



Intel® Advisor User Guide

Intel Corporation

Contents

Chapter 1: Intel® Advisor User Guide

Introduction	7
What's New in Intel® Advisor.....	8
Design and Optimization Methodology	12
Tutorials and Samples.....	16
Get Help and Support	17
Install and Launch Intel® Advisor	17
Install Intel® Advisor.....	18
Set Up Environment Variables	19
Set Up System to Analyze GPU Kernels	21
Set Up Environment to Offload SYCL, OpenMP* target, and OpenCL™	23
Applications to CPU	23
Launch Intel® Advisor	24
GUI Navigation Quick Start.....	27
Set Up Project.....	29
Configure Target Application	29
Limit the Number of Threads Used by Parallel Frameworks	30
Choose a Small, Representative Data Set	30
Build Target Application.....	31
Create Project.....	36
Configure Project	37
Configure Binary/Symbol Search Directories	45
Configure Source Search Directory	46
Binary/Symbol Search and Source Search Locations	46
Analyze Vectorization Perspective	48
Run Vectorization and Code Insights Perspective from GUI.....	50
Vectorization Accuracy Presets.....	51
Customize Vectorization and Code Insights Perspective	52
Run Vectorization and Code Insights Perspective from Command Line	58
Vectorization Accuracy Levels in Command Line	60
Explore Vectorization and Code Insights Results	62
Vectorization Report Overview	65
Examine Not-Vectorized and Under-Vectorized Loops	67
Analyze Loop Call Count	70
Investigate Memory Usage and Traffic.....	71
Find Data Dependencies.....	74
Analyze CPU Roofline	75
Run CPU / Memory Roofline Insights Perspective from GUI.....	77
CPU Roofline Accuracy Presets	78
Customize CPU / Memory Roofline Insights Perspective	79
Run CPU / Memory Roofline Insights Perspective from Command Line	84
CPU Roofline Accuracy Levels in Command Line	87
Explore CPU/Memory Roofline Results	88
CPU Roofline Report Overview	91
Examine Bottlenecks on CPU Roofline Chart.....	97
Examine Relationships Between Memory Levels	100
Compare CPU Roofline Results	105
Model Threading Designs.....	107

Run Threading Perspective from GUI.....	109
Customize Threading Perspective	110
Run Threading Perspective from Command Line	116
Threading Accuracy Levels in Command Line	118
Annotate Code for Deeper Analysis.....	119
Annotate Code to Model Parallelism	120
Annotations	131
Annotation Report.....	163
Explore Threading Results	169
Model Threading Parallelism.....	172
Suitability Report Overview	175
Choose Modeling Parameters in the Suitability Report	179
Fix Annotation-related Errors Detected by the Suitability Tool.....	181
Advanced Modeling Options.....	182
Reduce Parallel Overhead, Lock Contention, and Enable Chunking	183
Check for Dependencies Issues	184
Code Locations Pane	185
Filter Pane (Dependencies Report).....	186
Problems and Messages Pane	187
Dependencies Source Window	188
Add Parallelism to Your Program.....	192
Before You Add Parallelism: Choose a Parallel Framework.....	193
Add the Parallel Framework to Your Build Environment.....	196
Annotation Report.....	200
Replace Annotations with Intel® oneAPI Threading Building Blocks (oneTBB) Code	201
Replace Annotations with OpenMP* Code	206
Next Steps for the Parallel Program	219
Model Offloading to a GPU	221
Run Offload Modeling Perspective from GUI	223
Offload Modeling Accuracy Presets	225
Customize Offload Modeling Perspective	226
Run Offload Modeling Perspective from Command Line	233
Offload Modeling Accuracy Levels in Command Line	245
Run GPU-to-GPU Performance Modeling from Command Line	249
Explore Offload Modeling Results	253
Offload Modeling Report Overview	257
Examine Regions Recommended for Offloading	260
Examine Data Transfers for Modeled Regions	262
Check for Dependency Issues	266
Explore Performance Gain from GPU-to-GPU Modeling.....	267
Investigate Non-Offloaded Code Regions	270
Advanced Modeling Configuration	278
Model Application Performance on a Custom Target GPU Device ..	278
Check How Assumed Dependencies Affect Modeling	282
Manage Invocation Taxes	284
Enforce Offloading for Specific Loops	286
Analyze GPU Roofline	287
Run GPU Roofline Insights Perspective from GUI.....	288
GPU Roofline Accuracy Presets	289
Customize GPU Roofline Insights Perspective	290
Run GPU Roofline Insights Perspective from Command Line	295
GPU Roofline Accuracy Levels in Command Line	298
Explore GPU Roofline Results	300
Examine GPU Roofline Summary	303

Examine Bottlenecks on GPU Roofline Chart	305
Examine Kernel Details	310
Compare GPU Roofline Results	314
Design and Analyze Flow Graphs	316
Where to Find the Flow Graph Analyzer	316
Launching the Flow Graph Analyzer	316
Flow Graph Analyzer GUI Overview.....	317
Menus	319
Toolbars	321
Tabs.....	322
Main Canvas.....	325
Charts	325
Flow Graph Analyzer Workflows.....	327
Designer Workflow	327
Adding Nodes, Edges, and Ports.....	328
Modifying Node Properties.....	329
Viewing Edge Properties.....	331
Validating a Graph	331
Saving a Graph to a File.....	331
Generating C++ Stubs.....	332
Preferences	335
Scalability Analysis.....	338
Activating the Graph.....	338
Scalability Analysis Prerequisites	338
Running the Scalability Analysis	341
Exploring the Parallelism in a Concurrent Node	341
Showing Non-Parallel Nature of a Serial Node	341
Explore Parallelism Provided by the Topology of a Graph.....	342
Understanding Analysis Color Codes	344
Collecting Traces from Applications	344
Building an Application for Trace Collection	345
Collecting Trace Files	346
Nested Parallelism in Flow Graph Analyzer	352
Analyzer Workflow.....	353
Find Time Regions of Low Concurrency and Their Cause	353
Finding a Critical Path	354
Finding Tasks with Small Durations.....	355
Reduce Scheduler Overhead using Lightweight Policy	356
Identifying Tasks that Operate on Common Input	358
Support for SYCL	359
Experimental Support for OpenMP* Applications.....	367
Collecting Traces for OpenMP* Applications.....	368
OpenMP* Constructs in the Per-Thread Task View	369
OpenMP* Constructs in the Graph Canvas	370
Sample Trace Files.....	374
code_generation Samples	375
performance_analysis Samples	376
Additional Resources	380
Minimize Analysis Overhead.....	380
Collection Controls to Minimize Analysis Overhead	384
Loop Markup to Minimize Analysis Overhead	391
Filtering to Minimize Analysis Overhead.....	396
Execution Speed/Duration/Scope Properties to Minimize Analysis Overhead	397
Miscellaneous Techniques to Minimize Analysis Overhead	400

Analyze MPI Applications	403
Model MPI Application Performance on GPU	408
Control Collection with an MPI_Pcontrol Function	412
Manage Results	413
Open a Result	414
Rename an Existing Result	415
Delete a Result	415
Save Results to a Custom Location	415
Work with Standalone HTML Reports	416
Create a Read-only Result Snapshot	420
Create a Result Snapshot Dialog Box	421
Open a Result as a Read-only File in Visual Studio	422
Command Line Interface	423
advisor Command Line Interface Reference	423
advisor Command Action Reference	425
advisor Command Option Reference	435
Offload Modeling Command Line Reference	567
run_oa.py Options	569
collect.py Options	575
analyze.py Options	580
Generate Pre-configured Command Lines	589
Troubleshooting	591
Error Message: Application Sets Its Own Handler for Signal	593
Error Message: Cannot Collect GPU Hardware Metrics for the Selected GPU Adapter	594
Error Message: Memory Model Cache Hierarchy Incompatible	594
Error Message: No Annotations Found	595
Error Message: No Data Is Collected	596
Error Message: Stack Size Is Too Small	596
Error Message: Undefined Linker References to dlopen or dlsym	597
Problem: Broken Call Tree	598
Problem: Code Region is not Marked Up	599
Problem: Debug Information Not Available	600
Problem: No Data	601
Problem: Source Not Available	602
Problem: Stack in the Top-Down Tree Window Is Incorrect	603
Problem: Survey Tool does not Display Survey Report	604
Problem: Unexpected C/C++ Compilation Errors After Adding Annotations	605
Problem: Unexpected Unmatched Annotations in the Dependencies Report	606
Warning: Analysis of Debug Build	607
Warning: Analysis of Release Build	607
Reference	608
Data Reference	608
CPU Metrics	608
Accelerator Metrics	626
Dependencies Problem and Message Types	671
Dangling Lock	673
Data Communication	674
Data Communication, Child Task	675
Inconsistent Lock Use	676
Lock Hierarchy Violation	677
Memory Reuse	679
Memory Reuse, Child Task	680

Memory Watch	681
Missing End Site	682
Missing End Task.....	683
Missing Start Site.....	683
Missing Start Task	684
No Tasks in Parallel Site	685
One Task Instance in Parallel Site	685
Orphaned Task	686
Parallel Site Information	687
Thread Information	688
Unhandled Application Exception.....	688
Recommendation Reference.....	689
Vectorization Recommendations for C++	689
Vectorization Recommendations for Fortran	716
User Interface Reference.....	737
Dialog Box: Corresponding Command Line	737
Dialog Box: Create a Project.....	738
Dialog Box: Create a Result Snapshot	738
Dialog Box: Options - Assembly	739
Editor Tab	740
Dialog Box: Options - General.....	741
Dialog Box: Options - Result Location	742
Dialog Box: Project Properties - Analysis Target	743
Dialog Box: Project Properties - Binary/Symbol Search	750
Dialog Box: Project Properties - Source Search	751
Pane: Advanced View	752
Pane: Analysis Workflow	754
Pane: Roofline Chart.....	756
Pane: GPU Roofline Chart.....	759
Project Navigator Pane	763
Toolbar: Intel Advisor	764
Annotation Report.....	765
Window: Dependencies Source	765
Window: GPU Roofline Regions	768
Window: GPU Roofline Insights Summary	771
Window: Memory Access Patterns Source.....	772
Window: Offload Modeling Summary	773
Window: Offload Modeling Report - Accelerated Regions.....	779
Window: Perspective Selector	781
Window: Refinement Reports	781
Window: Suitability Report	785
Window: Suitability Source.....	786
Window: Survey Report	787
Window: Survey Source.....	790
Window: Threading Summary.....	791
Window: Vectorization Summary.....	793
Appendix.....	794
Data Sharing Problems	794
Data Sharing Problem Types.....	795
Problem Solving Strategies.....	798
Notational Conventions	807
Key Concepts.....	808
Glossary	809
Parallelism	811
Related Information.....	815

1

Intel® Advisor User Guide

This document provides a detailed overview of the Intel® Advisor functionality, workflows, and instructions.

Intel® Advisor is composed of a set of tools, or *perspectives*, to help ensure your Fortran, C, C++, SYCL, OpenMP*, Intel® oneAPI Level Zero (Level Zero), and OpenCL™ applications realize full performance potential on modern processors:

- **Vectorization and Code Insights:** Identify high-impact, under-optimized loops, what is blocking vectorization, and where it is safe to force vectorization. It also provides code-specific how-can-I-fix-this-issue recommendations. For details, see [Analyze Vectorization Perspective](#).
- **CPU / Memory Roofline Insights** and **GPU Roofline Insights:** Visualize actual performance against hardware-imposed performance ceilings (rooflines). They provide insights into where the bottlenecks are, which loops are worth optimizing for performance, what are the likely causes of bottlenecks and what should be the next optimization steps. For details, see [Analyze CPU Roofline](#) or [Analyze GPU Roofline](#).
- **Offload Modeling:** Identify high-impact opportunities to offload to GPU as well as the areas that are not advantageous to offload. It provides performance speedup projection on accelerators along with offload overhead estimation and pinpoints accelerator performance bottlenecks. For details, see the [Model Offloading to a GPU](#).
- **Threading:** Analyze, design, tune, and check threading design options without disrupting your normal development. For details, see [Model Threading Designs](#).

Flow Graph Analyzer is a part of the Intel® Advisor installation. Use it to visualize and analyze performance for applications that use the Intel® oneAPI Threading Building Blocks (oneTBB) flow graph interfaces. For details, see [Flow Graph Analyzer](#).

Intel® Advisor is available as a standalone product and as part of the Intel® oneAPI Base Toolkit.

- [Standalone Intel® Advisor](#)
- [Intel® oneAPI Base Toolkit](#)

Documentation for older versions of Intel® Advisor is available for download only. For a list of available documentation downloads by product version, see these pages:

- [Download Documentation for Intel® Parallel Studio XE](#)
- [Download Documentation for Intel® System Studio](#)

Start Here

- [Design and Optimization Methodology](#)
- [What's New in Intel® Advisor](#)
- [Install and Launch Intel® Advisor](#)
- [Get Started with Intel Advisor](#)
- [Intel Advisor Cookbook](#)

Introduction

This document provides a detailed overview of the product functionality, workflows, or perspectives, and instruction to use Intel® Advisor.

Use Intel® Advisor to check that your application realize full performance potential on modern hardware platforms (CPU, GPU) and get recommendations for where to add optimization.

With the Intel Advisor, you can:

- [Model your application performance on an accelerator](#)
- [Visualize performance bottlenecks on a CPU or GPU with a Roofline chart](#)
- [Check vectorization efficiency](#)

- [Prototype threading designs](#)

What's New in Intel® Advisor

This topic lists new high-level features and improvements in Intel® Advisor. For a full list of new features, see [Intel Advisor Release Notes](#).

Intel® Advisor 2023.0

GPU Profiling and Roofline on PVC-XT:

- Get actionable advice to design code that runs optimally on Intel® Data Center GPU MAX Series (formerly code named Ponte Vecchio).
- Discover GPU application performance characterization, such as bandwidth sensitivity, instruction mix, and cache-line use.
- Automated Roofline Analysis helps to identify and prioritize memory, cache, or compute bottlenecks and understand their likely causes.

Intel® Advisor 2022.1

• Usability:

- **Performance metrics in GPU Roofline Source view in the Intel Advisor GUI**

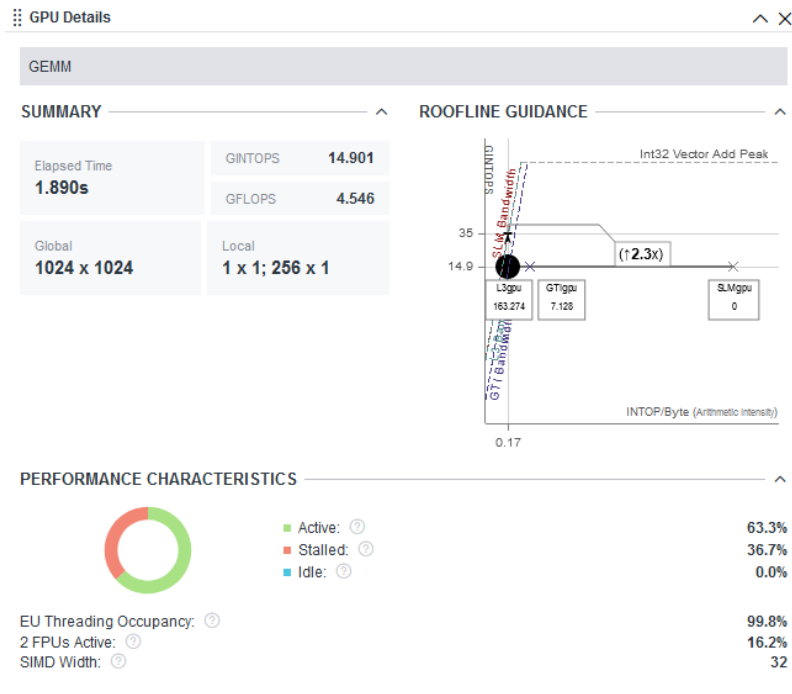
In the GPU Roofline Insights report, you can switch to **Source View** and examine the source code of your application with performance metrics for each kernel, such as elapsed time and memory traffic.

matrix_transpose_opt.cpp					
Line	Source	Elapsed Time	Traffic, GB	Instruction Count	SIMD Utilization
62					
63	h.parallel_for(
64	sycl::nd_range<2>(sycl::range<2>(N / blockSize,				
65	[=](sycl::nd_item<2> it) [[intel::reqd_sub_group	1.528s	475.781	54,983,917,568	29.0%
66	int gi = it.get_group(0);				
67	int gj = it.get_group(1);				
68					
69	sycl::ext::oneapi::sub_group sg = it.get_sub_group(0);				

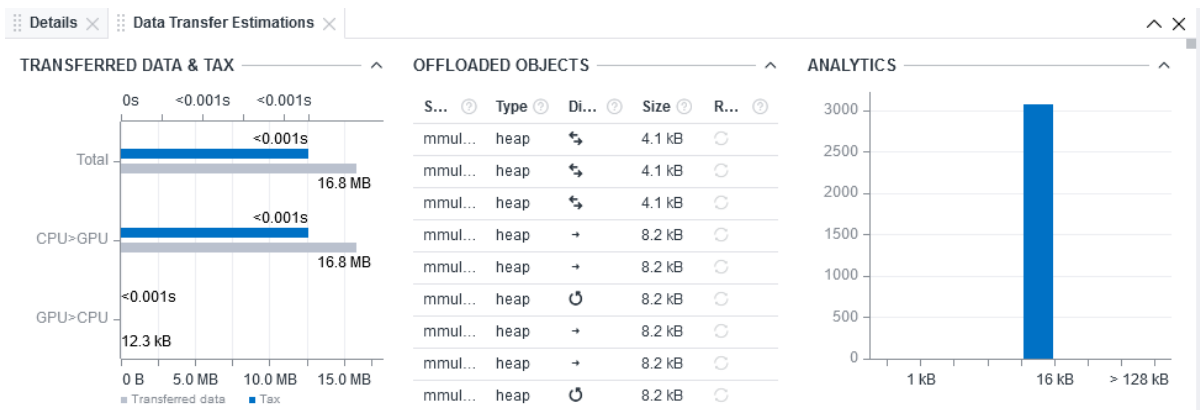
- **New panes in an interactive HTML Report: GPU Details in the GPU Roofline Insights perspective and Data Transfer Estimations in the Offload Modeling perspective**

The interactive HTML report, which combines Offload Modeling and GPU Roofline Insights results, now includes two new panes, which are similar to the panes with the same name in the Intel Advisor GUI report:

- GPU Roofline Insights perspective includes the **GPU Details** pane, which reports detailed code analytics for a selected kernel, such as Roofline guidance with the main limiting roof and estimated speedup after optimization, compute and memory bandwidth, memory usage metrics.



- Offload Modeling perspective includes the **Data Transfer Estimations** pane, which reports estimated data transferred between host and target devices in each direction and a list of offloaded objects.

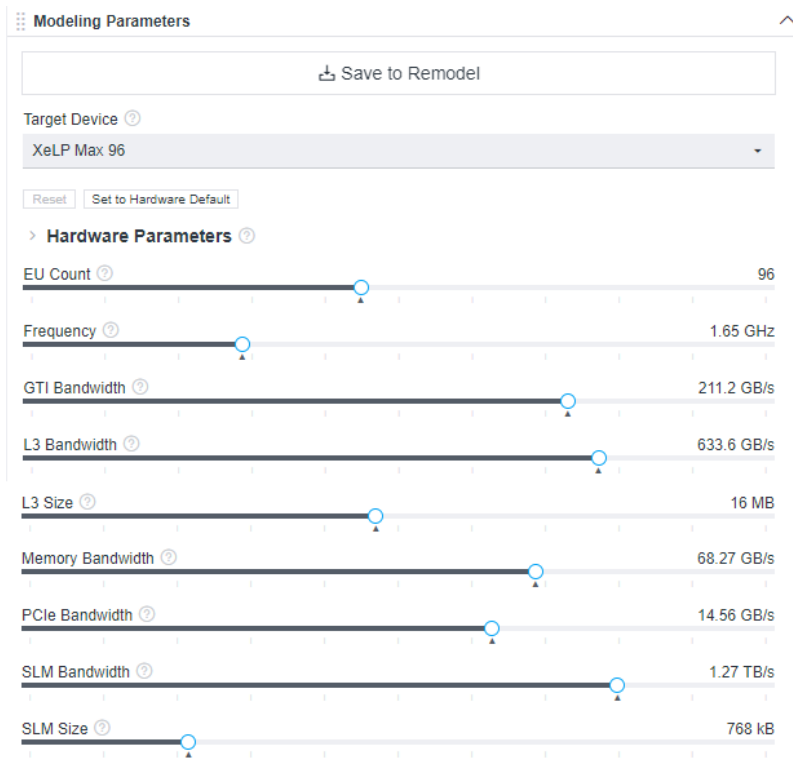


You can use the interactive HTML reports to analyze Intel Advisor results on a remote machine using your web browser or share the results. See [Work with Standalone HTML Reports](#) for details.

- Offload Modeling:**

- Adjustable hardware parameters in an interactive HTML report and GUI report**

Offload Modeling report includes a new **Modeling Parameters** pane, which shows available target devices for modeling and hardware configuration parameters for a selected device. Each parameter is a slider that you can adjust to a desired value to get a custom configuration for remodeling.



The pane is available in the interactive HTML report and Intel Advisor GUI report and has the same functionality. You can use it to:

- Examine device parameters that the application performance was modeled on to understand how they affect the estimated performance.
- Change the target device to compare the new configuration with the current modeled device.
- Adjust the parameters and remodel performance for a custom device. You can experiment with parameters to see how they affect the application performance or adjust the configuration to model performance for a future or a specific device not listed in the target devices. See the sections below for a full workflow.

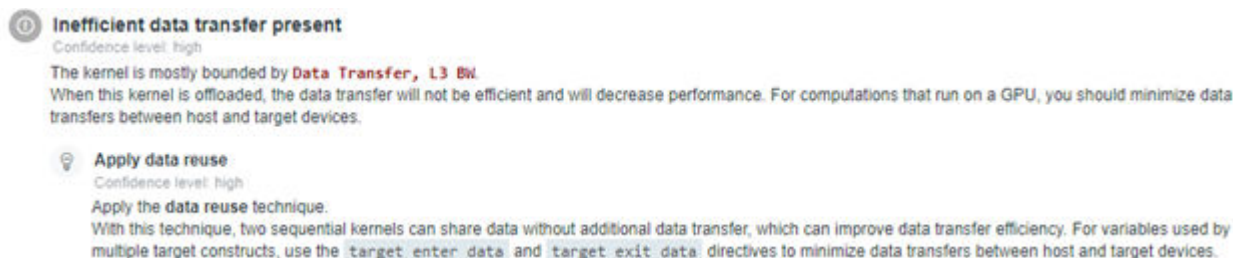
For CPU-to-GPU modeling, you can remodel performance using Intel Advisor CLI only.

See [Model Application Performance on a Custom Target GPU Device](#) for more information about how to work with the pane.

- **New recommendation to optimize data transfer costs with data reuse when porting your application from a CPU to a GPU**

Offload Modeling perspective introduces a new actionable recommendation for optimizing data transfer costs with data reuse before porting your application from a CPU to a GPU. Data reuse can help you improve the application performance on the GPU by optimizing data transfer efficiency.

The recommendation is reported in a Recommendations pane of the Accelerated Regions tab. The recommendation includes estimated performance characteristics and data reuse gain, as well as code snippet examples for applying data reuse techniques.



See [Examine Data Transfers for Modeled Regions](#) for details.

- **Documentation:**

- **Sample-based scenario for the Offload Modeling perspective in the Get Started with Intel Advisor document**

[Identify High-impact Opportunities to Offload to GPU](#) topic in the get started guide now uses a sample to introduce the main Offload Modeling features. You can download the sample or use your own application to follow this topic instructions and understand the basic Offload Modeling workflow with the Intel Advisor.

The following topics in the get started guide with a sample-based scenario are also available:

- [Discover Where Vectorization Pays Off The Most](#)
- [Prototype Threading Designs](#)

Intel® Advisor 2022.0

- **GPU Roofline:**

- **New recommendation to optimize GPU general purpose register file (GRF) usage and improve performance.**

The GPU Roofline Insights perspective introduces actionable recommendations for improving your application performance on GPU by optimizing GRF usage. The recommendations are reported in a **Recommendations** pane in the **GPU Roofline Regions** report. See [Get Recommendations](#) for details.

- **New GPU memory and compute metrics.**

The **GPU** pane in the **GPU Roofline Regions** tab introduces several new metrics. Some of the new metrics are:

Memory metrics:

- [GPU memory](#) usage summary
- [L3 shader](#) usage summary
- [Shared Local Memory \(SLM\)](#) usage summary
- [Register spilling](#) detection

Compute metrics:

- [FLOP operation](#) summary
- [INT operation](#) summary

- **Documentation:**

- **Command-line cheat sheet for quick reference**

Introduced a new downloadable [Intel Advisor command-line cheat sheet](#), which lists the most useful command-line interface (CLI) options. You can use this print-friendly PDF for quick reference on the Intel Advisor CLI.

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

NOTE

- [Download Documentation for Intel® Parallel Studio XE](#)
 - [Download Documentation for Intel® System Studio](#)
-

Design and Optimization Methodology

Intel® Advisor helps you to design and optimize high-performing Fortran, C, C++, SYCL, OpenMP*, and OpenCL™ code to realize full performance potential on modern computer architecture. You can measure your application performance, collect required data, and look at your code from different *perspectives* depending on your goal to dig deeper and get hints for optimization.

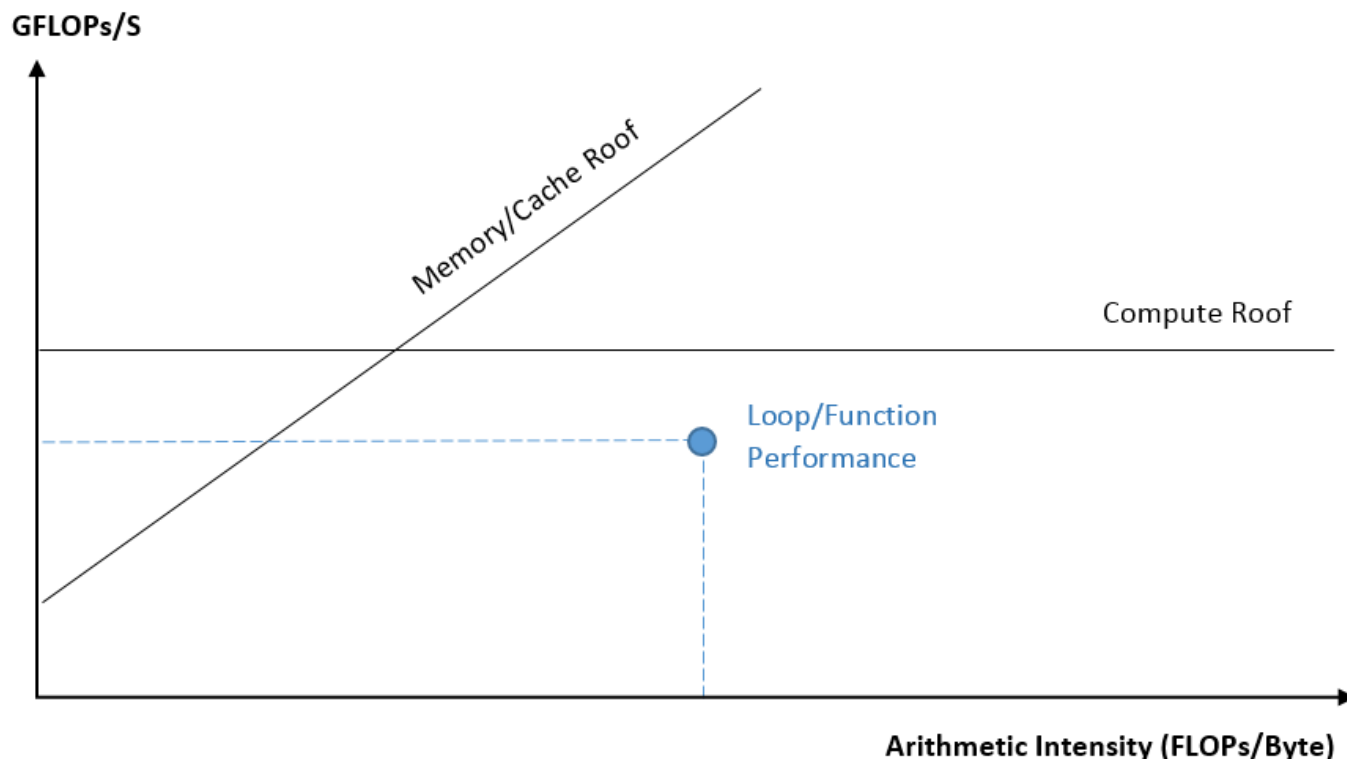
Visualize Performance Bottlenecks with Roofline Chart

When optimizing your C, C++, SYCL, or Fortran application, it is useful to know application's current and potential performance in relation to hardware-imposed limitations like memory bandwidth and compute capacity of a target platform that it runs on - a CPU or a GPU.

Roofline model of the Intel Advisor visualizes actual performance against hardware-imposed performance ceilings and helps you determine the main limiting factor (memory bandwidth or compute capacity) to provide an ideal road map of potential optimization steps. This analysis highlights loops that have the most headroom for improvement, which allows you to focus on areas that deliver the biggest performance payoff.

To generate a Roofline report, the Intel Advisor:

- Collects loop/function (for CPU) or OpenCL™ kernels (for GPU) timings and memory data.
- Measures the hardware limitations and collects floating-point and integer operations data.



The Roofline chart plots an application achieved performance and arithmetic intensity against the hardware maximum achievable performance:

- Arithmetic intensity (x axis) - measured in number of floating-point operations (FLOPs) and/or integer operations (INTOPs) per byte, based on the loop/function algorithm, transferred between CPU/VPU/GPU and memory.
- Performance (y axis) - measured in billions of floating-point operations per second (GFLOPS) and/or billions of integer operations per second (GINTOPS).

With the data collected, the Intel Advisor plots the Roofline chart:

- Execution time of each loop/function/kernel is reflected in the size and color of each dot. The dots on the chart correspond to OpenCL kernels for GPU Roofline, while for the CPU Roofline, they correspond to individual loops/functions.
- *Memory* bandwidth limitations are plotted as diagonal lines.
- *Compute* capacity limitations are plotted as horizontal lines.

For details on how to get the Roofline report and read the results, see [CPU / Memory Roofline Insights Perspective](#) or [GPU Roofline Insights Perspective](#).

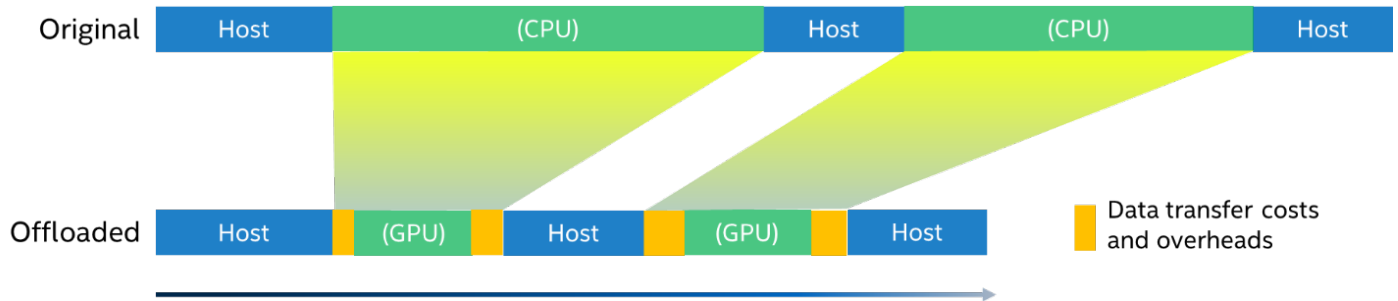
Model Offloading to Accelerator

When designing your application to offload to an accelerator, you might first want to:

- Estimate the offload benefit and overhead for each loop/function in your original C++ or Fortran code to make better decisions on which parts of code to offload
- Check performance gain for a SYCL, OpenCL™, or OpenMP* target application if you offload it to a different accelerator

Offload Modeling perspective of the Intel® Advisor can identify high-impact portions of a code that are profitable to offload to a target platform (for example, to a GPU) as well as the code regions that are not advantageous to offload. It can also predict the code performance if run on the target platform and lets you experiment with accelerator configuration parameters.

Offload Modeling takes measured baseline metrics and application characteristics as an input and applies an analytical model to estimate execution time and characteristics on a target platform.



Offload Modeling is based on three models:

- **Compute throughput model** counts arithmetic operations in a region on a baseline platform and estimates the execution time on a target platform required to achieve the same mix of arithmetic operations, considering it as bound by compute engines only.
- **Memory sub-system throughput model** traces memory accesses inside a region on a baseline platform and estimates the execution time on a target platform needed to transfer the same amount of memory. Memory traffic is measured using a cache simulator that reflects the target platform's memory configuration.
- **Offload data transfer analysis** measures memory accesses that are read from or written to a region and will need to be sent over a PCIe* if the region is offloaded to a target platform.

For details on how to run the Offload Modeling perspective and read the reports, see [Offload Modeling Perspective](#).

Check Vectorization Efficiency

Modern Intel® processors have extensions that support SIMD (single instruction, multiple data) parallelism with Intel® Streaming SIMD Extensions (Intel® SSE), Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® Advanced Vector Extensions 512 (Intel® AVX-512). To take advantage of SIMD instructions with the expanded vector width and achieve higher performance, applications need to be vectorized.

You can rely on your desired compiler - Intel® C++ Compiler Classic, Intel® Fortran Compiler Classic, Intel® oneAPI DPC++/C++ Compiler, GNU Compiler Collection (GCC)* - to auto-vectorize some loops, but serial constraints of programming languages limit the compiler's ability to vectorize some loops. The need arose for explicit vector programming methods to extend vectorization capability for supporting reductions, vectorizing:

- Outer loops
- Loops with user-defined functions
- Loops that the compiler assumes to have data dependencies

To improve the performance of CPU-bound applications on modern processors with vector processing units, you might use explicit vector programming apply structural changes for thread-level parallelism and SIMD-level parallelism.

Use the Vectorization and Code Insights perspective of the Intel Advisor to analyze your application run time behavior and identify application parts that will benefit most from vectorization. Vectorization and Code Insights perspective helps you to achieve the best performance using vectorization and identify:

- Where vectorization, or parallelization with threads, will pay off the most
- If vectorized loops are providing benefit, and if not, why not
- Un-vectorized loops and why they are not vectorized
- Performance problems in general

For details on how to run the perspective and read the reports, see [Vectorization and Code Insights Perspective](#).

Prototype Threading Designs

The best performance improvements from adding parallel execution (parallelism) to a program occur when many cores are busy most of the time doing useful work. Achieving this requires a lot of analysis, knowledge, and testing.

Because your serial program was not designed to allow parallel execution, as you convert parts of it to use parallel execution, you may encounter unexpected errors that occur only during parallel execution. Instead of wasting effort on portions of the program that use almost no CPU time, you should focus on the hotspots, and the functions between the main entry point and each hotspot.

If you add parallel execution to a program without proper preparation, unpredictable crashes, program hangs, and wrong answers can result from incorrect parallel task interactions. For example, you may need to add synchronization to avoid incorrect parallel task interactions, but this must be done carefully because locking overhead and serial synchronization can reduce the benefits of the parallel execution.

Threading perspective of the Intel Advisor helps you quickly prototype multiple threading options, project scaling on larger systems, optimize faster, and implement with confidence.

- Identify issues and fix them before implementing parallelism
- Add threading to C, C++, and Fortran code
- Prototype the performance impact of different threaded designs and project scaling on systems with larger core counts without disrupting development or implementation
- Find and eliminate data-sharing issues during design (when they're less expensive to fix)

The high-level parallel frameworks available for each programming language include:

Language	Available High-Level Parallel Frameworks
C	OpenMP
C++	Intel® oneAPI Threading Building Blocks (oneTBB) OpenMP
Fortran	OpenMP

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

For details on how to run the perspective and read the reports, see [Threading Perspective](#).

Using Amdahl's Law and Measuring the Program

There are two rules of optimization that apply to parallel programming:

- Focus on the part of the program that uses the most time.
- Do not guess, measure.

Amdahl's Law

In the context of parallel programming, Gene Amdahl formalized a rule called Amdahl's Law, which states that the speed-up that is possible from parallelizing one part of a program is limited by the portion of the program that still runs serially.

The consequence may be surprising: parallelizing the part of your program where it spends 80% of its time cannot speed it up by more than a factor of five, no matter how many cores you run it on.

Therefore, to get maximum benefit from parallelizing your program, you could add parallelism to all parts of your program as suggested by Amdahl's Law. However, it is more practical to find where it spends most of its time and focus on areas that can provide the most benefit.

Do Not Guess - Measure

This leads to another rule of optimization: *Do Not guess - Measure*. Programmers' intuitions about where their programs are spending time are notoriously inaccurate. Intel® Advisor includes a Survey tool you can use to profile your running program and measure where it spends its time.

After you add Intel® Advisor annotations to your program to mark the proposed parallel code regions, run the Suitability tool to predict the approximate maximum performance gain for the program and the annotated sites. These estimated performance gain values are based on a model of parallel execution that reflects the impact of Amdahl's law.

Tutorials and Samples

Intel® Advisor provides tutorials with step-by-step instructions on analyzing performance of applications with sample code.

Discover Where Vectorization Pays Off The Most

Get Started Guide: [Discover Where Vectorization Pays Off The Most](#)

Sample: included in the product package

Learning Objective: Use Vectorization report to:

- Identify loops that will benefit most from vectorization.
- Identify what is blocking effective vectorization.
- Increase the confidence that vectorization is safe.
- Explore the benefit of alternative data reorganizations.

Prototype Threading Designs

Get Started Guide: [Prototype Threading Designs](#)

Sample: included in the product package

Learning Objective: Demonstrates an end-to-end workflow you can ultimately apply to your own applications:

1. Survey the target executable to locate the loops and functions where your application spends the most time.
2. In the target sources, add Intel Advisor annotations to mark possible parallel tasks and their enclosing parallel sites.
3. Check Suitability to predict the maximum parallel performance speedup of the target based on these annotations.
4. Check Dependencies to predict parallel data sharing problems in the target based on these annotations.
5. If the predicted maximum speedup benefit is worth the effort to fix the predicted parallel data sharing problems, fix the problems.
6. Recheck Suitability to see how your fixes impact the predicted maximum speedup.
7. If the predicted maximum speedup benefit is still worth the effort to add parallelism to the target, replace the annotations with parallel framework code that enables parallel execution.

Use the Automated Roofline Chart to Make Optimization Decisions - C++ Sample

Windows* OS Tutorial: [HTML](#)

Sample: [Download](#) `roofline_demo_samples` sample. You can download source code or pre-collected results to save time.

Duration: 20 minutes (with pre-collected results)

Learning Objective: Use the Roofline chart to answer the following questions:

- What is the maximum achievable performance with your current hardware resources?
- Does your application work optimally on current hardware resources?
- If not, what are the best candidates for optimization?
- Is memory bandwidth or compute capacity limiting performance for each optimization candidate?

NOTE

- Sample applications are non-deterministic.
 - Sample applications are designed only to illustrate the Intel Advisor features and do not represent best practices for creating and optimizing code.
-

Get Help and Support

This topic explain the different options for accessing the Help documentation and technical support for Intel® Advisor.

Get Help

The documents provided with this release are available in HTML format. You can access the documentation:

- **For Windows* OS only:** From the **Start** menu, or **Start** screen, under the **Intel oneAPI [version]** group.
- **Help > Intel Advisor [version]**
- Access context-sensitive Help on active GUI elements:
 - In the **Advisor Workflow** tab and in the **Result** tab, click certain links to get specific help related to the underlined word.
 - In the **Result** tab, you can right-click an element to display its context menu. Certain context menus display a **What Should I Do Next?** menu item. Choose this menu item to get help specific to the active user interface element.
 - **F1 Help:** Press F1 to get help for an active dialog box, property page, pane, or window.

Get Support

The following links provide information and support on Intel® software products, including developer suite products:

- <https://www.intel.com/content/www/us/en/develop/tools.html>

At this site, you will find comprehensive product information, including:

- Links to each product, where you will find technical information such as white papers and articles
- Links to user forums
- Links to news and events
- <https://www.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit.html>

Intel® oneAPI Base Toolkit product page with links to download, support forums, knowledge base, and product documentation.

For detailed system requirements and additional support information, see the product [Release Notes](#).

Install and Launch Intel® Advisor

The following sections provide simple steps to quickly configure and run the Intel® Advisor graphical user interface (GUI) or command line interface (CLI).

- [Install Intel Advisor](#) as part of the Intel® oneAPI Base Toolkit or standalone.
- [Set up Intel Advisor environment variables](#) to launch Intel Advisor from command line or a terminal.
 - *Optional:* To analyze GPU kernels of your SYCL, OpenMP* target, or OpenCL™ application with the GPU Roofline Insights or Offload Modeling perspective, [configure your system to analyze GPU kernels](#).
 - *Optional:* To analyze SYCL, OpenMP target, or OpenCL application running on a CPU with the Offload Modeling perspective, [set up your system to offload the application to CPU](#).
- [Launch the Intel Advisor](#).

Quick steps to ramp up with the Intel Advisor are included in [Getting Started with Intel Advisor](#).

Install Intel® Advisor

Use this topic to download and install Intel® Advisor using oneAPI Installer and yum/APT package managers.

Intel® Advisor is available for download as:

- [Standalone installation](#)
- [Part of Intel® oneAPI Base Toolkit](#)

Depending on your internet connection, choose local or online installer.

To install Intel Advisor as **part of Intel® oneAPI Base Toolkit**, refer to [Installation Guide for Intel® oneAPI Toolkits](#).

NOTE Different major versions can co-exist with each other, but on Windows* OS, only one version of Intel Advisor can be integrated with Visual Studio* IDE.

On Windows* OS

1. Double-click the compressed self-extracting executable file as a user with administrative privileges.
2. To get a complete set of user interfaces (GUI front end and Visual Studio* IDE integration), select the **Recommended Installation** option. The default installation path is C:\Program Files (x86)\Intel\oneAPI. To change the installation path, select the **Custom Installation** option.

NOTE To perform silent, non-interactive installation, refer to [Intel® oneAPI Toolkits Installation Guide for Windows*](#).

3. Click the **Install** button to complete the installation.

On Linux* OS

1. Make sure to have read/write permissions for the /tmp directory and start the installation.
 - To install on the local system, run the installer using the following command:

```
sh <package-name>.sh
```

- If you want to install Intel Advisor for use by any user, you must do this as a root user. To install Intel Advisor to a network-mounted drive or shared file systems available for multiple users, run the following command:

```
sh <package-name>.sh --SHARED_INSTALL
```

2. To get a complete set of user interfaces (GUI front end and Eclipse* IDE integration), select the **Recommended Installation** option. The default installation path is /opt/intel/oneapi for root users and \$HOME/intel/oneapi for non-root users. To change the installation path, select the **Custom Installation** option.

3. Integrate Intel Advisor with Eclipse IDE by specifying the path to Eclipse IDE executable. Skip this step if you prefer to install Intel Advisor without integration into Eclipse IDE.
4. Click the **Install** button to complete the installation.

NOTE Intel Advisor is available for installation via [yum](#) and [APT](#) package managers.

System Requirements

See the list of [System Requirements](#) for more information.

See Also

After installation, consider the following next steps:

- [Set Up Environment Variables](#)
- [Set Up Environment to Analyze GPU Kernels](#)
- [Set Up Environment to Model Performance on GPU-Enabled Applications](#)

Set Up Environment Variables

Use this topic to get guidance on setting up environment variables for Intel® Advisor.

Set the environment variables if you want to:

- Run Intel Advisor command line interface
- Run Intel Advisor graphical user interface from command line (for example, on Linux OS)
- Compile your application with *Intel Advisor annotations* using additional include directories, so the compiler can find the include file that defines annotations

You can set the variables using *one* of the following methods:

- Recommended: Set up variables using a script.
- Set up variables manually. Use this method to set up variables for a custom Intel Advisor location or to set the variables permanently.

Default Installation Paths

In the instructions below, be sure to replace any values in brackets, such as `<version>` or `<install-dir>`. `<version>` is the Intel Advisor year and update version (for example, 2021.1). The default installation path for the application, `<install-dir>`, can be one the following:

- On **Linux* OS**:
 - `/opt/intel/oneapi` for root users
 - `$HOME/intel/oneapi` for non-root users
- On **Windows* OS**: `C:\Program Files (x86)\Intel\oneAPI`
For 32-bit systems, the `Program Files (x86)` folder is `Program Files`.
- On **macOS***: `/opt/intel/oneapi`

Set Up Environment Variables via Script

This is the recommended method to set up the Intel Advisor environment variables. In particular, use it if you want to run the Offload Modeling using the dedicated Python* scripts. The script automatically sets up all the required variables pointing to the Intel Advisor installation directory.

Linux OS and macOS

Run *one* of the following shell scripts:

```
source <install-dir>/setvars.sh
source <install-dir>/setvars.csh
```

The scripts set up the environment for the *latest* Intel Advisor version installed on your system.

NOTE If you want to set up environment for a lower version of the Intel Advisor installed on your system, also run one of the following Intel Advisor-specific scripts:

```
source <install-dir>/advisor/<version>/env/vars.sh
source <install-dir>/advisor/<version>/env/vars.csh
```

where *<version>* is the Intel Advisor version you want to use.

Windows OS

Run the following batch script:

```
<install-dir>\setvars.bat
```

The script sets up the environment for the latest Intel Advisor version installed on your system.

NOTE If you want to set up environment for a lower version of the Intel Advisor installed on your system, also run one of the following Intel Advisor-specific scripts:

```
source <install-dir>/advisor/<version>/env/vars.sh
source <install-dir>/advisor/<version>/env/vars.csh
```

where *<version>* is the Intel Advisor version you want to use.

Set Up Environment Variables Manually

Linux OS and macOS

1. Open a terminal.
2. Check the current definition of the environment variable. For example, with the `bash` shell, type:

```
env | grep ADVISOR_<version-year>_DIR
```

where *<version-year>* is a major Intel Advisor version installed on your system. For example, 2021.

If the variable is defined and points to the correct Intel Advisor installation directory, skip the steps below and continue to [launch the Intel Advisor](#).

3. Set the environment variable using the `export` command. Enter:

```
export ADVISOR_<version-year>_DIR="<install-dir>"
```

For example, for the Intel Advisor 2022 in the default installation directory:

```
export ADVISOR_2022_DIR="/opt/intel/oneapi/advisor/latest"
```

4. To set this variable permanently on the current system, add this definition to your `.login` or a similar shell initialization file.
5. Check the definition of the environment variable set:

```
env | grep ADVISOR_<version-year>_DIR
```

You should see the environment variable with its value printed to the terminal.

Windows OS

1. Open a command prompt.
2. Check the current definition of the environment variable. For example, type:


```
set ADVISOR_<version-year>_DIR
```

where <version-year> is a major Intel Advisor version installed on your system. For example, 2021.

If the variable is defined and points to the correct Intel Advisor installation directory, skip the steps below and continue to [launch the Intel Advisor](#).

3. Use a `set` command to set the environment variable. Type:

```
set ADVISOR_<version-year>_DIR="<install-dir>"
```

For example, for the Intel Advisor 2022:

```
set ADVISOR_2022_DIR="C:\Program Files (x86)\Intel\oneAPI\advisor\latest"
```

4. To set this variable permanently on the current system, add this definition to your system or user environment variables using **Control Panel > System and Security > System > Advanced system settings > Environment Variables...**

Additional Variables

Consider setting the following environment variables:

- To determine whether evaluation features have been activated, set the `ADVISOR_EXPERIMENTAL` environment variable.
- To locate the Intel® oneAPI Threading Building Blocks (oneTBB) include directory when working with programs that use oneTBB, set the `TBBROOT` environment variable. See [Defining the TBBROOT Environment Variable](#).
- **On Linux OS and macOS:** set the `BROWSER` environment variable to locate an installed HTML browser. This enables the display of Get Started, Tutorials or Help from the Intel® Advisor GUI **Help** menu.
- **On Linux OS and macOS:** set the `VISUAL` or `EDITOR` environment variable to specify an external editor to launch when you double-click a line in a **Source** window. `VISUAL` takes precedence over `EDITOR`.

Next Steps

[Launch Intel Advisor](#) from GUI or from command line interface.

See Also

[Set Up System to Analyze GPU Kernels](#)

[Set Up Environment to Offload SYCL, OpenMP* target, and OpenCL™ Applications to CPU](#)

[Limit the Number of Threads Used by Parallel Frameworks](#)

[Intel Advisor Annotation Definitions File](#)

Set Up System to Analyze GPU Kernels

To analyze performance of GPU kernels in your SYCL, OpenMP* target, or OpenCL™ application with the *GPU Roofline Insights* or *GPU-to-GPU Offload Modeling* perspective, you need to configure your system properly:

1. Make sure you have the Intel® Metrics Discovery Application Programming Interface. The library is included with the Intel® Advisor.
2. Install and configure a graphics processing unit (GPU) driver for your system.
3. For Linux* OS: Set up environment variables.

Important For the Offload Modeling perspective, make sure the kernels run with the oneAPI Level Zero back end.

Install Intel® Metrics Discovery Application Programming Interface

To collect GPU hardware metrics and GPU utilization data, Intel Advisor uses the Intel Metric Discovery Application Programming Interface library. This library is *delivered* with the Intel Advisor. If you already have the library installed and you want to use your local library, make sure you have the correct version as explained below.

NOTE If you see the *Cannot Collect GPU Hardware Metrics for the Selected GPU Adapter* error message, install the library as follows. The message means the Intel Advisor cannot access the library.

Windows* OS

Intel Metric Discovery Application Programming Interface library is part of a GPU driver package. You should have a driver version higher than 27.20.100.8280 for your system.

If you have a lower version of the driver, you can download it from <https://downloadcenter.intel.com/>.

Linux* OS

Intel Metrics Discovery Application Programming Interface library is supported on Linux OS with kernel version 4.14 or higher. You should have the Intel Metric Discovery Application Programming Interface library 1.6.0 or higher to support the selection of video adapters.

If you have a lower version of the library, you can build and install it from <https://github.com/intel/metrics-discovery>.

Install a GPU driver

To collect GPU hardware metrics, install Intel® software packages for general purpose GPU capabilities.

On Windows OS, install a GPU driver for your system from [Download Center](#).

On Linux OS, follows the instructions in the [GPGPU Installation Guides](#) to install and configure drivers for your operating system.

Set Up Environment Variables

On *Windows OS*, run the Survey step of the perspective as an *Administrator*.

On *Linux OS*, run the Survey step of the perspective with *root* privileges.

If you *do not* have root permissions on Linux OS, enable collecting GPU hardware metrics for non-privileged users as follows:

1. Add your username to the video group.

- a. To check if you are already in the `video` group, run:

```
groups | grep video
```

- b. If you are not part of the `video` group, add your username to it:

```
sudo usermod -a -G video <username>
```

- c. Type `groups` to verify that you successfully added your username to the video group . If video is not listed, log out and log back in.

2. For Ubuntu* 19.10 and higher: Add your username to the `render` group.

- a. To check if you are already in the `render` group, run:

```
groups | grep render
```

- b. If you are not part of the `render` group, add your username to it:

```
sudo usermod -a -G render <username>
```

- c. Type `groups` to verify that you successfully added your username to the `render` group . If `render` is not listed, log out and log back in.
3. Set the value of the `dev.i915.perf_stream_paranoid` `sysctl` option to 0:

```
sysctl -w dev.i915.perf_stream_paranoid=0
```

NOTE This command makes a temporary change that is lost on the next reboot. To change this option permanently, run:

```
echo dev.i915.perf_stream_paranoid=0 > /etc/sysctl.d/60-mdapi.conf
```

4. Open the grub file in the `/etc/default` directory.
5. Find `GRUB_CMDLINE_LINUX_DEFAULT` and type `i915.enable_hangcheck=0` between the "" to disable time limit and run OpenCL™ kernel for a longer period of time. Save the file and close.
6. Run the following command to update the configuration:

```
sudo update-grub
```

Next Steps

- [Set up environment variables](#) and run the Intel Advisor from a command line interface.
- Run the Intel Advisor from a graphical user interface and [set up a project](#) if you do not have one.

See Also

Model Offloading to a GPU Find high-impact opportunities to offload/run your code and identify potential performance bottlenecks on a target graphics processing unit (GPU) by running the Offload Modeling perspective.

Analyze GPU Roofline Measure and visualize the actual performance of GPU kernels using benchmarks and hardware metric profiling against hardware-imposed performance ceilings, as well as determine the main limiting factor, by running the GPU Roofline Insights perspective.

Set Up Environment to Offload SYCL, OpenMP* target, and OpenCL™ Applications to CPU

If you have an application that contains SYCL, C++/Fortran with OpenMP* target, or OpenCL™ code and prepared for offloading to a target device, you can analyze and model its potential performance on a different target device with the it with the Intel® Advisor.

To do this, use *CPU offload profiling* to offload your code temporarily to a CPU so that you can profile it and model its performance with the Offload Modeling perspective.

Important Offload your SYCL, C++/Fortran with OpenMP target, or OpenCL code to CPU *only* to analyze it with the CPU-to-GPU Offload Modeling workflow. To analyze it with the *GPU-to-GPU Offload Modeling workflow* or *GPU Roofline workflow*, [configure your system](#) to analyze GPU kernels instead.

Depending on your operating system, do one of the following:

Linux* OS

1. For *SYCL code*: Force offloading to a CPU using one of the following:
 - *Recommended*: Set the `SYCL_DEVICE_FILTER` environment variable as follows:

```
export SYCL_DEVICE_FILTER=opencl:cpu
```

- If your application uses a SYCL device selector:

1. In the application source code, add the following to specify the CPU as the target device:

```
sycl::cpu_selector
```

For details, see [Device selectors](#) in the SYCL Reference.

2. Rebuild the application,
2. For *OpenMP code*: Force offloading to a CPU with one of the following:
 - *Recommended*: To offload code to CPU, set the following environment variables:

```
export OMP_TARGET_OFFLOAD=MANDATORY
```

```
export LIBOMPTARGET_DEVICE_TYPE=CPU
```

```
export LIBOMPTARGET_PLUGIN=OPENCL
```

- To run code natively on CPU, set the following variable:

```
export OMP_TARGET_OFFLOAD=DISABLED
```

3. If your application uses *OpenCL code*: Configure your code to be offloaded to a CPU. Refer to the OpenCL documentation at <https://www.khronos.org/registry/OpenCL/> for specific instructions.

Windows* OS

1. Set the following environment variable to use the JIT profiling API:

```
set INTEL_JIT_BACKWARD_COMPATIBILITY=1
```

2. For *SYCL code*: Force offloading to a CPU using one of the following:
 - *Recommended*: Set the SYCL_DEVICE_FILTER environment variable as follows:

```
set SYCL_DEVICE_FILTER=opencl:cpu
```

- If your application uses a SYCL device selector:

1. In the application source code, add the following to specify the CPU as the target device:

```
sycl::cpu_selector
```

For details, see [Device selectors](#) in the SYCL Reference.

2. Rebuild the application,
3. For *OpenMP code*: Force offloading to a CPU with one of the following:
 - *Recommended*: To offload code to CPU, set the following environment variables:

```
set OMP_TARGET_OFFLOAD=MANDATORY
```

```
set LIBOMPTARGET_DEVICE_TYPE=CPU
```

```
set LIBOMPTARGET_PLUGIN=OPENCL
```

- To run code natively on CPU, set the following variable:

```
set OMP_TARGET_OFFLOAD=DISABLED
```

4. If your application uses *OpenCL code*: Configure your code to be offloaded to a CPU. Refer to the OpenCL documentation at <https://www.khronos.org/registry/OpenCL/> for specific instructions.

Next Steps

- [Set up a project](#) if you do not have one and run the Intel Advisor [from a graphical user interface](#).
- Run the Intel Advisor [from a graphical user interface](#) and [set up a project](#) if you do not have one.

Launch Intel® Advisor


This topic provides overview and simple steps for running Intel® Advisor graphical user interface (GUI) and command line interface (CLI).

Launch the Intel® Advisor GUI

The Intel® Advisor GUI is available:

- On the Microsoft Windows* OS: From the **Start** menu, choose **All Programs > Intel oneAPI [version] > Intel Advisor [version]**
- NOTE** Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

In the Microsoft Visual Studio*:

- From the **Tools** menu, choose **Intel Advisor [version] > Vectorization and Threading Advisor Analysis**.
- From the top toolbar: Click  Intel Advisor icon.

Important If you do not see the icon, right-click the toolbar and select **Intel Advisor** from the context menu.

- From a command line on Windows* or Linux* OS systems.
 - Set up the environment variables to be able to launch the GUI from the command line.
 - Run the `advisor-gui` command to open the Intel Advisor GUI.

To open a specific project or result, enter:

```
advisor-gui <path>
```

where `<path>` is one of the following:

- Full (absolute) path to a result file (*.advixe)
- Full path to a project file (config.advixeproj)
- Full path to a project directory. If there is no project file in the directory, the **Create a Project** dialog box opens and prompts you to create a project in the specified directory.

On Windows systems, if the path contains embedded spaces, enclose it in quotation marks.

NOTE By default, Intel Advisor opens a new graphical user interface (GUI). To switch back to an old GUI, set `ADVISOR_EXPERIMENTAL=advixe-gui` variable and re-open the Intel Advisor.

After opening the Intel Advisor, continue to [create a project](#) or run a perspective and view the results if you already have a project.

Launch the Intel Advisor CLI

Prerequisite: Set up environment variables to enable the command line interface.

To run the `advisor` command line interface, use the following syntax:

```
advisor <--action> [--action-options] [--global-options] -- <target-application> [target options]
```

where:

- `<--action>` is an Intel Advisor action to do, such as `collect` or `report`.
- `[--action-options]` and `[--global-options]` are options to modify action behavior.
- `<target-application>` is an application executable to analyze with optional `[target-options]` to apply to the target.

The `advisor` command line interface supports all Intel Advisor perspective and is the recommended method to run the Intel Advisor from command line.

You can also run the Offload Modeling perspective using Python* scripts as follows:

```
advisor-python <APM>/<offload-script>.py <project-dir> [--options] [-- <target-application>
[target-options]]
```

where:

- **<APM>** is the environment variable that points to the directory with the Intel Advisor scripts. It is \$APM for Linux* OS and %APM% for Windows* OS.
- **<offload-script>** is a script to run: `run_oa`, `collect`, or `analyze`.
- **<project-dir>** is a path to a project directory.
- **[--options]** is options to modify script behavior.
- **<target-application>** is an application executable to analyze with optional **[target-options]** to apply to the target.

When you run the first Intel Advisor analysis to a target application from the command line, it also creates a new project for the target.

Review the typical workflows for the Intel Advisor CLI in the dedicated topics for each perspective.

- [Run Vectorization and Code Insights Perspective from Command Line](#)
- [Run CPU / Memory Roofline Insights Perspective from Command Line](#)
- [Run Threading Perspective from Command Line](#)
- [Run Offload Modeling Perspective from Command Line](#)
- [Run GPU Roofline Insights Perspective from Command Line](#)

For details about the Intel Advisor command line syntax and options, see the [advisor Command Line Interface Reference](#).

Launch the Intel Advisor from a Docker* Container on Linux* OS

This section contains steps to run Intel® Advisor in a Docker* container. Containers allow you to set up and configure environments and distribute them using [images](#):

- You can install an image containing an environment pre-configured with all the tools you need, then develop within that environment.
- You can save an environment and use the image to move that environment to another system without additional setup.
- You can prepare containers with different sets of compilers, tools, libraries, or other components, as needed.

Set up the Docker container

1. Pull the Docker image from the oneAPI Containers Repository with the following commands:

```
image=amr-registry.caas.intel.com/oneapi/oneapi:base-dev-ubuntu18.04
docker pull "$image"
```

2. Run the Docker container using the following command:

```
docker run --cap-add=SYS_PTRACE -it "$image"
```

NOTE

- The `--device=/dev/dri` option enables the gpu (if available).
 - You can specify proxy information using options as follows: `-e http_proxy="$http_proxy" -e https_proxy="$https_proxy"`
-

3. For the rest of the steps in this section, run any commands from the command line prompt inside the Docker container.

For example, to set up the Mandelbrot sample, you can run:

```
cd /one-api-code-samples/HPC/mandelbrot
make
./main -dl
./main -t gpu # run on gpu
./main -t cpu # run on cpu
make clean
```

4. Run the following commands to source Intel Advisor variables:

```
source /opt/intel/oneapi/setvars.sh
```

NOTE This step is not required, but allows you to run tools from any directory, rather than using absolute file paths.

5. Now that your Docker container is running, you can run Advisor from the command line as you would without a container. For example:

```
advisor --collect=survey /bin/ls
```

When you run the first Intel Advisor analysis to a target application from the command line, it also creates a new project for the target.

For details about the Intel Advisor command line syntax and options, see the [advisor Command Line Interface Reference](#). Review the typical workflows for the Intel Advisor CLI in the dedicated topics for each perspective.

See Also

Set Up a Project

[Analyze Vectorization Perspective](#) Improve your application performance, get code-specific recommendations for how to fix vectorization issues and quick visibility into source code and assembly code by running the Vectorization and Code Insights perspective.

[Analyze CPU Roofline](#) Visualize actual performance against hardware-imposed performance ceilings by running the CPU / Memory Roofline Insights perspective. It helps you determine the main limiting factor (memory bandwidth or compute capacity) and provides an ideal roadmap of potential optimization steps.

[Model Threading Designs](#) Analyze, design, tune, and check threading design options without disrupting your normal development by running the Threading Perspective.

[Model Offloading to a GPU](#) Find high-impact opportunities to offload/run your code and identify potential performance bottlenecks on a target graphics processing unit (GPU) by running the Offload Modeling perspective.

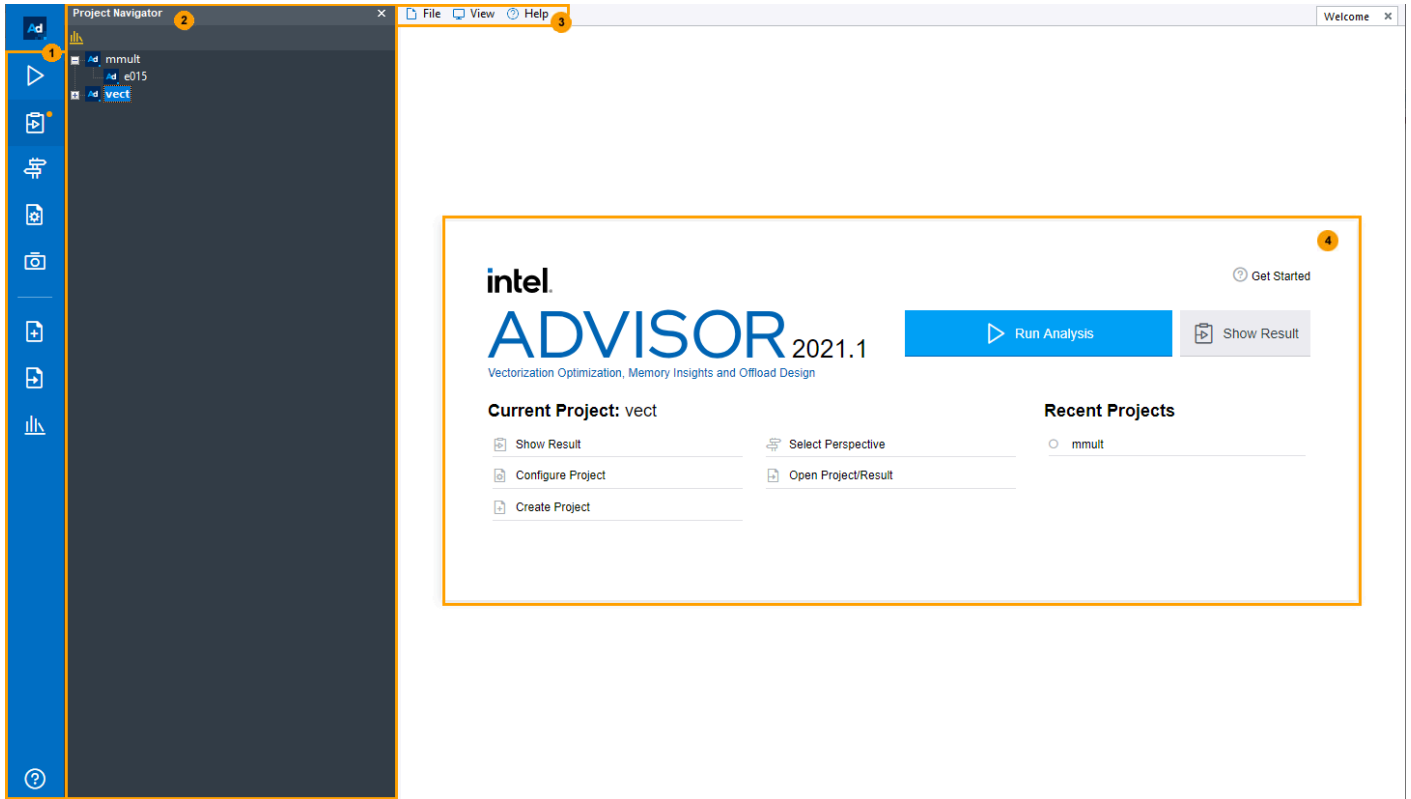
[Analyze GPU Roofline](#) Measure and visualize the actual performance of GPU kernels using benchmarks and hardware metric profiling against hardware-imposed performance ceilings, as well as determine the main limiting factor, by running the GPU Roofline Insights perspective.

GUI Navigation Quick Start

Use [Get Started with Intel® Advisor](#) to learn how to run perspectives using code samples and collect your first results.




Navigation Quick Start

After you launch the Intel® Advisor, a **Welcome** pane opens with the following controls:



1

Use the left-side toolbar for quick access to Intel Advisor projects and perspective controls. For example:

-  - Open the **Perspective Selector** window and select a perspective to run.
-  - [Create a project](#).
-  - Open an existing project.

2

Use the **Project Navigator** to view your projects and results based on the directory where the opened project resides.

3

Use the menu to create projects and dynamic analysis results, open projects and results, configure projects, set various options, open new panes, and access the Intel Advisor help.

4

Use the main **Welcome** window to create/open a project, configure current project, see recent projects, open the Get Started page.

See Also

[Set Up a Project](#)

Vectorization and Code Insights Perspective Improve your application performance, get code-specific recommendations for how to fix vectorization issues and quick visibility into source code and assembly code by running the Vectorization and Code Insights perspective.

CPU / Memory Roofline Insights Perspective Visualize actual performance against hardware-imposed performance ceilings by running the CPU / Memory Roofline Insights perspective. It helps you determine the main limiting factor (memory bandwidth or compute capacity) and provides an ideal roadmap of potential optimization steps.

Threading Perspective Analyze, design, tune, and check threading design options without disrupting your normal development by running the Threading Perspective.

Offload Modeling Perspective Find high-impact opportunities to offload/run your code and identify potential performance bottlenecks on a target graphics processing unit (GPU) by running the Offload Modeling perspective.

GPU Roofline Insights Perspective Measure and visualize the actual performance of GPU kernels using benchmarks and hardware metric profiling against hardware-imposed performance ceilings, as well as determine the main limiting factor, by running the GPU Roofline Insights perspective.

Set Up Project

To run Intel® Advisor, you need to create a project with your target executable. The project serves as a reusable container for:

- The location of a target executable, which is your compiled application
- Target executable sources and binaries
- A collection of configurable properties
- A previously collected analysis result

To set up a project:

1. **Optional: Configure your target application** to optimize it for analyses
2. **Build your target application** with optimal build settings
3. **Create and configure a project** with your target application

Configure Target Application

Intel® Advisor supports targets:

- Developed to run on Windows* or Linux* operating systems using the Intel® oneAPI DPC++/C++ Compiler, Intel® C++ Compiler Classic, Intel® Fortran Compiler Classic, or GNU* gcc compiler development environment
- That use C/C++, Fortran, or mixed Python* code for the portions that will run in parallel.
- That use SYCL, OpenCL™, or OpenMP* with **pragma omp target** (for C++) or **directive omp target** (for Fortran) code

The target executable must contain source symbol table debug information, so the Intel® Advisor can provide source line correlation and viewing sources.

Important To analyze an application with the Intel® Advisor, the application should take longer than 500 milliseconds to execute on CPU or GPU. If your application execution time is lower, it might cause inaccurate data sampling or a *No data is collected* error.

Before you start profiling your application and applying changes that should increase performance, you can configure the application as follows to optimize it for analyses:

- **Limit the number of threads used by parallel frameworks** to configure the application for threading.
- **Choose a small, representative data set** to reduce analysis overheads by reducing the amount of analyzed data.

Limit the Number of Threads Used by Parallel Frameworks

Intel® Advisor tools are designed to collect data and analyze *serial* programs. *Before* you use the Intel Advisor to examine a partially parallel program, modify your program so it runs as a serial program with a single thread within each parallel site.

Run Your Program as a Serial Program

To run the current version of your program as a serial program, you need to limit the number of threads to 1. To run your program with a single thread:

- With Intel® oneAPI Threading Building Blocks (oneTBB) , in the main thread create a `tbb::task_scheduler_init init(1);` object for the lifetime of the program and run the executable again. For example:

```
int main() {
    tbb::task_scheduler_init init(1);
    // ...rest of program...

    return 0;
}
```

The effect of `task_scheduler_init` applies separately to each user-created thread. So if the program creates threads elsewhere, you need to create a `tbb::task_scheduler_init init(1);` for that thread's lifetime as well. Use of certain oneTBB features can prevent the program from running serially. For more information, see the oneTBB documentation.

- With OpenMP*, do one of the following:
 - Set the OpenMP* environment variable `OMP_NUM_THREADS` to 1 before you run the program.
 - Omit the compiler option that enables recognition of OpenMP pragmas and directives. On Windows* OS, omit `/Qopenmp`, and on Linux* OS omit `-openmp`.

For more information, see your compiler documentation.

If you cannot remove the parallelism, you should add annotations to mark the parallel code regions and learn how parallel code will impact Intel Advisor tool reports.

See Also

[Build Target Application](#)

[Create a Project](#)

[Use Partially Parallel Programs with Intel Advisor](#)

s Choose a Small, Representative Data Set

When you run an analysis, the Intel® Advisor executes the target against the supplied data set. Data set size and workload have a direct impact on application execution time and analysis speed

For example, it takes longer to process a 1000x1000 pixel image than a 100x100 pixel image. A possible reason: You may have loops with an iteration space of 1...1000 for the larger image, but only 1...100 for the smaller image. The exact same code paths may be executed in both cases. The difference is the number of times these code paths are repeated.


You can control analysis cost without sacrificing completeness by minimizing this kind of unnecessary repetition from your target's execution.

Instead of choosing large, repetitive data sets, choose small, representative data sets that fully create tasks with minimal to moderate work per task. *Minimal* to *moderate* means just enough work to demonstrate all the different behaviors a task can perform.

Your objective: In as short a runtime period as possible, execute as many paths and the maximum number of tasks (parallel activities) as you can afford, while minimizing the repetitive computation within each task to the bare minimum needed for good code coverage.

Data sets that run in about ten seconds or less are ideal. You can always create additional data sets to ensure all your code is checked.

To modify the input data set using the Intel® Advisor GUI, do one of the following

- Specify the project properties for the target. For example:
 1. Either click **File > Project properties...** or the  icon on the Intel® Advisor toolbar. This displays the **Project Properties** dialog box.
 2. If needed, click the **Analysis Target** tab.
 3. In the **Target type** drop-down list, choose **Dependencies Analysis**.
 4. In the **Application parameters**, if your target's main entry point accepts command-line arguments, specify a value in this field. Either type a value, or click the **Modify...** button.
 5. When done, click **OK**.
- Modify the program's sources (perhaps using `#ifdef` directives) and rebuild the target.

On Windows* OS only: To modify the input data set in the Visual Studio IDE, do one of the following:

- Specify Properties for the project or configuration. For example, right-click the startup project's name to display the context menu:
 1. Choose **Properties > Configuration properties > Debugging**.
 2. Select the type of configuration this change will apply to by selecting the type under **Configuration**, such as **Active(Debug)**, **Debug**, **Release**, or **All Configurations**. By default, properties for **Debug** and **Release** configuration are maintained separately.
 3. Edit the **Command Arguments** to select the appropriate data set.
 4. Click **OK**.
- Specify a different startup project that already has a reduced data set.
- Modify the program's sources (perhaps using `#ifdef` directives) and rebuild the target.

Tip

- On Windows* OS only:** If you run this configuration often, consider creating a new configuration perhaps called **Dependencies** for this small data set.
- For the most current information on optimal C/C++ and Fortran build settings, see [Build Target Application](#).

Build Target Application

This section contains steps you should do before you begin running analyses on your application with Intel® Advisor. Do the following:

- Build an optimized binary of your application in **release** mode using settings designed to produce the most accurate and complete analysis results.
- Verify the resulting executable runs before trying to analyze it with the Intel® Advisor.

Important To analyze an application with the Intel® Advisor, the application should take longer than 500 milliseconds to execute on CPU or GPU. If your application execution time is lower, it might cause inaccurate data sampling or a *No data is collected* error.

Optimal C/C++ Settings

To Do This	For This	Optimal C/C++ Settings
Request full debug information (compiler and linker).	Vectorization and Code Insights	Linux* OS command line: <code>-g</code> Windows* OS command line:

To Do This	For This	Optimal C/C++ Settings
Request moderate optimization.	CPU / Memory Roofline Insights	<ul style="list-style-type: none"> • /Zi • /DEBUG <p>Microsoft Visual Studio* IDE:</p> <ul style="list-style-type: none"> • C/C++ > General > Debug Information Format > Program Database (/Zi) • Linker > Debugging > Generate Debug Info > Yes (/DEBUG)
	GPU Roofline Insights Offload Modeling Threading	
	Vectorization and Code Insights	<p>Linux* OS command line: -O2 or higher</p> <p>Windows* OS command line:</p> <ul style="list-style-type: none"> • /O2 or higher • /Ob1 (Threading only)
	CPU / Memory Roofline Insights GPU Roofline Insights Threading Offload Modeling	<p>Visual Studio* IDE:</p> <ul style="list-style-type: none"> • C/C++ > Optimization > Optimization > Maximum Optimization (Favor Speed) (/O2) or higher • C/C++ > Optimization > Inline Function Expansion > Only_inline (/Ob1) (Threading only)
Disable interprocedural optimizations that may inhibit the ability of Intel® Advisor to collect performance data.	Offload Modeling	<p>Linux* OS command line: -no-ipo</p> <p>Windows* OS command line: /Qipo-</p>
For Intel® C++ Compiler Classic / Intel® oneAPI DPC++/C++ Compiler only.		
Produce compiler diagnostics (necessary for version 15.0 of the Intel® C++ Compiler Classic; unnecessary for version 16.0 and higher).	Vectorization and Code Insights	<p>Linux* OS command line: -qopt-report=5</p> <p>Windows* OS command line: /Qopt-report:5</p>
	CPU / Memory Roofline Insights GPU Roofline Insights	<p>Visual Studio* IDE: C/C++ > Diagnostics [Intel C++] > Optimization Diagnostic Level > Level 5 (/Qopt-report:5)</p>
Enable vectorization.	Vectorization and Code Insights	<p>Linux* OS command line: -vec</p> <p>Windows* OS command line: /Qvec</p>

To Do This	For This	Optimal C/C++ Settings
Enable SIMD directives.	CPU / Memory Roofline Insights GPU Roofline Insights	Linux command line: <code>-simd</code> Windows* OS command line: <code>/Qsimd</code>
Enable generation of multi-threaded code based on OpenMP* directives.	Vectorization and Code Insights CPU / Memory Roofline Insights GPU Roofline Insights	Linux* OS command line: <code>-qopenmp</code> Windows* OS command line: <code>/Qopenmp</code> Visual Studio* IDE: C/C++ > Language [Intel C++] > OpenMP Support > Generate Parallel Code (/Qopenmp)
Search additional directory related to Intel Advisor annotation definitions.	Primarily Threading, but could also be useful for refinement analyses	Linux* OS command line: <code>-I\${ADVISOR_[product_year]_DIR}/include</code> Windows* OS command line: <code>/I"%ADVISOR_[product_year]_DIR%\include</code> Visual Studio* IDE: C/C++ > General > Additional Include Directories > \${ADVISOR_[product_year]_DIR} \include;%(AdditionalIncludeDirectories)
Search for unresolved references in multithreaded, dynamically linked libraries.	Threading only	Linux* OS command line: <code>-Bdynamic</code> Windows* OS command line: <code>/MD</code> or <code>/MDd</code> Visual Studio* IDE: C/C++ > Code Generation > Runtime Library > Multithread
Enable dynamic loading.	Threading only	Linux* OS command line: <code>-ldl</code>

Optimal Fortran Settings

To Do This	For This Tool	Optimal Fortran Settings
Request full debug information (compiler and linker).	Vectorization and Code Insights CPU / Memory Roofline Insights GPU Roofline Insights Threading	Linux* OS command line: <code>-g</code> Windows* OS command line: <ul style="list-style-type: none"> <code>/debug=full</code> <code>/DEBUG</code> Visual Studio* IDE: <ul style="list-style-type: none"> Fortran > General > Debug Information Format > Full (/debug=full) Linker > Debugging > Generate Debug Info > Yes (/DEBUG)
Request moderate optimization.	Vectorization and Code Insights CPU / Memory Roofline Insights GPU Roofline Insights Threading	Linux* OS command line: <code>-O2</code> or higher Windows* OS command line: <ul style="list-style-type: none"> <code>/O2</code> or higher <code>/Ob1</code> (Threading only) Visual Studio* IDE: <ul style="list-style-type: none"> Fortran > Optimization > Optimization > Maximize Speed or higher Fortran > Optimization > Inline Function Expansion > Only INLINE directive (/Ob1) (Threading only)
Produce compiler diagnostics (necessary for version 15.0 of the Intel® Fortran Compiler Classic; unnecessary for version 16.0 and higher).	Vectorization and Code Insights CPU / Memory Roofline Insights GPU Roofline Insights	Linux* OS command line: <code>-qopt-report=5</code> Windows* OS command line: <code>/Qopt-report:5</code> Visual Studio* IDE: Fortran > Diagnostics > Optimization Diagnostic Level > Level 5 (/Qopt-report:5)
Enable vectorization.	Vectorization and Code Insights CPU / Memory Roofline Insights GPU Roofline Insights	Linux* OS command line: <code>-vec</code> Windows* OS command line: <code>/Qvec</code>
Enable SIMD directives.	Vectorization and Code Insights	Linux* OS command line: <code>-simd</code> Windows* OS command line: <code>/Qsimd</code>

To Do This	For This Tool	Optimal Fortran Settings
Enable generation of multi-threaded code based on OpenMP* directives.	CPU / Memory Roofline Insights GPU Roofline Insights Vectorization and Code Insights CPU / Memory Roofline Insights GPU Roofline Insights	Linux* OS command line: <code>-qopenmp</code> Visual Studio* IDE: Fortran > Language > Process OpenMP Directives > Generate Parallel Code (/Qopenmp)
Search additional directory related to Intel Advisor annotation definitions.	Primarily Threading, but could also be useful for refinement analyses	Linux* OS command line: <ul style="list-style-type: none"> • <code>-I\${ADVISOR_[product_year]_DIR}/include/ia32</code> or <code>-I\${ADVISOR_[product_year]_DIR}/include/ia64</code> • <code>-L\${ADVISOR_[product_year]_DIR}/lib32</code> or <code>-L\${ADVISOR_[product_year]_DIR}/lib64</code> • <code>-ladvisor</code> Windows* OS command line: <ul style="list-style-type: none"> • <code>/I"%ADVISOR_[product_year]_DIR%\include\ia32</code> or <code>/I"%ADVISOR_[product_year]_DIR%\include\ia64</code> • <code>/L"%ADVISOR_[product_year]_DIR%\lib32</code> or <code>/L"%ADVISOR_[product_year]_DIR%\lib64</code> • <code>/ladvisor</code> or Visual Studio* IDE: <ul style="list-style-type: none"> • Fortran > General > Additional Include Directories > "\$(ADVISOR_[product_year]_DIR)\include\ia32\" or "\$(ADVISOR_[product_year]_DIR)\include\ia64\" • Linker > General > Additional Library Directories > "\$(ADVISOR_[product_year]_DIR)\lib32" or "\$(ADVISOR_[product_year]_DIR)\lib64" • Linker > Input > Additional Dependencies > .lib > libadvisor
Search for unresolved references in multithreaded, dynamically linked libraries.	Threading only	Linux* OS command line: <code>-shared-intel</code> Windows* OS command line: <code>/MD</code> or <code>/MDd</code> Visual Studio* IDE: Fortran > Libraries > Runtime Library > Multithread DLL (/libs:dll /threads) or Debug Multithread DLL (/libs:dll /threads /dbglibs)

To Do This	For This Tool	Optimal Fortran Settings
Enable dynamic loading.	Threading only	Linux* OS command line: <code>-ldl</code>

Create Project

Intel® Advisor is based on a project paradigm and requires that you create or open a project to enable analysis features. Think of a project as a reusable container for:

- The location of a compiled application
- A collection of configurable properties
- An analysis result

NOTE You can skip this step in the following cases:


- If you use Intel Advisor as a Microsoft Visual Studio* integration, as it creates a new project automatically when opened.
- If you use Intel Advisor from the command line interface, as it creates a new project automatically when you run the first analysis

To create an Intel® Advisor project from the GUI:

1. Open the **Create a Project** dialog box using any of the following options:

- Choose **File > New > Project....**

-

Click the  icon on the left-side toolbar.

- Click the Welcome page [Create Project](#) link.

2. In the **Create a Project** dialog box, configure the following:

Use This	To Do This
Project name field	Specify the name of the Intel® Advisor project. This might be similar to the target executable name. The project name is used for the project directory name: <ul style="list-style-type: none"> • A project file that identifies the target to be analyzed and a set of configurable attributes for running the target. • Results that allows you to view the collected data.
Location field and Browse button	Choose or create a directory to contain the project directory. Click the Browse button to browse to and select a directory where the project directory will be created. Project files should be located in a different directory than your source directories, such as a directory above the source directories or in a separate projects directory. You must have write permission to the specified directory and its subdirectories.
Create project button	After entering the Project name and specifying its Location , click Create project to create the project and its directory and display the Analysis Target tab of the Project Properties dialog box.

3. Click **Create Project** button.

A **Project Properties** dialog box opens where you can configure your target application and the project.

Continue to select a perspective and run it to analyze your application.

See Also

Launch Intel® Advisor

Run Vectorization and Code Insights Perspective from GUI

Run CPU / Memory Roofline Insights Perspective from GUI

Run Threading Perspective from GUI Steps to run the Threading perspective.

Run Offload Modeling Perspective from GUI

Run GPU Roofline Insights Perspective from GUI


Configure Project

After you [create a project](#), the **Project Properties** dialog box opens. In the **Analysis Target** tab, you can specify the target executable, set important project properties, and review current project properties.

Tip

Always check project property values before analyzing a new target.

For an existing project, you can also access this tab:

- From the Intel Advisor GUI, choose **File > Project Properties**.
- 
 Click the icon on the left-side toolbar.
- From the Visual Studio* menu, choose **Project > Intel Advisor [version] Project Properties...**

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

Analysis Target Tab Overview

In the **Analysis Target** tab, select an analysis type from list (on the left) to display and set project properties.

Analysis Type selector	Select an analysis type to configure. Different project properties are available in the Analysis Properties region depending on the analysis type selected. The following analysis types are available: <ul style="list-style-type: none"> • Survey Analysis Types <ul style="list-style-type: none"> • Survey Hotspots Analysis • Trip Counts and FLOP analysis • Suitability Analysis • Refinement Analysis Types <ul style="list-style-type: none"> • Memory Access Patterns Analysis • Dependencies Analysis • Performance Modeling Analysis
Analysis Properties	Set project properties for the analysis type selected in the Analysis Type region.

Analysis Target Tab Controls

The following table covers project properties applicable to all analysis types. To view controls applicable only to a specific analysis type, use the links immediately below:

- [Survey Analysis Controls](#)
- [Trip Counts and FLOPS Controls](#)
- [Suitability Analysis Controls](#)
- [MAP Analysis Controls](#)
- [Dependencies Analysis Controls](#)

NOTE To configure a project, it is enough to set only common properties.

Common Controls

The following controls are common for all analysis types. Specify the properties in the **Survey Hotspot Analysis** tab and check that the **Inherit settings from the Survey Hotspots Analysis Type** checkbox is enabled in other tabs to share the properties for all analyses.

Use This	To Do This
Target type drop-down	<ul style="list-style-type: none"> • Analyze an executable or script (choose Launch Application). • Analyze a process (choose Attach to Process). <p>If you choose Attach to Process, you can either inherit settings from the Survey Hotspots Analysis Type or specify the needed settings.</p>
Inherit settings from Visual Studio project checkbox and field (Visual Studio* IDE only)	<p>Inherit Intel Advisor project properties from the Visual Studio* startup project (enable).</p> <p>If enabled, the Application, Application parameters, and Working directory fields are pre-filled and cannot be modified.</p> <hr/> <p>NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.</p> <hr/>
Application field and Browse... button	<p>Select an analysis target executable or script.</p> <p>If you specify a script in this field, consider specifying the executable in the Advanced > Child application field (required for Dependencies analysis).</p>
Application parameters field and Modify... button	<p>Specify runtime arguments to use when performing analysis (equivalent to command line arguments).</p>
Use application directory as working directory checkbox	<p>Automatically use the value in the Application directory to pre-fill the Working directory value (enable).</p>
Working directory field and Browse... button	<p>Select the working directory.</p>
User-defined environment variables field and Modify... button	<p>Specify environment variables to use during analysis.</p>

Use This	To Do This
Managed code profiling mode drop-down	<ul style="list-style-type: none"> Automatically detect the type of target executable as Native or Managed, and switch to that mode (choose Auto). Collect data for native code and do not attribute data to managed code (choose Native). Collect data for both native and managed code, and attribute data to managed code as appropriate (choose Mixed). Consider using this option when analyzing a native executable that makes calls to managed code. Collect data for both native and managed code, resolve samples attributed to native code, and attribute data to managed source only (choose Managed). The call stack in the analysis result displays data for managed code only.
Child application field	<p>Analyze a file that is not the starting application. For example: Analyze an executable (identified in this field) called by a script (identified in the Application field).</p> <p>Invoking these properties could decrease analysis overhead.</p> <hr/> <p>NOTE For the Dependencies Analysis Type: If you specify a script file in the Application field, you must specify the target executable in the Child application field.</p> <hr/>
Modules radio buttons, field, and Modify... button	<ul style="list-style-type: none"> Analyze specific modules and disable analysis of all other modules (click the Include only the following module(s) radio button and choose the modules). Disable analysis of specific modules and analyze all other modules (click the Exclude only the following module(s) radio button and choose the modules). <p>Including/excluding modules could minimize analysis overhead.</p>
Use MPI launcher checkbox	<p>Generate a command line (enable) that appears in the Get command line field based on the following parameters:</p> <ul style="list-style-type: none"> Select MPI Launcher - Intel or another vendor Number of ranks - Number of instances of the application Profile ranks - All or a range of ranks to profile
Automatically stop collection after (sec) checkbox and field	<p>Stop collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could minimize analysis overhead.</p>

Survey Analysis-Specific Controls

Use This	To Do This
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p>

Use This	To Do This
	<hr/> <p>Tip The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p> <hr/>
Sampling Interval selector	<p>Set the wait time between each analysis collection CPU sample while your target application is running.</p> <p>Increasing the wait time could decrease analysis overhead.</p>
Collection data limit, MB selector	<p>Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses.</p> <p>Decreasing the limit could decrease analysis overhead.</p>
Callstack unwinding mode drop-down list	<p>Set to After collection if:</p> <ul style="list-style-type: none"> Survey analysis runtime overhead exceeds 1.1x. A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions. <p>Otherwise, set to During Collection. This mode improves stack accuracy but increases overhead.</p>
Stitch stacks checkbox	<p>Restore a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload (enable).</p> <p>Disable if Survey analysis runtime overhead exceeds 1.1x.</p>
Analyze MKL Loops and Functions checkbox	<p>Show Intel® oneAPI Math Kernel Library (oneMKL) loops and functions in Intel Advisor reports (enable).</p> <p>Enabling could increase analysis overhead.</p>
Analyze Python loops and functions checkbox	<p>Show Python* loops and functions in Intel Advisor reports (enable).</p> <p>Enabling could increase analysis overhead.</p>
Analyze loops that reside in non-executed code paths checkbox	<p>Collect a variety of data during analysis for loops that reside in non-executed code paths, including loop assembly code, instruction set architecture (ISA), and vector length (enable).</p> <p>Enabling could increase analysis overhead.</p> <hr/> <p>NOTE Analyzing non-executed code paths in binaries that target multiple ISAs (contain multiple code paths) is available only for binaries compiled using the <code>-ax</code> (Linux* OS) / <code>Qax</code> (Windows* OS) option with an Intel compiler.</p> <hr/>
Enable registry spill/fill analysis checkbox	<p>Calculate the number of consecutive load/store operations in registers and related memory traffic (enable).</p>

Use This	To Do This
	Enabling could increase analysis overhead.
Enable static instruction mix analysis checkbox	Statically calculate the number of specific instructions present in the binary (enable). Enabling could increase analysis overhead.
Source caching drop-down list	<ul style="list-style-type: none"> Delete source code cache from a project with each analysis run (default; choose Clear cached files). Keep source code cache within the project (choose Keep cached files).

Trip Counts and FLOP Analysis-Specific Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds.
Trip Counts / Collect information about Loop Trip Counts checkbox	Measure loop invocation and execution (enable).
FLOP / Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage checkbox	Measure floating-point operations, integer operations, and memory traffic (enable).
Callstacks / Collect callstacks checkbox	Collect call stack information when performing analysis (enable). Enabling could increase analysis overhead.
Capture metrics for dynamic loops and functions checkbox	Collect metrics for dynamic Just-In-Time (JIT) generated code regions.
Capture metrics for stripped binaries checkbox	Collect metrics for stripped binaries. Enabling could increase analysis overhead.

Use This	To Do This
Cache Simulation / Enable Memory-Level Roofline with cache simulation checkbox	Model multiple levels of cache for data, such as counts of loaded or stored bytes for each loop, to plot the Roofline chart for all memory levels (enable). Enabling could increase analysis overhead.
Cache simulator configuration field	Specify a cache hierarchy configuration to model (enable and specify hierarchy). The hierarchy configuration template is: $[num_of_level1_caches]:[num_of_ways_level1_connected]:$ $[level1_cache_size]:[level1_cacheline_size]/$ $[num_of_level2_caches]:[num_of_ways_level2_connected]:$ $[level2_cache_size]:[level2_cacheline_size]/$ $[num_of_level3_caches]:[num_of_ways_level3_connected]:$ $[level3_cache_size]:[level3_cacheline_size]$ For example: 4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l is the hierarchy configuration for: <ul style="list-style-type: none"> • Four eight-way 32-KB level 1 caches with line size of 64 bytes • Four four-way 256-KB level 2 caches with line size of 64 bytes • One sixteen-way 6-MB level 3 cache with line size of 64 bytes
Data Transfer Simulation / Data transfer simulation mode drop-down	Select a level of details for data transfer simulation: <ul style="list-style-type: none"> • Off - Disable data transfer simulation analysis. • Light - Model data transfers between host and device memory. • Full - Model data transfers, attribute memory objects to loops that accessed the objects, and track accesses to stack memory.

Suitability Analysis-Specific Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. <hr/> Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds. <hr/>
Sampling Interval selector	Set the wait time between each analysis collection sample while your target application is running. Increasing the wait time could decrease analysis overhead.

Use This	To Do This
Collection data limit, MB selector	Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses. Decreasing the limit could decrease analysis overhead.

Memory Access Patterns Analysis-Specific Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Suppression mode group box	<ul style="list-style-type: none"> Report possible memory issues in system modules (choose the Show problems in system modules radio button). Do not report possible memory issues in system modules (choose the Suppress problems in system modules radio button).
Loop call count limit selector	Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances. Supplying a non-zero value could decrease analysis overhead.
Instance of interest selector	Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes. Supplying a non-zero value could decrease analysis overhead.
Report stack variables checkbox	Report stack variables for which memory access strides are detected (enable). Enabling could increase analysis overhead.
Report heap allocated variables checkbox	Report heap-allocated variables for which memory access strides are detected (enable). Enabling could increase analysis overhead.
Enable CPU cache simulation checkbox	Model cache misses, cache misses and cache line utilization, or cache misses and loop footprint (enable and select desired options). Enabling could increase analysis overhead.
Cache associativity drop-down list	Set the cache associativity for modeling CPU cache behavior. You can set the value to the following power-of-two integers: 1, 2, 4, 8, 16.
Cache sets drop-down list	Set the cache set size (in bytes) for modeling CPU cache behavior. You can set the value to the following power-of-two integers: 256, 512, 1024, 2048, 4096, 8192.
Cache line size drop-down list	Set the cache line size (in bytes) to model CPU cache behavior. You can set the value to the following power-of-two integers: 4, 8, 16, 32, ..., up to 65536.
Cache simulation mode drop-down list	Set the focus for modeling CPU cache behavior: <ul style="list-style-type: none"> Model cache misses only.

Use This	To Do This
	<ul style="list-style-type: none"> • Model cache misses and memory footprint of a loop. Calculation: Cache line size x Number of unique cache lines accessed during simulation. • Model cache misses and cache line utilization.

Dependencies Analysis Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Suppression mode radio buttons	<ul style="list-style-type: none"> • Report possible dependencies in system modules (choose the Show problems in system modules radio button). • Do not report possible dependencies in system modules (choose the Suppress problems in system modules radio button).
Loop call count limit selector	Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances. Supplying a non-zero value could decrease analysis overhead.
Instance of interest selector	Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes. Supplying a non-zero value could decrease analysis overhead.
Analyze stack variables checkbox	Analyze parallel data sharing for stack variables (enable). Enabling could increase analysis overhead.
Filter stack variables by scope checkbox	Enable to report: <ul style="list-style-type: none"> • Variables initiated inside the loop as potential dependencies (warning) • Variables initialized outside the loop as dependencies (error) Enabling could increase analysis overhead.
Reduction Detection / Filter reduction variables checkbox	Mark all potential reductions by a specific diagnostic (enable). Enabling could increase analysis overhead.
Markup type checkbox	<p>Select loops/functions by pre-defined markup algorithm. Supported algorithms are:</p> <ul style="list-style-type: none"> • GPU generic - Select loops executed on a GPU. • OpenMP - Select OpenMP* loops. • SYCL - Select SYCL loops. • OpenCL - Select OpenCL™ loops. • DAAL - Select Intel® oneAPI Data Analytics Library loops. • TBB - Select Intel® oneAPI Threading Building Blocks loops. <hr/> <p>NOTE This option is available only from the Analysis Workflow pane for the Offload Modeling perspective.</p> <hr/>

Performance Modeling Properties

Use This	To Do This
Device configuration	Select a pre-defined hardware configurations from a drop-down list to model application performance on.
Other parameters	Enter a space-separated list of command-line parameters. For a full list of available options, see Command Option Reference .


Configure Binary/Symbol Search Directories

You need to configure binary/symbol search directories if your target application has non-standard directories with the supporting files needed to execute and analyze the target. By default, if you do not specify source search directory, Intel Advisor searches the standard OS directories. See [Binary/Symbol Search Locations](#) for details.





With Visual Studio* on Windows* OS, you can instead use the Visual Studio solution and project capabilities to search for specific directories.

Tab Location

To access this tab:

- From the Intel Advisor GUI, choose **File > Project Properties**. Then click the **Binary/Symbol Search** tab.
- Click the  icon on the left-side toolbar.
- From the Visual Studio* menu, choose **File > Intel Advisor [version] Project Properties...**. Then click the **Binary/Symbol Search** tab.

Tab Controls

Use This	To Do This
 button	On a row containing Add new search location , click to browse for directories to include in the search list. You can also type directly in the row. In addition to local directories, you can specify a symbol server URL.
 and  buttons	Change the search order of the selected directory by moving it up or down. To select multiple rows, use the Ctrl or Shift keys.
 button	Delete a selected directory row(s).
Search recursively checkbox	Enable to search the specified location subdirectories. To use recursive search, the lines must provide only a directory name and omit a file name. Using a recursive search for multiple directories may slow processing and could lead to unexpected results.

See Also

- [Binary/Symbol Search and Source Search Locations](#)

Configure Source Search Directory

You need to configure source search directories to specify the source search locations needed to execute and analyze your target application. By default, if you do not specify source search directory, Intel Advisor searches the directories from the collected result. See [Source Search Locations](#) for details.


With Visual Studio, some source locations are pre-populated from the Visual Studio startup project into the internal representation of Intel Advisor project properties, so you may not need to add new row(s).

Tip






For Threading Advisor only: Intel® Advisor does not automatically populate source locations after you create a project using the Intel® Advisor GUI, so you must specify one or more locations to find application annotations. View the **Annotation Report** to verify all project annotations are found.

Tab Location

To access this tab:

- From the Intel® Advisor GUI, choose **File > Project Properties**. Then click the **Source Search** tab.
- Click the  icon on the left-side toolbar.
- From the Visual Studio* menu, choose **File > Intel Advisor [version] Project Properties....** Then click the **Source Search** tab.

Tab Controls

Use This	To Do This
 button	On a row containing Add new search location , click to browse for directories to include in the search list. You can also type directly in the row.
 and  buttons	Change the search order of the selected directory by moving it up or down. To select multiple rows, use the Ctrl or Shift keys.
 button	Delete a selected directory row(s).
Search recursively checkbox	Enable to search the specified location subdirectories. To use recursive search, the lines must provide only a directory name and omit a file name. Using a recursive search for multiple directories may slow processing and could lead to unexpected results.
Mask text box	Specify the file name mask pattern(s) to ignore (skip) using wildcard characters, such as an asterisk (*). For example, you can skip certain file suffixes.
File text box	Specify the file(s) to ignore (skip) using an absolute path. To delete a row, use the  button.

See Also

- [Binary/Symbol Search and Source Search Locations](#)

Binary/Symbol Search and Source Search Locations

When using the Intel Advisor Standalone GUI:

- If you specify binary and symbol locations to search using the **Binary/Symbol Search** tab, they will be searched *in addition* to the [default binary and symbol locations](#).
- If you specify source locations to search using the **Source Search** tab, they will be searched *in addition* to the [default source search locations](#).

Binary/Symbol Search Locations

Intel Advisor searches binary and symbol files in default locations and in locations specified in the **Binary/Symbol Search** tab (if specified).

The following lists describe the order and default locations that are searched. As indicated below, some directory searches examine the specified directory and its subdirectories, while other searches do not examine its subdirectories.

The search order on **Windows* OS** systems is the following:

1. Search for binary and symbol files in the directories specified in the **Binary/Symbol Search** tab and their subdirectories (if enabled in the tab).
2. Search for symbol files in the directories near the related (corresponding) binary file(s) just found, such as a library:
 - Check in the directory of the corresponding binary file, using the corresponding name.
 - Check in the directory of the corresponding binary file, using a related name. For example, for `app.dll` where a file `app_x86.pdb` is present, also search for file `app.pdb`.
3. When using an integrated Visual Studio project, the directories provided by the Visual Studio project pre-populate the corresponding directories in the internal representation of the **Binary/Symbol Search** tab (for example, Visual Studio binary locations pre-populate the Project Properties binary locations). So, the Visual Studio project's directories are searched and are specific to the selected configuration.

For symbol files, also search using symbol server paths specified in the **Binary/Symbol Search** tab in the following notation: `srv*C:\localsymbols*http://msdl.microsoft.com/download/symbols` and/or provided in Visual Studio **Tools > Options > Debugging > Symbols**.

4. Search for binary files in this standard Windows OS system directory:


```
%SYSTEMROOT%\system32\drivers
```

 (subdirectories are not searched)
5. Search for symbol files in these standard Windows OS system directories:
 - All directories specified in the environment variable `_NT_SYMBOL_PATH` (subdirectories are not searched)
 - `srv*%SYSTEMROOT%\symbols` (symbol downstream or cache path)
 - `%SYSTEMROOT%\symbols\dll` (subdirectories are not searched)

The search order on **Linux* OS** systems is the following:

1. Search for binary and symbol files in the directories specified in the **Binary/Symbol Search** tab and their subdirectories (if enabled in the tab).
2. Search for binary files in directories from the collected result that provide an absolute path name. If the file name `vmlinux` is present, search these directories:
 - `/usr/lib/debug/lib/modules/`uname -r`/vmlinux`
 - `/boot/vmlinuz-`uname -r``
3. Search for symbol files in the directories near the related (corresponding) binary file(s) just found, such as a library:
 - Check in the directory of the corresponding binary file, using the corresponding name.
 - Check in the directory of the corresponding binary file, using a related name. For example, for `app.dll` where a file `app_x86.pdb` is present, also search for file `app.pdb`.
 - Search in the `.debug` subdirectory.
4. Search for binary files in these standard Linux OS system directories:

- `/lib/modules` (subdirectories are not searched)
 - `/lib/modules/`uname -r`/kernel` (subdirectories are searched)
5. Search for symbol files in these standard Linux OS system directories:
- `usr/lib/debug`(subdirectories are not searched)
 - `/usr/lib/debug` with appended path to the corresponding binary file, such as `/usr/lib/debug/usr/bin/ls.debug`

Source Search Locations

A limited set of default source locations are used in addition to the locations specified in the **Source Search** tab. With Intel Advisor, you can use this tab to indicate whether or not the default source locations (listed below) will be searched.

NOTE

When using the Intel Advisor GUI, you must specify one or more new rows (locations) in the **Source Search** tab so Intel Advisor tools can find your application's annotations.

The following list describes the order and default locations that are searched. As indicated below, some directory searches examine the specified directory and its subdirectories, while other searches do not examine its subdirectories.

1. Search for source files in the directories specified in the **Source Search** tab. With Intel Advisor, you can indicate whether the subdirectories of these directories should be searched.
2. Search for source files in directories from the collected result that provide an absolute path name.
3. When using an integrated Visual Studio project, the source directories provided by the Visual Studio project pre-populate the corresponding source directories in the internal representation of the **Source Search** tab. So, the Visual Studio project's source directories are searched for source files, and they apply to all configurations. **When using Microsoft Visual Studio***: Search for source files in Visual Studio project directories.
4. **On Linux OS systems**: Search for source files in these standard Linux locations (does not search subdirectories):

```
/usr/src
/usr/src/linux-headers-`uname -r`
```

See Also

[Binary/Symbol Search Tab](#)

[Source Search Tab](#)

Analyze Vectorization Perspective

Improve your application performance, get code-specific recommendations for how to fix vectorization issues and quick visibility into source code and assembly code by running the Vectorization and Code Insights perspective.

The Vectorization and Code Insights perspective can help you to identify:

- Where vectorization, or parallelization with threads, will pay off the most
- If vectorized loops are providing benefit, and if not, why not
- Not vectorized loops and why they are not vectorized
- Memory usage issues
- Performance insights and problems in general

How It Works

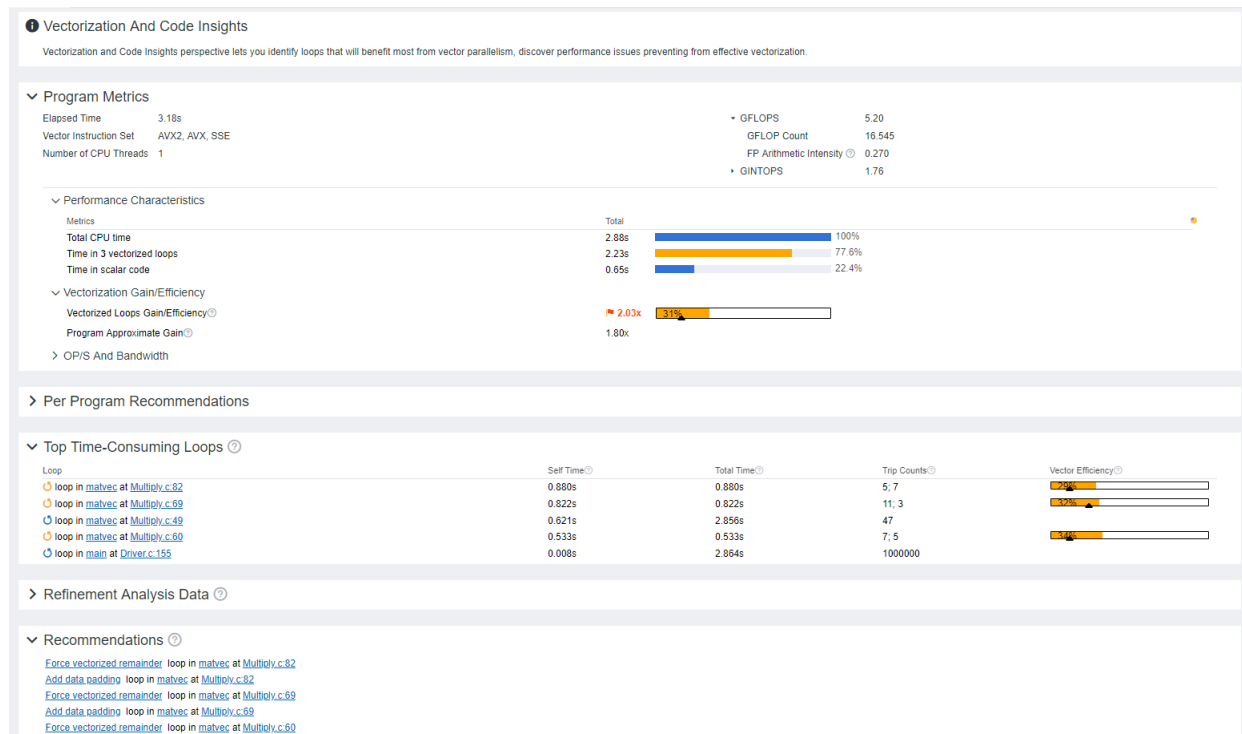
The Vectorization and Code Insights perspective includes the following steps:

1. Get integrated compiler report data and performance data by running a **Survey** analysis.
2. Identify the number of times loops are invoked and execute and the number of floating-point and integer operations by running the **Characterization** analysis. It measures the call count/loop count and iteration count metrics for your application. Enable to make better decisions about your vectorization strategy for particular loops, as well as optimize already-vectorized loops.
3. Check for various memory issues by running the **Memory Access Patterns (MAP)** analysis. It can warn you about non-contiguous memory accesses, unit stride vs. non-unit stride accesses, or other issues. Enable to identify issues that could lead to significant vector code execution slowdown or block automatic vectorization by the compiler.
4. Check for data dependencies in loops the compiler did not vectorize by running the **Dependencies** analysis. The Dependencies analysis checks for real data dependencies and if real dependencies are detected, provides additional details to help resolve them. Choose to identify and better characterize real data dependencies that could make forced vectorization unsafe.

Vectorization Summary

Vectorization and Code Insights perspective collects data about your application performance, including the following:

- Performance metrics, including vectorization efficiency for the whole application and for each vectorized loop/function
- Top five time-consuming loops sorted by self time
- Integrated compiler report data and code-specific recommendations for fixing performance issues



See Also

[Run Vectorization and Code Insights Perspective from GUI](#)

[Run Vectorization and Code Insights Perspective from Command Line](#)

[Vectorization Report Navigation Overview](#) Review the controls available in the main report of the Vectorization and Code Insights perspective of the Intel® Advisor.

Run Vectorization and Code Insights Perspective from GUI

Prerequisite:

In the graphical-user interface (GUI): [Create a project](#) and specify an analysis target and target options.

To configure and run the Vectorization and Code Insights perspective from the GUI:

1. From the **Analysis Workflow** tab, configure the perspective and set analysis properties, depending on desired results:
 - Select a collection accuracy level with analysis properties preset for a specific result:
 - **Low**: Get the basic insights about vectorized and un-vectorized loops in your code.
 - **Medium**: Identify the number of times loops execute to make better decisions about your vectorization strategy.
 - **High**: Analyze application memory usage and performance values that help you make better decisions about your vectorization strategy in details.
 - Select the analyses and properties manually to adjust the perspective flow to your needs. The accuracy level is set to **Custom**.








The higher accuracy value you choose, the higher runtime overhead is added to your application. The **Overhead** indicator shows the overhead for the selected configuration. For the **Custom** accuracy, the overhead is calculated automatically for the selected analyses and properties.

By default, accuracy is set to Low. For more information, see [Vectorization Accuracy Presets](#).

2. If you want to limit the Characterization, Memory Access Patterns, and/or Dependencies analyses to one or more specific loops/functions instead of analyzing the whole application:
 - From a Survey report generated: Mark one or more un-vectorized loops by enabling the corresponding checkboxes in the report.

3. Click  button to run the perspective.


While the perspective is running, you can do the following in the **Analysis Workflow** tab:

- Control the perspective execution:
 - Stop data collection and see the already collected data: Click the  button.
 - Pause data collection: Click the  button.
 - Cancel data collection and discard the collected data: Click the  button.
- Expand an analysis with  to control the analysis execution:
 - Pause the analysis: Click the  button.
 - Stop the currently running analysis and start the next analysis selected: Click the  button.
 - Interrupt execution of all selected analyses and see the already collected data: Click the  button.

To run the Vectorization and Code Insights perspective with the Low accuracy from the command line interface:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

See [Run Vectorization and Code Insights Perspective from Command Line](#) for details.

NOTE To [generate command lines](#) for selected perspective configuration, click the  **Command Line** button.

Once the data is collected, the Survey report opens showing a **Summary** tab. Depending on the selected accuracy level and perspective properties, continue to investigate the results:

- [Examine Not-Vectorized and Under-Vectorized Loops](#)
- [Examine Loop Call Count](#)
- [Investigate Memory Usage and Traffic](#)
- [Identify Data Dependencies in Your Application](#)

NOTE

- After you run the Vectorization and Code Insights perspective, the collected **Survey** data becomes available for all other perspectives. If you switch to another perspective, you can skip the Survey step and run only perspective-specific analyses.
- If the Survey analysis does not collect enough data to produce a report, it displays a *Target executed too quickly or does not contain debug symbols* message. Increase the target workload or data to run the analysis for at least a few seconds, check whether debug information is specified as a build option, or specify a different target application.

Vectorization Accuracy Presets

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium	High
Overhead	1.1x	5 - 8x	10 - 40x
Goal	Get basic insights about how well your application is vectorized and how you can improve vectorization efficiency	Get more insights about how well your application is vectorized and the number of iterations in loops/functions	Get detailed insights about your application performance, including performance issues and detailed optimization recommendations
Analyses	Survey	Survey + Characterization (Trip Counts)	Survey + Characterization (Trip Counts, FLOP, Call Stacks) + Memory Access Patterns
Result	Basic Survey report	Survey report extended with trip count data	Extended Survey report with trip counts and floating-point and integer operations (FLOP and INTOP)

Comparison / Accuracy Level	Low	Medium	High
			Memory Access Patterns with memory traffic data and memory usage issues

You can choose custom accuracy and set a custom perspective flow for your application. For more information, see [Customize Vectorization and Code Insights Perspective](#).



NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).

Customize Vectorization and Code Insights Perspective

Customize the perspective flow to better fit your goal and your application.

If you change any of the analysis settings from the **Analysis Workflow** tab, the accuracy level changes to **Custom** automatically. With this accuracy level, you can customize the perspective flow and/or analysis properties.

To change the properties of a specific analysis:

1. Expand the analysis details on the **Analysis Workflow** pane with .
2. Select desired settings.
3. For more detailed customization, click the gear  icon. You will see the **Project Properties** dialog box open for the selected analysis.
4. Select desired properties and click **OK**.

For a full set of available properties, click the  icon on the left-side pane or go to **File > Project Properties**.

The following tables cover project properties applicable to the analyses in the Vectorization and Code Insights perspective.

Common Properties

Use This	To Do This
Target type drop-down	<ul style="list-style-type: none"> Analyze an executable or script (choose Launch Application). Analyze a process (choose Attach to Process). <p>If you choose Attach to Process, you can either inherit settings from the Survey Hotspots Analysis Type or specify the needed settings.</p>
Inherit settings from Visual Studio project checkbox and field (Visual Studio* IDE only)	<p>Inherit Intel Advisor project properties from the Visual Studio* startup project (enable).</p> <p>If enabled, the Application, Application parameters, and Working directory fields are pre-filled and cannot be modified.</p> <p>NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.</p>

Use This	To Do This
Application field and Browse... button	Select an analysis target executable or script. If you specify a script in this field, consider specifying the executable in the Advanced > Child application field (required for Dependencies analysis).
Application parameters field and Modify... button	Specify runtime arguments to use when performing analysis (equivalent to command line arguments).
Use application directory as working directory checkbox	Automatically use the value in the Application directory to pre-fill the Working directory value (enable).
Working directory field and Browse... button	Select the working directory.
User-defined environment variables field and Modify... button	Specify environment variables to use during analysis.
Managed code profiling mode drop-down	<ul style="list-style-type: none"> Automatically detect the type of target executable as Native or Managed, and switch to that mode (choose Auto). Collect data for native code and do not attribute data to managed code (choose Native). Collect data for both native and managed code, and attribute data to managed code as appropriate (choose Mixed). Consider using this option when analyzing a native executable that makes calls to managed code. Collect data for both native and managed code, resolve samples attributed to native code, and attribute data to managed source only (choose Managed). The call stack in the analysis result displays data for managed code only.
Child application field	<p>Analyze a file that is not the starting application. For example: Analyze an executable (identified in this field) called by a script (identified in the Application field).</p> <p>Invoking these properties could decrease analysis overhead.</p> <hr/> <p>NOTE For the Dependencies Analysis Type: If you specify a script file in the Application field, you must specify the target executable in the Child application field.</p> <hr/>
Modules radio buttons, field, and Modify... button	<ul style="list-style-type: none"> Analyze specific modules and disable analysis of all other modules (click the Include only the following module(s) radio button and choose the modules). Disable analysis of specific modules and analyze all other modules (click the Exclude only the following module(s) radio button and choose the modules). <p>Including/excluding modules could minimize analysis overhead.</p>

Use This	To Do This
Use MPI launcher checkbox	<p>Generate a command line (enable) that appears in the Get command line field based on the following parameters:</p> <ul style="list-style-type: none"> • Select MPI Launcher - Intel or another vendor • Number of ranks - Number of instances of the application • Profile ranks - All or a range of ranks to profile
Automatically stop collection after (sec) checkbox and field	<p>Stop collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could minimize analysis overhead.</p>

Survey Analysis Properties

Use This	To Do This
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <hr/> <p>Tip The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p> <hr/>
Sampling Interval selector	<p>Set the wait time between each analysis collection CPU sample while your target application is running.</p> <p>Increasing the wait time could decrease analysis overhead.</p>
Collection data limit, MB selector	<p>Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses.</p> <p>Decreasing the limit could decrease analysis overhead.</p>
Callstack unwinding mode drop-down list	<p>Set to After collection if:</p> <ul style="list-style-type: none"> • Survey analysis runtime overhead exceeds 1.1x. • A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions. <p>Otherwise, set to During Collection. This mode improves stack accuracy but increases overhead.</p>
Stitch stacks checkbox	<p>Restore a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload (enable).</p> <p>Disable if Survey analysis runtime overhead exceeds 1.1x.</p>
Analyze MKL Loops and Functions checkbox	<p>Show Intel® oneAPI Math Kernel Library (oneMKL) loops and functions in Intel Advisor reports (enable).</p>

Use This	To Do This
	Enabling could increase analysis overhead.
Analyze Python loops and functions checkbox	Show Python* loops and functions in Intel Advisor reports (enable). Enabling could increase analysis overhead.
Analyze loops that reside in non-executed code paths checkbox	Collect a variety of data during analysis for loops that reside in non-executed code paths, including loop assembly code, instruction set architecture (ISA), and vector length (enable). Enabling could increase analysis overhead. NOTE Analyzing non-executed code paths in binaries that target multiple ISAs (contain multiple code paths) is available only for binaries compiled using the <code>-ax</code> (Linux* OS) / <code>Qax</code> (Windows* OS) option with an Intel compiler.
Enable registry spill/fill analysis checkbox	Calculate the number of consecutive load/store operations in registers and related memory traffic (enable). Enabling could increase analysis overhead.
Enable static instruction mix analysis checkbox	Statically calculate the number of specific instructions present in the binary (enable). Enabling could increase analysis overhead.
Source caching drop-down list	<ul style="list-style-type: none"> Delete source code cache from a project with each analysis run (default; choose Clear cached files). Keep source code cache within the project (choose Keep cached files).

Trip Counts and FLOP Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds.
Collect information about Loop Trip Counts checkbox	Measure loop invocation and execution (enable).

Use This	To Do This
Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage checkbox	Measure floating-point operations, integer operations, and memory traffic (enable).
Collect stacks checkbox	Collect call stack information when performing analysis (enable). Enabling could increase analysis overhead.
Capture metrics for dynamic loops and functions checkbox	Collect metrics for dynamic Just-In-Time (JIT) generated code regions.

Memory Access Patterns Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Suppression mode group box	<ul style="list-style-type: none"> Report possible memory issues in system modules (choose the Show problems in system modules radio button). Do not report possible memory issues in system modules (choose the Suppress problems in system modules radio button).
Loop call count limit selector	Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances. Supplying a non-zero value could decrease analysis overhead.
Instance of interest selector	Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes. Supplying a non-zero value could decrease analysis overhead.
Report stack variables checkbox	Report stack variables for which memory access strides are detected (enable). Enabling could increase analysis overhead.
Report heap allocated variables checkbox	Report heap-allocated variables for which memory access strides are detected (enable). Enabling could increase analysis overhead.
Enable CPU cache simulation checkbox	Model cache misses, cache misses and cache line utilization, or cache misses and loop footprint (enable and select desired options). Enabling could increase analysis overhead.
Cache associativity drop-down list	Set the cache associativity for modeling CPU cache behavior. You can set the value to the following power-of-two integers: 1, 2, 4, 8, 16.
Cache sets drop-down list	Set the cache set size (in bytes) for modeling CPU cache behavior. You can set the value to the following power-of-two integers: 256, 512, 1024, 2048, 4096, 8192.

Use This	To Do This
Cache line size drop-down list	Set the cache line size (in bytes) to model CPU cache behavior. You can set the value to the following power-of-two integers: 4, 8, 16, 32, ..., up to 65536.
Cache simulation mode drop-down list	Set the focus for modeling CPU cache behavior: <ul style="list-style-type: none"> • Model cache misses only. • Model cache misses and memory footprint of a loop. Calculation: Cache line size x Number of unique cache lines accessed during simulation. • Model cache misses and cache line utilization.

Dependencies Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Suppression mode radio buttons	<ul style="list-style-type: none"> • Report possible dependencies in system modules (choose the Show problems in system modules radio button). • Do not report possible dependencies in system modules (choose the Suppress problems in system modules radio button).
Loop call count limit selector	Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances. Supplying a non-zero value could decrease analysis overhead.
Instance of interest selector	Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes. Supplying a non-zero value could decrease analysis overhead.
Analyze stack variables checkbox	Analyze parallel data sharing for stack variables (enable). Enabling could increase analysis overhead.
Filter stack variables by scope checkbox	Enable to report: <ul style="list-style-type: none"> • Variables initiated inside the loop as potential dependencies (warning) • Variables initialized outside the loop as dependencies (error) Enabling could increase analysis overhead.
Reduction Detection / Filter reduction variables checkbox	Mark all potential reductions by a specific diagnostic (enable). Enabling could increase analysis overhead.
Markup type checkbox	Select loops/functions by pre-defined markup algorithm. Supported algorithms are: <ul style="list-style-type: none"> • GPU generic - Select loops executed on a GPU. • OpenMP - Select OpenMP* loops. • SYCL - Select SYCL loops. • OpenCL - Select OpenCL™ loops.

Use This	To Do This
	<ul style="list-style-type: none"> • DAAL - Select Intel® oneAPI Data Analytics Library loops. • TBB - Select Intel® oneAPI Threading Building Blocks loops. <hr/> <p>NOTE This option is available only from the Analysis Workflow pane for the Offload Modeling perspective.</p> <hr/>

Run Vectorization and Code Insights Perspective from Command Line

Vectorization and Code Insights perspective includes several analyses that you can run depending on the desired result. The main analysis is the Survey, which collects performance data for loops and functions in your application and identifies under-vectorized and non-vectorized loops/functions. The Survey analysis is enough to get the basic insights about your application performance.

Tip See [Intel Advisor cheat sheet](#) for quick reference on command line interface.

Prerequisites

Set [Intel Advisor environment variables](#) with an automated script to enable the command line interface (CLI).

Run Vectorization and Code Insights Perspective

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

1. Run the Survey analysis.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

2. Run the Characterization analysis to collect trip counts and FLOP data:

```
advisor --collect=tripcounts --flop --stacks --project-dir=./advi_results -- ./myApplication
```

3. *Optional:* Run the Memory Access Patterns analysis for loops/functions with the *Possible Inefficient Memory Access Patter* issue.

```
advisor --collect=map --select=has-issue --project-dir=./advi_results -- ./myApplication
```

4. *Optional:* Run the Dependencies analysis to check for loop-carried dependencies in loops/functions with the *Assumed dependency present* issue:

```
advisor --collect=dependencies --project-dir=./advi_results --select=has-issue -- ./myApplication
```

You can view the results in the Intel Advisor graphical user interface (GUI), print a summary to a command prompt/terminal, or save to a file. See [View the Results](#) below for details.

Analysis Details

The Vectorization and Code Insights workflow includes the following analyses:

1. Survey to collect initial performance data.
2. Characterization with trip counts and FLOP data to collect additional performance details.
3. Memory Access Patterns (optional) to identify memory traffic data and memory usage issues.
4. Dependencies (optional) to identify loop-carried dependencies.

Each analysis has a set of additional options that modify its behavior and collect additional performance data. The more analyses you run and option you use, the more useful data about your application you get.

Consider the following options:

Characterization Options

To run the Characterization analysis, use the following command line action: `--collect=tripcounts`.

Recommended action options:

Options	Description
<code>--flop</code>	Collect data about floating-point and integer operations, memory traffic, and mask utilization metrics for AVX-512 platforms.
<code>--stacks</code>	Enable advanced collection of call stack data.
<code>--enable-cache-simulation</code>	Model CPU cache behavior on your target application.
<code>--cache-config=<config></code>	Set the cache hierarchy to collect modeling data for CPU cache behavior. Use with <code>enable-cache-simulation</code> . The value should follow the template: [<num_of_caches>]: [<num_of_ways_caches_connected>]: [<cache_size>]:[<cacheline_size>] for each of three cache levels separated with a /.
<code>--cachesim-associativity=<num></code>	Set the cache associativity for modeling CPU cache behavior: 1 2 4 8 (default) 16. Use with <code>enable-cache-simulation</code> .
<code>--cachesim-mode=<mode></code>	Set the focus for modeling CPU cache behavior: <code>cache-misses</code> <code>footprint</code> <code>utilization</code> . Use with <code>enable-cache-simulation</code> .

See [advisor Command Option Reference](#) for more options.

Memory Access Patterns Options

The Memory Access Patterns analysis is optional because it adds a high overhead. To run the Memory Access Patterns analysis, use the following command line action: `--collect=map`.

Recommended action options:

Options	Description
<code>--select=<string></code>	Select loops for the analysis by loop IDs, source locations, or criteria such as <code>scalar</code> , <code>has-issue</code> , or <code>markup=<markup-mode></code> . This option is required. See select for more selection options.
<code>--enable-cache-simulation</code>	Model CPU cache behavior on your target application.

Options	Description
<code>--cachesim-cacheline-size=<num></code>	Set the cache line size (in bytes) for modeling CPU cache behavior: 4 8 16 32 64 (default) 128 256 512 1024 2048 4096 8192 16384 32768 65536. Use with <code>enable-cache-simulation</code> .
<code>--cachesim-sets=<num></code>	Set the cache set size (in bytes) for modeling CPU cache behavior: 256 512 1024 2048 4096 (default) 8192. Use with <code>enable-cache-simulation</code> .

See [advisor Command Option Reference](#) for more options.

Dependencies Options

The Dependencies analysis is optional because it adds a high overhead and is mostly necessary if you have scalar loops/functions in your application. To run the Dependencies analysis, use the following command line action: `--collect=dependencies`.

Recommended action options:

Options	Description
<code>--select=<string></code>	Select loops for the analysis by loop IDs, source locations, criteria such as <code>scalar</code> , <code>has-issue</code> , or <code>markup=<markup-mode></code> . This option is required. See select for more selection options.
<code>--filter-reductions</code>	Mark all potential reductions with a specific diagnostic.

See [advisor Command Option Reference](#) for more options.

Next Steps

Continue to [explore the Vectorization and Code Insights results](#) with a preferred method. For details about the metrics reported, see [CPU and Memory Metrics](#).

See Also

[Analyze Vectorization Perspective](#) Improve your application performance, get code-specific recommendations for how to fix vectorization issues and quick visibility into source code and assembly code by running the Vectorization and Code Insights perspective.

[Command Line Interface](#) This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

[Minimize Analysis Overhead](#)

[Analyze MPI Applications](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

Vectorization Accuracy Levels in Command Line

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

In CLI, each accuracy level corresponds to a set of commands with specific options that you should run one by one to get a desired result.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium	High
Overhead	1.1x	5 - 8x	10 - 40x
Goal	Get basic insights about how well your application is vectorized and how you can improve vectorization efficiency	Get more insights about how well your application is vectorized and the number of iterations in loops/functions	Get detailed insights about your application performance, including performance issues and detailed optimization recommendations
Analyses	Survey	Survey + Characterization (Trip Counts)	Survey + Characterization (Trip Counts, FLOP, Call Stacks) + Memory Access Patterns
Result	Basic Survey report	Survey report extended with trip count data	Extended Survey report with trip counts and floating-point and integer operations (FLOP and INTOP) Memory Access Patterns with memory traffic data and memory usage issues

You can generate commands for a desired accuracy level from the Intel Advisor GUI. See [Generate Command Lines from GUI](#) for details.

NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).

Consider the following command examples.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Low Accuracy

To run the Vectorization and Code Insights perspective with the low accuracy:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

Medium Accuracy

To run the Vectorization and Code Insights perspective with the medium accuracy:

1. Run the Survey analysis:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

2. Run the Trip Counts analysis:

```
advisor --collect=tripcounts --enable-data-transfer-analysis --project-dir=./advi_results -- ./myApplication
```

High Accuracy

To run the Vectorization and Code Insights perspective with the high accuracy:

1. Run the Survey analysis:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

2. Run the Trip Counts and FLOP analysis:

```
advisor --collect=tripcounts --flop --stacks --project-dir=./advi_results -- ./myApplication
```

3. Run the Memory Access Pattern analysis for the loops that have the *Possible Inefficient Memory Access Pattern* issue:

```
advisor --collect=map --select=has_issue --project-dir=./advi_results -- ./myApplication
```

You can view the results in the Intel Advisor GUI or generate an interactive HTML report.

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[Run Vectorization and Code Insights Perspective from Command Line](#)

[Minimize Analysis Overhead](#)

Explore Vectorization and Code Insights Results

Intel® Advisor provides several ways to view the Vectorization and Code Insights results.

View Result in CLI

If you run the Vectorization and Code Insights perspective from command line, you can print the results to a terminal or a command prompt and save them to a .txt, .csv, or .xml file.

For example, to generate the Survey report:

```
advisor --report=survey --project-dir=./advi_results
```

You should see a similar result:

ID	Function Call Sites	Total	Self		
Type	Why No Vectorization ...				
	and Loops	Time			
Time			...		
14	[loop in main at mmult_serial.cpp:79]	0.495s	0.495s	Vectorized Versions 1	vectorization possible but seems inefficient ...
6	-[loop in main at mmult_serial.cpp:79]	0.275s	0.275s	Vectorized	
	(Body)		...		
3	-[loop in main at mmult_serial.cpp:79]	0.205s	0.205s	Vectorized	
	(Body)		...		
7	-[loop in main at mmult_serial.cpp:79]	0.015s	0.015s		
	Peeled		...		
11	-[loop in main at mmult_serial.cpp:79]	0s	0s	Remainder	vectorization possible but seems inefficient ...
4	[loop in main at mmult_serial.cpp:79]	0.510s	0.015s	Scalar	inner loop was already vectorized ...
12	[loop in main at mmult_serial.cpp:79]	0.510s	0s	Scalar Versions	1 inner loop was already vectorized ...
5	-[loop in main at mmult_serial.cpp:79]	0.510s	0s	Scalar	inner loop was already vectorized ...

The result is also saved into a text file `advisor-survey.txt` located at `./advi_results/eNNN/hsNNN`.

You can generate a report for any analysis you run. The *generic* report command looks as follows:

```
advisor --report=<analysis-type> --project-dir=<project-dir> --format=<format>
```

where:

- *<analysis-type>* is the analysis you want to generate the results for. For example, *survey* for the Survey report, *top-down* for the Survey report in a top-down view, *map* for the Memory Access Patterns, or *dependencies* for the Dependencies report.
- *--format=<format>* is a file format to save the results to. *<format>* is *text* (default), *csv*, *xml*.

You can also generate a report with the data from all analyses run and save it to a CSV file with the *--report=joined* action as follows:

```
advisor --report=joined --report-output=<path-to-csv>
```

where *--report-output=<path-to-csv>* is a path and a name for a *.csv* file to save the report to. For example, */home/report.csv*. This option is required to generate a joined report.

See [advisor Command Line Interface Reference](#) for more options.

View Result in GUI

If you run the *Vectorization and Code Insights* perspective from command line, a project is created automatically in the directory specified with *--project-dir*. All the collected results and analysis configurations are stored in the *.advixeproj* project, that you can view in the Intel Advisor.

To open the project in GUI, you can run the following command:

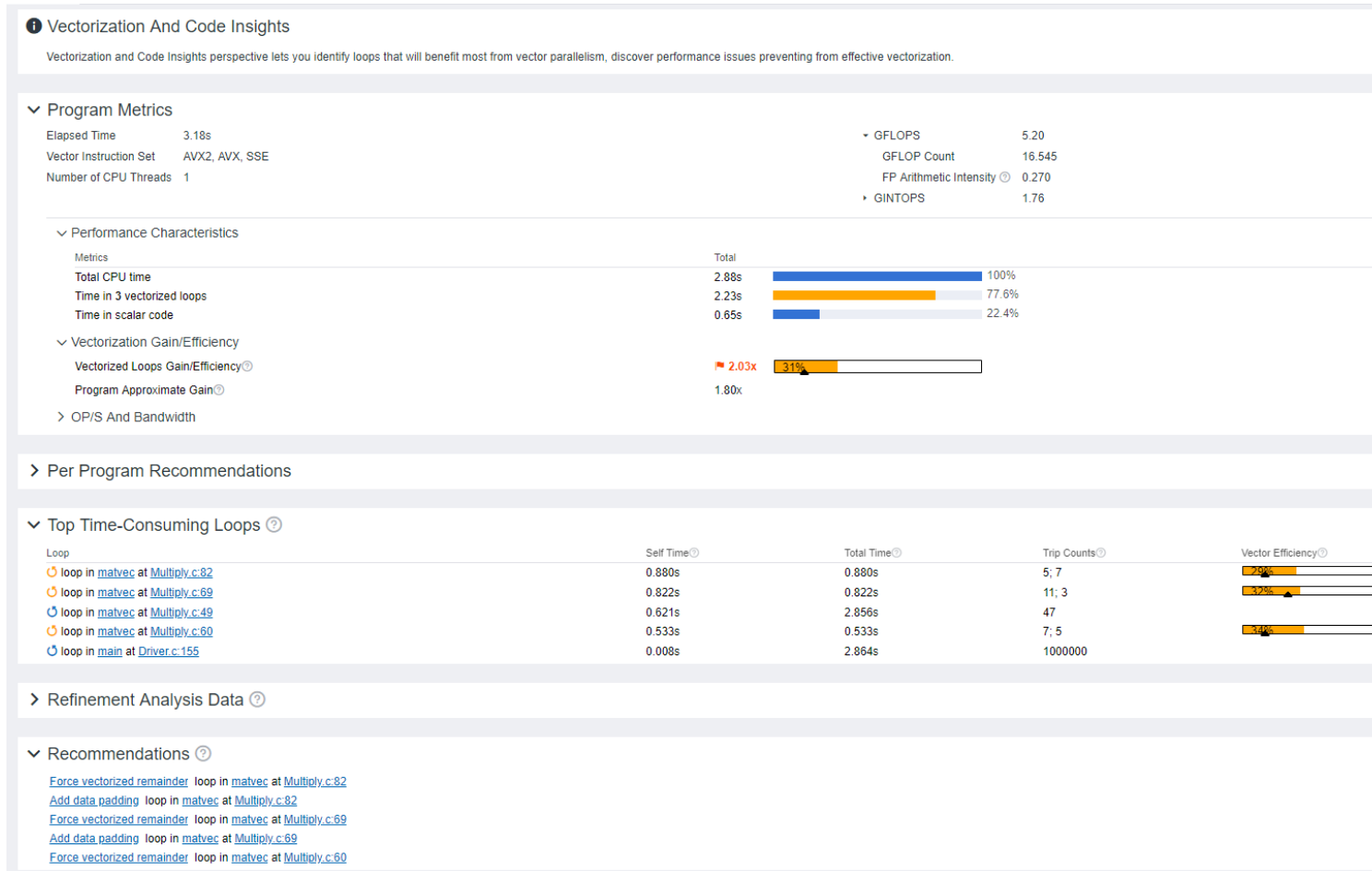
```
advisor-gui <project-dir>
```

NOTE If the report does not open, click **Show Result** on the Welcome pane.

If you run the *Vectorization and Code Insights* perspective from GUI, the result is opened automatically after the collection finishes.

You first see a Vectorization Summary report that includes the overall information about vectorized and not vectorized loops/functions in your code and the vectorization efficiency, including:


- Performance metrics of your program and the speedup for the vectorized loops/functions
- Top five time-consuming loops and top five optimization recommendations with the highest confidence



Save a Read-only Snapshot

A snapshot is a read-only copy of a project result, which you can view at any time using the Intel Advisor GUI. You can save a snapshot for a project using Intel Advisor GUI or CLI.



To save an active project result as a read-only snapshot from GUI: Click the  button in the top ribbon of the report. In the **Create a Result Snapshot** dialog box, enter the snapshot details and save it.

To save an active project result as a read-only snapshot from CLI:

```
advisor --snapshot --project-dir=<project-dir> [--cache-sources] [--cache-binaries] --<snapshot-path>
```

where:

- `--cache-sources` is an option to add application source code to the snapshot.
- `--cache-binaries` is an option to add application binaries to the snapshot.
- `<snapshot-path>` is a path and a name for the snapshot. For example, if you specify `/tmp/new_snapshot`, a snapshot is saved in a `tmp` directory as `new_snapshot.adviceexpz`. You can skip this and save the snapshot to a current directory as `snapshotXXX.adviceexpz`.

To open the result snapshot in the Intel Advisor GUI, you can run the following command:

```
advisor-gui <snapshot-path>
```

You can visually compare the saved snapshot against the current active result or other snapshot results.

See [Create a Read-only Result Snapshot](#) for details.

Result Interpretation

When you run the Vectorization and Code Insights perspective, depending on a configuration chosen, the report can show different levels of details:

-
- [Examine Not-Vectorized and Under-Vectorized Loops](#)
- [Analyze Loop Call Count](#)
- [Investigate Memory Usage and Traffic](#)
- [Find Data Dependencies](#)

For a general overview of the report, see [Vectorization Report Overview](#).

See Also

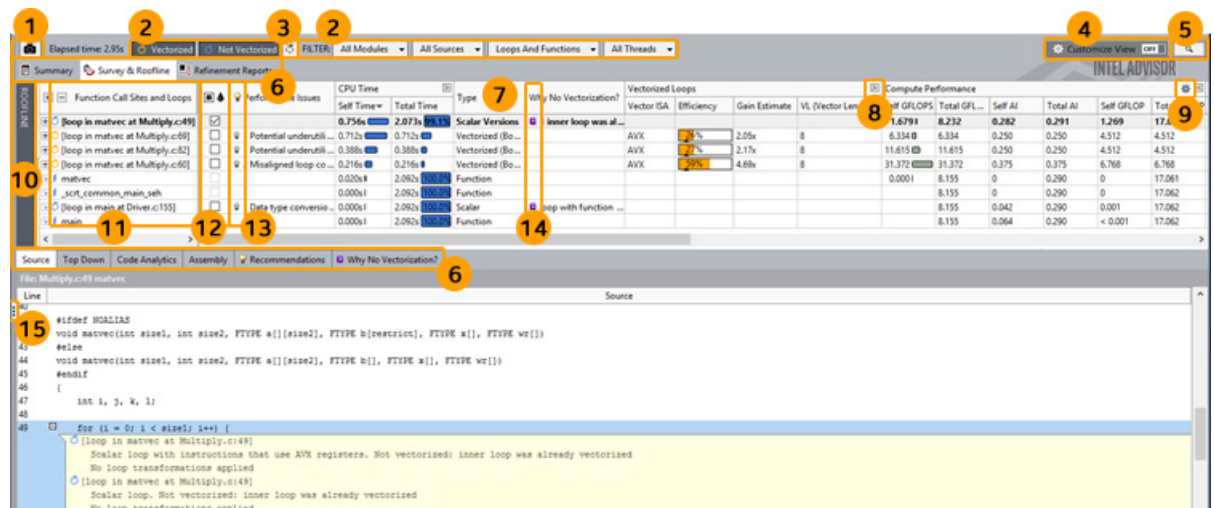
[Run Vectorization and Code Insights Perspective from GUI](#)

[Run Vectorization and Code Insights Perspective from Command Line](#)

[CPU Metrics](#) This reference section describes the contents of data columns in **Survey** and **Refinement Reports** of the Vectorization and Code Insights, CPU / Memory Roofline Insights, and Threading perspectives.

Vectorization Report Overview

Review the controls available in the main report of the Vectorization and Code Insights perspective of the Intel® Advisor.



There are many controls available to help you focus on the data most important to you, including the following:

- 1 Click the control to save a read-only result snapshot you can view any time.
Intel Advisor stores only the most recent analysis result. Visually comparing one or more snapshots to each other or to the most recent analysis result can be an effective way to judge performance improvement progress.
To open a snapshot, choose **File > Open > Result...**
- 2 Click the various **Filter** controls to temporarily limit displayed data based on your criteria.
- 3 Click the control to view loops in non-executed code paths for various instruction set architectures (ISAs). Prerequisites:
 - Compile the target application for multiple code paths using the Intel compiler.

- Enable the **Analyze loops in not executed code path** checkbox in **Project Properties > Analysis Target > Survey Hotspots Analysis**.

4 This toggle control currently combines two features: The **View Configurator** and the **Smart Mode** filter.

- **View Configurator** - Toggle on the **Customize View** control to choose the view layout to display: **Default**, **Smart Mode**, or a customized view layout. To create a customized view layout you can apply to this and other projects:
 - 1**Click the **Settings** control next to the **View Layout** drop-down list to open the **Configure Columns** dialog box.
 - 2**Choose an existing view layout in the **Configuration** drop-down list.
 - 3**Enable/disable columns to show/hide.

Outcome: *Copy n* is added to the name of the selected view layout in the **Configuration** drop-down list.

 - 4**Click the **Rename** button and supply an appropriate name for the customized view layout.
 - 5**Click **OK** to save the customized view layout.
- **Smart Mode Filter** - Toggle on the **Customize View** control to temporarily limit displayed data to the top potential candidates for optimization based on **Total CPU Time** (the time your application spends actively executing a function/loop and its callees). In the **Top** drop-down list, choose one of the following:
 - The **Number** of top loops/functions to display
 - The **Percent of Total CPU Time** the displayed loops/functions must equal or exceed

5 Click the button to search for specific data.

6 Click the tab to open various Intel Advisor reports or views.

7 Right-click a column header to:

- Hide the associated report column.
- Resume showing all available report columns.
- Open the **Configure Columns** dialog box (see #4 for more information).

8 Click the toggle to show all available columns in a column set, and resume showing a limited number of preset columns in a column set.





9 Click the control to:

- Show options for customizing data in a column or column set.
- Open the **Configure Columns** dialog box (see #4 for more information).

For example, click the control in the **Compute Performance** column set to:

- Show data for floating-point operations only, for integer operations only, or for the sum of floating-point and integer operations.
- Determine what is counted as an integer operation in integer calculations:
 - Choose **Show Pure Compute Integer Operations** to count only ADD, MUL, IDIV, and SUB operations.
 - Choose **Show All Operations Processing Integer Data** to count ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shift, and rotate operations.

10 Click the control to show/hide a chart that helps you visualize actual performance against hardware-imposed performance ceilings, as well as determine the main limiting factor (memory bandwidth or compute capacity), thereby providing an ideal roadmap of potential optimization steps.

- 11 Click a data row in the top of the **Survey Report** to display more data specific to that row in the bottom of the **Survey Report**. Double-click a loop data row to display a **Survey Source** window. To more easily identify data rows of interest:
 -  = Vectorized function
 -  = Vectorized loop
 -  = Scalar function
 -  = Scalar loop
- 12 Click a checkbox to mark a loop for deeper analysis.
- 13 If present, click the image to display code-specific *how-can-I-fix-this-issue?* information in the **Recommendations** pane.
- 14 If present, click the image to view the reason automatic compiler vectorization failed in the **Why No Vectorization?** pane.
- 15 Click the control to show/hide the **Workflow** pane.

Examine Not-Vectorized and Under-Vectorized Loops

Accuracy Level

Low

Enabled Analyses




Survey







Result Interpretation

After running the Vectorization and Code Insights perspective with Low accuracy, you get a basic vectorization report, which shows not-vectorized and under-vectorized loops, and other performance issues.


In the Survey report:

1. Sort by the **Self-Time** and/or **Total-Time** column to find top time-consuming loops.

Self Time▼	
0,891s	 11,5%
0,281s	
0,266s	

2. Check whether your target loop or function is vector or scalar. Intel Advisor helps you to differentiate vector and scalar using the following icons:
 -  - vectorized function
 -  - vectorized loop
 -  - scalar function
 -  - scalar loop
3. Use filters to hide the code sides that you do not want to tweak now:  **Vectorized** and  **Not Vectorized**
4. Decide what loops or functions to investigate:
 - If loop/function is scalar
 - If loop/function is vectorized

If Loop/Function is Scalar

If the target loop/function is scalar ( or *f*), you need to understand why the compiler did not vectorize the loop/function.

Several reasons are possible:

NOTE

See [OpenMP* Pragmas Summary](#) in the Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference for more information about the directives mentioned below.

Possible Reason	To Confirm	To Do
Assumed dependency	Refer to Why No Vectorization? column. Search for <i>Vector dependence prevents vectorization</i> issue.	Run the Dependencies analysis . <ul style="list-style-type: none"> If no dependencies are found, force vectorization with the <code>omp simd</code> directive or provide other vectorization recommendations to compiler. If dependencies are confirmed, resolve them, or move to the next loop.
Function call in the loop	Refer to Why No Vectorization? column. Search for issues: <ul style="list-style-type: none"> <i>Function call present</i> <i>Indirect function call present</i> <i>Serialized user function call present</i> 	For issue: <i>Function call present</i> , do one of the following: <ul style="list-style-type: none"> Inline function into the loop. Vectorize the function with the <code>omp declare simd</code> directive. For issues <i>Indirect function call present</i> or <i>Serialized user function call present</i> , refer to guidelines in the Recommendations tab.
Compiler-assumed inefficient vectorization	Refer to Why No Vectorization? column. Search for the <i>Loop vectorization possible but seems inefficient</i> issue.	Try forcing vectorization with the <code>omp simd</code> directive. If forcing vectorization doesn't provide tangible results, consider experimenting with other directives. To better understand performance implications and potential speed-up, consider running additional analyses: <ul style="list-style-type: none"> Trip Counts Memory Access Patterns
Other	Refer to <ul style="list-style-type: none"> Why No Vectorization? column Vector Issues column 	Study the Compiler Diagnostic Details and Advisor Recommendations to resolve the issues.

If Loop/Function is Vectorized

If the target loop is vectorized (🔄 or f), ensure vector efficiency is above 90%.

If efficiency is below 90%, consider the following:

Possible Reason	To Confirm	To Do
ISA	Refer to Vectorized Loops/ Vector ISA column to check the ISA version used in the application.	Change the target ISA by specifying corresponding compiler flags.
Inefficient peel/remainder	Refer to Vector Issues column. Search for the <i>Inefficient Peel/Reminder</i> issue. Or check if the time spent in peel/remainder is significant.	Resolve the issues: <ul style="list-style-type: none"> • Check Recommendations tab. • Run the Trip Counts analysis.
Possible inefficient memory access	Refer to Vector Issues column. Search for the <i>Possible Inefficient Memory Access</i> issue. Refer to Instruction Set Analysis/Traits column. Search for the following traits: <ul style="list-style-type: none"> • extracts • inserts • gather • scatter 	Run the Memory Access Patterns analysis.
Type conversions present	Refer to Instruction Set Analysis/Traits column. Search for the <i>Type Conversions</i> metric.	Remove redundant type conversions from float to double that might lead to smaller vector length and reduced vectorization efficiency.
Unaligned vector access in loop	Refer to Advanced/ Vectorization Details column. Search for the <i>Unaligned access in vector loop</i> metric.	Align data.
Register pressure	Refer to Vector Issues column. Search for the <i>Vector register spilling possible</i> issue.	Resolve the issue by doing one of the following: <ul style="list-style-type: none"> • Decrease loop unroll factor. • Split the loop into smaller parts.
Potential underutilization of FMA instructions	Refer to Vector Issues column. Search for the <i>Potential underutilization of FMA instructions</i> issue.	Resolve the issue by doing one of the following: <ul style="list-style-type: none"> • Change the target ISA. • Explicitly enable FMA generation and vectorization.
Other	Refer to Vector Issues column.	Follow the Intel Advisor recommendations to resolve the issues.

Next Steps

- Investigate Memory Usage and Traffic
- Find Data Dependencies

Analyze Loop Call Count

Accuracy Level

Medium

Enabled Analyses

Survey + Trip Counts (Characterization)

NOTE Collecting additional data may substantially increase report generation time. There is a variety of techniques available to minimize data collection, result size, and execution time. Check [Minimize Analysis Overhead](#).

Result Interpretation

After you run the Vectorization and Code Insights perspective with *medium* accuracy and Trip Counts collection enabled, Intel® Advisor dynamically identifies the number of times loops are invoked and execute and extends the basic vectorization report with the Trip Counts data. Use Trip Counts data to analyze parallelism granularity more deeply and fine-tune vector efficiency and capability.

Function Call Sites and Loops		Trip Counts					
		Average	Min	Max	Call Count	Iteration D...	Loop Instance Total Time
🔍 [loop in main at Driver.c:171]		6	6	6	47000000	< 0.001s	< 0.001s
🔍 [loop in main at Driver.c:164]		6	6	6	47000000	< 0.001s	< 0.001s
🔍 [loop in main at Driver.c:158]		47	47	47	1000000	< 0.001s	< 0.001s
f main					1		0.763s
🔍 [loop in main at Driver.c:155]		1000000	1000000	1000000	1	< 0.001s	0.763s

By default, the **Trip Counts** column shows only **Average** and **Call Count** metrics. Look for the following to find good candidates for optimization:

- Detect loops with too-small trip counts and trip counts that are not a multiple of vector length.
- A high number in the **Call Count** column means there is an outer loop in the selected loop call chain with high trip count values.
- If the loop has a low trip count value, the outer loop could be a better candidate for parallelization (threading/vectorization).

To optimize such loops, follow the Intel® Advisor **Recommendations** for the loop/function, for example, use specific recommended pragmas to provide the information about loop trip counts to a compiler.

Next Steps

For further investigation, you can run the Vectorization and Code Insights perspective with a higher accuracy level or with different configurations:

- [Examine Not-Vectorized and Under-Vectorized Loops](#)
- [Investigate Memory Usage and Traffic](#)
- [Find Data Dependencies](#)

Investigate Memory Usage and Traffic

Accuracy Level

High

Enabled Analyses

Survey with register spill/fill analysis + Trip Counts, FLOP, Call Stacks (Characterization) + Memory Access Patterns

NOTE Collecting additional data may substantially increase report generation time. There is a variety of techniques available to minimize data collection, result size, and execution time. Check [Minimize Analysis Overhead](#).

Result Interpretation

After you run the Vectorization and Code Insights perspective with *high* accuracy and full Characterization and Memory Access Patterns steps enabled, Intel® Advisor:


- Extends the Survey report with the **Compute Performance** and **Memory** metrics.
- Adds Memory Access Patterns data to the **Refinement Report** tab.

In the Survey report

Function Call Sites and Loops	Compute Performance						Memory
	Self GFLOPS	Self AI	Self GINTOPS	Self INT AI	Self Giga OPS	Self Overall AI	
[loop in main at Driver.c:171]	9.385	0.444	2.933	0.139	12.318	0.583	10.152
[loop in main at Driver.c:164]	36.732	0.375	4.591	0.047	41.323	0.422	18.048
[loop in main at Driver.c:158]	6.930	2.375	1.094	0.375	8.024	2.750	0.376
main		0.063		0.036		0.098	< 0.001
[loop in main at Driver.c:155]		0.083		0.167		0.250	0.012

Use the FLOP data to analyze application memory usage and performance values that help you make better decisions about your vectorization strategy.

Compute Performance column

You can configure this column to show only metrics for a specific type of operations used in your application. Click the  control in the **Compute Performance** column set header and choose the desired drop-down option to:

- Show data for floating-point operations only, for integer operations only, or for both floating-point and integer operations.
- Determine what is counted as an integer operation in integer calculations:
 - Choose **Show Pure Compute Integer Operations** to count only ADD, MUL, IDIV, and SUB operations.
 - Choose **Show All Operations Processing Integer Data** to count ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shift, and rotate operations.

Select a specific loop/function to see the details about FLOP and/or integer operation utilization in the **Code Analytics** tab:

Statistics for FLOP ▾
And Data Transfers

Self ☐ Total ☒

	Per loop	Per Iteration	Per Instance
GFLOP [?]	6.77	2.40e-08	1.44e-07
GFLOPS [?]		36.73	
AI [?]		0.38	
Mask Utilization [?]		-	
L1 Gb [?]	18.05	6.40e-08	3.84e-07
L1 Gb/s [?]		97.95	
Elapsed Time [?]	0.18s	6.53e-10s	3.92e-09s

Memory column

You can configure this column to show only metrics for one or more specific memory levels and specific types of operations. Click the gear icon in the **Memory** column set header and choose the desired drop-down option to determine which columns to display in the grid:

NOTE This data is only available if cache simulation is enabled. By default, Intel® Advisor collects only L1 traffic, so you will not be able to select memory levels or loads/stores.

- Show data for **L1**, **L2**, **L3**, or **DRAM** memory metrics, or show a **Customized Column Layout**.
- Show data for memory load operations only, store operations only, or the sum of both.

You can choose to hide the current column, **Show All Columns**, or customize the columns displayed in the grid by choosing **Configure Column Layouts**.

You can use the traffic and AI data reported in the Compute Performance and Memory columns to find the best candidates to examine the memory usage for in the Memory Access Patterns tab. For example, bad access pattern has stronger impact on loops/functions with higher AI value suggesting that you start with optimizing their memory usage.

In the Refinement report

Important Before running the Memory Access Patterns analysis, select loops/functions from the  column in the Survey report.

Get information about types of memory access in selected loops/functions, how you traverse your data, and how it affects your vector efficiency and cache bandwidth usage by running the Memory Access Patterns analysis.

Memory access patterns affect how data is loaded into and stored from the vector registers.

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate			
				Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memory Footprint	
[icon] [loop in main at Driver.c: ...]	No Information Available	75% / 0% / 25%	Mixed Strides	9KB	10KB	0B	S
[icon] [loop in main at Driver.c: ...]	RAW:1 WAW:2	No Information Available	No Information Available	No Information Available	No Information Available	No Information Available	lc
[icon] [loop in main at Driver.c: ...]	No Information Available	67% / 17% / 17%	Mixed Strides	9KB	9KB	0B	lc
<pre> 156 #ifdef NOFUNCALL 157 int i, j, l, m, sumx; 158 for (i = 0; i < sizel; i++) { 159 b[i] = 0; 160 #ifdef ALIGNED </pre>							
[icon] [loop in main at Driver.c: ...]	No Dependencies Found	No Information Available	No Information Available	No Information Available	No Information Available	No Information Available	lc
[icon] [loop in main at Driver.c: ...]	No Information Available	100% / 0% / 0%	All Unit Strides	160B	320B	0B	lc
[icon] [loop in main at Driver.c: ...]	No Information Available	50% / 0% / 50%	Mixed Strides	7KB	7KB	0B	lc

Memory Access Patterns Report			Dependencies Report		Recommendations					
ID	Stride	Type	Source	Nested Function	Variable references	Max. Per-Instruction Addr. Range	Modules	Site Name	Access Type	
P1	6	Constant stride	Driver.c:165		a	9KB	vec_samples.exe	loop_site_2	Read	
P3		Gather stride	Driver.c:172			9KB	vec_samples.exe	loop_site_2	Read	
P6		Parallel site information	Driver.c:158				vec_samples.exe	loop_site_2		
P14	0	Uniform stride	Driver.c:165		x	192B	vec_samples.exe	loop_site_2	Read	
P16	0	Uniform stride	Driver.c:172		x	192B	vec_samples.exe	loop_site_2	Read	
P18	1	Unit stride	Driver.c:165		wr	140B	vec_samples.exe	loop_site_2	Read	
P21	1	Unit stride	Driver.c:172		b	184B	vec_samples.exe	loop_site_2	Write	

To analyze how the data structure layout affects performance, pay attention to the following:

- Look for loops/functions that *do not* have "All Unit Strides" in the Access Pattern column to find optimization candidates.
- Strides Distribution** column reports the ratio of stride types for a selected loop/function and is color-coded:
 - Blue** is unite/uniform stride, which means that the instruction access memory sequentially or within the distance of one element.
 - Yellow** is constant stride, which means that the instructions access memory with the constant step of more than one element.
 - Red** is irregular stride, which means that the instructions access memory addresses that change by an unpredictable number of elements from iteration to iteration.

For vectorization, **unit** stride is the preferred distribution. Your goal is to eliminate irregular strides and minimize the constant stride to optimize memory usage.

- Click a loop in the top pane to see a detailed report for this loop below in the **Memory Access Patterns Report** tab.
 - Review details for each stride type that contributes to the loop/function with corresponding source locations.
 - Review the size of the strides, variables accessed, and source locations and modules.
- To optimize memory access patterns, follow the Intel® Advisor **Recommendations** for specific loops/functions.

In the **Memory Analysis Patterns Report** tab at the bottom of the **Refinement Reports** double-click a line to view the selected operation's source code.

Associated **Memory Access Patterns Source** window, from top left to bottom right:

- View Activation** pane - Enable or disable views shown in the Source view.
- Source View** pane - View source code of the selected loop/function.
- Assembly View** pane - View assembly source of the selected loop/function.
- Details View** pane - View details of the selected site.

Next Steps

- Run [CPU / Memory Roofline Insights](#) perspective to get a detailed view about your application performance.
- Cookbook: [Optimize Memory Access Patterns Using Loop Interchange and Cache Blocking Technique](#)

Find Data Dependencies

Prerequisites

Collect Survey data and select loops for the analysis from the 🔥 column in the Survey report.

Accuracy Level

Custom

Enabled Analyses

Dependencies

NOTE Collecting Dependencies data may substantially increase report generation time. There are a variety of techniques available to minimize data collection, result size, and execution time. Check [Minimize Analysis Overhead](#).

Result Interpretation

For safety purposes, compiler is often conservative when assuming data dependencies. The Dependencies analysis checks for real data dependencies in loops the compiler did not vectorize because of assumed dependencies and provides recommendations to help resolve the dependencies if detected.

NOTE The Dependencies analysis is not enabled in any of the accuracy presets by default. Select it manually from the **Analysis Workflow** tab before executing the perspective.

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern
[loop in main at Driver.c: ...]	No Information Available	<div><div></div>75% / 0% / 25%</div>	Mixed Strides
[loop in main at Driver.c: ...]	RAW:1 WAW:2	No Information Available	No Information Available
<pre>153 //start timing the matrix multiply code 154 startTime = clock_it(); 155 for (k = 0;k < REPEATNTIMES;k++) { 156 #ifdef NOFUNCCALL 157 int i, j, l, m, sumx;</pre>			
[loop in main at Driver.c: ...]	No Information Available	<div><div></div>67% / 17% / 17%</div>	Mixed Strides
<			



Memory Access Patterns Report	Dependencies Report	Recommendations
-------------------------------	---------------------	-----------------

Problems and Messages						
ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	loop_site_77	Driver.c	vec_samples.exe	✓ Not a problem
P5		Read after write dependency	loop_site_77	Driver.c	vec_samples.exe	New
P6		Write after write dependency	loop_site_77	Driver.c	vec_samples.exe	New
P7		Write after write dependency	loop_site_77	Driver.c	vec_samples.exe	New

- Click a loop in the top pane to see a detailed report for each dependency found in this loop below in the **Dependencies Report** tab.
- Use the **Dependencies Report** to view each reported problem and its associated code locations.
 - If no dependencies found, it is safe to force vectorization.
 - For loops/functions with real dependencies, Intel Advisor reports dependency type and severity in the Loop-Carried Dependency column in the top pane.
- Use the **Dependencies Source** window to view the focus and related source code regions to help you understand the cause of the reported problem.
- Use the **Code Locations** window to view the focus and related source code regions to help you understand the cause of the reported problem.
- To learn about a reported problem, right-click its name in the **Dependencies Report, Problems and Messages** pane and select **What Should I Do Next?**. This displays the help topic for that problem type with recommendations on how to resolve the dependency.
- Double-click a problem in the **Dependencies Report, Problems and Messages** pane to open the Dependencies Source tab and examine the problem in more detail.

Dependencies Report Overview

In the **Dependencies Report** tab at the bottom of the **Refinement Report**, review the following panes:

- **Problems and Messages** pane - Select the problems that you want to analyze by viewing their associated observations.
- **Code Locations** pane - View details about the code locations for the selected problem in the **Dependencies Report** window. Icons identify the focus code location  and related code location .
- **Filters** pane - Filter contents of the report tab.

Associated **Dependencies Source** window, from top left to bottom right:

- **Focus Code Location** pane - Use this pane to explore source code associated with focus code location in the **Dependencies Source** window.
- **Focus Code Location Call Stack** pane - Use this pane to select which source code appears in the **Focus Code Location** pane in the **Dependencies Source** window.
- **Related Code Locations** pane - Use this pane to explore source code associated with related code locations (related to the focus code location) in the **Dependencies Source** window.
- **Related Code Location Call Stack** pane - Use this pane to select which source code appears in the **Related Code Location** pane.
- **Code Locations** pane - Use this pane to view the details about the code location for the selected problem in the **Dependencies Report** window.
- **Relationship Diagram** pane - Use this pane to view the relationships among code locations for the selected problem.

Next Steps

[Dependencies Problem and Message Types Reference](#)

Analyze CPU Roofline

Visualize actual performance against hardware-imposed performance ceilings by running the CPU / Memory Roofline Insights perspective. It helps you determine the main limiting factor (memory bandwidth or compute capacity) and provides an ideal roadmap of potential optimization steps.

Use the **Roofline** chart to answer the following questions:

- What is the maximum achievable performance with your current hardware resources?
- Does your application work optimally on current hardware resources?
- If not, what are the best candidates for optimization?

- Is memory bandwidth or compute capacity limiting performance for each optimization candidate?

How It Works

The CPU / Memory Roofline Insights perspective includes the following steps:

1. Collect loop/function timings using the **Survey** analysis.
2. Collect floating-point and/or integer operations data, memory traffic data, and measure the hardware limitations of your hardware using the **FLOP** analysis in the **Characterization** step.

At this step, Intel® Advisor collects:

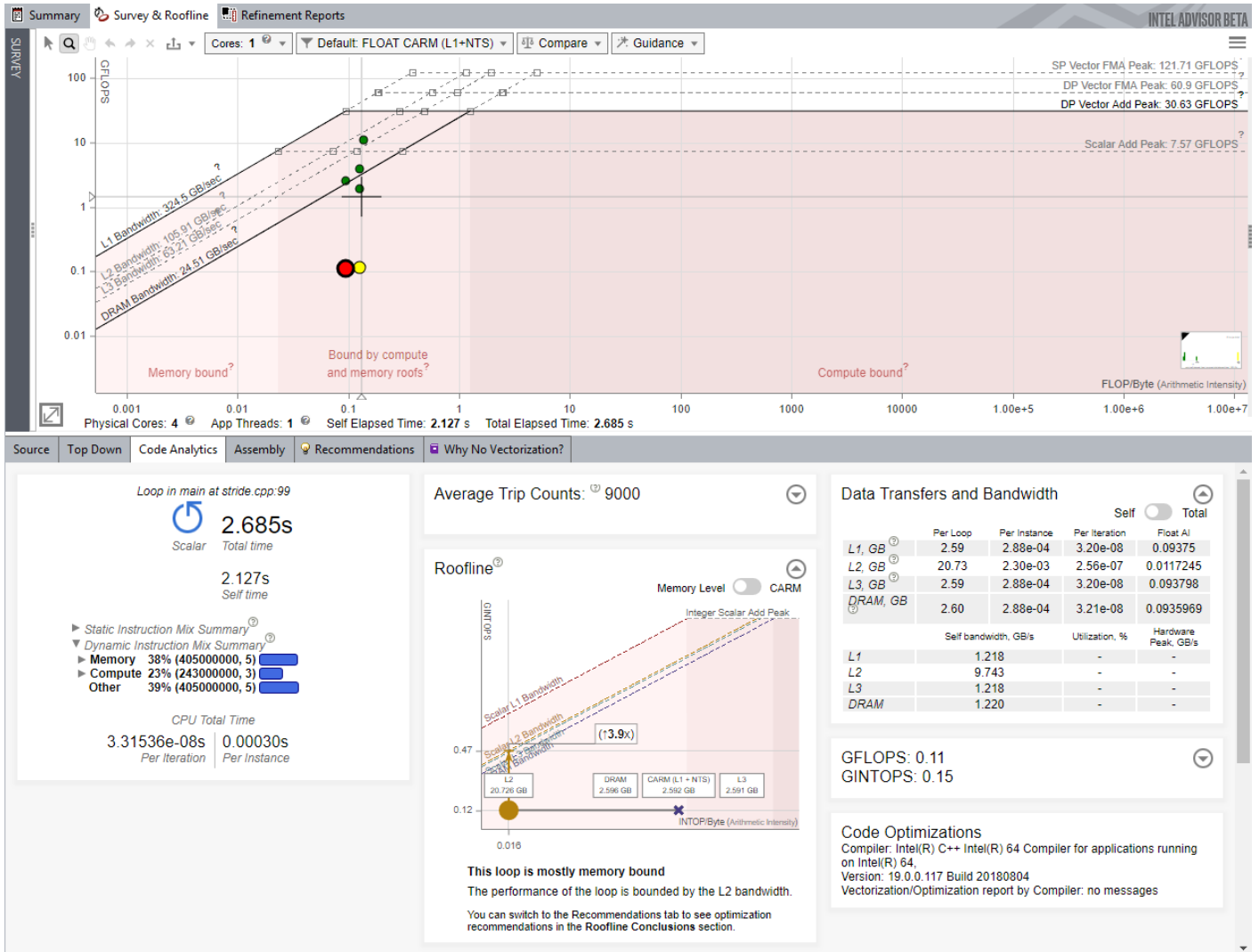
- **Compute** operations (floating-point operations (FLOP) and integer operations (INTOP)):
 - **FLOP** is calculated as a sum of the following classes of instructions multiplied by their iteration count: FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND
 - **INTOP** is calculated by default as a sum of the following classes of instructions multiplied by their iteration count: ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates.
- **Memory traffic** data that is calculated as a product of memory operations and the amount of bytes in the register accessed by the function/loop. For memory traffic calculation, Intel Advisor counts the following classes of memory instructions:
 - scalar and vector MOV instructions
 - GATHER/SCATTER instructions
 - VBM12 compress/expand instructions

NOTE This collection can take three to four times longer than the Survey analysis.

CPU Roofline Report

The **Roofline** chart plots an application's **achieved performance** and **arithmetic intensity** against the hardware **maximum achievable performance**:

- Arithmetic intensity (x axis) - measured in number of floating-point operations (FLOPs) and/or integer operations (INTOPs) per byte, based on the loop/function algorithm, transferred between CPU/VPU and memory
- Performance (y axis) - measured in billions of floating-point operations per second (GFLOPS) and/or billions of integer operations per second (GINTOPS)



See Also

[Run CPU / Memory Roofline Insights Perspective from GUI](#)

[Run CPU / Memory Roofline Insights Perspective from Command Line](#)

[CPU Roofline Report Overview](#) Review the controls available in the main report of the CPU / Memory Roofline Insights perspective of the Intel® Advisor.

Run CPU / Memory Roofline Insights Perspective from GUI

Prerequisite: In the graphical-user interface (GUI): [Create a project](#) and specify an analysis target and target options.


To run the CPU / Memory Roofline Insights perspective from the GUI:

- Configure the perspective and set analysis properties, depending on desired results:
 - Select a collection accuracy level with analysis properties preset for a specific result:
 - Low:** Get the basic CPU Cache-Aware Roofline chart with self data metrics.
 - Medium:** Get the detailed Memory-Level Roofline chart with total data metrics and an additional memory usage report.
 - Select the analyses and properties manually to adjust the perspective flow to your needs. The accuracy level is set to **Custom**.








The higher accuracy value you choose, the higher runtime overhead is added to your application. The **Overhead** indicator shows the overhead for the selected configuration. For the **Custom** accuracy, the overhead is calculated automatically for the selected analyses and properties.

By default, accuracy is set to **Low**. For more information, see [CPU Roofline Accuracy Presets](#).

2. *Optional:* If you want check for loop-carried dependency, select the **Dependencies** analysis. For more information about the Dependencies analysis and report, see [Find Data Dependencies](#).

3. Run the perspective: click  button.


While the perspective is running, you can do the following in the **Analysis Workflow** tab:

- Control the perspective execution:
 - Stop data collection and see the already collected data: Click the  button.
 - Pause data collection: Click the  button.
 - Cancel data collection and discard the collected data: Click the  button.
- Expand an analysis with  to control the analysis execution:
 - Pause the analysis: Click the  button.
 - Stop the currently running analysis and start the next analysis selected: Click the  button.
 - Interrupt execution of all selected analyses and see the already collected data: Click the  button.

To run the CPU / Memory Roofline Insights perspective with the Low accuracy from the command line interface:

```
advisor --collect=roofline --project-dir=./advi_results -- ./myApplication
```

See [Run CPU / Memory Roofline Insights from Command Line](#) for details.

NOTE To [generate command lines](#) for selected perspective configuration, click the  **Command Line** button.

Once the CPU / Memory Roofline Insights perspective collects data, the report opens showing a **Summary** tab. Continue to investigate the results:

- [Examine Bottlenecks on CPU Roofline Chart](#)
- [Examine Relationships Between Memory Levels](#)

NOTE After you run the CPU / Memory Roofline Insights perspective, the collected Survey data becomes available for all other perspectives. If you switch to another perspective, you can skip the Survey step and run only perspective-specific analyses.

CPU Roofline Accuracy Presets

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium
Overhead	5 - 10x	15 - 50x
Goal	Analyze how well your application uses memory and compute resources of a CPU and determine the main limiting factor (memory bandwidth or compute capacity)	Analyze how well your application uses CPU memory at different cache levels in more details
Analyses	Survey + Characterization (FLOP)	Survey + Characterization (Trip Counts and FLOP with call stacks for all memory levels) + Memory Access Patterns
Result	Cache-aware CPU Roofline for L1 cache	Memory-level CPU Roofline with call stacks (for L1, L2, L3, DRAM) Memory Access Patterns

You can choose custom accuracy and set a custom perspective flow for your application. For more information, see [Customize CPU / Memory Roofline Insights Perspective](#).



NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).


Customize CPU / Memory Roofline Insights Perspective

Customize the perspective flow to better fit your goal and your application.

If you change any of the analysis settings from the **Analysis Workflow** tab, the accuracy level changes to **Custom** automatically. With this accuracy level, you can customize the perspective flow and/or analysis properties.

To change the properties of a specific analysis:

1. Expand the analysis details on the **Analysis Workflow** pane with .
2. Select desired settings.
3. For more detailed customization, click the *gear*  icon. You will see the **Project Properties** dialog box open for the selected analysis.
4. Select desired properties and click **OK**.

For a full set of available properties, click the  icon on the left-side pane or go to **File > Project Properties**.

The following tables cover project properties applicable to the analyses in the CPU / Memory Roofline Insights perspective.

Common Properties

Use This	To Do This
Target type drop-down	<ul style="list-style-type: none"> Analyze an executable or script (choose Launch Application). Analyze a process (choose Attach to Process).

Use This	To Do This
	If you choose Attach to Process , you can either inherit settings from the Survey Hotspots Analysis Type or specify the needed settings.
Inherit settings from Visual Studio project checkbox and field (Visual Studio* IDE only)	<p>Inherit Intel Advisor project properties from the Visual Studio* startup project (enable).</p> <p>If enabled, the Application, Application parameters, and Working directory fields are pre-filled and cannot be modified.</p> <hr/> <p>NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.</p> <hr/>
Application field and Browse... button	<p>Select an analysis target executable or script.</p> <p>If you specify a script in this field, consider specifying the executable in the Advanced > Child application field (required for Dependencies analysis).</p>
Application parameters field and Modify... button	Specify runtime arguments to use when performing analysis (equivalent to command line arguments).
Use application directory as working directory checkbox	Automatically use the value in the Application directory to pre-fill the Working directory value (enable).
Working directory field and Browse... button	Select the working directory.
User-defined environment variables field and Modify... button	Specify environment variables to use during analysis.
Managed code profiling mode drop-down	<ul style="list-style-type: none"> Automatically detect the type of target executable as Native or Managed, and switch to that mode (choose Auto). Collect data for native code and do not attribute data to managed code (choose Native). Collect data for both native and managed code, and attribute data to managed code as appropriate (choose Mixed). Consider using this option when analyzing a native executable that makes calls to managed code. Collect data for both native and managed code, resolve samples attributed to native code, and attribute data to managed source only (choose Managed). The call stack in the analysis result displays data for managed code only.
Child application field	<p>Analyze a file that is not the starting application. For example: Analyze an executable (identified in this field) called by a script (identified in the Application field).</p> <p>Invoking these properties could decrease analysis overhead.</p>

Use This	To Do This
	<p>NOTE</p> <p>For the Dependencies Analysis Type: If you specify a script file in the Application field, you must specify the target executable in the Child application field.</p>
Modules radio buttons, field, and Modify... button	<ul style="list-style-type: none"> Analyze specific modules and disable analysis of all other modules (click the Include only the following module(s) radio button and choose the modules). Disable analysis of specific modules and analyze all other modules (click the Exclude only the following module(s) radio button and choose the modules). <p>Including/excluding modules could minimize analysis overhead.</p>
Use MPI launcher checkbox	<p>Generate a command line (enable) that appears in the Get command line field based on the following parameters:</p> <ul style="list-style-type: none"> Select MPI Launcher - Intel or another vendor Number of ranks - Number of instances of the application Profile ranks - All or a range of ranks to profile
Automatically stop collection after (sec) checkbox and field	<p>Stop collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could minimize analysis overhead.</p>

Survey Analysis Properties

Use This	To Do This
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <p>Tip</p> <p>The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p>
Sampling Interval selector	<p>Set the wait time between each analysis collection CPU sample while your target application is running.</p> <p>Increasing the wait time could decrease analysis overhead.</p>
Collection data limit, MB selector	<p>Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses.</p> <p>Decreasing the limit could decrease analysis overhead.</p>

Use This	To Do This
Callstack unwinding mode drop-down list	<p>Set to After collection if:</p> <ul style="list-style-type: none"> Survey analysis runtime overhead exceeds 1.1x. A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions. <p>Otherwise, set to During Collection. This mode improves stack accuracy but increases overhead.</p>
Stitch stacks checkbox	<p>Restore a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload (enable).</p> <p>Disable if Survey analysis runtime overhead exceeds 1.1x.</p>
Analyze MKL Loops and Functions checkbox	<p>Show Intel® oneAPI Math Kernel Library (oneMKL) loops and functions in Intel Advisor reports (enable).</p> <p>Enabling could increase analysis overhead.</p>
Analyze Python loops and functions checkbox	<p>Show Python* loops and functions in Intel Advisor reports (enable).</p> <p>Enabling could increase analysis overhead.</p>
Analyze loops that reside in non-executed code paths checkbox	<p>Collect a variety of data during analysis for loops that reside in non-executed code paths, including loop assembly code, instruction set architecture (ISA), and vector length (enable).</p> <p>Enabling could increase analysis overhead.</p> <hr/> <p>NOTE</p> <p>Analyzing non-executed code paths in binaries that target multiple ISAs (contain multiple code paths) is available only for binaries compiled using the <code>-ax</code> (Linux* OS) / <code>Qax</code> (Windows* OS) option with an Intel compiler.</p> <hr/>
Enable registry spill/fill analysis checkbox	<p>Calculate the number of consecutive load/store operations in registers and related memory traffic (enable).</p> <p>Enabling could increase analysis overhead.</p>
Enable static instruction mix analysis checkbox	<p>Statically calculate the number of specific instructions present in the binary (enable).</p> <p>Enabling could increase analysis overhead.</p>
Source caching drop-down list	<ul style="list-style-type: none"> Delete source code cache from a project with each analysis run (default; choose Clear cached files). Keep source code cache within the project (choose Keep cached files).

Trip Counts and FLOP Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds.
Trip Counts / Collect information about Loop Trip Counts checkbox	Measure loop invocation and execution (enable).
FLOP / Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage checkbox	Measure floating-point operations, integer operations, and memory traffic (enable).
Callstacks / Collect callstacks checkbox	Collect call stack information when performing analysis (enable). Enabling could increase analysis overhead.
Capture metrics for dynamic loops and functions checkbox	Collect metrics for dynamic Just-In-Time (JIT) generated code regions.
Capture metrics for stripped binaries checkbox	Collect metrics for stripped binaries. Enabling could increase analysis overhead.
Cache Simulation / Enable Memory-Level Roofline with cache simulation checkbox	Model multiple levels of cache for data, such as counts of loaded or stored bytes for each loop, to plot the Roofline chart for all memory levels (enable). Enabling could increase analysis overhead.
Cache simulator configuration field	Specify a cache hierarchy configuration to model (enable and specify hierarchy). The hierarchy configuration template is: <code>[num_of_level1_caches]:[num_of_ways_level1_connected]:</code> <code>[level1_cache_size]:[level1_cacheline_size]/</code> <code>[num_of_level2_caches]:[num_of_ways_level2_connected]:</code> <code>[level2_cache_size]:[level2_cacheline_size]/</code> <code>[num_of_level3_caches]:[num_of_ways_level3_connected]:</code> <code>[level3_cache_size]:[level3_cacheline_size]</code> For example: <code>4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l</code> is the hierarchy configuration for: <ul style="list-style-type: none"> • Four eight-way 32-KB level 1 caches with line size of 64 bytes • Four four-way 256-KB level 2 caches with line size of 64 bytes • One sixteen-way 6-MB level 3 cache with line size of 64 bytes

Use This	To Do This
Data Transfer Simulation / Data transfer simulation mode drop-down	Select a level of details for data transfer simulation: <ul style="list-style-type: none"> • Off - Disable data transfer simulation analysis. • Light - Model data transfers between host and device memory. • Full - Model data transfers, attribute memory objects to loops that accessed the objects, and track accesses to stack memory.

Run CPU / Memory Roofline Insights Perspective from Command Line

To plot a Roofline chart, the Intel® Advisor does the following:

1. Collect OpenCL™ kernels timings and memory data using the Survey analysis with GPU profiling.
2. Measure the hardware limitations and collect floating-point and integer operations data using the Characterization analysis with GPU profiling.

Intel® Advisor calculates compute operations (FLOP and INTOP) as a weighted sum of the following groups of instructions: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Intel Advisor automatically determines data type in the collected operations using the `dst` register.

Tip See [Intel Advisor cheat sheet](#) for quick reference on command line interface.

Prerequisites

Set [Intel Advisor environment variables](#) with an automated script to enable the `advisor` command line interface (CLI).

Plot a CPU Roofline Chart

There are two methods to run the CPU Roofline. Use *one* of the following:

- Run the shortcut `--collect=roofline` command line action to execute the Survey and Characterization analyses with a single command. This method is recommended to run the CPU / Memory Roofline Insights perspective, but it does not support MPI applications.
- Run the Survey and Characterization analyses with the `--collect=survey` and `--collect=tripcounts` command actions separately one by one. This method is recommended if you want to analyze an MPI application.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Method 1. Run the Shortcut Command

To collect data for a CPU Roofline chart with a shortcut, run the following command:

```
advisor --collect=roofline --project-dir=./advi_results -- ./myApplication
```

This command collects data for a basic CPU Roofline chart based on the Cache-Aware Roofline model. You can add other option to the command to collect more data. See **Analysis Details** below for more options.

Method 2. Run the Analyses Separately

Use this method if you want to analyze an MPI application.

1. Run the Survey analysis.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```


2. Run the Characterization analysis to collect trip counts and FLOP data:

```
advisor --collect=tripcounts --flop --project-dir=./advi_results -- ./myApplication
```

These commands collect data for a basic CPU Roofline chart based on the Cache-Aware Roofline model. You can add other option to the command to collect more data. See **Analysis Details** below for more options.

You can view the results in the Intel Advisor graphical user interface (GUI), or generate an interactive HTML report. See View the Results below for details.

Analysis Details

The CPU / Memory Roofline Insights workflow includes the following analyses:

1. Roofline to plot a Roofline chart. This step sequentially runs the Survey and Characterization (trip counts and FLOP) analyses.
2. Memory Access Patterns (optional) to identify memory traffic data and memory usage issues.
3. Dependencies (optional) to identify loop-carried dependencies that might limit offloading.

Each analysis has a set of additional options that modify its behavior and collect additional performance data. The more analyses you run and option you use, the more useful data about your application you get.

Consider the following options:

Roofline Options

To run the Roofline analysis, use the following command line action: `--collect=roofline`.

NOTE You can also use this options with `--collect=tripcounts` if you want to run the analyses separately.

Recommended action options:

Options	Description
<code>--stacks</code>	Enable advanced collection of call stack data. Use this option to get a CPU Roofline with callstacks.
<code>--enable-cache-simulation</code>	Model CPU cache behavior on your target application. Use this option to get a Memory-level CPU Roofline that shows data for all memory levels.
<code>--cache-config=<config></code>	Set the cache hierarchy to collect modeling data for CPU cache behavior. Use with <code>enable-cache-simulation</code> . The value should follow the template: [<num_of_caches>]: [<num_of_ways_caches_connected>]: [<cache_size>]:[<cacheline_size>] for each of three cache levels separated with a /.
<code>--cachesim-associativity=<num></code>	Set the cache associativity for modeling CPU cache behavior: 1 2 4 8 (default) 16. Use with <code>enable-cache-simulation</code> .
<code>--cachesim-mode=<mode></code>	Set the focus for modeling CPU cache behavior: <code>cache-misses</code> <code>footprint</code> <code>utilization</code> . Use with <code>enable-cache-simulation</code> .

See [advisor Command Option Reference](#) for more options.

Memory Access Patterns Options

The Memory Access Patterns analysis is *optional* because it adds a high overhead. This analysis does not add more information to the CPU Roofline chart. The results are added to the Refinement report, which you can view from GUI or from CLI. Use it to understand the Memory-Level Roofline chart better and get more detailed optimization recommendations.

To run the Memory Access Patterns analysis, use the following command line action: `--collect=map`.

Recommended action options:

Options	Description
<code>--select=<string></code>	Select loops for the analysis by loop IDs, source locations, or criteria such as <code>scalar</code> , <code>has-issue</code> , or <code>markup=<markup-mode></code> . This option is required. See select for more selection options.
<code>--enable-cache-simulation</code>	Model CPU cache behavior on your target application.
<code>--cachesim-cacheline-size=<num></code>	Set the cache line size (in bytes) for modeling CPU cache behavior: 4 8 16 32 64 (default) 128 256 512 1024 2048 4096 8192 16384 32768 65536. Use with <code>enable-cache-simulation</code> .
<code>--cachesim-sets=<num></code>	Set the cache set size (in bytes) for modeling CPU cache behavior: 256 512 1024 2048 4096 (default) 8192. Use with <code>enable-cache-simulation</code> .

See [advisor Command Option Reference](#) for more options.

Dependencies Options

The Dependencies analysis is *optional* because it adds a high overhead and is mostly necessary if you have scalar loops/functions in your application. This analysis does not add more information to the CPU Roofline chart. The results are added to the Refinement report, which you can view from GUI or from CLI. Use it to get more detailed optimization recommendations.

To run the Dependencies analysis, use the following command line action: `--collect=dependencies`.

Recommended action options:

Options	Description
<code>--select=<string></code>	Select loops for the analysis by loop IDs, source locations, criteria such as <code>scalar</code> , <code>has-issue</code> , or <code>markup=<markup-mode></code> . This option is required. See select for more selection options.
<code>--filter-reductions</code>	Mark all potential reductions with a specific diagnostic.

See [advisor Command Option Reference](#) for more options.

Next Steps

Continue to [explore the CPU / Memory Roofline Insights results](#) with a preferred method. For details about the metrics reported, see [CPU and Memory Metrics](#).

See Also

[CPU / Memory Roofline Insights Perspective](#) Visualize actual performance against hardware-imposed performance ceilings by running the CPU / Memory Roofline Insights perspective. It helps you determine the main limiting factor (memory bandwidth or compute capacity) and provides an ideal roadmap of potential optimization steps.

[Command Line Interface](#) This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

[Minimize Analysis Overhead](#)

[Analyze MPI Applications](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

CPU Roofline Accuracy Levels in Command Line

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

In CLI, each accuracy level corresponds to a set of commands with specific options that you should run one by one to get a desired result.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium
Overhead	5 - 10x	15 - 50x
Goal	Analyze how well your application uses memory and compute resources of a CPU and determine the main limiting factor (memory bandwidth or compute capacity)	Analyze how well your application uses CPU memory at different cache levels in more details
Analyses	Survey + Characterization (FLOP)	Survey + Characterization (Trip Counts and FLOP with call stacks for all memory levels) + Memory Access Patterns
Result	Cache-aware CPU Roofline for L1 cache	Memory-level CPU Roofline with call stacks (for L1, L2, L3, DRAM) Memory Access Patterns

You can generate commands for a desired accuracy level from the Intel Advisor GUI. See [Generate Command Lines from GUI](#) for details.

NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).

Consider the following command examples.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Low Accuracy

To run the CPU / Memory Roofline Insights perspective with the low accuracy:

```
advisor --collect=roofline --project-dir=./advi_results -- ./myApplication
```

Medium Accuracy

To run the CPU / Memory Roofline Insights perspective with the medium accuracy:

1. Generate the Memory-level Roofline report with call stacks:

```
advisor --collect=roofline --stacks --enable-data-transfer-analysis --project-dir=./advi_results
-- ./myApplication
```

2. Run the Memory Access Pattern analysis for the loops that have the *Possible Inefficient Memory Access Pattern* issue:

```
advisor --collect=map --select=has-issue --project-dir=./advi_results -- ./myApplication
```

You can view the results in the Intel Advisor GUI or [generate an interactive HTML report](#).

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[Run CPU / Memory Roofline Insights from Command Line](#)

[Minimize Analysis Overhead](#)

Explore CPU/Memory Roofline Results

Intel® Advisor provides several ways to view the CPU / Memory Roofline Insights results.

View Results in GUI

If you run the CPU / Memory Roofline Insights perspective from command line, a project is created automatically in the directory specified with `--project-dir`. All the collected results and analysis configurations are stored in the `.advixeproj` project, which you can view in the Intel Advisor.

To open the project in GUI, run the following command:

```
advisor-gui <project-dir>
```

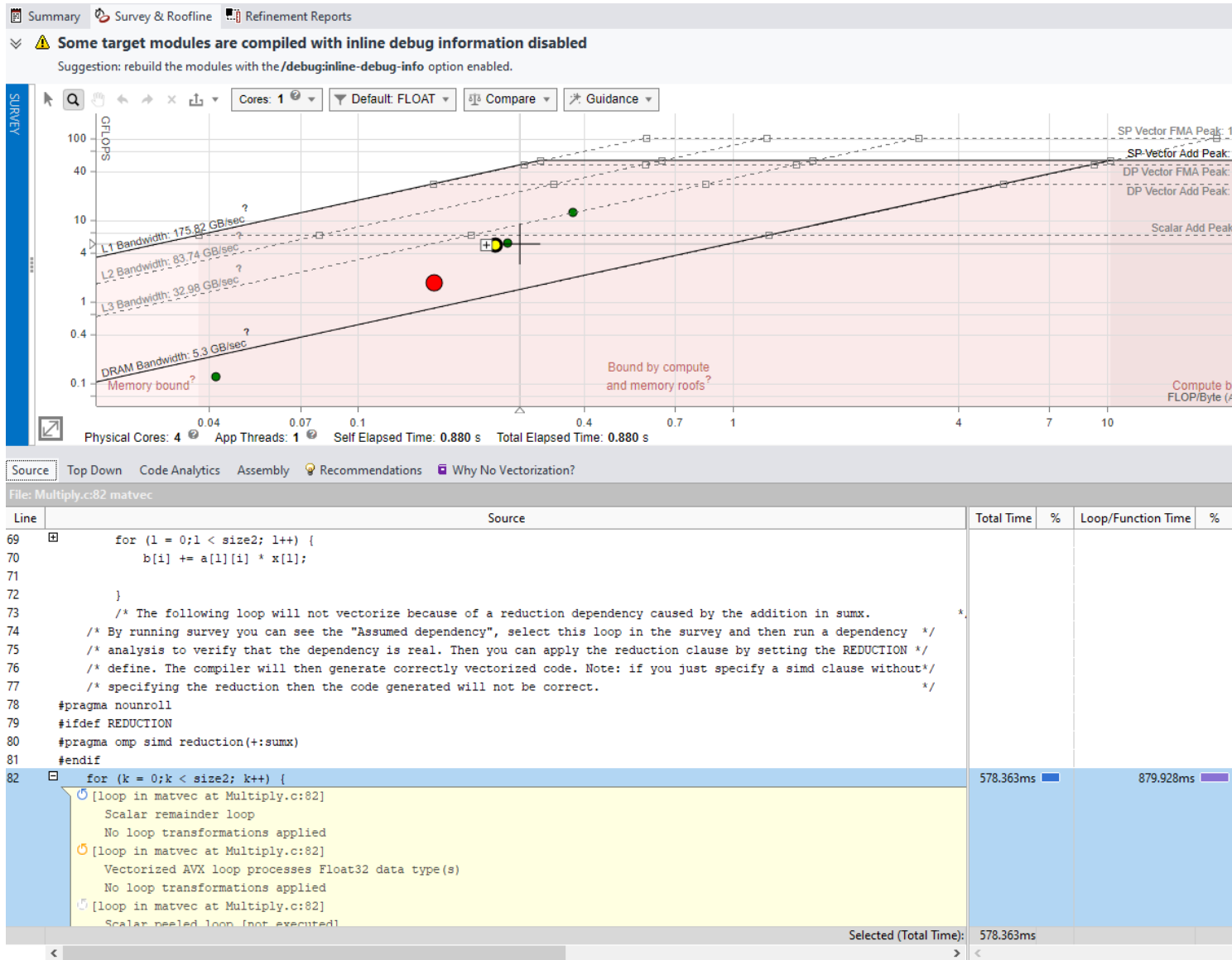
NOTE If the report does not open, click **Show Result** on the Welcome pane.

If you run the CPU / Memory Roofline Insights perspective from GUI, the result is opened automatically after the collection finishes.

You will see the CPU Roofline report that includes:

- Roofline chart that plots an application's achieved performance and arithmetic intensity against the CPU maximum achievable performance
- Additional information about your application in the **Advanced View** pane under the chart, including source code, detailed code analytics for trip counts and FLOP/INTOP data, optimization recommendations, and compiler diagnostics

Select a dot on the Roofline chart to see details for the selected loop in all tabs of the **Advanced View** pane



View an Interactive HTML Report

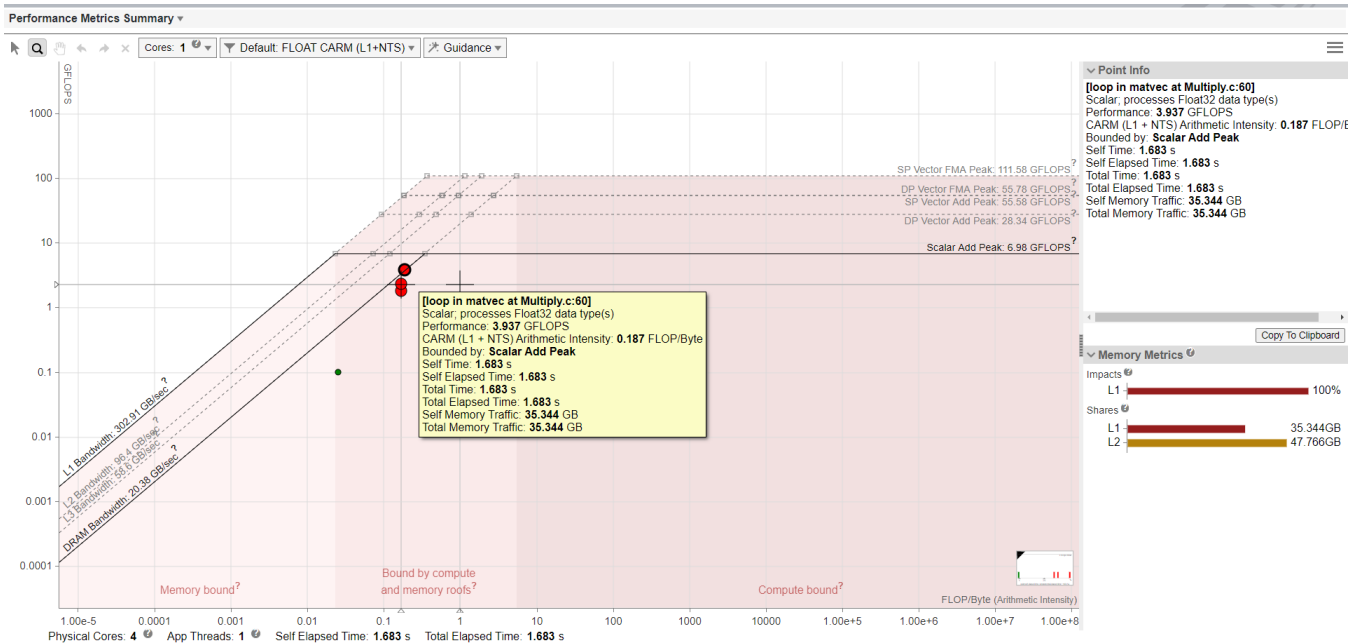
Intel Advisor enables you to export an interactive HTML report for the CPU Roofline chart, which you can open in your preferred browser and share.

When you open the report, you see the CPU Roofline chart with the selected configuration. In this report, you can:

- Expand the **Performance Metrics Summary** drop-down to view the summary performance characteristics for your application.
- Double-click a dot on the chart to see a roof ruler that point to exact roofs that bound the dot.
- Hover over a dot to see a detailed tooltip with performance metrics.

If you have a Memory-level Roofline report, you can also:

- Select memory levels to show dots for from the filter drop-down list on the chart.
- Double-click a dot on the chart to expand it for other memory levels and see roof rulers.




For details on exporting HTML reports, see [Work with Standalone HTML Reports](#).

Save a Read-only Snapshot

A snapshot is a read-only copy of a project result, which you can view at any time using the Intel Advisor GUI. You can save a snapshot for a project using Intel Advisor GUI or CLI.



To save an active project result as a read-only snapshot from GUI: Click the  button in the top ribbon of the report. In the **Create a Result Snapshot** dialog box, enter the snapshot details and save it.

To save an active project result as a read-only snapshot from CLI:

```
advisor --snapshot --project-dir=<project-dir> [--cache-sources] [--cache-binaries] --<snapshot-path>
```

where:

- `--cache-sources` is an option to add application source code to the snapshot.
- `--cache-binaries` is an option to add application binaries to the snapshot.
- `<snapshot-path>` is a path and a name for the snapshot. For example, if you specify `/tmp/new_snapshot`, a snapshot is saved in a `tmp` directory as `new_snapshot.advixeexpz`. You can skip this and save the snapshot to a current directory as `snapshotXXX.advixeexpz`.

To open the result snapshot in the Intel Advisor GUI, you can run the following command:

```
advisor-gui <snapshot-path>
```

You can visually compare the saved snapshot against the current active result or other snapshot results.

See [Create a Read-only Result Snapshot](#) for details.

Result Interpretation

When you run the CPU / Memory Roofline Insights perspective from GUI, depending on a configuration chosen, the chart shows a different level of details:

- [Examine Bottlenecks on CPU Roofline Chart](#)
- [Examine Relationships Between Memory Levels](#)

For a general overview of the report, see [CPU Roofline Report Overview](#).

See Also

[Run CPU / Memory Roofline Insights Perspective from GUI](#)

[Run CPU / Memory Roofline Insights Perspective from Command Line](#)

[Compare CPU Roofline Results](#) Use the *Roofline Compare* functionality to display Roofline chart data from other Intel® Advisor results or non-archived snapshots for comparison purposes to track optimization progress.

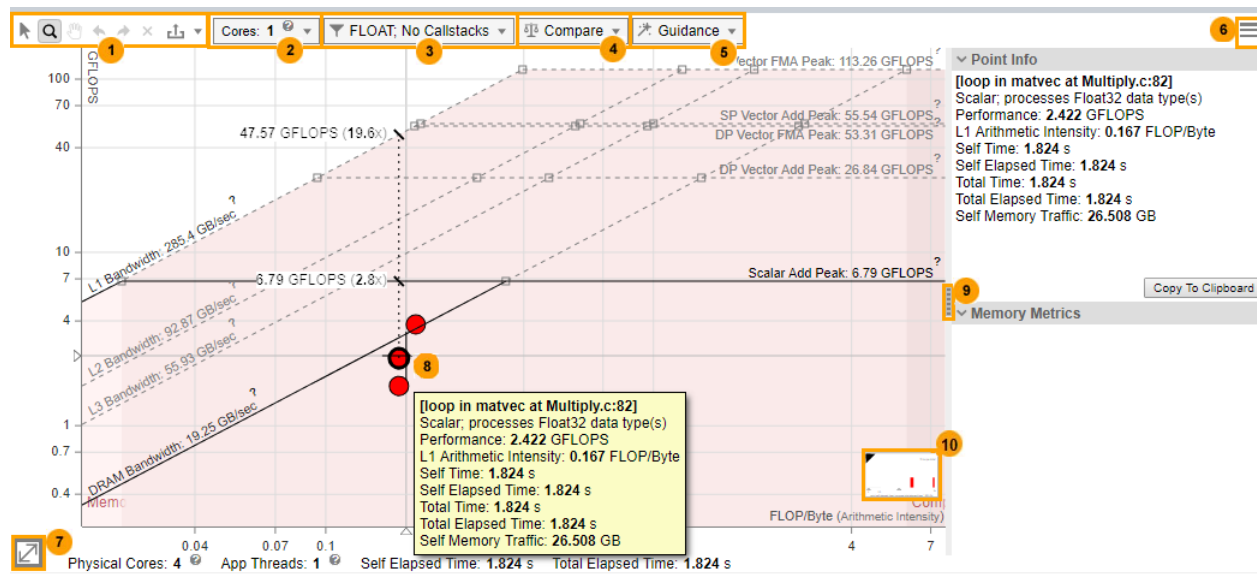
[CPU Metrics](#) This reference section describes the contents of data columns in **Survey** and **Refinement Reports** of the Vectorization and Code Insights, CPU / Memory Roofline Insights, and Threading perspectives.

CPU Roofline Report Overview

Review the controls available in the main report of the CPU / Memory Roofline Insights perspective of the Intel® Advisor.

Basic Roofline Chart (Low Accuracy)

There are several controls to help you focus on the **Roofline** chart data most important to you, including the following.



- 1
 - **Select Loops by Mouse Rect:** Select one or more loops/functions by tracing a rectangle with your mouse.
 - **Zoom by Mouse Rect:** Zoom in and out by tracing a rectangle with your mouse. You can also zoom in and out using your mouse wheel.
 - **Move View By Mouse:** Move the chart left, right, up, and down.
 - **Undo or Redo:** Undo or redo the previous zoom action.
 - **Cancel Zoom:** Reset to the default zoom level.
 - **Export as x:** Export the chart as a dynamic and interactive HTML or SVG file that does not require the Intel Advisor viewer for display. Use the arrow to toggle between the options.
- 2 Use the **Cores** drop-down toolbar to:
 - Adjust rooflines to see practical performance limits for your code on the host system.

- Build roofs for single-threaded applications (or for multi-threaded applications configured to run single threaded, such as one thread-per-rank for MPI applications. (You can use Intel Advisor filters to control the loops displayed in the **Roofline** chart; however, the **Roofline** chart does not support the **Threads** filter.)

Choose the appropriate number of CPU cores to scale roof values up or down:

- 1 – if your code is single-threaded
- Number of cores equal or close to the number of threads – if your code has fewer threads than available CPU cores
- Maximum number of cores – if your code has more threads than available CPU cores

By default, the number of cores is set to the number of threads used by the application (even values only).

You'll see the following options if your code is running on a multisocket PC:

- Choose **Bind cores to 1 socket** (default) if your application binds memory to one socket. For example, choose this option for MPI applications structured as one rank per socket.

NOTE This option may be disabled if you choose a number of CPU cores exceeding the maximum number of cores available on one socket.

- Choose **Spread cores between all n sockets** if your application binds memory to all sockets. For example, choose this option for non-MPI applications.

- 3**
- Toggle the display between floating-point (FLOP), integer (INT) operations, and mixed operations (floating-point and integer).
 - *If you collected Roofline with Calltacks:* Enable the display of Roofline with Callstacks additions to the **Roofline** chart.

- 4**
- Display **Roofline** chart data from other Intel Advisor results or non-archived snapshots for comparison purposes.

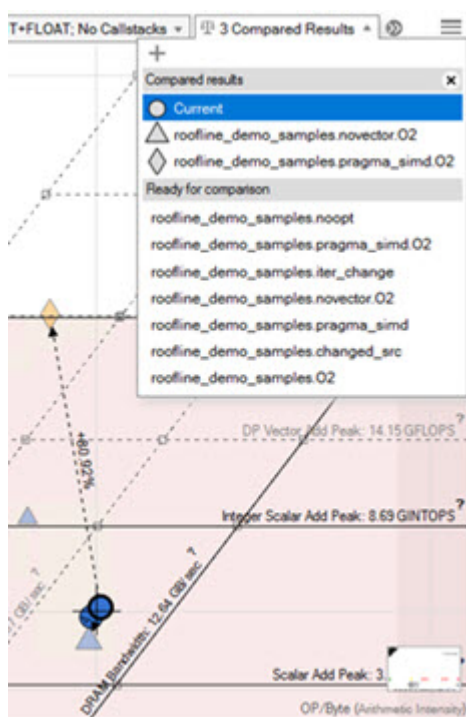
Use the drop-down toolbar to:

- Load a result/snapshot and display the corresponding filename in the **Compared Results** region.
- Clear a selected result/snapshot and move the corresponding filename to the **Ready for comparison** region.

- **Note:** Click a filename in the **Ready for comparison** region to reload the result/snapshot.
- Save the comparison itself to a file.

NOTE The arrowed lines showing the relationship among loops/functions do not reappear if you upload the comparison file.

Click a loop/function dot in the current result to show the relationship (arrowed lines) between it and the corresponding loop/function dots in loaded results/snapshots.



- 5 Add visual indicators to the Roofline chart to make the interpretation of data easier, including performance limits and whether loops/functions are memory bound, compute bound, or both.

Use the drop-down toolbar to:

- Show a vertical line from a loop/function to the nearest and topmost performance ceilings by enabling the **Display roof rulers** checkbox. To view the ruler, hover the cursor over a loop/function. Where the line intersects with each roof, labels display hardware performance limits for the loop/function.
- *If you collected Roofline for All Memory Levels:* Visually emphasize the relationships among displayed memory levels and roofs and for a selected loop/function dot by enabling the **Show memory level relationships** checkbox.
- Color the roofline zones to make it easier to see if enclosed loops/functions are fundamentally memory bound, compute bound, or bound by compute and memory roofs by enabling the **Show Roofline boundaries** checkbox.

The preview picture is updated as you select guidance options, allowing you to see how changes will affect the Roofline chart's appearance. Click **Apply** to apply your changes, or **Default** to return the Roofline chart to its original appearance.

Once you have a loop/function's dots highlighted, you can zoom and fit the Roofline chart to the dots for the selected loop/function by once again double-clicking the loop/function or pressing **SPACE** or **ENTER** with the loop/function selected. Repeat this action to return to the original Roofline chart view.

To hide the labeled dots, select another loop/function, or double-click an empty space in the Roofline chart.

- 6
- **Roofline View Settings:** Adjust the default scale setting to show:
 - The optimal scale for each **Roofline** chart view
 - A scale that accommodates all **Roofline** chart views
 - **Roofs Settings:** Change the visibility and appearance of roofline representations (lines):

- Enable calculating roof values based on single-threaded benchmark results instead of multi-threaded.
- Click a **Visible** checkbox to show/hide a roofline.
- Click a **Selected** checkbox to change roofline appearance: display a roofline as a solid or a dashed line.
- Manually fine-tune roof values in the **Value** column to set hardware limits specific to your code.
- **Loop Weight Representation:** Change the appearance of loop/function weight representations (dots):
 - **Point Weight Calculation:** Change the **Base Value** for a loop/function weight calculation.
 - **Point Weight Ranges:** Change the **Size**, **Color**, and weight **Range (R)** of a loop/function dot. Click the + button to split a loop weight range in two. Click the - button to merge a loop weight range with the range below.
 - **Point Colorization:** color loop/function dots by weight ranges or by type (vectorized or scalar). You can also change the color of loop with no self time.

You can save your Roofs Settings or Point Weight Representation configuration to a JSON file or load a custom configuration.

7 Zoom in and out using numerical values.

8 Click a loop/function dot to:

- Outline it in black.
- Display metrics for it.
- Display corresponding data in other window tabs.

Right-click a loop/function dot or a blank area in the **Roofline** chart to perform more functions, such as:

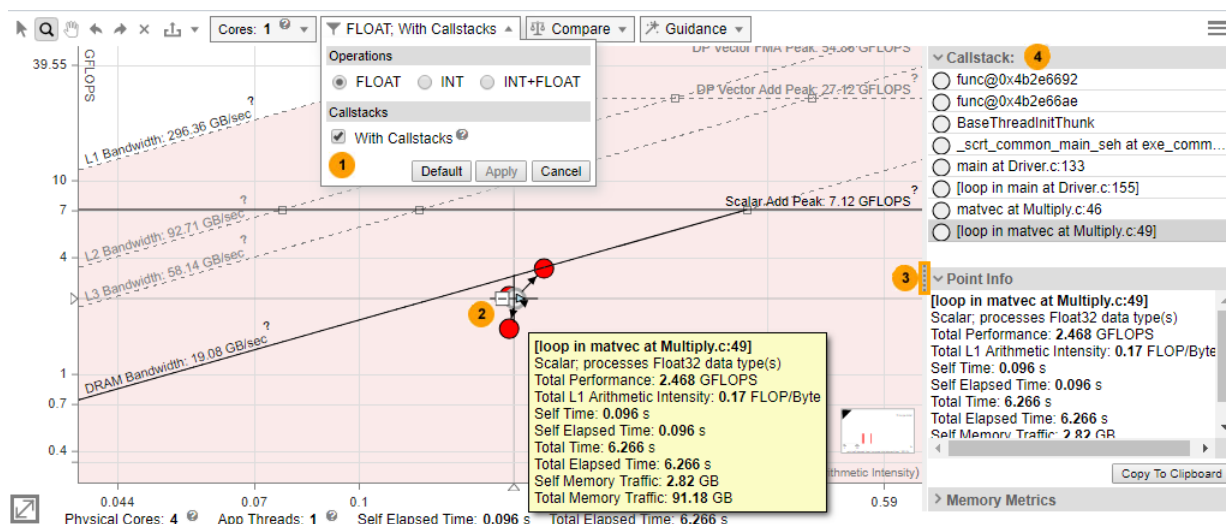
- Further simplify the **Roofline** chart by filtering out (temporarily hiding a dot), filtering in (temporarily hiding all other dots), and clearing filters (showing all originally displayed dots).
- Copy data to the clipboard.

9 Show/hide the metrics pane:

- Review the basic performance metrics in the **Point Info** pane.
- *If you collected the Roofline for All Memory Levels:* Review how efficiently the loop/function uses cache and what memory level bounds the loop/function in the **Memory Metrics** pane.

10 Display the number and percentage of loops in each loop weight representation category.

Roofline with Callstacks Chart (Medium Accuracy)

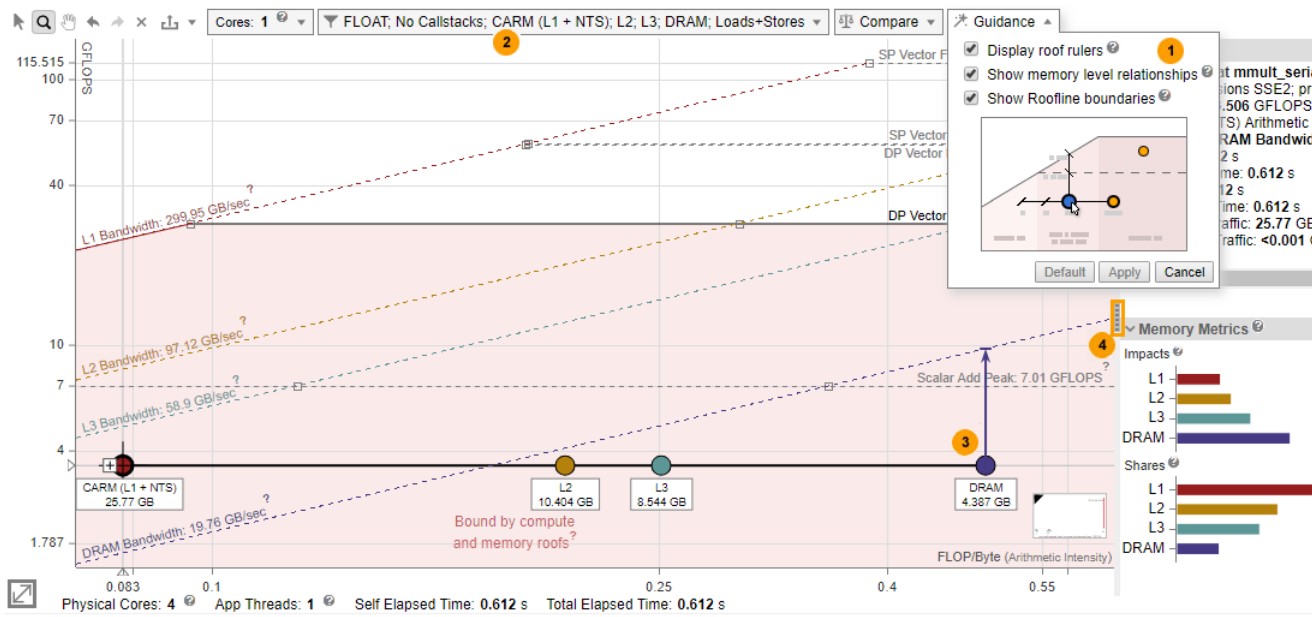


- 1 Enable the display of Roofline with Callstacks additions to the Roofline chart.
- 2 Show/hide loop/function descendants:
 - Click a loop/function dot control to collapse descendant dots into the parent dot.
 - Click a loop/function dot control to show descendant dots and their relationship via visual indicators to the parent dot.

You can also right-click a loop/function dot to open the context menu and expand/collapse the loop/function subtree.
- 3 Show/hide the **Callstack** and other panes.
- 4
 - Click an item in the **Callstack** pane to flash the corresponding loop/function dot in the Roofline chart.
 - Right-click an item in the **Callstack** pane to open the context menu and expand/collapse the item subtree.

You can also click an item in the **Callstack** pane to flash the corresponding loop/function dot in the **Roofline** chart.

Memory-Level Roofline Chart (Medium Accuracy)



- 1 Visually emphasize the relationships among displayed memory levels and roofs for a selected loop/function dot by enabling the **Show memory level relationships** checkbox.

NOTE This checkbox is enabled by default.

- 2 Use the drop-down toolbar to:
 - Select the **Memory Level(s)** to show for each loop/function in the chart (L1, L2, L3, DRAM).
 - Select which **Memory Operation Types(s)** to display data for in the Roofline chart: **Loads**, **Stores**, or **Loads and Stores**.
- 3 Double-click a dot or select a dot and press **SPACE** or **ENTER** to examine how the relationships between displayed memory levels and roofs:
 - Labeled dots are displayed, representing memory levels for the selected loop/function. Lines connect the dots to indicate that they correspond to the selected loop/function.

NOTE If you have chosen to display only some memory levels in the chart using the **Memory Level** option, unselected memory levels are displayed with X marks.

- An arrowed line is displayed, pointing to the memory level roofline that bounds the selected loop. If the arrowed line cannot be displayed, a message will pop up with instructions on how to fix it.
- 4 Show/hide the **Memory Metrics** and other panes.

In the **Memory Metrics** pane:

 - Review the time spent processing requests for each memory level reported in the **Impacts** histogram. A big value indicates a memory level that bounds the selected loop.
 - Review an amount of data that passes through each memory level reported in the **Shares** histogram.

Examine Bottlenecks on CPU Roofline Chart

Accuracy Level

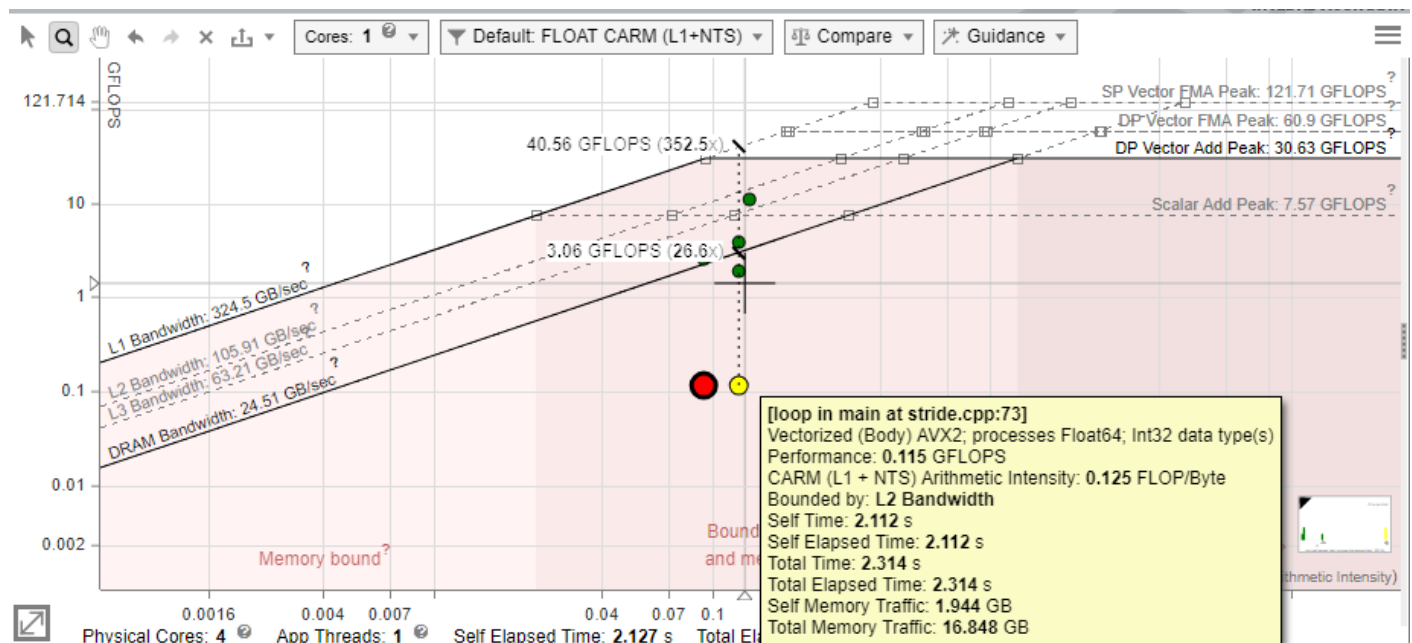
Low

Enabled Analyses

Survey + FLOP (Characterization)

Result Interpretation

The farther a dot is from the topmost roofs, the more room for improvement there is. In accordance with Amdahl's Law, optimizing the loops that take the largest portion of the program's total run time will lead to greater speedups than optimizing the loops that take a smaller portion of the run time.



NOTE This topic describes data as it is shown in the CPU Roofline report in the Intel Advisor GUI. You can also view the result in an [HTML report](#), but data arrangement and panes may vary.

- By dot size and color, identify loops that take most of total program time and/or located very low in the chart. For example:
 - Small, green dots take up relatively little time, so are likely not worth optimizing.
 - Large, red dots take up the most time, so the best candidates for optimization are the large, red dots with a large amount of space between them and the topmost roofs.

NOTE You can switch between coloring the dots by execution time and coloring the dots by type (scalar or vectorized) in the roof view menu on the right.

- Depending on the dots position, identify what the loops are bounded by. Intel® Advisor marks the roofline zones on the chart to help you identify what roofs bound the loop:
 - Loop is bounded by memory roofs.

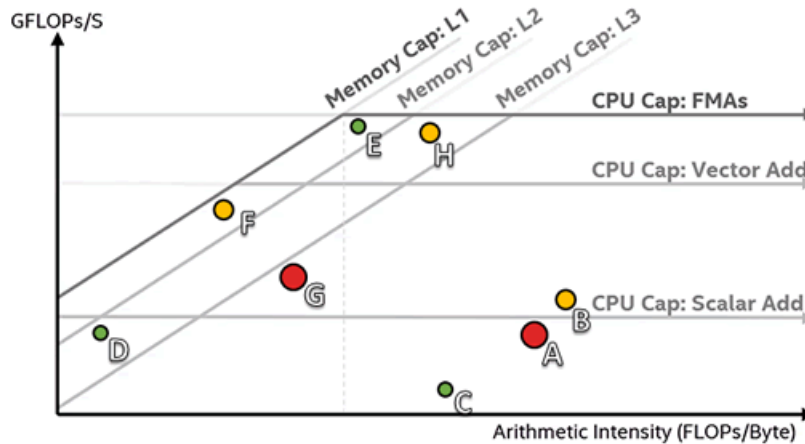
- Loop is bounded by compute roofs.
- Loop is bounded by both memory and compute roofs.
- In the **Recommendations** tab, scroll down to the **Roofline Guidance** section that provides you hints on next optimization steps for a selected loop/function.

The roofs above a dot represent the restrictions preventing it from achieving a higher performance, although the roofs below can contribute somewhat. Each roof represents the maximum performance achievable without taking advantage of a particular optimization, which is associated with the next roof up. Depending on a dot position, you can try the following optimizations.

NOTE For more precise optimization recommendations, see the Roofline Guidance in **Code Analytics** and Roofline Conclusions in **Recommendations** tabs.

Dot Position	Reason	To Optimize
Below a memory roof (DRAM Bandwidth, L1 Bandwidth, so on)	The loop/function uses memory inefficiently.	Run a Memory Access Patterns analysis for this loop. <ul style="list-style-type: none"> • If MAP analysis suggests cache optimization, make any appropriate optimizations. • If cache optimization is impossible, try reworking the algorithm to have a higher AI.
Below Vector Add Peak	The loop/function under-utilizes available instruction sets.	Check Traits column in the Survey report to see if FMAs are used. <ul style="list-style-type: none"> • If FMA is not used, try altering your code or compiler flags to induce FMA usage.
Just above Scalar Add Peak	The loop/function is undervectorized.	Check vectorization efficiency and performance issues in the Survey. Follow the recommendations to improve it if it's low.
Below Scalar Add Peak	The loop/function is scalar.	Check the Survey report to see if the loop vectorized. If not, try to get it to vectorize if possible. This may involve running Dependencies to see if it's safe to force it.

In the following **Roofline** chart representation, loops A and G (large red dots), and to a lesser extent B (yellow dot far below the roofs), are the best candidates for optimization. Loops C, D, and E (small green dots) and H (yellow dot) are poor candidates because they do not have much room to improve or are too small to have significant impact on performance.



Some algorithms are incapable of breaking certain roofs. For instance, if Loop A in the example above cannot be vectorized due to dependencies, it cannot break the Scalar Add Peak.

Tip If you cannot break a memory roof, try to rework your algorithm for higher arithmetic intensity. This will move you to the right and give you more room to increase performance before hitting the memory bandwidth roof. This would be the appropriate approach to optimizing loop F in the example, as well as loop G if its cache usage cannot be improved.

Analyze Specific Loops

Select a dot on the chart, open the **Code Analytics** tab to view detailed information about the selected loop:

- Refer to **Loop Information** pane to examine total time, self time, instruction sets used, and instruction mix for the selected loop. Intel Advisor provides:
 - Static instruction mix data that is based on static assembly code analysis within a call stack. Use static instruction mix to examine instruction sets in the inner-most functions/loops.
 - Dynamic instruction mix that is based on dynamic assembly code analysis. This metric represents the total count of instructions executed by your function/loop. Use dynamic instruction mix to examine instruction sets in the outer loops and in complex loop-nests.

Intel Advisor automatically determines the data type used in operations. View the classes of instructions grouped by categories in instruction mix:

Category	Instruction Types
Compute (FLOP and INTOP)	ADD, MUL, SUB, DIV, SAD, MIN, AVG, MAX, ABS, SIN, SQRT, FMA, RCCP, SCALE, FCOM, V4FMA, V4VNNI
Memory	<ul style="list-style-type: none"> scalar and vector MOV instructions GATHER/SCATTER instructions VBMI2 compress/expand instructions
Mixed	Compute instructions with memory operands
Other	MOVE, CONTROL FLOW, SYNC, OTHER

NOTE Intel Advisor counts FMA and VNNI instructions as more than 1 operation depending on the size of the data type and/or the type of vector registers used.

- Refer to **Roofline** pane for more details about a specific roof that bounds the loop:

- View roofs with number of threads, data types, and instructions mix used in the loop
- Identify what exactly bounds the selected loop - memory, compute, or both memory and compute
- Determine the exact roof that bounds the loop and estimates a potential speedup for the loop in the callout if you optimize it for this roof
- Refer to **Statistics for operations** pane to view the count of operations collected during Characterization analysis. Depending on the operations you need, use a drop-down list to choose FLOP, INTOP, FLOP +INTOP or All Operations. Switch between Self and Total data using the toggle in the top right-hand corner of the pane.

Intel Advisor calculates **floating-point operations (FLOP)** as a sum of the following classes of instructions multiplied by their iteration count: FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND

Integer operations (INTOP) are calculated in two modes:

- **Potential INT operations (default)** that include loop counter operations that are not strictly calculations (for example, INC/DEC, shift, rotate operations). In this case, INTOP is a sum of the following instructions multiplied by their iteration count: ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates
- **Strict INT operations (available in Python* API only)** that include only calculation operations. In this case, INTOP is a sum of the following instructions multiplied by their iteration count: ADD, MUL, IDIV, SUB

Next Steps

- [Identify Bottlenecks Iteratively: Cache-Aware Roofline](#)

Examine Relationships Between Memory Levels

Accuracy Level

Medium

Enabled Analyses

Survey + Characterization (Trip Counts and FLOP, Call Stacks, Memory-Level) + Memory Access Patterns

Result Interpretation

In the **Medium** accuracy preset, the Intel® Advisor extends the basic Roofline capability and [collects metrics for all memory levels](#) and the [callstack data](#), which allows you to analyze your application in more detail. Roofline chart uses the results of Memory Access Patterns analysis to understand what bounds the loop and build recommendations in Roofline Guidance.

For information about Memory Access Patterns data interpretation, refer to [Investigate Memory Usage and Traffic](#).

NOTE This topic describes data as it is shown in the CPU Roofline report in the Intel Advisor GUI. You can also view the result in an [HTML report](#), but data arrangement and panes may vary.

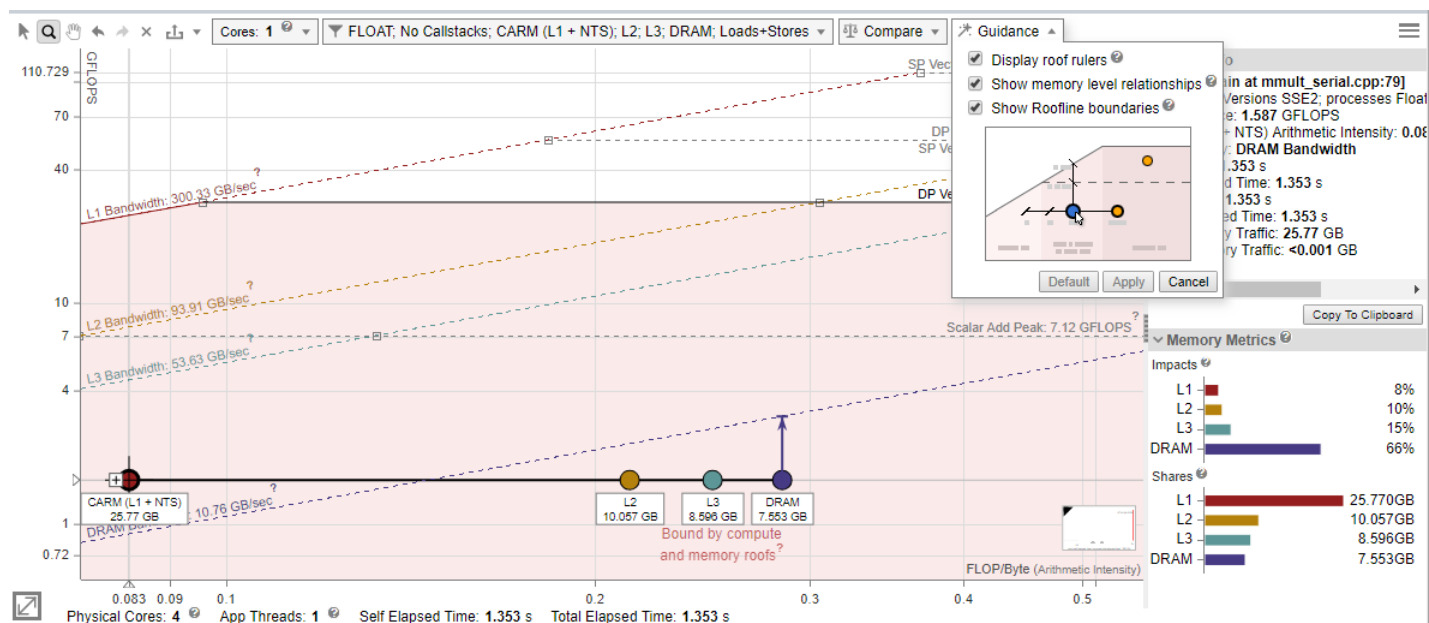
Memory-Level Roofline

The Memory-Level Roofline allows you to examine each loop at different cache levels and arithmetic intensities and provides precise insights into which cache level causes the performance bottlenecks.

The Memory-Level Roofline can help you to:

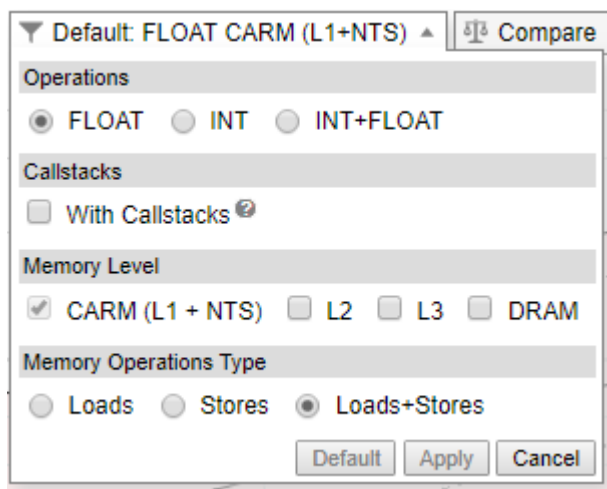
- Determine which loops are limited by cache

- Find inefficient access patterns
- Locate loops that can benefit from vectorization or threading optimizations

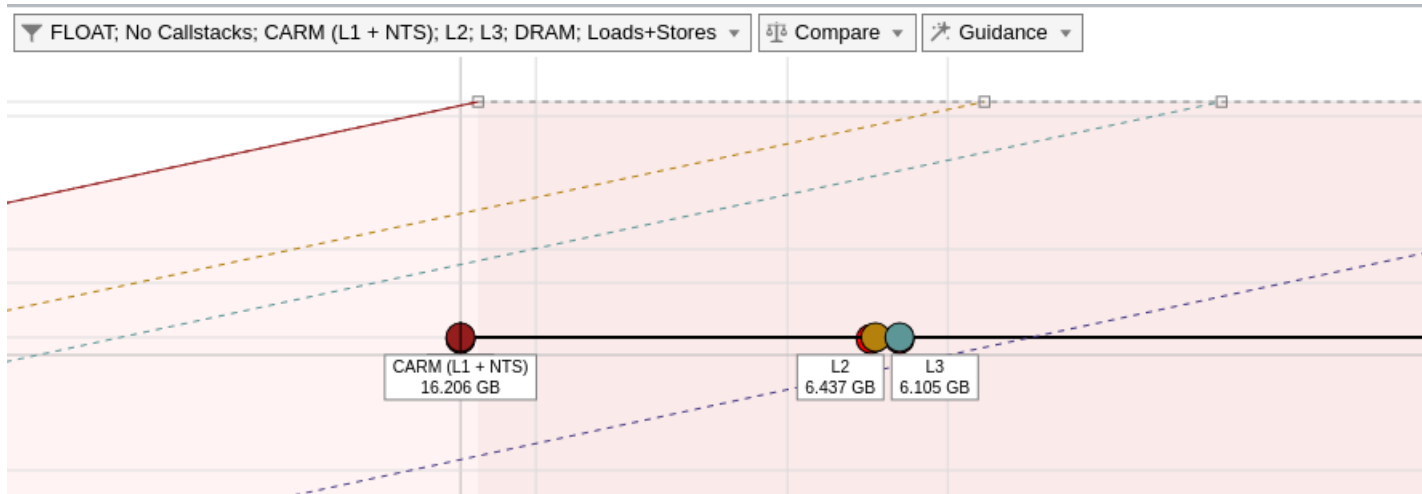


To configure the Memory-Level Roofline chart:

1. Expand the filter pane in the Roofline chart toolbar.
2. In the Memory Level section, select the memory levels you want to see metrics for.



3. Click **Apply**.
4. In the Roofline chart, double-click a loop to examine how the relationships between displayed memory levels and roofs. Labeled dots are displayed, representing memory levels with arithmetic intensity for the selected loop/function; lines connect the dots to indicate that they correspond to the selected loop/function.



Tip By default, the Memory-Level Roofline chart is generated for the system cache configuration. You can also generate the chart for a custom cache configuration:

1. Go to **Project Properties > Trip Count and FLOP**.
2. In the Cache simulator field, click **Modify**.
3. Click **Add** and enter/select the desired cache configurations.
4. Re-run the Roofline with the **Medium** accuracy.

Memory-Level Roofline Data

Intel® Advisor collects integrated traffic data for all traffic types between a CPU and different memory subsystem using cache simulation. With this data, Intel® Advisor counts the number of data transfers for a given cache level and computes AI for each loop and each memory level.

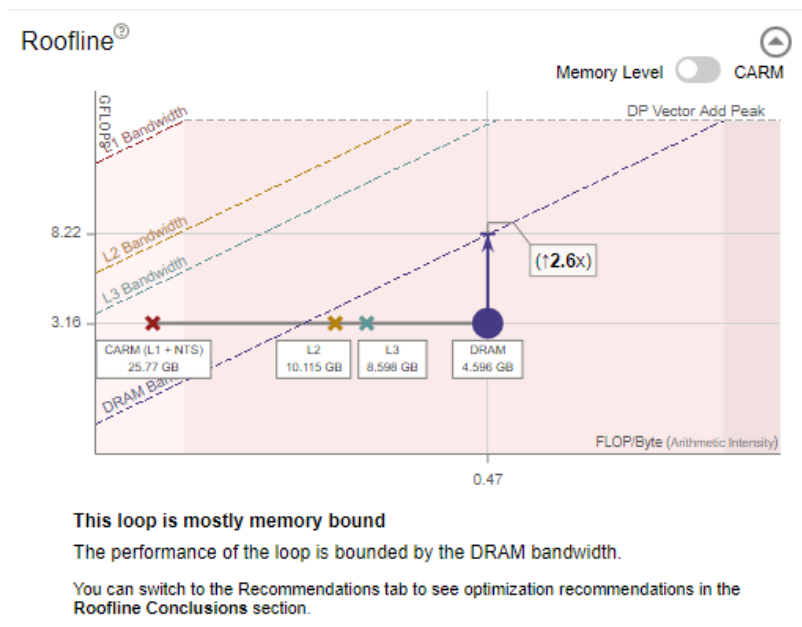
Review the changes in the traffic from one memory level to another and compare it to respective to identify the memory hierarchy bottleneck for the kernel and determine optimization steps based on this information.

- The *vertical distance* between memory dots and their respective roofline shows how much you are limited by a given memory subsystem. If a dot is close to its roof line, it means that the kernel is limited by the performance of this memory level.
- The horizontal distance between memory dots indicates how efficiently the loop/function uses cache. For example, if L3 and DRAM dots are very close on the horizontal axis for a single loop, the loop/function uses L3 and DRAM similarly. This mean that it does not use L3 and DRAM efficiently. You can try to improve re-usage of data in the code to change arithmetic intensity for all loops/functions and improve application performance. For more precise advice, see the **Roofline Guidance** in the **Code Analytics** tab.
- *Arithmetic intensity* determines the order in which dots are plotted, which can provide some insight into your code's performance. For example, the L1 dot should be the largest and first plotted dot on the chart from left to right. However, memory access type, latency, or technical issues can change the order of the dots. Continue to run the [Memory Access Pattern](#) analysis to investigate this issue.

To examine a *specific* loop in more details, select a dot on the chart and open the **Code Analytics** tab below the chart:

- Review the amount of data transferred for the selected loop/function and a specific roof that bounds the loop in the **Roofline** pane. Use this pane to analyze deeper a selected loop/function:
 - It shows only roofs with number of threads, data types, and instructions mix used in the loop.
 - It identifies what exactly bounds the selected loop - memory, compute, or both memory and compute.

- It determines exact roof that bounds the loop and estimates a potential speedup for the loop in the callout if you optimize it for this roof.



- Review the memory metrics for different memory levels (L1, L2, L3 and DRAM) and the number of operations transferred (FLOP and INTOP) in the **Data transfers and Bandwidth** table. This indicates the amount of *self* data (excluding data from inner loops/functions) or *total* data (including data from inner loops/functions) transferred, memory level bandwidth, and percentage of memory used at each memory level.

NOTE Total data transfers are available only if you collect Roofline with Callstacks.

- Review the amount of data processed at different memory levels for the selected loop in the **Memory Metrics** pane. The pane shows two histograms:
 - Review the time spent processing requests for each memory level reported in the **Impacts** histogram. A big value indicates a memory level that bounds the selected loop. Examine the difference between the two largest bars to see how much throughput you can gain if you reduce the impact on your main bottleneck. It also gives you a long-time plan to reduce your memory bound limitations as once you will solve the problems coming from the widest bar, your next issue will come from the second biggest bar and so on. Ideally, a developer would like to see the L1 as the most impactful memory in the application for each loop.
 - Review an amount of data that passes through each memory level reported in the **Shares** histogram.

NOTE Metrics in the **Memory Metrics** pane calculated for a dominant operation type in the selected loop (FLOAT or INT) and based on the total data aggregating all callstacks. Hover over the ? icon for the whole pane to see the tooltip that indicates the dominant type.

Roofline with Callstacks

Intel® Advisor basic Roofline model, the Cache-Aware Roofline Model (CARM), offers *self data* capability. Intel® Advisor Roofline with Callstacks feature extends the basic model with *total data* capability:

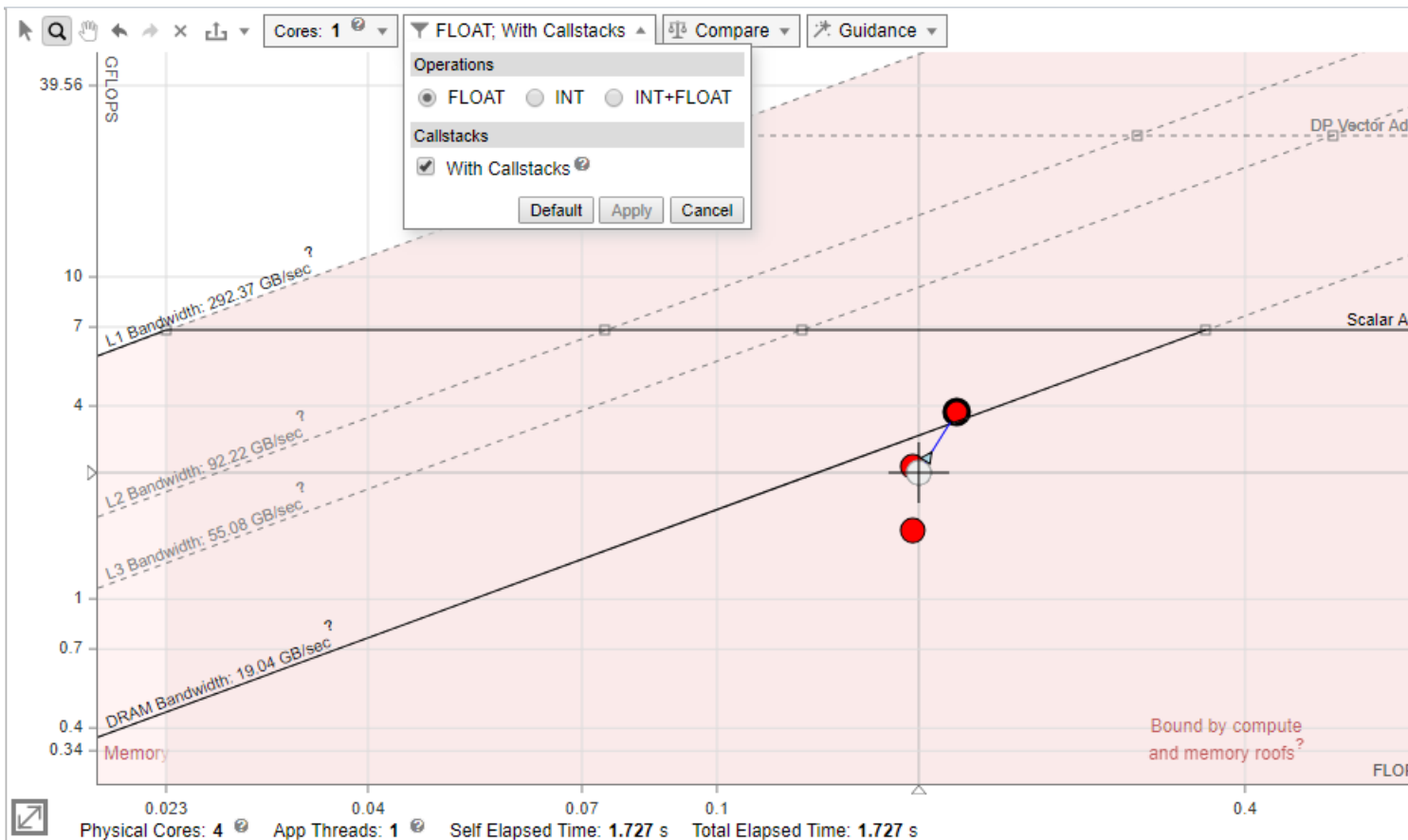
- Self data = Memory access, FLOPs, and duration related only to the loop/function itself and excludes data originating in other loops/functions called by it

- Total data = Data from the loop/function itself and its inner loops/functions

The total-data capability in the Roofline with Callstacks feature can help you:

- Investigate the source of loops/functions instead of just the loops/functions themselves.
- Get a more accurate view of loops/functions that behave differently when called under different circumstances.
- Uncover design inefficiencies higher up the call chain that could be the root cause of poor performance by smaller loops/functions.

To view the callstacks, enable the **With Callstacks** checkbox in the **Roofline** chart.



To show/hide dot descendants:

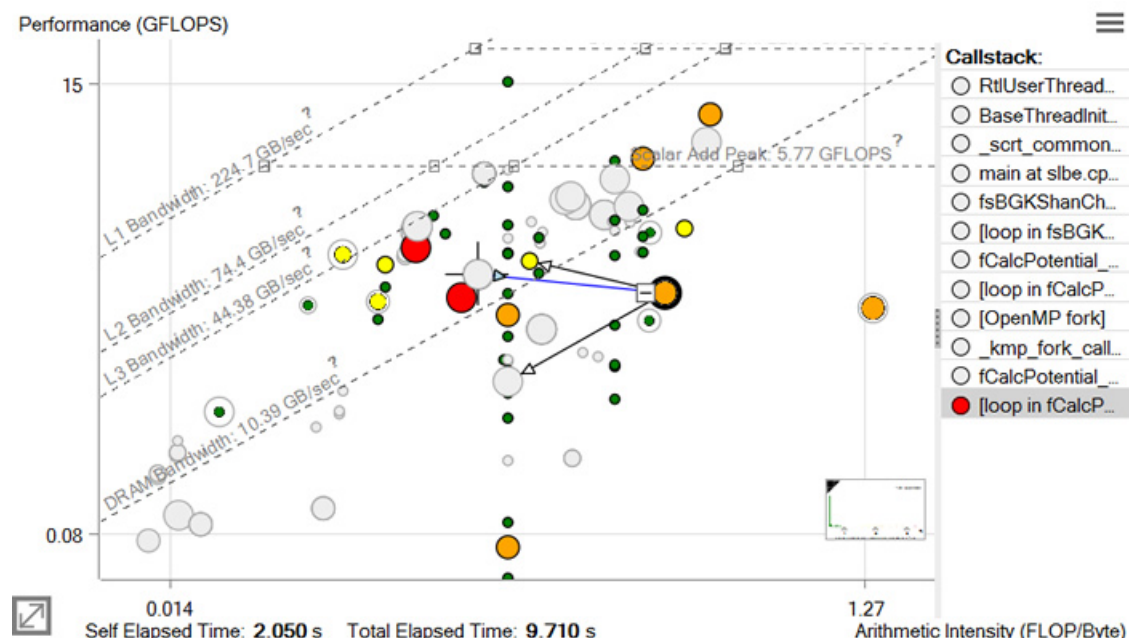
- Click a loop/function dot control to collapse descendant dots into the parent dot.
- Click a loop/function dot control to show descendant dots and their relationship with visual indicators to the parent dot.

Roofline with Callstacks Chart Data

The following **Roofline** chart representation shows some of the added benefits of the Roofline with Callstacks feature, including:

- A navigable, color-coded **Callstack** pane that shows the entire call chain for the selected loop/function, but excludes its callees
- Visual indicators (caller and callee arrows) that show the relationship among loops and functions
- The ability to simplify dot-heavy charts by collapsing several small loops into one overall representation

Loops/functions with no self data are grayed out when expanded and in color when collapsed. Loops/functions with self data display at the coordinates, size, and color appropriate to the data when expanded, but have a gray halo of the size associated with their total time. When such loops/functions are collapsed, they change to the size and color appropriate to their total time and, if applicable, move to reflect the total performance and total arithmetic intensity.



See Also

[Examine Bottlenecks on CPU Roofline Chart](#)

[Compare CPU Roofline Results](#) Use the *Roofline Compare* functionality to display Roofline chart data from other Intel® Advisor results or non-archived snapshots for comparison purposes to track optimization progress.

Compare CPU Roofline Results

Use the *Roofline Compare* functionality to display Roofline chart data from other Intel® Advisor results or non-archived snapshots for comparison purposes to track optimization progress.

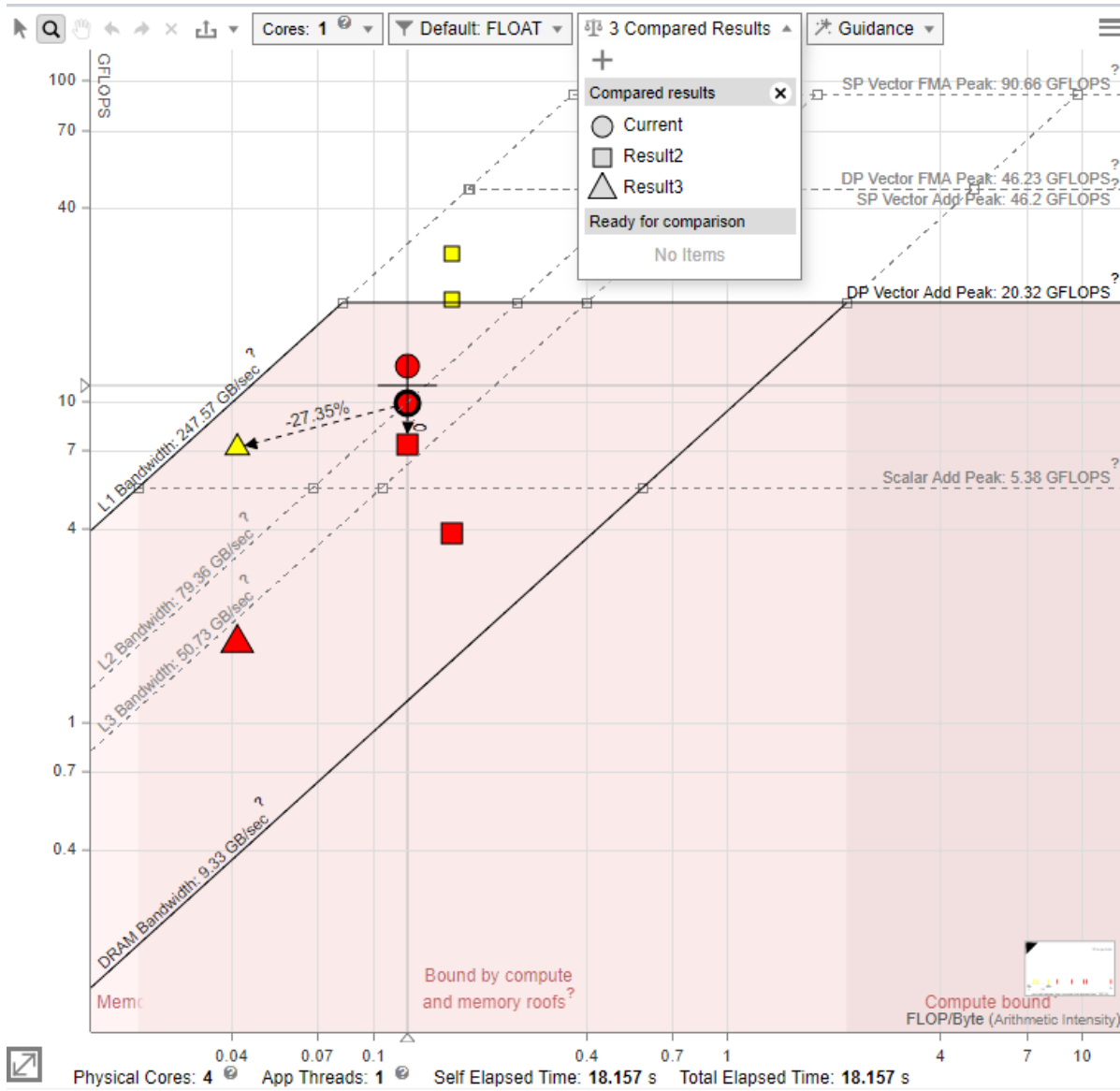
Prerequisites

To compare the GPU Roofline results, make sure to get the following:

- A baseline GPU Roofline *result* or *snapshot*
- One or more GPU Roofline results or snapshots of the *same* application with an optimization applied

To compare the results:

1. Open a baseline GPU Roofline result/snapshot.
2. From the **Compare** drop-down toolbar, click **+** to load a comparison result/snapshot. You can load multiple results/snapshots for comparison one by one.

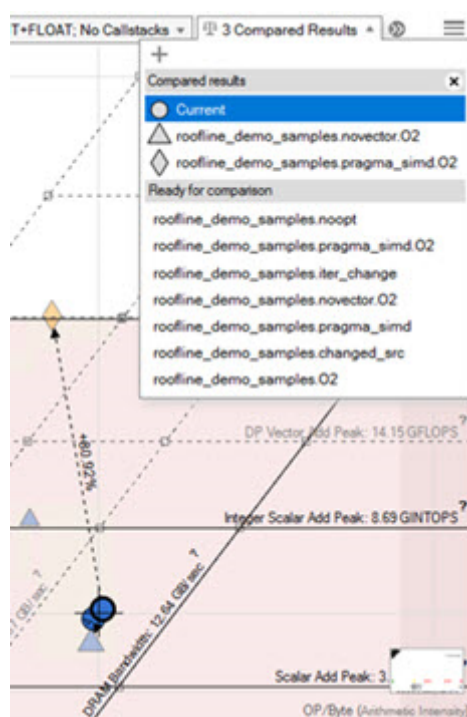


When the comparison is uploaded:

- The filenames for uploaded results/snapshots are displayed in the **Compared Results** region.
- Similar loops/functions from all compared results are recognized automatically. They are connected with a dashed arrow line. The performance improvement between the loops/functions is shown above the line, in per cent. The improvement is calculated as the *difference* in FLOPS, INTOPS, or OPS and Total Time.

NOTE The arrows showing the relationship among loops/functions do not reappear if you upload a new comparison file.

- Loops from different snapshots are shown as different icons on the chart. For example, on the picture below, the baseline loops are shown as circles and comparison loops are triangles and diamonds.



- To highlight all dots from a specific compared result, open the **Compare** drop-down and hover over the result name.
- Each time you change the Roofline configuration or filter the dots on the chart, the comparison is updated automatically.
- You can remove a selected result from **Compared Results** by hovering over it and clicking the **X** icon. The result is removed from the chart and appears in the **Ready for comparison** region. Click a name in the **Ready for comparison** region to reload the result back to the chart.
- You can save the comparison itself to a file using the export feature.

NOTE To find the same loops/functions among the results, Intel Advisor compares several loop/function features, such as their type, nesting level, source code file name and line, and function name. When a certain threshold of similar or equal features is reached, the two loops/functions are considered a match and connected with a dashed line. However, this method still has few limitations. Sometimes, there can be no match for the same loop/function if one is optimized, parallelized, or moved in the source code to four or more lines from the original place. Intel Advisor tries to ensure some balance between matching source code changes and false positives.

Model Threading Designs

Analyze, design, tune, and check threading design options without disrupting your normal development by running the Threading Perspective.

The Threading Perspective can help you to:

- Model different threading designs for your application
- Prototype project scaling on systems with larger core counts
- Find performance issues and fix them before implementing parallelism
- Find and eliminate data-sharing issues during design

How It Works

The Threading perspective includes the following steps:

1. Run the **Survey** analysis to find candidates for parallelizing.
2. Add parallel site and task annotations to your code and re-build your application.
3. Run **Suitability** analysis to view proposed parallel design options.
4. Run **Dependencies** analysis to identify stoppers for adding parallel code.

Threading Summary

Threading perspective reports information about your application performance recommends you loops/functions to parallelize with the highest gain:

- View the main performance metrics of your program with execution time details.
- View optimization recommendations that help you to improve the overall performance of your application and separate loops/functions.
- Examine how different parallel design options affect performance of annotated loops/functions and view estimated gain for each option. Check if annotated loops have dependencies that can be show-stoppers while parallelizing your code.

Summary
Survey & Roofline
Refinement Reports
Annotation Report
Suitability Report

Threading Perspective

Threading Perspective lets you analyze, design, tune, and check threading options without disrupting your development.

Program Metrics

Elapsed Time 5.39s Number of CPU Threads 1

Vector Instruction Set None

Performance Characteristics

Metrics	Total	
Total CPU time	5.00s	100%
Time in scalar code	5.00s	100%

> Vectorization Gain/Efficiency (Not Available)

Per Program Recommendations

Higher instruction set architecture (ISA) available
Consider recompiling your application using a higher ISA. [Show more](#)

Top Time-Consuming Loops

Consider adding parallel site and task annotations around these time-consuming loops found during Survey analysis.

Loop	Self Time	Total Time	Trip Counts
loop in setQueen at nqueens_serial.cpp:132	0.449s	4.996s	14
loop in solve at nqueens_serial.cpp:156	<0.001s	4.996s	14
loop in setQueen at nqueens_serial.cpp:103	1.934s	1.934s	4

Suitability And Dependencies Analysis Data

These annotated parallel sites were detected:

Site Location	Maximum Site Gain	Dependencies
loop in solve at nqueens_serial.cpp:154	6.4685634666754614	WAR:1 WAW:1

Recommendations

[align_loop_title](#) loop in [setQueen](#) at [nqueens_serial.cpp:103](#)

[align_loop_title](#) loop in [setQueen](#) at [nqueens_serial.cpp:132](#)

See Also

[Run Threading Perspective from GUI](#) Steps to run the Threading perspective.

[Run Threading Perspective from Command Line](#)

[Annotate Code for Deeper Analysis](#)

[Model Threading Parallelism](#)

Run Threading Perspective from GUI

Steps to run the Threading perspective.

In the **Analysis Workflow** pane, select the Threading perspective. The perspective can be executed at the following collection accuracy levels:










- **Low** - Find candidates for parallelizing.
- **Medium** - Model parallel design options and determine whether there are dependencies limiting parallelizing.
- **Custom** - Customize the perspective flow and properties.


In the Threading perspective, collection accuracy levels match the steps you should take. By default, accuracy is set to Low.

NOTE The higher accuracy value you choose, the higher runtime overhead is added to your application. The **Overhead** indicator shows the overhead for the selected configuration.

Prerequisites: In the graphical-user interface (GUI): [Create a project](#) and specify an analysis target and target options.

To configure and run the Threading perspective from GUI, do the following:

1. Select **Low** accuracy level to enable the Survey analysis and run the perspective by clicking  button.
You will get a Survey report that shows the execution times of your functions and loops.
2. Sort the report data by **Total Time** to identify functions and loops with the longest execution time. These loops/functions are the best candidates to apply parallelization for.
3. In your source code, annotate sites and tasks to model threading for and re-build your application. For more information on annotations and how to apply them, see [Annotate Code for Deeper Analysis](#) section.
4. Select **Medium** accuracy level and run the Threading perspective by clicking  button.
While the perspective is running, you can do the following in the **Analysis Workflow** tab:
 - Control the perspective execution:
 - Stop data collection and see the already collected data: Click the  button.
 - Pause data collection: Click the  button.
 - Cancel data collection and discard the collected data: Click the  button.
 - Expand an analysis with  to control the analysis execution:
 - Pause the analysis: Click the  button.
 - Stop the currently running analysis and start the next analysis selected: Click the  button.
 - Interrupt execution of all selected analyses and see the already collected data: Click the  button.

NOTE To [generate command lines](#) for selected perspective configuration, click the  **Command Line** button.

To run the Threading perspective with the Medium accuracy from the command line interface:

1. Run the Survey analysis:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

2. Collect trip count data:

```
advisor --collect=tripcounts --project-dir=./advi_results -- ./myApplication
```

3. Run the Suitability analysis for annotated loops:

```
advisor --collect=suitability --project-dir=./advi_results -- ./myApplication
```

4. Run the Dependencies analysis:

```
advisor --collect=dependencies --project-dir=./advi_results -- ./myApplication
```

See [Run Threading Perspective from Command Line](#) for details.



After running the perspective as describes above, you get a [Suitability report](#) showing predicted options for parallelizing and a [Dependencies report](#) showing whether you can implement parallel design without disrupting your code.

Customize Threading Perspective

Customize the perspective flow to better fit your goal and your application.

If you change any of the analysis settings from the **Analysis Workflow** tab, the accuracy level changes to **Custom** automatically. With this accuracy level, you can customize the perspective flow and/or analysis properties.

To change the properties of a specific analysis:

1. Expand the analysis details on the **Analysis Workflow** pane with .
2. Select desired settings.
3. For more detailed customization, click the *gear*  icon. You will see the **Project Properties** dialog box open for the selected analysis.
4. Select desired properties and click **OK**.

The following tables cover project properties applicable to analyses in the Threading perspective.

Common Properties

Use This	To Do This
Target type drop-down	<ul style="list-style-type: none"> Analyze an executable or script (choose Launch Application). Analyze a process (choose Attach to Process). <p>If you choose Attach to Process, you can either inherit settings from the Survey Hotspots Analysis Type or specify the needed settings.</p>
Inherit settings from Visual Studio project checkbox and field (Visual Studio* IDE only)	<p>Inherit Intel Advisor project properties from the Visual Studio* startup project (enable).</p> <p>If enabled, the Application, Application parameters, and Working directory fields are pre-filled and cannot be modified.</p> <hr/> <p>NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.</p> <hr/>
Application field and Browse... button	Select an analysis target executable or script.

Use This	To Do This
	If you specify a script in this field, consider specifying the executable in the Advanced > Child application field (required for Dependencies analysis).
Application parameters field and Modify... button	Specify runtime arguments to use when performing analysis (equivalent to command line arguments).
Use application directory as working directory checkbox	Automatically use the value in the Application directory to pre-fill the Working directory value (enable).
Working directory field and Browse... button	Select the working directory.
User-defined environment variables field and Modify... button	Specify environment variables to use during analysis.
Managed code profiling mode drop-down	<ul style="list-style-type: none"> Automatically detect the type of target executable as Native or Managed, and switch to that mode (choose Auto). Collect data for native code and do not attribute data to managed code (choose Native). Collect data for both native and managed code, and attribute data to managed code as appropriate (choose Mixed). Consider using this option when analyzing a native executable that makes calls to managed code. Collect data for both native and managed code, resolve samples attributed to native code, and attribute data to managed source only (choose Managed). The call stack in the analysis result displays data for managed code only.
Child application field	<p>Analyze a file that is not the starting application. For example: Analyze an executable (identified in this field) called by a script (identified in the Application field).</p> <p>Invoking these properties could decrease analysis overhead.</p> <hr/> <p>NOTE For the Dependencies Analysis Type: If you specify a script file in the Application field, you must specify the target executable in the Child application field.</p> <hr/>
Modules radio buttons, field, and Modify... button	<ul style="list-style-type: none"> Analyze specific modules and disable analysis of all other modules (click the Include only the following module(s) radio button and choose the modules). Disable analysis of specific modules and analyze all other modules (click the Exclude only the following module(s) radio button and choose the modules). <p>Including/excluding modules could minimize analysis overhead.</p>

Use This	To Do This
Use MPI launcher checkbox	<p>Generate a command line (enable) that appears in the Get command line field based on the following parameters:</p> <ul style="list-style-type: none"> • Select MPI Launcher - Intel or another vendor • Number of ranks - Number of instances of the application • Profile ranks - All or a range of ranks to profile
Automatically stop collection after (sec) checkbox and field	<p>Stop collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could minimize analysis overhead.</p>

Survey Analysis Properties

Use This	To Do This
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <hr/> <p>Tip The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p> <hr/>
Sampling Interval selector	<p>Set the wait time between each analysis collection CPU sample while your target application is running.</p> <p>Increasing the wait time could decrease analysis overhead.</p>
Collection data limit, MB selector	<p>Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses.</p> <p>Decreasing the limit could decrease analysis overhead.</p>
Callstack unwinding mode drop-down list	<p>Set to After collection if:</p> <ul style="list-style-type: none"> • Survey analysis runtime overhead exceeds 1.1x. • A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions. <p>Otherwise, set to During Collection. This mode improves stack accuracy but increases overhead.</p>
Stitch stacks checkbox	<p>Restore a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload (enable).</p> <p>Disable if Survey analysis runtime overhead exceeds 1.1x.</p>
Analyze MKL Loops and Functions checkbox	<p>Show Intel® oneAPI Math Kernel Library (oneMKL) loops and functions in Intel Advisor reports (enable).</p>

Use This	To Do This
	Enabling could increase analysis overhead.
Analyze Python loops and functions checkbox	Show Python* loops and functions in Intel Advisor reports (enable). Enabling could increase analysis overhead.
Analyze loops that reside in non-executed code paths checkbox	Collect a variety of data during analysis for loops that reside in non-executed code paths, including loop assembly code, instruction set architecture (ISA), and vector length (enable). Enabling could increase analysis overhead. NOTE Analyzing non-executed code paths in binaries that target multiple ISAs (contain multiple code paths) is available only for binaries compiled using the <code>-ax</code> (Linux* OS) / <code>Qax</code> (Windows* OS) option with an Intel compiler.
Enable registry spill/fill analysis checkbox	Calculate the number of consecutive load/store operations in registers and related memory traffic (enable). Enabling could increase analysis overhead.
Enable static instruction mix analysis checkbox	Statically calculate the number of specific instructions present in the binary (enable). Enabling could increase analysis overhead.
Source caching drop-down list	<ul style="list-style-type: none"> Delete source code cache from a project with each analysis run (default; choose Clear cached files). Keep source code cache within the project (choose Keep cached files).

Suitability Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds.
Sampling Interval selector	Set the wait time between each analysis collection sample while your target application is running.

Use This	To Do This
	Increasing the wait time could decrease analysis overhead.
Collection data limit, MB selector	Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses. Decreasing the limit could decrease analysis overhead.

Dependencies Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Suppression mode radio buttons	<ul style="list-style-type: none"> Report possible dependencies in system modules (choose the Show problems in system modules radio button). Do not report possible dependencies in system modules (choose the Suppress problems in system modules radio button).
Loop call count limit selector	Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances. Supplying a non-zero value could decrease analysis overhead.
Instance of interest selector	Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes. Supplying a non-zero value could decrease analysis overhead.
Analyze stack variables checkbox	Analyze parallel data sharing for stack variables (enable). Enabling could increase analysis overhead.
Filter stack variables by scope checkbox	Enable to report: <ul style="list-style-type: none"> Variables initiated inside the loop as potential dependencies (warning) Variables initialized outside the loop as dependencies (error) Enabling could increase analysis overhead.
Reduction Detection / Filter reduction variables checkbox	Mark all potential reductions by a specific diagnostic (enable). Enabling could increase analysis overhead.
Markup type checkbox	Select loops/functions by pre-defined markup algorithm. Supported algorithms are: <ul style="list-style-type: none"> GPU generic - Select loops executed on a GPU. OpenMP - Select OpenMP* loops. SYCL - Select SYCL loops. OpenCL - Select OpenCL™ loops. DAAL - Select Intel® oneAPI Data Analytics Library loops. TBB - Select Intel® oneAPI Threading Building Blocks loops.

Use This	To Do This
	<p>NOTE This option is available only from the Analysis Workflow pane for the Offload Modeling perspective.</p>

Trip Counts and FLOPs Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds.
Trip Counts / Collect information about Loop Trip Counts checkbox	Measure loop invocation and execution (enable).
FLOP / Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage checkbox	Measure floating-point operations, integer operations, and memory traffic (enable).
Callstacks / Collect callstacks checkbox	Collect call stack information when performing analysis (enable). Enabling could increase analysis overhead.
Capture metrics for dynamic loops and functions checkbox	Collect metrics for dynamic Just-In-Time (JIT) generated code regions.
Capture metrics for stripped binaries checkbox	Collect metrics for stripped binaries. Enabling could increase analysis overhead.
Cache Simulation / Enable Memory-Level Roofline with cache simulation checkbox	Model multiple levels of cache for data, such as counts of loaded or stored bytes for each loop, to plot the Roofline chart for all memory levels (enable). Enabling could increase analysis overhead.
Cache simulator configuration field	Specify a cache hierarchy configuration to model (enable and specify hierarchy). The hierarchy configuration template is:

Use This	To Do This
	<pre>[num_of_level1_caches]:[num_of_ways_level1_connected]: [level1_cache_size]:[level1_cacheline_size]/ [num_of_level2_caches]:[num_of_ways_level2_connected]: [level2_cache_size]:[level2_cacheline_size]/ [num_of_level3_caches]:[num_of_ways_level3_connected]: [level3_cache_size]:[level3_cacheline_size]</pre> <p>For example: 4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l is the hierarchy configuration for:</p> <ul style="list-style-type: none"> • Four eight-way 32-KB level 1 caches with line size of 64 bytes • Four four-way 256-KB level 2 caches with line size of 64 bytes • One sixteen-way 6-MB level 3 cache with line size of 64 bytes
Data Transfer Simulation / Data transfer simulation mode drop-down	<p>Select a level of details for data transfer simulation:</p> <ul style="list-style-type: none"> • Off - Disable data transfer simulation analysis. • Light - Model data transfers between host and device memory. • Full - Model data transfers, attribute memory objects to loops that accessed the objects, and track accesses to stack memory.

Run Threading Perspective from Command Line

Threading perspective includes several steps that you are recommended to run one by one:

1. Collect performance metrics and find candidates for parallelizing using a **Survey** analysis.
2. **Annotate** manually loops/functions to model parallelization for.
3. Model parallel design options and estimate speedup for the annotated loops using a **Suitability** analysis.
4. Check for loop-carried **dependencies** to make sure the loops/functions are safe to parallelize.

Tip See [Intel Advisor cheat sheet](#) for quick reference on command line interface.

Prerequisites

Set [Intel Advisor environment variables](#) with an automated script to enable the `advisor` command line interface (CLI).

Run Threading Perspective

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

1. Run the Survey analysis.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

2. Run the Characterization analysis to collect trip counts and FLOP data.

```
advisor --collect=tripcounts --project-dir=./advi_results --flop -- ./myApplication
```

3. View the Survey report to identify candidates for parallelization. For example, run the following command to print the report in command line:

```
advisor --report=survey --project-dir=<project-dir>
```


Consider analyzing loops/functions with high total time.

4. In the application source code, [annotate loops/functions of interest](#) to model parallelization for. [Rebuild the application](#) as usual to make the annotations available for the Intel Advisor.
5. Run the Suitability analysis to model threading for the annotated loops/functions:

```
advisor --collect=suitability --project-dir=./advi_results -- ./myApplication
```

6. Run the Dependencies analysis to check for loop-carried dependencies in the annotated loops:

```
advisor --collect=dependencies --project-dir=./advi_results -- ./myApplication
```

You can view the results in the Intel Advisor graphical user interface (GUI), print a summary to a command prompt/terminal, or save to a file. See [View the Results](#) below for details.

Analysis Details

Each analysis in the Threading perspective has a set of additional options that modify its behavior and collect additional performance data.

Consider the following options:

Characterization Options

To run the Characterization analysis, use the following command line action: `--collect=tripcounts`.

Recommended action options:

Options	Description
<code>--flop</code>	Collect data about floating-point and integer operations, memory traffic, and mask utilization metrics for AVX-512 platforms.
<code>--stacks</code>	Enable advanced collection of call stack data.

Dependencies Options

To run the Dependencies analysis, use the following command line action: `--collect=dependencies`.

Recommended action options:

Options	Description
<code>--filter-reductions</code>	Mark all potential reductions with a specific diagnostic.

See [advisor Command Option Reference](#) for more options.

Next Steps

Continue to [explore threading results](#). For details about the metrics reported, see [CPU and Memory Metrics](#).

See Also

[Threading Perspective](#) Analyze, design, tune, and check threading design options without disrupting your normal development by running the Threading Perspective.

[Command Line Interface](#) This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

[Minimize Analysis Overhead](#)

[Analyze MPI Applications](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

Threading Accuracy Levels in Command Line

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

In CLI, each accuracy level corresponds to a set of commands with specific options that you should run one by one to get a desired result.

For the Threading perspective, you are recommended to run the accuracy levels one by one to get a Threading report.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium
Overhead	1.1x	5 - 8x
Goal	Find candidates for parallelization	Model threading parallelism and check for loop-carried dependencies
Analyses	Survey	Survey + Characterization (Trip Counts) + Suitability + Dependencies
Result	Basic Survey report	Survey report extended with trip count data Dependencies report Suitability report with parallel performance modeled for annotated loops

You can generate commands for a desired accuracy level from the Intel Advisor GUI. See [Generate Command Lines from GUI](#) for details.

NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).

Consider the following command examples.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Low Accuracy

First, run the Threading perspective with low accuracy to find candidates for parallelizing based on Survey analysis results.

Run the analysis as follows:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

You can view the generated results in the Intel Advisor GUI or in the CLI. The loops/functions with high total time are the best candidates for parallelization. [Annotate the loops/functions of interest](#) to model parallelism.

Medium Accuracy

Prerequisite: [Annotate loops/functions](#) to model parallelization for. Rebuild the application.

Run the commands as follows:

1. Run the Survey analysis:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

2. Collect trip count data:

```
advisor --collect=tripcounts --project-dir=./advi_results -- ./myApplication
```

3. Run the Suitability analysis to model threading parallelism for the annotated loops:

```
advisor --collect=suitability --project-dir=./advi_results -- ./myApplication
```

4. Run the Dependencies analysis for the annotated loops:

```
advisor --collect=dependencies --project-dir=./advi_results -- ./myApplication
```

You can view the generated results in the Intel Advisor GUI or in the CLI.

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[Run Threading from Command Line](#)

[Minimize Analysis Overhead](#)

Annotate Code for Deeper Analysis

Before you can *mark* the best parallel opportunities by adding Intel® Advisor annotations, you need to *choose* likely places to add parallelism. This section provides a series of topics that explain factors to consider as you examine the candidate code regions and their execution and choose candidate places.

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

The operations of a serial program execute one after another in a well-defined order, starting at the beginning, continuing to the end, and then stopping. A parallel program, on the other hand, is made up of *tasks* - portions of the program that may execute independently on separate cores. Tasks can either be implemented in separate functions or in iterations of a loop.

You mark your proposed code regions by adding Intel® Advisor annotations that identify the:

- **Parallel site:** A code region that contains one or more parallel tasks. Execution of a parallel site constrains the time during which the tasks that it contains can execute. Although execution of a parallel site begins when its execution reaches its beginning, its execution terminates only after all tasks that started within it have completed. In parallel frameworks, this corresponds to the *join* location in the code where all tasks have completed.
- **Parallel tasks:** Task code regions run independently, at the same time as other tasks within the parallel site and the enclosing parallel site itself. Also, each task can have multiple instances of its code executing. As shown in the table below, there are two forms of task annotations:
 - For a loop with only a single task, add a single iteration task annotation within the two site annotations.
 - For other code, add a task annotation pair to mark the task region's begin and end within the two site annotations.

Characteristics of Parallel Site Code	Parallel Site and Task Annotations	Comments and Limitations
<p>A loop that requires only a single task. For simple loops, begin with the type of task annotation, unless the task does not include the entire loop body.</p> <p>Example code: <code>nqueens_Advisor C/C++ sample and nqueens Fortran and C# samples</code></p>	<p>Add three annotations to mark:</p> <ul style="list-style-type: none"> The parallel site region by adding site begin and site end annotations. The parallel task loop by adding a single iteration task annotation at the start of the loop body. 	<p>Based on the Suitability tool performance predictions, you may want to try using multiple tasks. In this case, remove the single iteration task annotation and replace it with task begin and task end annotations for each task (see the next row).</p> <p>If the loop structure is complex, you may need to mark the task begin and task end region by using the task annotations in the next row.</p>
<p>Complex loop, code that allows multiple tasks, or non-loop code</p> <p>Example code: <code>stats C++ sample</code></p>	<p>Add four annotations to mark:</p> <ul style="list-style-type: none"> The parallel site region by adding site begin and site end annotations. Each parallel task region by adding task begin and task end annotations. 	

After you choose several places to add parallelism, view the data displayed in the **Survey Report** window. Use this data and your code editor to add annotations to mark the candidate parallel sites and their task(s). Make sure that these annotations are executed by the selected target executable.

The site and task annotations enable the Intel® Advisor Suitability and Dependencies tools to predict your serial program's execution as a parallel program. These tools perform extensive analysis of your running serial program to provide data needed to help you decide the best place(s) to add parallelism.

To take advantage of the Intel® Advisor parallel design capabilities, experiment with different possible parallel code regions by modifying the site and task annotations and their locations, rebuilding your application's target, and running the Suitability and Dependencies tools again.

The following figure illustrates the `nqueens_Advisor C/C++ sample` code to show the task (blue background) and its enclosing parallel site (orange background).



Before you convert your serial program into a parallel program, you need to:

- Understand where your program is spending its time.
- Decide how to divide that work up into tasks that can execute in parallel.

Annotate Code to Model Parallelism

After identifying candidates for parallelizing, mark up serial parts of your code where you plan to add parallelism using Intel® Advisor annotations.

Before Annotating Code for Deeper Analysis

Before you can *mark* the best parallel opportunities by adding annotations, you need to *choose* likely places to add parallelism. This section introduces several topics that explain factors you should consider as you closely examine the candidate code regions and their execution.

Each code region where you might add parallelism consists of a single *parallel site* and one or more *parallel tasks* enclosed within the parallel site. Each parallel site defines the scope of parallel execution. You can have multiple parallel sites in a program.

No matter how much you improve one part of your program, the program cannot complete any faster than the part that you did not speed up. So, focus your efforts on the parts of your program that use the most time.

Use the **Survey Report** provided by the Survey tool to help you understand where your program spends its time.

Use Amdahl's Law and Measure the Program

There are two rules of optimization that apply to parallel programming:

- Focus on the part of the program that uses the most time.
- Do not guess, measure.

Amdahl's Law

In the context of parallel programming, Gene Amdahl formalized a rule called Amdahl's Law, which states that the speed-up that is possible from parallelizing one part of a program is limited by the portion of the program that still runs serially.

The consequence may be surprising: parallelizing the part of your program where it spends 80% of its time cannot speed it up by more than a factor of five, no matter how many cores you run it on.

Therefore, to get maximum benefit from parallelizing your program, you could add parallelism to all parts of your program as suggested by Amdahl's Law. However, it is more practical to find where it spends most of its time and focus on areas that can provide the most benefit.

Do Not Guess - Measure

This leads to another rule of optimization: *Do Not guess - Measure*. Programmers' intuitions about where their programs are spending time are notoriously inaccurate. Intel® Advisor includes a Survey tool you can use to profile your running program and measure where it spends its time.

After you add Intel® Advisor annotations to your program to mark the proposed parallel code regions, run the Suitability tool to predict the approximate maximum performance gain for the program and the annotated sites. These estimated performance gain values are based on a model of parallel execution that reflects the impact of Amdahl's law.

See Also

[Task Organization and Annotations](#)

Task Organization and Annotations

You will choose a region of code to execute as a task. This region is the *static extent* of the task. The task includes not just its static extent, but also any other code that is called from the static extent when it executes - this is the *dynamic extent*.

In addition to choosing tasks, you will also decide which tasks can execute in parallel with one another. To do this, you will choose *parallel sites*. A parallel site, like a task, has a static extent which is a block of code and a dynamic extent which includes all the code that is called from it.

NOTE

If you have a loop with a single task and the task includes the entire loop body, you can use the simplified parallel site with one iteration task annotation. The remainder of this topic and this group of topics describe the more complex case where multiple tasks are needed within a parallel site.

The execution of tasks with the serial execution done by Intel® Advisor works like this:

1. A parallel site begins when execution reaches the begin-site annotation.
2. A task is created when execution reaches the begin-task annotation. The task executes independently, in parallel with any other tasks that are already executing, including the parallel site itself.
3. When the execution of a task reaches an end-task annotation, the task terminates. Intel® Advisor end-task annotations do not allow or require an end-task label, so be aware that in some cases the task's execution could reach a task-end annotation for a different task, which can impact the predicted parallel performance.
4. When execution reaches the end-site annotation for the parallel site, Intel® Advisor predicts that execution suspends (waits) until all tasks that were created within it have terminated, after which execution exits the parallel site.

With C/C++ code, note that `goto`, `break`, `continue`, `return`, and `throw` statements must not bypass the end of the static extent of a task or parallel site! With Fortran code, such statements include `goto` and `return`. You may need to add extra `end` annotations before these operations so the Intel® Advisor tools will correctly model the end of a site or task.

Because you will later add parallel framework code after you no longer need the Intel® Advisor annotations, you need to be aware of the requirements of the parallel framework. For example, some parallel frameworks might not allow a branch out of a task, such as a loop task. Whenever possible, plan your tasks to suit the needs of the parallel framework code. The annotations are present only while you need Intel® Advisor to help you predict the proposed parallel behavior and make decisions about the best locations for your tasks.

After you decide where the parallel sites and tasks are in your program, add source annotations.

See Also

[Annotate Parallel Sites and Tasks](#)

[Site and Task Annotations for Simple Loops with One Task](#)

[Copy Annotations and Build Settings Using the Annotation Assistant Pane](#)

Annotate Parallel Sites and Tasks

You add *annotations* into your program to mark the tasks and parallel sites. The annotations are one-line macro uses or function calls that have no effect on the behavior of your program.

Annotations allow you to mark your tentative decisions about your program's task structure before you modify the program to use parallel execution. Annotations are used by the Intel® Advisor Suitability and Dependencies tools.

After you decide on the parallel site(s) and task(s), add the annotations into your source code.

To simplify adding Intel® Advisor annotations:

- When using the Microsoft Visual Studio* code editor, you can use the Annotation Wizard.

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

- With any editor, use the annotation assistant in the **Survey Report** window, **Survey Source** window, or the **No Data** message to copy example annotation code and build settings.

Code examples throughout this group of topics illustrate the use of these annotations.

As you use Intel® Advisor to investigate possible code regions for adding parallel execution, you will find some areas are not feasible. Adding a comment to explain why that site (or task) was not chosen may help later. For example, with C/C++ code:

```
...
// Investigated the following function call as a parallel task and dismissed
// June 2014. Need to first re-write the function to improve parallel
// performance and fix the data race.
//
// ANNOTATE_TASK_BEGIN(func1);
```

-
-
-

See Also

[Task Patterns](#)

[Intel Advisor Annotation Definitions File](#)

[Annotation Types Summary](#)

[Copy Annotations and Build Settings Using the Annotation Assistant Pane](#)

[Add Annotations Using the Annotation Wizard](#)

[Add Parallelism](#)

Task Patterns

To summarize:

- You choose *parallel sites* in your program.
- You choose *tasks* in your parallel sites.
- Tasks in a parallel site can execute in parallel with one another and with tasks in an outer parallel site, but not in parallel with tasks in unrelated parallel sites.

You are free to arrange your sites and tasks any way that you want, but there are several simple, common patterns that you will probably want to use.

The following sections describe the process of identifying task patterns, as well as information about data parallelism and task parallelism.

Multiple Parallel Sites

You may be able to introduce parallelism independently in more than one place in a program.

For example, consider a C/C++ program with the general structure:

```
initialize(data);
while (!done) {
    display_on_screen(data);
    update(data);
}
```

You might be able to parallelize the display and update operations independently:

```
display_on_screen(data)
{
    ANNOTATE_SITE_BEGIN(site_display);
    for (each block of data) {
        ANNOTATE_ITERATION_TASK(task_display);
        display the block of data;
    }
    ANNOTATE_SITE_END();
}
update(data)
{
    ANNOTATE_SITE_BEGIN(site_update);
    for (each block of data) {
        ANNOTATE_ITERATION_TASK(task_update);
        update the block of data;
    }
    ANNOTATE_SITE_END();
}
```

Each iteration of the main loop would still do the display and then the update, but the display and update operations could be performed much faster.

Depending on your program, you need to decide whether to implement multiple parallel sites at the same or at different times:

- When two parallel sites are truly disjoint or have overlapping functions that are purely functional and do not show problems reported by the Dependencies tool, you can consider parallelizing those sites separately at different times.
- When considering multiple parallel sites that overlap on the same call trees - such as multiple sites that call the same (common) utility functions - consider parallelizing or not parallelizing the entire set of parallel sites at the same time.

You need to determine the cause of each dependency and fix it. If you have multiple parallel sites that overlap on the same call trees - such as multiple sites that call the same utility functions (common code) - read the help topic [Fixing Problems in Code Used by Multiple Parallel Sites](#).

See Also

[Data and Task Parallelism](#)

[Using Partially Parallel Programs with Intel Advisor Tools](#)

[Data Sharing Problems](#)

[Fixing Problems in Code Used by Multiple Parallel Sites](#)

Data and Task Parallelism

This topic describes two fundamental types of program execution - data parallelism and task parallelism - and the task patterns of each.

Data Parallelism

In many programs, most of the work is done processing items in a collection of data, often in a loop. The data parallelism pattern is designed for this situation. The idea is to process each data item or a subset of the data items in separate task instances. In general, the parallel site contains the code that invokes the processing of each data item, and the processing is done in a task.

In the most common version of this pattern, the serial program has a loop that iterates over the data items, and the loop body processes each item in turn. The data parallelism pattern makes the whole loop a parallel site, and the loop body is a task. Consider this C/C++ simple loop:

```
ANNOTATE_SITE_BEGIN(sitename);
for (int I = 0; I != n; ++I) {
    ANNOTATE_ITERATION_TASK(task_process);
    process(a[i]);
}
ANNOTATE_SITE_END();
```

The following C/C++ code shows a situation where the data items to be processed are in the nodes of a tree. The recursive tree walk is part of the serial execution of the parallel site - only the `process_node` calls are executed in separate tasks.

```
ANNOTATE_SITE_BEGIN(sitename);
process_subtree(root);
ANNOTATE_SITE_END(sitename);
...
void process_subtree(node) // in the dynamic extent of the parallel site
{
    ANNOTATE_TASK_BEGIN(task_process);
    process_node(node);
    ANNOTATE_TASK_END();
    for (child = first_child(node);
        child;
        child = next_child(child) )
    {
```



```
        process_subtree(child);
    }
}
```

In the data parallelism pattern, the parallel site usually contains a single task.

The sample `tachyon_Advisor` demonstrates data parallelism.

Task Parallelism

When work is divided into several activities which you cannot parallelize individually, you may be able to take advantage of the task parallelism pattern.

NOTE

The word *task* in *task parallelism* is used in the general sense of an activity or job. It is just a coincidence that we use the same word to refer to "a body of code that is executed independently of other bodies of code".

In this pattern, you have multiple distinct task bodies in a parallel site performing different activities at the same time.

Suppose that neither the display nor the update operation from the previous example can be parallelized individually. You still might be able to do the display and the update simultaneously. Consider this C/C++ code:

```
initialize(data);
while (!done) {
    old_data = data;
    ANNOTATE_SITE_BEGIN(sitename);
    ANNOTATE_TASK_BEGIN(task_display);
    display_on_screen(old_data);
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(task_updatedata);
    update(data);
    ANNOTATE_TASK_END();
    ANNOTATE_SITE_END();
}
```

The most obvious shortcoming of the task-parallel pattern is that it cannot take advantage of more cores than the number of distinct tasks. In this example, any more than two cores would be wasted. On the other hand, the task parallel pattern may be applicable to programs that simply do not fit the data parallel pattern - some parallelism may be better than none.

The tasks used in task parallelism are not limited to called functions. For example, consider this C/C++ code that creates two tasks that separately increment variables `x` and `y`:

```
main() {
    ANNOTATE_SITE_BEGIN(sitename);
    ANNOTATE_TASK_BEGIN(task_x);
    X++;
    ANNOTATE_TASK_END();

    ANNOTATE_TASK_BEGIN(task_y);
    Y++;
    ANNOTATE_TASK_END();
    ANNOTATE_SITE_END();
}
```

The sample `stats` demonstrates task parallelism.

See Also

[Mixing and Matching Tasks](#)

[Annotations](#)

Mix and Match Tasks

You can combine the data parallel and task parallel patterns. Continuing with the display/update example, suppose that you can parallelize the update operation, but not the display operation. Then you could execute the display operation in parallel with multiple tasks from the update operation. Consider this C/C++ code:

```
initialize(data);
while (!done) {
    old_data = data;
    ANNOTATE_SITE_BEGIN(sitename);
    ANNOTATE_TASK_BEGIN(task_display);
    display_on_screen(old_data);
    ANNOTATE_TASK_END();
    update(data);
    ANNOTATE_SITE_END();
}
display_on_screen(data)
{
    . . .
}
update(data)
{
    for (each block of data) {
        ANNOTATE_TASK_BEGIN(task_update);
        update the block of data;
        ANNOTATE_TASK_END();
    }
}
```

See Also

[Choosing the Tasks](#)

[Annotations](#)

Choose the Tasks

When choosing tasks, you should consider task interactions and the factors that influence how large a task should be. The following sections describe the process of choosing the tasks.

Task Interactions and Suitability

If your tasks access the same memory locations, then, left to themselves, they will tend to trip over each other. You can solve this by adding synchronization code to make sure the tasks are well-behaved when they access shared memory locations, but synchronization code can be tedious to add and hard to get right, and it is easy to end up with tasks that spend more time doing synchronization than doing work.

You can use the Suitability tool to provide performance data that helps you choose your tasks wisely.

It is better to minimize data access conflicts in the first place by choosing your tasks wisely. It can be hard to tell, just by looking at your code, where all the sharing problems will be, which is why you will learn how to automate the process by using the Dependencies tool.

However, you can make a good guess whether two proposed tasks are mostly independent of each other or are completely intertwined.

See Also

[How Big Should a Task Be?](#)

[Model Threading Parallelism](#)

[Dependencies Analysis](#)

How Big Should a Task Be?

The ideal task size is very dependent on the details of your program. Here are a few general considerations to keep in mind.

Task Overhead

In general, if your program can keep most of the cores on your system busy doing useful work, then it will be using the system about as efficiently as possible. There are two parts to this: keeping the cores busy, and doing useful work.

It takes time to start a new task. If your tasks are too small, then your program may spend more time creating tasks than it saves by running them in parallel - the cores are kept busy, but not doing useful work.

Load Balance

On the other hand, very large tasks can reduce parallelism: your parallel program cannot finish any more quickly than the longest-running task. A rule of thumb is to try to have the number of tasks in a parallel site be at least several times larger than the number of cores available, so that there will always be some work to do when a core is free.

Choosing the Right Level

You will often have the opportunity to create tasks at different loop nesting levels or function call depths. This may provide an easy way to choose your task size. For example, consider the C/C++ code:

```
for (i = 0; i != N; ++i) {
    for (j = 0; j != N; ++j) {
        x[i, j] = y[i, j] * z[j, i];
    }
}
```

The inner loop body is too small to be a useful task. You can view the Suitability Report for a task's Average Instance Time. The entire inner loop might be more suitable:

```
ANNOTATE_SITE_BEGIN(sitename);
for (i = 0; i < N; ++i) {
    ANNOTATE_ITERATION_TASK(task_process_array);
    for (j = 0; j < N; ++j) {
        x[i, j] = y[i, j] * z[j, i];
    }
}
ANNOTATE_SITE_END();
```

Blocking

If you have a loop which seems like an obvious place to introduce parallelism, but the loop body is too small to make a good task, consider grouping several iterations together. When you specify a loop body as a parallel construct, Intel® oneAPI Threading Building Blocks and OpenMP* will automatically group multiple loop iterations together to create tasks of an appropriate size. Therefore, given a simple loop, the question is not whether the loop body is the right size for a good task, but whether the total loop execution time can be divided up into chunks of the right size.

For example, there is only one loop level here, and its body looks too small to be a good task:

```
for (i = 0; i < 100000; ++i) {
    a[i] = b[i] * c[i];
}
```

Go ahead and choose it, and it may run as though you had written it as:

```
ANNOTATE_SITE_BEGIN(sitename);
for (i = 0; i < 100000; i += 1000) {
    ANNOTATE_ITERATION_TASK(task_calculate_a);
    for (j = i; j < i + 1000; ++j) {
        a[j] = b[j] * c[j];
    }
}
ANNOTATE_SITE_END();
```

Sizing to Avoid Interactions

It is not uncommon for loop iterations or other potential task bodies to be almost independent at one level, but have many interactions at other levels. In this case, it may be worth accepting a less than perfect program gain in exchange for simpler programming and cleaner code.

The outer loop of the Sudoku problem generator repeatedly calls the `generate()` function to generate problems. There are opportunities for introducing parallelism at many different levels in the problem generation function, but the individual calls to `generate()` are almost perfectly independent, and each call to `generate()` takes less than a second. Parallelizing the outermost loop would be a trivial project. No user is likely to care if it takes 0.8 seconds instead of 0.2 seconds to generate a single problem, and the speedup for generating more than a handful of problems should be nearly perfect.

Using the Survey Report

Ultimately, choosing your tasks is more of an art than a science. Locations close to the root of the call tree tend to form larger tasks, but may have more conflicts on shared variables; locations toward the leaves of the call tree tend to be smaller, causing problems with task overhead, but typically have fewer conflicts. We can offer some rules of thumb. Start by looking at a function `F` that uses a significant portion of the time of the program part you are trying to improve - remember Amdahl's law!

- If almost all of the time spent in `F` is spent in a block of code that is executed many times in a loop, then that block of code may be a prime candidate for a data-parallel task.
- If `F` is basically just a wrapper around a call to a function `G`, then look at `G` instead.
- If almost all of the time in `F` is spent in multiple calls to a function `G` that is too large to be a good task, then you may want to enclose the calls to `G` in a parallel site, but introduce the actual tasks inside `G` or another function that is called from `G`.
- If the time spent in `F` is distributed across a number of distinct activities, you should consider whether it is better to apply the task parallelism pattern to `F`, or to use the multiple parallel sites pattern to look for parallelism in each of the activities.

Recursion

Recursive algorithms can present a special challenge. The problem occurs when you have a large amount of time spent in a function that only does a small amount of work in any one invocation, but that is called recursively a great many times. The actual work may be data-parallel, but the function body is too small to be a useful task by itself, and the blocking strategy (see Blocking above) is harder to apply to a recursive algorithm.

The general solution is to use a threshold to control recursive parallelism. For example, a recursive sort might solve sub-problems in parallel only if they are above a certain threshold size.

See Also

[Using Partially Parallel Programs with Intel Advisor Tools](#)
[Data and Task Parallelism](#)

Use Partially Parallel Programs with Intel® Advisor

Intel® Advisor tools are designed to collect data and analyze *serial* programs. If you have a partially parallel program, *before* you use the Intel® Advisor Suitability and Dependencies tools to examine it to add more parallelism, read the guidelines in this topic and modify your program so it runs as a serial program with a single thread within each parallel site.

Run Your Program as a Serial Program

To run the current version of your program as a serial program, you need to limit the number of threads to 1. To run your program with a single thread:

- With Intel® oneAPI Threading Building Blocks (oneTBB), in the main thread create a `tbb::task_scheduler_init init(1);` object for the lifetime of the program and run the executable again. For example:

```
int main() {
    tbb::task_scheduler_init init(1);
    // ...rest of program...

    return 0;
}
```

The effect of `task_scheduler_init` applies separately to each user-created thread. So if the program creates threads elsewhere, you need to create a `tbb::task_scheduler_init init(1);` for that thread's lifetime as well. Use of certain oneTBB features can prevent the program from running serially. For more information, see the oneTBB documentation.

- With OpenMP*, do one of the following:
 - Set the OpenMP* environment variable `OMP_NUM_THREADS` to 1 before you run the program.
 - Omit the compiler option that enables recognition of OpenMP pragmas and directives. On Windows* OS, omit `/Qopenmp`, and on Linux* OS omit `-openmp`.

For more information, see your compiler documentation.

Add or Remove Intel® Advisor Annotations

Intel® Advisor site, task, and lock annotations are used by the Suitability and Dependencies tools. You can add Intel® Advisor parallel site and task annotations to mark the already parallel code regions. For example, the `nqueens_Advisor` sample `nqueens_cilk.cpp`:

```
...
ANNOTATE_SITE_BEGIN(solve);
cilk_for(int i=0; i<size; i++) {
    // try all positions in first row using separate array for each recursion
    ANNOTATE_ITERATION_TASK(setQueen);
    int * queens = new int[size];
    setQueen(queens, 0, i);
}
ANNOTATE_SITE_END();
```

If needed, you can comment out annotations, or add preprocessor directives by using conditional compilation. For example, use the `#ifdef`, `#ifndef`, and `#endif` preprocessor directives:

```
...
// Comment out the next line to hide the annotations.
#define ANNOTATE_ON
.
.
.
```

```

#ifdef ANNOTATE_ON
    ANNOTATE_SITE_BEGIN(solve);
#endif
#ifndef ANNOTATE_ON
    // add parallel code here
    .
    .
    .
#ifdef ANNOTATE_ON
    ANNOTATE_SITE_END();
#endif
...

```

After you add the parallel framework code and test it, you can remove the annotations.

Effect of Parallel Code on Intel® Advisor Tools' Reports

Because Intel® Advisor tools are designed to collect data and analyze *serial* program targets.

Parallel code that creates one or more threads within any annotated parallel site usually cause the Suitability or Dependencies tool reports to contain unreliable data. To use these two tools, there must be only a single thread within each parallel site. Also, when using parallel frameworks that use dynamic scheduling or work stealing at run-time, execution times can be assigned to the wrong source code.

If you use the Survey tool to profile your program, the **Self Time** in the Survey Report shows the sum of the CPU time for all threads. However, because Intel® Advisor's purpose is to analyze serial code, some of the time used by parallel code may be added to the wrong places. For example, **Self Time** may be added to the parallel framework run-time system entry points instead of the caller(s) in the thread that entered the parallel region. Also in the Survey Report, when examining parallel code, some entry points may be parallel framework run-time system entry points instead of the expected functions or loops. Similarly, in the Survey Source window, for a parallel code region the **Total Time** (and **Loop Time**) shows the sum of the CPU time for all threads.

Because Intel® Advisor's purpose is to analyze serial code, in the Suitability Report:

- Intel® Advisor assumes there is only a single thread (no parallelism) within any annotated parallel site, including its task(s) and lock(s). When only a single thread executes within a parallel site (as expected), the results for *that site* may be correct. If the application has multiple parallel sites, and one or more sites were executed by multiple threads, the next two items apply.
- If multiple threads execute within *any* parallel site, the reported **Maximum Program Gain** and that site's **Impact on Program Gain** values are not reliable. To obtain correct values, ensure that only a single thread executes for all parallel sites (see Run Your Program as a Serial Program above).
- If multiple threads execute within a parallel site, the results for that site will be unpredictable and its values will not be reliable. Also, if one thread executes the parallel site annotations and a second thread executes the task annotation(s), the site may appear to not have any tasks and the tasks may appear to not execute within a site. To obtain correct values, ensure that only a single thread executes within each parallel site (see Run Your Program as a Serial Program above).
- Any work-stealing constructs within the site will cause extra time to be added to the suspended site and/or task. All Suitability Report times are approximate.

Similarly in the Dependencies Report, if any parallel site uses multiple threads, this may prevent certain problems from being detected and reported by the Dependencies tool. To obtain correct values, ensure that only a single thread executes within each parallel site (see Run Your Program as a Serial Program above).

See Also

[Model Threading Parallelism](#)

[Using Intel® Inspector and Intel® VTune™ Profiler](#)

Annotations

You add Intel® Advisor annotations to mark the places in serial parts of your program where Intel® Advisor tools should assume your program's parallel execution and synchronization will occur. Later, after you modify your program to prepare it for parallel execution, you replace these annotations with parallel framework code that enables parts of your program to execute in parallel.

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

Annotations are either subroutine calls or macro uses, depending on which language you are using, so they can be processed by your current compiler. The annotations do not change the computations of your program, so your application runs normally.

The three main types of annotations mark the location of:

- A *parallel site*. A parallel site encloses one or more tasks and defines the scope of parallel execution. When converted to a parallel code, a parallel site executes initially using a single thread.
- One or more *parallel tasks* within the parallel site. Each task encountered during execution of a parallel site is modeled as being possibly executed in parallel with the other tasks and the remaining code in the parallel site. When converted to parallel code, the tasks will run in parallel. That is, each instance of a task's code may run in parallel on separate cores, and the multiple instances of that task's code also runs in parallel with multiple instances of any other tasks within the same parallel site.
- Locking synchronization, where mutual exclusion of data access must occur in the parallel program.

In addition, there are:

- Annotations that stop and resume data collection. Data collection occurs while the target executes. These annotations allow you to skip uninteresting parts of the target program's execution.
- Special-purpose annotations used in less common cases.

The three Intel Advisor tools recognize the three main types of annotations and the Stop and Resume Collection annotations. Only the Dependencies tool processes the special-purpose annotations.

Use the parallel site and task annotations to mark the code regions that are candidates for adding parallelism. These annotations enable the Intel® Advisor Suitability and Dependencies tools to predict your serial program's parallel behavior. For example:

- The Suitability tool runs your program and uses parallel site and task boundaries to predict your parallel program's approximate performance characteristics.
- The Dependencies tool runs your program and uses parallel site and task boundaries to check for data races and other data synchronization problems.

One common use of sites and tasks is to enclose an entire loop within a parallel site, and to enclose the body of the loop in a task. For example, the following C/C++ code shows a simple loop that uses two parallel site annotations and one task annotation from the `nqueens_Advisor` sample. The three added annotations and the line that includes the annotation definitions appear in a **bold font** below.

```
#include "advisor-annotate.h"
...
void solve() {
int * queens = new int[size]; //array representing queens placed on a chess board...
ANNOTATE_SITE_BEGIN(solve);
for(int i=0; i<size; i++) {
    // try all positions in first row
    ANNOTATE_ITERATION_TASK(setQueen);
    setQueen(queens, 0, i);
}
```

```

ANNOTATE_SITE_END();
...
}

```

The following code from the Fortran `nqueens` sample shows the use of parallel site and task Fortran annotations, such as `call annotate_site_begin("label")`. The three added annotations and the line that references the annotation definitions module (the `use` statement) appear in a **bold** font below.

```

use advisor_annotate
...
! Main solver routine
subroutine solve (queens)
  implicit none
  integer, intent(inout) :: queens(:)
  integer :: i
  call annotate_site_begin("solve")
  do i=1,size
    ! try all positions in first row
    call annotate_iteration_task("setQueen")
    call SetQueen (queens, 1, i)
  end do
  call annotate_site_end
end subroutine solve

```

The following code from the C# `nqueens` sample on Windows* OS systems shows the use of parallel site and task C# annotations, such as `Annotate.SiteBegin("label");`. The three added annotations and the line that allows use of the annotation definitions (`using` directive) appear in a **bold** font below.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

```

using AdvisorAnnotate;
...
public void Solve()
{
  int[] queens = new int[size]; //array representing queens on a chess board. Index is row
  position, value is column.
  Annotate.SiteBegin("solve");
  for (int i = 0; i < size; i++)
  {
    Annotate.IterationTask("setQueen");
    // try all positions in first row
    SetQueen(ref queens, 0, i);
  }
  Annotate.SiteEnd();
  ...
}

```

To simplify adding annotations:

- When using the Microsoft Visual Studio* code editor, you can use the Annotation Wizard.
- With any editor, use the annotation assistant in the **Survey** windows or the **No Data** message. The annotation assistant displays example annotated code and build settings that you can copy to your application's code.

If you manually type annotations, you should place each annotation on a separate line and use the correct data type for annotation arguments. With C/C++ code, do not place annotations in macros so that references go to the correct source location.

You can experiment by modifying annotations and running the tools again to locate the best places to add parallelism.

For each source compilation module that contains annotations, in addition to adding the annotations, you need to:

- In files where you add annotations, add a source line to reference the Intel Advisor file that defines the annotations:
 - For C/C++ modules, include the `advisor-annotate.h` header file by adding either `#include "advisor-annotate.h"` or `#include <advisor-annotate.h>`.
 - For Fortran compilation units, add the `use advisor_annotate` statement.
 - For C# modules (on Windows* OS), add the `using AdvisorAnnotate;` directive.
- Specify the Intel Advisor include directory when you build your C/C++ or Fortran application, so the compiler can find this include file. Similarly, you need to add the C# annotations definition file to your C# project.
- For native applications, add the build (compiler and linker) settings.

Annotation Types

Annotation Types Summary

You can use different kinds of Intel® Advisor annotations to mark where you propose to have parallel sites, tasks, locks, or perform special actions. These annotations are:

- Parallel site annotations
- Parallel task annotations
- Parallel lock annotations
- Annotations that let you pause and resume data collection
- Special-purpose annotations

To be useful, a parallel site must contain at least one task. Code within a parallel task can be executed by multiple threads independently of other instances of itself and also other parallel tasks. Many tasks are code within a loop, or they could be a single statement that does an iterative operation. After you use the Survey or similar profiling tool to locate where your program spends its time, you will see two general types of parallel code regions (parallel sites):

- **A simple loop that requires only a single task.** For the common case where the Survey tool identifies a simple loop structure whose iterations consume much of an application's CPU time and the entire loop body should be a task, you may only need a single task within a parallel site. Unless your time-consuming code is not in a loop or has task(s) in a complex loop, start with this simple form. Add annotations to mark the beginning and end of the parallel site around the loop, and add one task-iteration annotation at the start of the loop body. This annotation form is the easiest to convert to parallel code.
- **Code whose characteristics require multiple tasks.** Depending on the application code characteristics, you may need multiple tasks. For example, you may have statements that can each become separate tasks, or complex or nested loop structures where you need multiple tasks to meet scalability requirements. In this case, add site annotations to mark the beginning and end of the parallel site region and also task annotations that mark the beginning and end of each task.

The two task annotation types use the same parallel site annotations. The following table lists the annotations by category type, including the syntax for the C/C++, Fortran, and C# languages. Each has a link to its detailed description.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

Optional arguments are identified using square brackets, such as `annotation([int expr])`.

NOTE

To help you add annotations, use the Intel Advisor annotation assistant in the **Survey** windows or the **No Data** message to copy and add code snippets or the Annotation Wizard if you use the Microsoft Visual Studio* code editor (see [Inserting Annotations Using the Annotation Wizard](#)). You also need to add the reference to the annotations definitions file.

Brief Description	Name
Site and task annotations for a parallel site that contains a loop with a single task:	
Start a parallel site that contains a single task in a loop.	C/C++: <code>ANNOTATE_SITE_BEGIN(sitename);</code> Fortran: <code>call annotate_site_begin(sitename)</code> C#: <code>Annotate.SiteBegin(sitename);</code>
Mark an iterative parallel task in a loop. Place this annotation near the start of the loop body within the parallel site's execution.	C/C++: <code>ANNOTATE_ITERATION_TASK(taskname);</code> Fortran: <code>call annotate_iteration_task(taskname)</code> C#: <code>Annotate.IterationTask(taskname);</code>
End a parallel site. The parallel site terminates only after all tasks that started within it have completed.	C/C++: <code>ANNOTATE_SITE_END([sitename]); // sitename is optional</code> Fortran: <code>call annotate_site_end</code> C#: <code>Annotate.SiteEnd();</code>
Site and task annotations for parallel site code that contains multiple tasks (all other situations):	
Start a parallel site that contains multiple tasks, or task(s) within non-loop code or complex loop code.	C/C++: <code>ANNOTATE_SITE_BEGIN(sitename);</code> Fortran: <code>call annotate_site_begin(sitename)</code> C#: <code>Annotate.SiteBegin(sitename);</code>
Start a parallel task. Must execute within a parallel site that contains multiple tasks, or task(s) within non-loop code or complex loop code.	C/C++: <code>ANNOTATE_TASK_BEGIN(taskname);</code> Fortran: <code>call annotate_task_begin(taskname)</code> C#: <code>Annotate.TaskBegin(taskname);</code>
End a parallel task. Must execute within a parallel site that contains multiple tasks, or task(s) within non-loop code or complex loop code.	C/C++: <code>ANNOTATE_TASK_END([taskname]); //taskname is optional</code> Fortran: <code>call annotate_task_end</code> C#: <code>Annotate.TaskEnd();</code>
End a parallel site. The parallel site terminates only after all tasks that started within it have completed.	C/C++: <code>ANNOTATE_SITE_END([sitename]); // sitename is optional</code> Fortran: <code>call annotate_site_end</code> C#: <code>Annotate.SiteEnd();</code>
Lock Annotations: describe synchronization locations.	
Acquire a lock (0 is a valid address). Must occur within a parallel site.	C/C++: <code>ANNOTATE_LOCK_ACQUIRE(pointer-expression);</code> Fortran: <code>call annotate_lock_acquire(address)</code>

Brief Description	Name
Release a lock. Must occur within a parallel site.	C#: <code>Annotate.LockAcquire([int expr]);</code> <code>// this C# argument is optional</code>
	C/C++: <code>ANNOTATE_LOCK_RELEASE(pointer-expression);</code> Fortran: <code>call annotate_lock_release(address)</code> C#: <code>Annotate.LockRelease([int expr]);</code> <code>// this C# argument is optional</code>
Pause Collection and Resume Collection Annotations: lets you pause data collection to skip uninteresting code.	
Pause Collection. The target program continues to execute.	C/C++: <code>ANNOTATE_DISABLE_COLLECTION_PUSH;</code> Fortran: <code>call annotate_disable_collection_push()</code> C#: <code>Annotate.DisableCollectionPush();</code>
Resume Collection after it was stopped by a Pause Collection annotation.	C/C++: <code>ANNOTATE_DISABLE_COLLECTION_POP;</code> Fortran: <code>call annotate_disable_collection_pop()</code> C#: <code>Annotate.DisableCollectionPop();</code>
Special-purpose Annotations: describe certain memory allocations to avoid false conflicts, disable reporting of problems or analysis, or enable reporting more detail for memory accesses. These apply only to the Dependencies tool. For their syntax, see the Special-purpose Annotations help topic.	

See Also

[Intel Advisor Annotation Definitions File](#)

[Site and Task Annotations for Simple Loops With One Task](#)

[Site and Task Annotations for Loops with Multiple Tasks](#)

[Adding Annotations in Your Source Code](#)

[Lock Annotations](#)

[Pause Collection and Resume Collection Annotations](#)

[Special-purpose Annotations](#)

[Annotating Code for Deeper Analysis](#)

[Copying Annotations and Build Settings Using the Annotation Assistant Pane](#)

[Inserting Annotations Using the Annotation Wizard](#)

Annotation General Characteristics

Usage

Annotations typically expand to calls to one or more functions, with minimal additional code. When you run the Suitability or Dependencies tools, the calls are instrumented during data collection.

Most annotations must be used in pairs that will execute in a begin-end sequence, such as the parallel site annotations for a site with a single task:

- For C/C++: `ANNOTATE_SITE_BEGIN(sitename);` and `ANNOTATE_SITE_END();`
- For Fortran: call `annotate_site_begin(sitename)` and call `annotate_site_end`
- For C#: `Annotate.SiteBegin(sitename);` and `Annotate.SiteEnd();`

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

Any mismatched annotations show up as error during data collection.

For example, if your C/C++ code has an `ANNOTATE_SITE_BEGIN();` that is executed, but no corresponding `ANNOTATE_SITE_END();`, you will see a message, such as: Error: Missing end site when you run the Suitability or Dependencies tool.

You can also use annotations when they are dynamically paired. This lets you annotate code regions that might have more than one exit point. For example, consider this parallel site with multiple tasks:

```
//Show that an end task annotation should be repeated for a jump out of a loop
ANNOTATE_SITE_BEGIN(for_site1);
  ANNOTATE_TASK_BEGIN(for_taskA);
  for ()
  {
    if ()
    {
      ANNOTATE_TASK_END();
      break;
      ANNOTATE_TASK_END(); // unreachable!
    }
    ANNOTATE_TASK_BEGIN(for_taskB);
    ...
    ANNOTATE_TASK_END();
  }
ANNOTATE_SITE_END();
```

With C/C++, when you add annotations after a loop that executes only one statement without opening and closing braces (`{` and `}`), add opening and closing braces to allow multi-statement execution of both the original statement and the added annotation statement.

From a program source perspective, the annotation macros expand as a single executable statement (or to nothing if null expansion is used). This allows annotations to be used in locations requiring a single statement safely, as in this example:

```
...
if (!initialized)
    ANNOTATE_RECORD_ALLOCATION(my_buffer, my_buffer_size);
...
```

Guidelines for Placing Annotations in Source Code

Intel Advisor guidelines for placing annotations in source code are similar to debugger breakpoint limitations. The rules include:

- Place each annotation on a separate statement line. That is, do not place multiple annotations in a single statement line.
- With C/C++ code, do not place annotations inside preprocessor macros.

The following shows correct coding using one annotation per statement line:

```
ANNOTATE_TASK_BEGIN(foo);
    call xyz();
ANNOTATE_TASK_END();
```

If you do not follow these guidelines, you may see unexpected Unmatched annotations in the **Dependencies Report** window (see the Troubleshooting topic below) or annotation-related errors in the Suitability Report window.

Semantics

When you run the Suitability or Dependencies tool to collect interactions between your tasks, the execution of annotations and their implications for other operations are tracked by the tool during serial execution, and the results of analysis are displayed in the corresponding Report.

When you run the Dependencies tool, the primary problems of interest are the data interactions that need attention. However, some semantic errors in the use of the annotations in your program may also be reported.

See Also

[Site and Task Annotations for Simple Loops With One Task...](#)

[Dependencies Analysis](#)

[Copying Annotations and Build Settings Using the Annotation Assistant Pane](#)

[Troubleshooting Unexpected Unmatched Annotations in the Dependencies Report](#)

[Fixing Annotation-related Errors Detected by the Suitability Tool](#)

[Inserting Annotations Using the Annotation Wizard](#)

[Data Sharing Problems](#)

Site and Task Annotations for Simple Loops With One Task

Parallel site annotations mark the beginning and end of the parallel site. In contrast, to mark an entire simple loop body as a task, you only need a single iteration task annotation in the common case where the Survey tool identifies a single simple loop that consumes much of an application's time. In many cases, a single time-consuming simple loop structure may be the only task needed within a parallel site. This annotation form is also the easiest to convert to parallel code.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

NOTE

If the task's code does not include the entire loop body, or if you need multiple tasks in one parallel site or for complex loops, use the task begin-end annotation pair to mark each task.

Use the general site/task annotation form for time-consuming code not in a loop, for complex loops containing task(s), or cases that require multiple tasks within a parallel site.

Syntax: Simple Loops With One Task

Parallel site annotations mark the parallel site that wraps the loop:

```
C/C++:      ANNOTATE_SITE_BEGIN(sitename); and ANNOTATE_SITE_END();
```

Fortran:	<code>call annotate_site_begin(sitename) and call annotate_site_end</code>
C#:	<code>Annotate.SiteBegin(sitename); and Annotate.SiteEnd();</code>

The iteration task annotation occurs within the parallel site. Place this annotation near the start of the loop body to mark an entire simple loop body as a task:

C/C++:	<code>ANNOTATE_ITERATION_TASK(taskname);</code>
Fortran:	<code>call annotate_iteration_task(taskname)</code>
C#:	<code>Annotate.IterationTask(taskname);</code>

For the C/C++ `ANNOTATE_SITE_END();` annotation, the `sitename` argument is optional.

The `sitename` and `taskname` must follow the rules for annotation name arguments:

- For C/C++ code, the `sitename` must be an ASCII C++ identifier. This should be a name you will recognize when it appears in Intel Advisor tool reports.
- For Fortran code, the `sitename` must be a character constant. This should be a name you will recognize when it appears in Intel Advisor tool reports.
- For C# code, the `sitename` must be a string. This name should be a string that you will easily remember when it appears in Intel Advisor tool reports.

Examples: Simple Loops With One Task

The following C/C++ code fragment shows a parallel site for a loop with a single task, where the task includes the entire simple loop body:

```
...
ANNOTATE_SITE_BEGIN(sitename);
for (i=0; i<N; i++) {
    ANNOTATE_ITERATION_TASK(taskname);
    func(i);
}
ANNOTATE_SITE_END();
...
```

The following Fortran code fragment shows a parallel site for a loop with a single task, where the task includes the entire simple loop body:

```
...
call annotate_site_begin("sitename")
do i=1,size
    call annotate_iteration_task("taskname")
    call func(i)
end do
call annotate_site_end
...
```

The following C# code fragment shows a parallel site for a loop with a single task, where the task includes the entire simple loop body:

```
...
Annotate.SiteBegin("sitename");
for (int i = 0; i < N; i++) {
    Annotate.IterationTask("taskname");
    func(i);
}
```

```

}
Annotate.SiteEnd();
...

```

With Visual Studio projects, parallel sites may span project boundaries, but the parallel sites and their related annotations should be placed within the set of projects that the startup project depends on. You may need to use the Visual Studio* Project Dependencies context menu item to add appropriate dependencies - see the help topic [Troubleshooting Unexpected Unmatched Annotations](#).

The `nqueens_Advisor` C++ sample and the `nqueens_Fortran` Fortran sample demonstrate this form of site/task annotations. For example, the C++ annotated code in `nqueens_annotated.cpp`:

```

ANNOTATE_SITE_BEGIN(solve);
for(int i=0; i<size; i++) {
    // try all positions in first row
    // create separate array for each recursion
    ANNOTATE_ITERATION_TASK(setQueen);
    // int * queens = new int[size]; //array representing queens placed on a chess ...
    // ADVISOR COMMENT: This is incidental sharing because all the tasks are using ...
    setQueen(queens, 0, i);
}
ANNOTATE_SITE_END();

```

The help topic [Annotating Parallel Sites and Tasks](#) describes adding parallel sites and tasks.

See Also

[Site and Task Annotations with Multiple Tasks](#)

[Annotating Parallel Sites and Tasks](#)

[Dependencies Analysis](#)

[Annotation General Characteristics](#)

[Inserting Annotations Using the Annotation Wizard](#)

[Copying Annotations and Build Settings Using the Annotation Assistant Pane](#)

[Troubleshooting Unexpected Unmatched Annotations](#)

Site and Task Annotations for Parallel Sites with Multiple Tasks

Parallel site annotations mark the beginning and end of the parallel site. Similarly, begin-end parallel task annotations mark the start and end of each task region. Use this begin-end task annotation pair if there are multiple tasks in a parallel site, if the task code does not include all of the loop body, or for complex loops or code that requires specific task begin-end boundaries, including multiple task end annotations.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

Syntax: Parallel Sites with Multiple Tasks

Parallel site annotations that mark the parallel site:

C/C++:	<code>ANNOTATE_SITE_BEGIN(sitename); and ANNOTATE_SITE_END();</code>
Fortran:	<code>call annotate_site_begin(sitename) and call annotate_site_end</code>
C#:	<code>Annotate.SiteBegin(sitename); and Annotate.SiteEnd();</code>

Parallel task annotations that mark each task within the parallel site:

C/C++:	<code>ANNOTATE_TASK_BEGIN(taskname); and ANNOTATE_TASK_END();</code>
--------	--

Fortran:	call annotate_task_begin(taskname) and call annotate_task_end
C#:	Annotate.TaskBegin(taskname); and Annotate.TaskEnd();

For the C/C++ `ANNOTATE_TASK_END()`; annotation, the `taskname` argument is optional.

The `taskname` must follow the rules for annotation name arguments:

- For C/C++ code, the `taskname` must be an ASCII C++ identifier. This should be a name you will recognize when it appears in Intel Advisor tool reports.
- For Fortran code, the `taskname` must be a character constant. This should be a name you will recognize when it appears in Intel Advisor tool reports.
- For C# code, the `taskname` must be a string. This name should be a string that you will easily remember when it appears in Intel Advisor tool reports.

If you previously used site and task annotations for simple loops with one task and need to convert the task to this general, multiple task form, replace the single iteration loop annotation with a pair of task begin and task end annotations that mark the task region. Both forms use the same parallel site annotations.

Examples: Parallel Site, Multiple Tasks Not in a Loop

The `stats` C++ sample application shows task parallelism with multiple tasks that are in a parallel site but not in a loop. In this case, several related statements do a lot of computation work and each can be a separate task:

```
ANNOTATE_SITE_BEGIN(MySite1);
cout << "Start calculating running average..."<<endl;
ANNOTATE_TASK_BEGIN(MyTask1);
runningAvg(vals, SIZE, rnAvg);
ANNOTATE_TASK_END(MyTask1);

cout << "Start calculating running standard deviation..."<<endl;
ANNOTATE_TASK_BEGIN(MyTask2);
runningStdDev(vals, SIZE, rnStdDev);
ANNOTATE_TASK_END(MyTask2);
ANNOTATE_SITE_END(MySite1);
```

In addition to calling functions that perform the computations, there are other cases where the Survey tool may indicate that a single statement consumes a lot of CPU time. For example, a Fortran array assignment for a very large array.

Examples: Parallel Site, Multiple Tasks Within a Loop

The annotations in the following C/C++ code fragment specify that each iteration of the loop can be two separate tasks, potentially running in parallel with any other iteration and the other task.

```
...
ANNOTATE_SITE_BEGIN(sitename);
for (I=0; i<N; I++) {
    ANNOTATE_TASK_BEGIN(task1);
    func1(I);
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(task2);
    func2(I);
    ANNOTATE_TASK_END();
}
ANNOTATE_SITE_END();
...
```


The following Fortran code fragment also shows the Fortran site and task annotations, where each iteration of the loop can be two separate tasks, potentially running in parallel with any other iteration and the other task.

```
...
call annotate_site_begin("sitename ")
do i=1,size
  call annotate_task_begin("task1")
  call func1(i)
  call annotate_task_end
  call annotate_task_begin("task2")
  call func2(i)
  call annotate_task_end
end do
call annotate_site_end
...
```

The following C# code fragment also shows the C# site and task annotations, where each iteration of the loop can be two separate tasks, potentially running in parallel with any other iteration and the other task.

```
...
Annotate.SiteBegin("sitename");
for (int i = 0; i < N; i++) {
  Annotate.TaskBegin("task1");
  func1(i);
  Annotate.TaskEnd();
  Annotate.TaskBegin("task2");
  func2(i);
  Annotate.TaskEnd();
}
Annotate.SiteEnd();
...
```

The code for each task will be marked between *task begin* and *task end* annotation pairs inside a parallel site. Code that is not executed in any task is executed by the thread entering the site, which may run in parallel with the identified tasks. In this example, the loop control code that increments *i* and the compares *i* with *N* is assumed to be executed separately from the explicitly specified tasks. This means that you may see conflicts between tasks, and the code outside of any task.

When you use the Dependencies tool on the above code, the tool would report data conflicts on global data accessed by either *func1* or *func2* on a later loop iteration.

The help topic [Annotating Parallel Sites and Tasks](#) describes adding parallel sites and tasks.

Parallel Site and Task Placement

Consider the following C/C++ code:

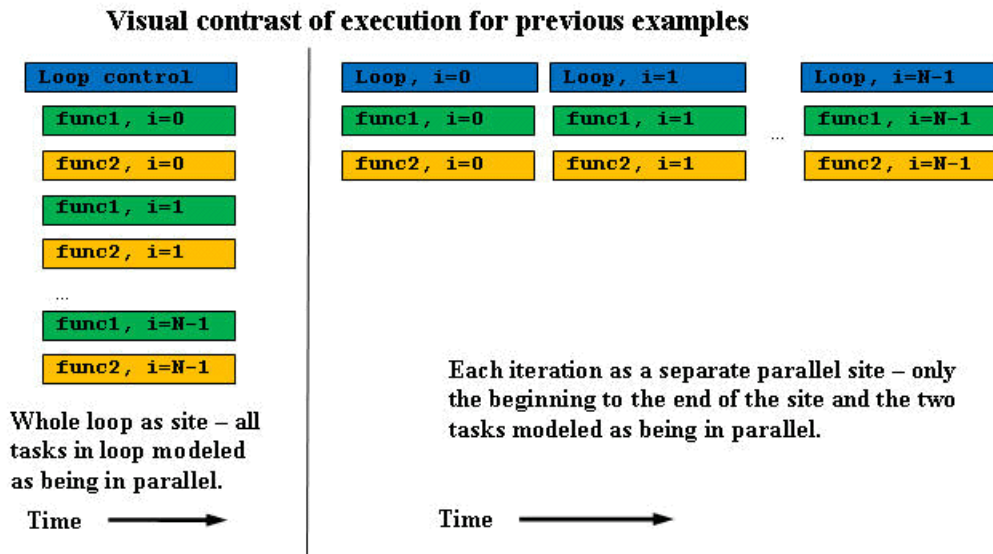
```
...
ANNOTATE_SITE_BEGIN(sitename);
for (i=0; i<N; i++) {
  ANNOTATE_ITERATION_TASK(taskname);
  func(i);
}
ANNOTATE_SITE_END();
...
```

```
...
for (i=0; i<N; i++) {
  ANNOTATE_SITE_BEGIN(sitename);
  ANNOTATE_TASK_BEGIN(taskfunc1);
  func1(i);
  ANNOTATE_TASK_END();
  ANNOTATE_TASK_BEGIN(taskfunc2);
  func2(i);
  ANNOTATE_TASK_END();
  ANNOTATE_SITE_END();
}
...
```

In the simple case on the left, the single annotated site encapsulates the entire loop. This causes all of the iterations of the loop to potentially run all at the same time. Use this simple form of loop annotations (two site annotations and one iteration task annotation) for loops whenever possible.

In the case on the right, you are not specifying that all of the loop iterations will run in parallel, but rather that the opportunities for parallelism are only within a single iteration of the loop. In this case, only the invocations of `func1` and `func2` from one loop iteration at a time are considered as sources of potential parallelism. So, in the case on the right, you will never see conflicts between successive invocations of `func1`, because you are specifying that you do not intend to run them in parallel.

Graphically comparing what the model considers to be in parallel for these two cases, with time progressing from left to right for each case:



The boxes shown overlapping vertically above are modeled as being executed in parallel.

The execution of `ANNOTATE_TASK_BEGIN(taskname)` and `ANNOTATE_TASK_END()` pair delimits the dynamic extent of a task. Each time the annotations are executed during Intel Advisor Dependencies or Suitability analysis to collect interactions between tasks, a dynamic extent is identified that is associated with the most closely containing dynamic site. Each task is assumed to be independent and able to be run in parallel with all other tasks inside the containing sites.

Task annotations in a multiple-task parallel site must use the following rules:

- According to execution paths, each begin task annotation must be terminated by an end task annotation.
- Task boundaries must be within parallel site boundaries.
- The argument to the task annotations follow the rules for annotation name arguments.

The only times tasks are not modeled to be executing in parallel are:

1. When tasks are using synchronization, the specific code inside the synchronized region will not be modeled to be in parallel with other code synchronized using the same lock addresses.
2. When one task creates another task, the code of the parent task executed before the second task is created is assumed to execute before the task creation. However, any code executed after the task creation is assumed to be in parallel with the nested task. For example:

```
...
ANNOTATE_SITE_BEGIN(sitename);
for (I=0; i<N; I++) {
    ANNOTATE_TASK_BEGIN(taskfunc1a);
    func1a(I);
    ANNOTATE_TASK_END(taskfunc1a);
}
```

```

    func2(I);
    ANNOTATE_TASK_END();
    func1b(I);
    ANNOTATE_TASK_END();
}
ANNOTATE_SITE_END();
...

```

In this example, `func1a(I)` is not in parallel with either `func2(I)` or `func1b(I)`. However, `func2(I)` and `func1b(I)` are modeled as being executed in parallel. This semantic interpretation allows modeling of recursion where nested calls create tasks that execute in parallel. In this example, note that while this parallel relationship holds for tasks inside one iteration, tasks from different loop iterations will all be in parallel because they have no special relationship. For example, `func1a(I)` from one loop iteration may be executed concurrently with `func2(I)` in a different iteration.

While you are checking Dependencies, the Dependencies tool assumes that all tasks in a given site may execute in parallel unless there is explicit synchronization. For example, in this case all `N` iterations of `func1` and `func2` will execute in parallel.

```

...
ANNOTATE_SITE_BEGIN(sitename);
for (I=0; i<N; I++) {
    ANNOTATE_TASK_BEGIN(taskfunc1);
    func1(I);
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(taskfunc2);
    func2(I);
    ANNOTATE_TASK_END();
}
ANNOTATE_SITE_END();
...

```

If you want to model other kinds of relationships, for example `func2` invocations will have some form of serialization, that constraint needs to be expressed using lock annotations that mark a lock that is acquired and released for the duration of that task's execution.

To select where to add task annotations may take some experimentation, considering factors such as average instance time and number of iterations (provided in the Suitability Report). If your parallel site has nested loops and the computation time used by the innermost loop is small, consider adding task annotations around the next outermost loop. See help topics such as [How Big Should a Task Be?](#).

See Also

[Lock Annotations](#)

[Inserting Annotations Using the Annotation Wizard](#)

[Copying Annotations and Build Settings Using the Annotation Assistant Pane](#)

[Annotating Parallel Sites and Tasks](#)

[How Big Should a Task Be?](#)

[Dependencies Analysis](#)

[Fixing Sharing Problems](#)

Lock Annotations

Lock annotations mark where you expect you will be adding explicit synchronization.

Syntax

C/C++:	ANNOTATE_LOCK_ACQUIRE(pointer-expression); and ANNOTATE_LOCK_RELEASE(pointer-expression);
Fortran:	call annotate_lock_acquire(address) and call annotate_lock_release(address)
C#:	Annotate.LockAcquire([int expr]); and Annotate.LockRelease([int expr]); (for each annotation, its argument is optional)

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

With C/C++ and Fortran programs, all of the lock annotations use an address value to represent distinct locks in your final program. You can use the address value 0 to represent a global “lock” that is the same across the entire program. With C# programs, the argument is an `int` with a default value of 0 (zero).

Intel recommends that you start by using a default lock, unless you need additional locks for performance scaling.

The modeling step is aware of the standard locking routines in the Windows* OS API, as well as Intel® oneAPI Threading Building Blocks (oneTBB) and OpenMP*, so there is no need to annotate existing locking. Lock annotations are only required for cases where you are not already using synchronization.

The lock-acquire and lock-release annotations denote points in your program where you intend to acquire and release locks. These annotations take a single parameter, which is an address that you choose.

For example, if you decided you would have a lock used only for *glob_variable*, you specify the same memory address for all cases where you are protecting access to *glob_variable*, to represent that specific lock. The sample below uses the variable's address to represent the lock that will be associated with *glob_variable*.

You typically can use one of the following four values, using a finer granularity of synchronization when necessary:

- The value of 0 (zero) to represent a single unspecified lock that is the same across the entire program.
- The address of a data structure or other aggregation of data. This represents using a single lock for the collection of data.
- The address of a member of the data collection. This represents finer-grained locking than the previous value and provides better performance.
- A variable representing a lock as you move toward final parallel code.

This C/C++ example shows the intent for the parallel program to acquire and release a lock around the access to the global variable *glob_variable* in each task:

```
...
extern int glob_variable = 0;
...
ANNOTATE_SITE_BEGIN(sitename);
for (I=0; i<N; I++) {
    ANNOTATE_TASK_BEGIN(taskfunc1);
    func1(I);
    ANNOTATE_LOCK_ACQUIRE(&glob_variable);
    glob_variable++;
    ANNOTATE_LOCK_RELEASE(&glob_variable);
    func2(I);
    ANNOTATE_TASK_END();
}
ANNOTATE_SITE_END();
...
```

This Fortran example also shows the intent to acquire and release a lock around the access to the global variable *glob_variable* in each task:

```
...
integer :: glob_variable = 0

call annotate_site_begin("sitename")
do i=1,size
  call annotate_task_begin("taskfunc1")
  call func1(i)
  call annotate_lock_acquire(0)
  glob_variable = glob_variable + 1
  call annotate_lock_release(0)
  call func2(i)
  call annotate_task_end
end do
call annotate_site_end
...
```

This C# example also shows the intent to acquire and release a lock around the access to the global variable *glob_variable* in each task:

```
...
public int glob_variable {
    get{return nrOfSolutions;}
    set{nrOfSolutions = value;}
}

Annotate.SiteBegin("sitename");
for (int i = 0; i < N; i++) {
    Annotate.TaskBegin("taskfunc1");
    func1(i);
    Annotate.LockAcquire();
    glob_variable++;
    Annotate.LockRelease();
    func2(i);
    Annotate.TaskEnd();
}
Annotate.SiteEnd();
...
```

The following C/C++ example is a typical use of a data item's address. It shows the use of an *Entity* address, where there is a vector of integers that are each going to have an associated lock, because the program is counting random elements of the array that will be accessed by different tasks, some of which may occasionally have the same random value. The text from adding annotations appears in **bold** below.

```
struct Entity {
    int val;
};
...
std::vector<Entity> v;
...

for (int I=0; i<v.size()*10000; I++) {
    int random_int = random_n();
    ANNOTATE_LOCK_ACQUIRE(&v[random_int]);
    v[random_int].val++;
    ANNOTATE_LOCK_RELEASE(&v[random_int]);
}
...
```

Use Lock Annotations

Lock addresses are the basis of lock annotations, and each lock address corresponds to the intent to create a unique lock, or other synchronization mechanism, in the final program. Tasks sharing a parallel site are modeled as executing in parallel unless you describe synchronization using lock addresses, or known locking mechanisms.

See Also

[Special-purpose Annotations](#)

[Synchronize Independent Updates](#)

[Data Sharing Problems](#)

[Insert Annotations Using the Annotation Wizard](#)

[Copy Annotations and Build Settings Using the Annotation Assistant Pane](#)

Pause Collection and Resume Collection Annotations

The Pause Collection and Resume Collection annotations let you stop and resume data collection to skip uninteresting parts of the target program's execution. If you pause data collection, the target executable continues to execute until you resume data collection. Pausing data collection minimizes the amount of data collected and speeds up the analysis of large applications.

In addition to these annotations, you can click certain buttons on the side command toolbar to pause or resume data collection:

- You can start the Survey and Suitability tools with data collection either paused or enabled. For example, the **Start Paused** button starts executing the target being analyzed with data collection (analysis) disabled. Also, once the tool is started, you can pause and resume data collection by using the **Pause** or **Resume** buttons or by executing the equivalent Pause Collection and Resume Collection annotations.
- You start the Dependencies tool with data collection enabled, but you can pause data collection either by using the **Pause** button or by executing the equivalent Pause Collection annotation. You can add Pause Collection and Resume Collection annotations as described below.

Pause Collection

This annotation completely stops the analysis of your program until the matching Resume Collection (disable-collection-pop) annotation is executed. Use this annotation to reduce the analysis overhead for certain uninteresting parts of your program. This annotation is recognized by the Dependencies, Survey, and Suitability tools. Because this annotation completely disables monitoring of most annotations, add it carefully in your source code, such as outside a parallel site. If there are multiple pushes, all have to be popped before re-enabling collection.

Syntax:

C/C++:	<code>ANNOTATE_DISABLE_COLLECTION_PUSH;</code>
Fortran:	<code>call annotate_disable_collection_push()</code>
C#:	<code>Annotate.DisableCollectionPush();</code>

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

This annotation takes no arguments.

NOTE

For C/C++, this annotation does not have an argument list.

Resume Collection

This annotation resumes the analysis previously stopped by a Pause Collection (disable-collection-push) annotation. This annotation is recognized by the Dependencies, Survey, and Suitability tools. Because the Pause Collection annotation completely disables monitoring of most annotations, add this Resume Collection annotation carefully in your source code, such as outside a parallel site.

Syntax:

C/C++:	<code>ANNOTATE_DISABLE_COLLECTION_POP;</code>
Fortran:	<code>call annotate_disable_collection_pop()</code>
C#:	<code>Annotate.DisableCollectionPop();</code>

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

This annotation takes no arguments.

Special-purpose Annotations

All Intel Advisor special-purpose annotations are recognized by the Dependencies tool, which observes memory accesses in great detail. Some of these annotations prevent the Dependencies tool from reporting all or specific data sharing problems, while one (Observe Uses of Storage) provides more detail about memory accesses.

NOTE

In the C/C++ syntax descriptions below, `addresses` and `sizes` are C++ expressions. Similarly, the Fortran `var` is a Fortran integer address.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

This topic describes the following special-purpose annotations:

- [Inductive Expressions Uses](#)
- [Reduction Uses](#)
- [Observe Uses of Storage](#)
- [Clear Uses of Storage](#)
- [Disable Observation Annotations](#)
- [Enable Observation Annotations](#)
- [Memory Allocation Annotations](#)

Inductive Expressions Uses

Induction variables (such as `++i`) can often be eliminated when you add parallel framework code. Use this annotation to disable reporting data sharing problems for the specified memory region. This annotation is only recognized by the Dependencies tool.

Terminate this annotation with a [Clear Uses of Storage](#) annotation.

Syntax:

C/C++:	<code>ANNOTATE_INDUCTION_USES(address, size);</code>
Fortran:	<code>call annotate_induction_uses(var)</code>
C#:	Not supported

- `address` is a C++ identifier or expression that provides information about the memory region for this annotation.
- `size` is a C++ identifier or expression that provides information about the memory region for this annotation.
- `var` is a Fortran integer address that provides information about the memory region for this annotation.

Reduction Uses

Reduction variables (such as `sum += data[i]`) can often be replaced with reduction operations when you add parallel framework code. Use this annotation to disable reporting data sharing problems for the specified memory region. This annotation is only recognized by the Dependencies tool.

Terminate this annotation with a [Clear Uses of Storage](#) annotation. For example, with C/C++ code:

```
ANNOTATE_REDUCTION_USES(&sum, 4);
    sum += a[i];
ANNOTATE_CLEAR_USES(&sum);
```

Syntax:

C/C++:	<code>ANNOTATE_REDUCTION_USES(address, size);</code>
Fortran:	<code>call annotate_reduction_uses(var)</code>
C#:	Not supported

- `address` is a C++ identifier or expression that provides information about the memory region location for this annotation.
- `size` is a C++ identifier or expression that provides information about the memory region location for this annotation.
- `var` is a Fortran integer address that provides information about the memory region for this annotation.

Observe Uses of Storage

Use this annotation to report all accesses to the specified memory region. For example, this can help you find all of the uses of a variable to determine how you should refactor your code. This annotation gets reported as a `Memory watch` remark message in the Dependencies Report. This annotation is only recognized by the Dependencies tool.

NOTE

For performance reasons, this annotation may not report memory access for variables stored on the stack.

To terminate this annotation, add a [Clear Uses of Storage](#) annotation.

Syntax:

C/C++:	<code>ANNOTATE_OBSERVE_USES(address, size);</code>
Fortran:	<code>call annotate_observe_uses(var)</code>
C#:	Not supported

- `address` is a C++ expression that provides information about the memory region location for this annotation.
- `size` is a C++ expression that provides information about the memory region location for this annotation.

- `var` is a Fortran integer address that provides information about the memory region for this annotation.

Clear Uses of Storage

Use this annotation to terminate these annotations: Inductive Expressions Uses, Reduction Uses, and Observe Uses of Storage. For example, when the C/C++ `ANNOTATE_CLEAR_USES()`; annotation terminates `ANNOTATE_OBSERVE_USES()`; , the Dependencies tool stops reporting all uses of the specified variable. This annotation is only recognized by the Dependencies tool.

Syntax:

C/C++:	<code>ANNOTATE_CLEAR_USES(address);</code>
Fortran:	<code>call annotate_clear_uses(var)</code>
C#:	Not supported

- `address` is a C++ identifier or expression that provides information about the memory region location for this annotation.
- `var` is a Fortran integer address that provides information about the memory region for this annotation.

Disable Observation Annotations

This annotation disables the reporting of problems until the matching Enable Observation Annotation is executed. After executing this annotation, the Dependencies tool does not report problems but continues to monitor other annotations so it can resume reporting problems if a matching Enable Observation Annotation is executed. This can be useful to suppress Dependencies problems that are false-positives or not useful in your program. Unlike `ANNOTATE_CLEAR_USES`; - which applies to a specific memory area - this annotation remains active until a `disable-observation-pop` annotation is executed to enable annotations. This annotation is only recognized by the Dependencies tool.

Syntax:

C/C++:	<code>ANNOTATE_DISABLE_OBSERVATION_PUSH;</code>
Fortran:	<code>call annotate_disable_observation_push()</code>
C#:	<code>DisableObservationPush();</code>

This annotation takes no arguments.

Enable Observation Annotations

This annotation enables the reporting of Dependencies stopped by a previous Disable Observation Annotation was executed to disable observation annotations. This annotation is only recognized by the Dependencies tool.

Syntax:

C/C++:	<code>ANNOTATE_DISABLE_OBSERVATION_POP;</code>
Fortran:	<code>call annotate_disable_observation_pop()</code>
C#:	<code>Annotate.DisableObservationPop();</code>

This annotation takes no arguments.

Memory Allocation Annotations

Memory allocation annotations apply only to C/C++ programs. They describe non-standard or user-defined memory allocations to avoid false conflicts reported by the Dependencies tool. Only use these Memory allocation annotations if you see false conflicts related to memory allocation in the Dependencies tool. This annotation is only recognized by the Dependencies tool.

Heap-allocated memory can be freed and then reused. If the same memory region is allocated during one task, then freed, and then re-allocated for use by a second task, this can confuse Dependencies tool analysis, because it appears as if two threads were accessing the same parallel memory region without synchronization. When the program runs in parallel runs in parallel, each thread could allocate different memory, so there is not really a data race.

The Dependencies tool understands the standard library memory allocation routines, such as `malloc` and `free`, operator `new`, and so on. However, if you have a user-defined memory allocator, the Dependencies tool may not accurately understand the memory relationships between different tasks. If your application utilizes a user-defined memory allocator, you may need to use these annotations to help the Dependencies tool understand the relationships. You place:

- `ANNOTATE_RECORD_ALLOCATION` *after* a call to your non-standard or user-defined allocator.
- `ANNOTATE_RECORD_DEALLOCATION` *before* the call to your non-standard or user-defined deallocator.

If you do not have such an allocator you can skip these annotations.

If you do have a user-defined memory allocator and you omit these annotations, you may see the effects as **Memory reuse** problems for the storage that is actually allocated by your allocator, and **Data communication** problems for the control information used by the allocator.

Syntax:

C/C++:	<code>ANNOTATE_RECORD_ALLOCATION(address, size);</code> and <code>ANNOTATE_RECORD_DEALLOCATION(address);</code>
Fortran:	Not supported
C#:	Not supported

`ANNOTATE_RECORD_ALLOCATION(address, size);` specifies the storage allocated by a user-memory allocator with a specific `address` and `size`:

1. The `address` is a C++ expression that provides information about the memory region location for this annotation.
2. The `size` is a C++ expression that provides information about the memory region size for this annotation.

Use `ANNOTATE_RECORD_DEALLOCATION(address);` each time your deallocator is freeing memory.

Static Loop Scheduling Annotations

Loop scheduling annotation inform the Suitability tool that the following loop will be divided into equal-sized (or as equal as possible) chunks. By default, chunk size is `loop_count/number_of_threads`.

Syntax:

C/C++:	<code>ANNOTATE_AGGREGATE_TASK;</code>
Fortran:	Not supported
C#:	Not supported

See Also

[Tips for Annotation Use with C/C++ Programs](#)

Pause Collection and Resume Collection Annotations

Annotation General Characteristics

Annotation Types Summary

Inserting Annotations Using the Annotation Wizard

Copying Annotations and Build Settings Using the Annotation Assistant Pane

Annotation Definitions Files

Intel® Advisor provides macro or routine definitions that enable use of its annotations for each language:

- For C/C++, the `advisor-annotate.h` header file defines macros that begin with `ANNOTATE_`, so you can use annotations such as `ANNOTATE_SITE_BEGIN()` ;.
- For Fortran, the `advisor_annotate` module declares subroutines starting with `annotate_`, so you can call annotations such as `annotate_site_begin()`.
- For C# on Windows* OS systems, the `AdvisorAnnotate` header declares an `Annotate` class containing member routines, so you can use annotations such as `Annotate.SiteBegin()` ;.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

Reference the Annotation Definitions from Your Source Files

Before you add Intel® Advisor annotations into your source files, you need to reference the definitions for the Intel® Advisor annotations:

- For C/C++, add: `#include "advisor-annotate.h" or #include <advisor-annotate.h>` (see Including the Annotations Header File in C/C++ Sources).
- For Fortran, add: use `advisor_annotate`
- For C#, add: using `AdvisorAnnotate`; (Windows OS systems only)

Where to Add USE Statements in Fortran Programs

Fortran does not have file scope declarations, so the `USE` statement needs to be inside the subroutine, function or main program where the annotation(s) appear. For example:

```
program F_example

! The main program does not contain annotations, do not add use advisor_annotate here!
! some code . . .
!

subroutine F_sub
! This subroutine contains annotations, so add the use advisor_annotate statement
use advisor_annotate
! some code . . .
! add Intel Advisor site and task annotations around compute intensive code
! For example, begin a parallel site: call annotate_site_begin(site1)
!
end subroutine F_sub
! some code . . .
end program F_example
```

If the call is in a module procedure, the `USE` statement can be at the module level. For more details about placing `USE` statements, see your Fortran compiler documentation.

Specify Build Settings

Specific build settings are needed for each language. Certain build settings are needed for each module that contains Intel® Advisor annotations, such as specifying the directory where the annotations definitions are located. For C/C++ and Fortran applications, other build (compiler and linker) settings are needed for all modules in an application, such as full debug information. Read the Build Settings... topics by clicking the links below under See Also for your language.

Redistribute the Annotations Definition File(s)

You only need annotations in your code when you are using the Intel® Advisor Suitability and Dependencies tools to predict your serial program's parallel behavior. Before you distribute your application, you will typically replace these annotations when you add the parallel framework code. However, because the annotations do not change how your applications runs unless you use Intel® Advisor tools, you can distribute your application with the annotations still present.

For information about redistributing the annotation definition files, see the installed End User License Agreement (EULA.rtf or EULA.txt) and the `redist.txt` file installed in the Intel® Advisor.../documentation/<locale> directory.

Special Considerations for C/C++ Applications

With C/C++ programs:

- If your program encounters errors when you include the `advisor-annotate.h` file, see [Handling Compilation Issues that Appear After Adding advisor-annotate.h](#) (primarily for Windows systems).
- **On Windows OS systems:** If you do need to modify the `advisor-annotate.h` file, you can add a copy of it for a specific project or solution. If the version of `advisor-annotate.h` changes, you will need to update your copies of the file. See Adding a Copy of the Annotations Include File to Your Visual Studio Project.

If you do not need to modify this file, you can reference the same installed `advisor-annotate.h` from multiple projects or solutions as a read-only file. If you use the Intel® Advisor environment variable and the version of Intel® Advisor `advisor-annotate.h` changes, you only need to change this reference if the environment variable name changes, such as for a major version. Thus, using a read-only version can minimize future maintenance.

- **On Linux* OS systems:** Except in very rare circumstances, you can reference the same installed `advisor-annotate.h` from multiple projects or solutions as a read-only file.
- Since the annotations do not change the values computed by your program, you can change the expansions of the macro, or suppress expansion altogether, as described in Controlling the Expansions in `advisor-annotate.h`.

Reference the Annotations Definitions Directory

You need to specify the directory containing the Intel® Advisor definition file as an additional include directory when you compile your program. Intel® Advisor installs its annotation definition files into a default directory on your system. For example:

- With a Visual Studio project or solution for a C/C++ or Fortran application, you need to specify the property **Additional Include path**. You can use the environment variable `ADVISOR_<version>_DIR` followed by the `include` directory.
- With the C/C++ or Fortran command line, use the compiler option `-Idir` (Linux* OS) or `/Idir` (Windows* OS), where `dir` is the directory containing the annotation definition files. You can use the environment variable `ADVISOR_<version>_DIR` followed by the `include` directory.
- With Fortran modules, you also need to specify the library name and directory of the annotations definitions to the linker.
- With a Visual Studio project or solution for a C# program, you need to specify **Properties > Add > Existing Item** and browse to and select the annotations definitions file.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

Tip

For the most current information on optimal C/C++ and Fortran build settings, see [Build Your Target Application](#).

Add a Copy of the C/C++ Annotation Definition File to Your Visual Studio Project*

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

If you do not want to refer to the installed C/C++ annotation include header file, you can reference a solution- or project-specific copy of it.

To add a project-specific annotations include file to your Visual Studio project:

1. In Solution Explorer, right-click the project where you want to create the Intel® Advisor annotation header file.
2. Click **Add > New Item...** The Add New Item dialog box opens.
3. Under **Installed Templates**, click **Intel Advisor [version]**.
4. In the middle column, click **advisor-annotate**.
5. Type a file name for this include file, such as `advisor-annotate.h` for the C/C++ header file.
6. Verify the directory containing the solution- or project-specific header file and click **Add**.

In Solution Explorer, a copy of the header file appears as a file under the project folder.

See Also

[Including the Annotations Header File in C/C++ Sources](#)

[Inserting Annotations Using the Annotation Wizard](#)

Include the Annotations Header File in C/C++ Sources

When you add annotations to your C/C++ source files, you also need to include the Intel® Advisor annotation header file `advisor-annotate.h` in those files. Use the code editor to type the line or use the context menu item to add a `#include` directive.

To include the annotations C/C++ header file, specify one of the following forms listed below:

Use the quoted form to have the preprocessor first search for the header file in the same directory as the source file that contains the <code>#include</code> directive, and then other directories (see your compiler documentation for details).	<code>#include "advisor-annotate.h"</code>
Use the angle bracket form to have the preprocessor first search for the header file in the directory specified by the <code>/I</code> option (Additional Include Directories), and then other directories (see your compiler documentation for details).	<code>#include <advisor-annotate.h></code>

To use the include file with Fortran sources, see [Intel® Advisor Annotation Definitions File](#).

See Also

[Insert Annotations Using the Annotation Wizard](#)

[Set Intel Advisor Environment Variables](#) Use this topic to get guidance on setting up environment variables for Intel® Advisor.

Add Annotations into Your Source Code

You can add Intel® Advisor annotations in your source code by:

- Copying annotations with the [annotation assistant](#) in the **Survey Report** window, **Survey Source** window, or the **No Data** message. Use the annotation assistant to copy the main annotations for parallel sites, tasks, and locks. For example, the annotation assistant appears in the lower part of the **Survey Report** window in the **Assistance** tab.
- **On Windows* OS only:** When using the Visual Studio* code editor, you can use the [Annotation Wizard](#) to select an annotation type and add the annotations and their arguments into your code. You can use the Annotation Wizard to add parallel site, task, lock, pause/resume data collection, and special-purpose annotations.

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

NOTE

If your sources include huge source files that contain annotations, be aware that only the first 8 MB of each file will be parsed for annotations. If not all of your annotations are being parsed in such huge source files, consider breaking each huge source file into several source files.

Insert Annotations Using the Annotation Wizard

Adding annotations requires you to reference the annotation definitions include file as well as include it from each source file that contains Intel® Advisor annotations.

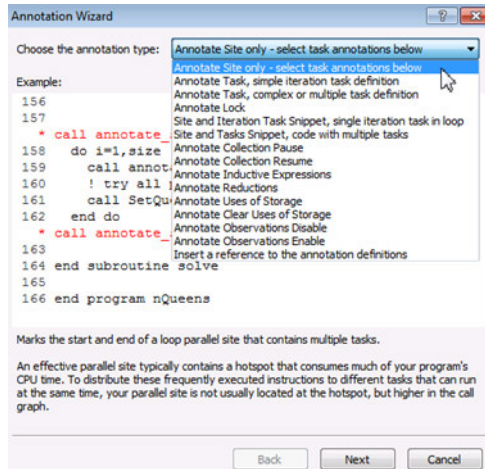
NOTE

The Annotation Wizard is supported only in the Microsoft Visual Studio* code editor. Alternatively, you can copy annotation code snippets using any editor.

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

Use the annotation wizard to add Intel® Advisor annotations to your program.

1. In the Visual Studio* editor, select the code section that you wish to annotate.
2. Right-click to open the context menu and select **Intel Advisor [version]> Annotation Wizard...** The Annotation Wizard opens with the default **Annotate Site – select task annotations below** annotation selected:



- From the **Choose the Annotation Type** drop-down, choose the appropriate annotation type. For example, if you want to add the parallel site (parallel code region) annotations and a single task annotation within that site, start by selecting the site code block and choose **Site and Iteration Task Snippet, single iteration task in loop**. In other cases, you may need to add two separate annotations - one for the site and one for the task(s). In this case, after adding them, move individual annotation lines around your existing site and task(s) code. Your code appears in the **Example** section of the dialog, with the annotation line(s) highlighted in red font.
- Click **Next** to configure the parameters of the opening annotation line. For Annotation types that include parameters, page 2 of the wizard appears. Site, task, and other annotations take name arguments. You should replace the added name with a name that helps you quickly identify its source location. For example, if `MySite1` is the argument to a site annotation, replace it with a meaningful function or loop name. The added name must be unique amongst the annotations in this project. For Annotation types that do not include parameters, go to step 8.
- Specify the parameter values for the first parameter in the Annotation type, or use the default text that appears in the wizard. The highlighted annotation line now has your specified parameter value entered in the annotation line.
- Click **Next** to configure, or keep the default text for the next parameter. Repeat for all the parameters. The highlighted annotation line now has all parameters filled in with values entered in the annotation line.
- Click **Next** to go to page 3 of the wizard and review the annotation line(s) before adding it to you code.
- Click **Finish** to add the annotation line with your specified parameters to your code. The Wizard closes and the editor shows the annotation lines added to the code.

The annotation line(s) are added in the code editor.

If a loop only executes a single statement and does not contain an opening brace ({) to allow multi-statement execution, add braces ({ and }) around the existing statement and the annotation.

Code After Adding a Pair of Parallel Site Annotations

The following screen capture shows the C/C++ annotation lines for an annotation of type **Annotate Site**, where the site name `MySite1` parameter was replaced by typing a meaningful name `queens`:

```

void setQueen(int queens[], int row, int col) {
    //check all previously placed rows for attacks

    //ADVISOR COMMENT: The accesses to the "queens" array in this function create an :
    //ADVISOR COMMENT: Each task should have its own copy of the "queens" array
    //ADVISOR COMMENT: Look at the solve() function to see how to fix this issue
    ANNOTATE_SITE_BEGIN(queens);
    for(int i=0; i<row; i++) {
        // vertical attacks
        if (queens[i]==col) {
            return;
        }
        // diagonal attacks
        if (abs(queens[i]-col) == (row-i) ) {
            return;
        }
    }
    ANNOTATE_SITE_END();
    // column is ok, set the queen
    //ADVISOR COMMENT: See comment at top of function
    queens[row]=col;
}

```

Annotation Wizard - Page 1

To access this dialog box: in the Visual Studio code editor, right-click to open the context menu, and select **> Annotation Wizard**.

The **Annotation Wizard** helps you add annotations into your code. After you are done adding annotations, rebuild your program.

Use page 1 of the wizard to select the type of annotation from the **Choose the annotation type** drop down list.

The wizard shows the annotation line(s) that will be added into your code, in red font.

- For annotations that have start and end lines, it adds them around your selected code lines with placeholders for the parameters.
- For annotations that have only one line, it adds that annotation line before your selected code line, with placeholders for the parameters.

After choosing the annotation type, click **Next** to go to the next wizard page and fill in the parameters for the annotation.

See Also

[Annotation Wizard - Page 2](#)

[Annotation Wizard - Page 3](#)

Annotation Wizard - Page 2

The **Annotation Wizard** helps you add annotations into your code. After you are done adding annotations, rebuild your target executable.

Use page 2 of the wizard to define the annotation parameters. Replace the placeholder text in the **Specify annotation parameter** field with the parameters you want to define for the annotation, or keep the default text. For example, for an **Annotation Site** annotation, replace the `<site_name>` placeholder text with a meaningful site name.

The wizard shows your parameters within the annotation line(s).

See Also

- [Annotation Wizard - Page 1](#)
- [Annotation Wizard - Page 3](#)

Annotation Wizard - Page 3

The **Annotation Wizard** helps you add annotations into your code. After you are done adding annotations, rebuild your program.






Use page 3 of the wizard to check the annotation line(s) that you defined using the wizard and verify that the line(s) and insertion location(s) are correct:

- If you are satisfied with the annotation line(s), click **Finish** to add the line(s) to your code.
- To revise the annotation, click **Back** and revise the annotation type, or the parameters defining the annotation.
- To change the location of the added lines, click **Cancel**, select a different range of code lines and right-click and choose **Intel Advisor [version]> Annotation Wizard** to add annotation lines around the new selection.

Copy Annotations and Build Settings Using the Annotation Assistant Pane

The Intel® Advisor provides an annotation assistant near the bottom of the **Survey Report** and **Survey Source** windows, as well as with the **No Data** message. Use this assistant to view and copy selected annotated code snippets and build setting information into a code editor.

The assistant provides a drop-down list under **Example:** from which you select one of the following:

Select This	To Do This
Iteration Loop, Single Task	<p>View and copy an annotation code snippet for a simple loop structure, where the task's code includes the entire loop body. Use this common task structure when only a single task is needed within a parallel site. For example code, see the help topic Site and Task Annotations for Simple Loops With One Task.</p> <p>Click the  Copy to Clipboard button to copy the selected snippet to the clipboard.</p>
Loop, One or More Tasks (bounded)	<p>View and copy an annotation code snippet for a loops where the task code does not include all of the loop body, or for complex loops or code that requires specific task begin-end boundaries, including multiple task end annotations. Also use this structure when multiple tasks are needed within a parallel site. For example code, see the help topic Site and Task Annotations for Parallel Sites with Multiple Tasks.</p> <p>Click the  Copy to Clipboard button to copy the selected snippet to the clipboard.</p>
Function, One or More Tasks (bounded)	<p>View and copy an annotation code snippet for code that calls multiple functions (task parallelism). Use this structure when multiple tasks are needed within a parallel site. For example code, see the help topic Site and Task Annotations for Parallel Sites with Multiple Tasks.</p> <p>Click the  Copy to Clipboard button to copy the selected snippet to the clipboard.</p>
Pause/Resume Collection	<p>View and copy an annotation code snippet whose annotations temporarily pause data collection and later resume it. This lets you skip uninteresting parts of the target program's execution to minimize the data collected and speed up the analysis of large applications. Add these annotations outside a parallel site.</p> <p>Click the  Copy to Clipboard button to copy the selected snippet to the clipboard.</p>
Build Settings	<p>View and copy build settings. The Build Settings are specific to the language in use.</p> <p>Click the  Copy to Clipboard button to copy the selected snippet to the clipboard.</p>

Site, task, and other annotations take name arguments. You should replace the placeholder name with a name that helps you quickly identify its source location. For example, in place of `MySite5` in the argument to a site annotation, replace it with a meaningful function or loop name. The name you add must be unique amongst the annotations in this project.

Insert Annotations in the Visual Studio* Code Editor

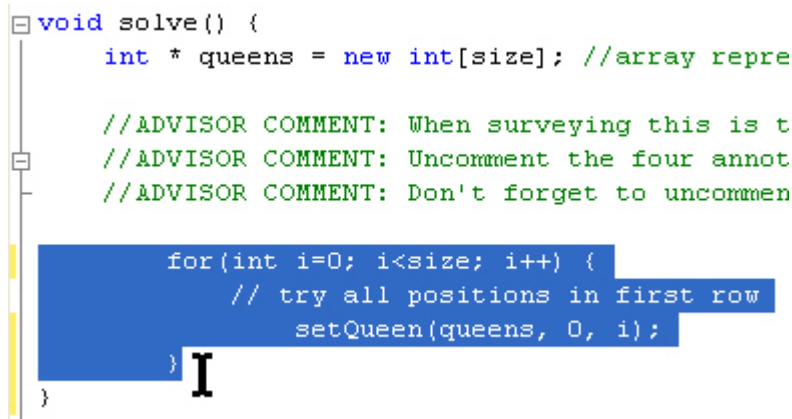
NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

To add Intel® Advisor annotations into your source files, you can use the Visual Studio* code editor. Intel® Advisor simplifies the process of adding annotations so you do not need to type the annotation names. Alternatively, you can use the annotation assistant in the Survey Report or Survey Source windows, or when using the Visual Studio code editor, the Annotation Wizard (on Windows* OS systems). Alternatively, you can type the exact macro name and its arguments manually.

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

To add Intel® Advisor annotations:

1. Open the source file into which you want to add Intel® Advisor annotations in your code editor. You should start with the outermost code regions, such as a parallel site, and then add the tasks within the boundaries of the enclosing site.
2. Select the code region around which you will add your first annotation, such as a parallel site. Carefully include the correct group of lines, including any opening and closing braces ({ and }) needed. For example:



```
void solve() {
    int * queens = new int[size]; //array repre

    //ADVISOR COMMENT: When surveying this is t
    //ADVISOR COMMENT: Uncomment the four annot
    //ADVISOR COMMENT: Don't forget to uncommen

    for(int i=0; i<size; i++) {
        // try all positions in first row
        setQueen(queens, 0, i);
    }
}
```

3. Within the highlighted code region, right-click the mouse to display the context menu. Select **Intel Advisor [version]** and the type of annotation to be added, such as **Intel Advisor [version]> Annotate Site**. For example:
4. This adds the selected type of annotations. For the begin site annotation, Intel® Advisor adds a unique annotation identifier as an argument.

```

        ANNOTATE_SITE_BEGIN(MySite1);
        for(int i=0; i<size; i++) {
            // try all positions in first row
            setQueen(queens, 0, i);
        }
        ANNOTATE_SITE_END(MySite1);
    }

```

You should replace the added name with a name that helps you quickly identify its source location. For example, in place of `MySite1` in the argument to `ANNOTATE_SITE_BEGIN()` and `ANNOTATE_SITE_END()` shown above, you might instead type the word `solve` (the function name). The added name must be unique amongst the annotations in this project. Annotation name arguments for:

- C/C++ code use an ASCII C++ identifier.
- Fortran code use a character constant.
- C# code use a string (Windows OS only)

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

Choose a string that you will easily remember when it appears in Intel Advisor tool reports. Other annotations use address or size arguments.

5. To add more annotations in the same file, repeat this process from step 2. To add annotations in a different file, repeat this process from step 1.

This enables you to quickly add annotations into the appropriate source files.

This wizard provides only the more frequently used annotations, so some annotations are not available in this wizard. Either use the Survey windows' annotation assistant to copy other annotations or type the annotations into your code editor.

If a C/C++ loop only executes a single statement and does not contain an opening brace ({) to allow multi-statement execution, add braces ({ and }) around the existing statement and the annotation.

See Also

[Annotation General Characteristics](#)

[Inserting Annotations Using the Annotation Wizard](#)

[Copying Annotations and Build Settings Using the Annotation Assistant Pane](#)

[Annotation Types Summary](#)

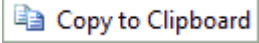
Insert Annotations in a Text Editor

To add Intel® Advisor annotations into your source files on a Linux* system, you can use any text editor. Intel® Advisor simplifies the process of locating where to add annotations.

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

To add Intel® Advisor annotations:

1. Open the Intel® Advisor GUI and the relevant project.
2. In the **File** menu, select **Options**.
3. Select **Editor** to display the **Options - Editor** dialog box. Follow the instructions to associate an editor with your source language(s).

4. For your project, open the **Survey Source** window.
5. Double-click a source line to display the specified editor opened to the corresponding source location.
6. Use the annotation assistant pane in the lower part of the **Survey Source** window to select the type of annotation you want to add.
7. Copy the selected annotations from the annotation assistant pane by clicking the  button.
8. Paste the copied annotations into your editor.
9. You may need to move some annotation lines around.
10. Repeat as needed for other annotations from step 4.

This enables you to quickly add annotations into the appropriate source files.

See Also

[Annotation General Characteristics](#)

[Annotation Types Summary](#)

[Copying Annotations and Build Settings Using the Annotation Assistant Pane](#)

Tips for Annotation Use with C/C++ Programs

The following topics provide tips related to using annotations with C/C++ programs:

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

- Depending on your particular environment, you may want to control the expansion of macros in `advisor-annotate.h` by using the `ANNOTATE_EXPAND_NULL` environment variable. See the help topic [Controlling the Expansion of advisor-annotate.h](#).
- Tips for Windows* OS only:
 - Because the `advisor-annotate.h` header file includes `windows.h`, including `advisor-annotate.h` may cause type and symbols conflicts, which result in unexpected compiler messages. See the help topic [Handling Compilation Issues that Appear After Adding advisor-annotate.h](#).
 - If you run into certain unexpected problems, you need to learn how `advisor-annotate.h` and `libittnotify.dll` interact. See the help topic [advisor-annotate.h and libittnotify.dll](#).

Control the Expansion of `advisor-annotate.h`

Depending on your particular environment, you may want to control the expansion of C/C++ macros in `advisor-annotate.h` at the inclusion site.

Defining `ANNOTATE_EXPAND_NULL` before you include `advisor-annotate.h` causes the annotation macros to have a null expansion, which will remove the actions from your code. If you have a project where some configurations, or some users, want to have annotations, while others should not have them present, this control may be helpful.

```
#define ANNOTATE_EXPAND_NULL
#include "advisor-annotate.h"
```

You can also do this by defining the value as part of your compilation command using the `/D` option. For example:

```
/DANNOTATE_EXPAND_NULL
```

See Also

[Handling Compilation Issues that Appear After Adding `advisor-annotate.h`](#)

[Set Up Project](#)

[Including the Annotations Header File in C/C++ Sources](#)

Handle Compilation Issues that Appear After Adding `advisor-annotate.h`

NOTE

This topic primarily applies to Windows systems. It is possible for similar errors to occur on Linux systems.

Symptoms

On Windows* systems, the `advisor-annotate.h` header file includes `windows.h` to define some types and functions. As a result, in some cases including `advisor-annotate.h` may cause compilation errors. For example, the following conflict for the type `UINT`:

```
error C2371: 'UINT' : redefinition; different basic types
```

On Linux systems, something similar could occur under certain very specific conditions when using a different header file for operating system threading software.

Possible Correction Strategies

To fix this problem, you can use a declaration/definition approach, where all uses of `advisor-annotate.h` other than one generate a set of declarations, and `windows.h` is only needed in a single implementation module. In all cases, you `#define` either `ANNOTATE_DECLARE` or `ANNOTATE_DEFINE` just *before* the `#include "advisor-annotate.h"` as follows:

1. In nearly all modules that contain annotations, insert `#define ANNOTATE_DECLARE` just before `#include "advisor-annotate.h"`. This causes `advisor-annotate.h` to declare an external function, and not include `windows.h` (or Linux equivalent), which avoids the type/symbol conflicts with the operating system threading header file, such as `windows.h`.
2. In a single module that either does not include annotations or does not have type/symbol conflicts with `windows.h`, you insert `#define ANNOTATE_DEFINE` just before `#include "advisor-annotate.h"`. This causes `advisor-annotate.h` to define the global function to resolve the external reference and the `#include "advisor-annotate.h"` is the only one that uses the operating system threading header file `windows.h` (or Linux equivalent). These two lines can be placed in an otherwise empty `.cpp` file.

One way to do this is to add an empty `.cpp` to your project with two lines in it, shown as `empty.cpp` below.

For example, on Windows systems:

```
//File foo.cpp/.h:
...
// Insert #define ANNOTATE_DECLARE in all modules that contain annotations just before the
// #include "advisor-annotate.h". This prevents inclusion of windows.h to avoid the
// type/symbol conflicts.

#define ANNOTATE_DECLARE
#include "advisor-annotate.h"
...
// annotation uses
ANNOTATE_SITE_BEGIN(MySite1)
```

```
...  
    ANNOTATE_SITE_END()  
...
```

```
//File empty.cpp:  
// Insert #define ANNOTATE_DEFINE just before the #include "advisor-annotate.h" in only one  
module.  
// This single implementation file (.cpp/.cxx) causes windows.h to be included, and the  
support  
// routine is defined as a global routine called from the various annotation uses.  
#define ANNOTATE_DEFINE  
#include "advisor-annotate.h"  
...
```

If the problem persists, please request support, such as by using the support forum.

advisor-annotate.h and libittnotify.dll

NOTE

This topic is provided for reference, but it should not be needed. If you read this because you believe you need it to understand a problem, please provide feedback (see the release notes for support information).

Code using `advisor-annotate.h` should work when running your application regardless of whether or not you are running your application under Visual Studio using Intel® Advisor on Windows* OS. However, should you run into problems, this topic provides a few implementation details that might be helpful to understand issues.

Each compilation unit that includes `advisor-annotate.h`, and that has one or more annotations in it, will have a global inline routine named `__AnnotateRoutine`. This routine is called from the various locations where you have used the `ANNOTATE*` macros, and will be used to invoke one or more routines in `libittnotify.dll` (on Windows OS) or `libittnotify.so` (on Linux* OS).

`__AnnotateRoutine` is an inline function that will have only one copy per executable in your program. So, if you are only working on modeling semantic behavior in a single Dynamic Link Library (DLL), you will only have one copy of it in that one library. If you have multiple executables where you have annotations, there will be a single copy of `__AnnotateRoutine` for each executable.

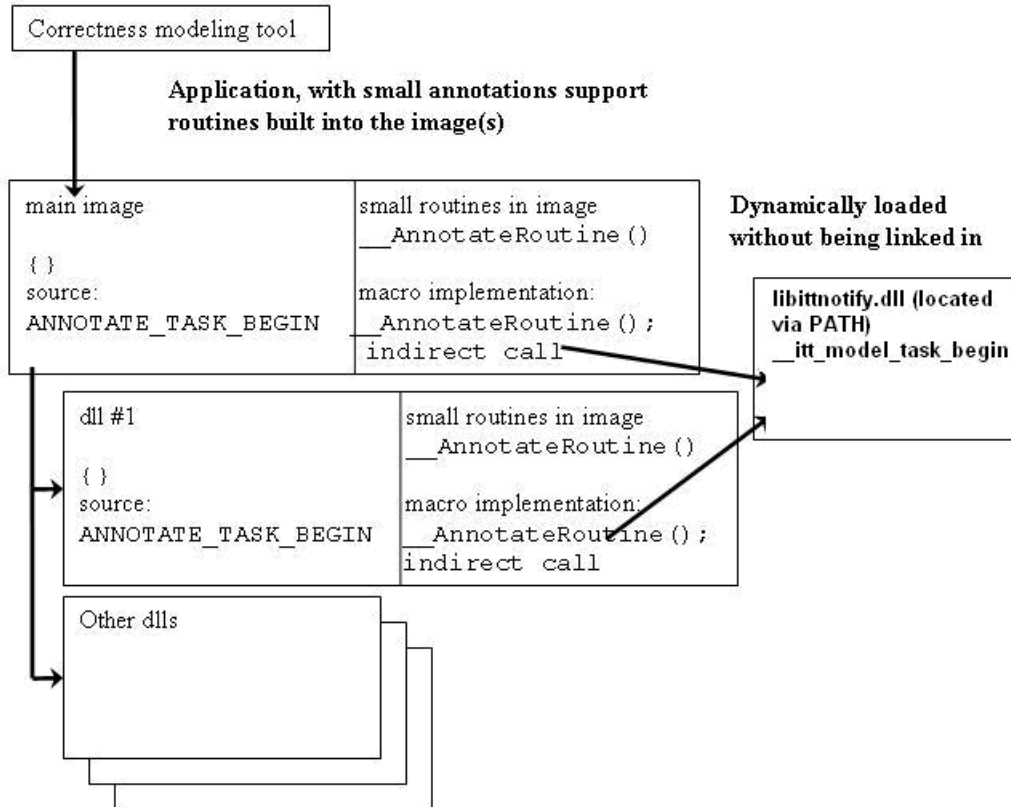
When the `ANNOTATE` macros are used, the first one executed in a given executable attempts to load `libittnotify.dll` from the current path.

Once the library is loaded, calls from the annotations will go to the `__itt_model*` routines in the library. If an expected routine is not found, the code asserts.

The following figure shows what happens when you have a main executable and a DLL that both have annotations:

Implementation relationship between images, ANNOTATE* macros, __AnnotateRoutine and __itt_model* routines

Modeling tool runs program




Finally, the annotation routines - __itt_model* in the .dll - by themselves are only markers for a tool that interprets the calls. Unless the program is run under the tool, these routines will not do anything. The intent is that the application will run normally when it is not run under the tool.

See Also

[Set Up Project](#)

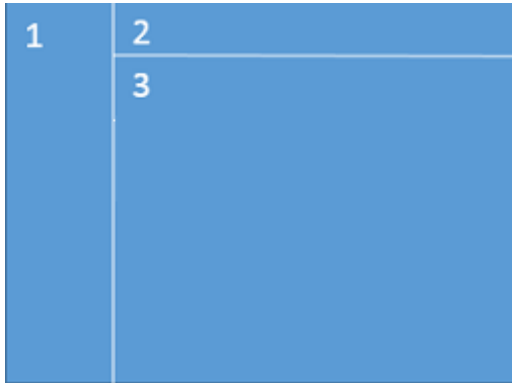
Annotation Report

To access this window, in the Result tab, click the **Annotation Report** button. Alternatively, if you are using the **Advisor Workflow** tab, click the  button below **2. Annotate Sources** or **5. Add Parallel Framework**.

The **Annotation Report** window lists all annotations found during source scanning or running the Suitability and Dependencies tools. It lists the annotation type, source location, and annotations label in a table-like grid format, where each annotation appears on a separate row. Intel® Advisor updates the listed annotations when changes occur to the specified source directories. For example, when you save a source file with a code editor.




Annotation Report Layout

1. Analysis Workflow Tab
2. Result Tab
3. Annotation Report window grid



Use This	To Do This
Analysis Workflow tab	Run a tool of your choice and see results in the Result tab.
Result Tab	Select between available reports.
Annotation Report window grid	View a summary of the annotations found as well as data collected by the Suitability and Dependencies tools. Each annotation's data appears on a separate row in the grid. The columns are explained below.
Right-click a row in the Annotation Report window grid	Displays a context menu that lets you expand or collapse code snippets, edit corresponding source code using a code editor, copy data to the clipboard, or display context-sensitive help.

To sort the grid using a column's values, click on the column's heading. The columns of the grid are:

Use This Column	To Do This
Annotation	<p>View the type of annotation, such as Site, Task, or Lock.</p> <p>To show or hide a code snippet showing the annotation, click the  icon next to its name.</p> <p>For information about each annotation type, see the help topic Summary of Annotation Types.</p> <p>To view the source associated with an annotation in your code editor, double-click its name or a line in the code snippet (or right-click and select Edit Source from the context menu) in this column.</p> <ul style="list-style-type: none"> On Windows* OS: <ul style="list-style-type: none"> When using Visual Studio, the Visual Studio code editor appears with the file open at the corresponding location. When using the Intel® Advisor GUI, the file type association (or Open With dialog box) determines the editor used. On Linux* OS: When using the Intel® Advisor GUI, the editor defined by the Options > Editor dialog box appears with the file open at the corresponding location.
Source Location	<p>View the name of the source file that contains the annotation and the line number.</p> <p>Icons indicate where source is available  or not available .</p> <p>To view the source, double-click its name (or right-click and select Edit Source) in this column. The code editor appears.</p>
Annotation Label	View the annotation's label (name).

Use This Column	To Do This
	To view the source associated with an annotation, double-click its name (or right-click and select Edit Source) in this column. The code editor appears.


When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Clear Description of Storage Row

Use this special-purpose annotation to stop tracking references to a memory location by the **Dependencies** tool. This information can help you understand what code accesses a memory location. When you have learned enough, simply remove this annotation.

To view the source code for this annotation, click the  icon.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Disable Observations in Region Row

This special-purpose annotation disables the reporting of problems until the matching enable annotation `ANNOTATE_DISABLE_OBSERVATION_POP`; is executed. Use this annotation to suppress reported problems that are false-positives, or not useful in you.

To view the source code for this annotation, click the  icon.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Pause Collection Row

This special-purpose annotation temporarily stops (pauses) the analysis of your program's execution until the matching Resume Collection annotation (disable-collection-pop) is executed. Use this annotation to reduce the tool analysis overhead and reported data for certain parts of your program while running the Dependencies, Survey, and Suitability tools.

To view the source code for this annotation, click the  icon.


When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Inductive Expression Row

This special-purpose annotation marks a line that updates an expression that is inductive in a loop.

To view the source code for this annotation, click the  icon.

Inductive expressions cause dependence cycles which normally prevent parallelizing a loop, but it is possible to compute the value of the expression if you know the iteration number. You may have to re-write the inductive expression to compute the value based on the iteration number when the loop is translated to parallel code.

For example, if `i++` is the iteration variable of your loop, the parallel framework that you use may automatically fix this for you. For example, by using `cilk_for`. Otherwise, you may need to fix it manually. A common example is with `j+=3`, and `i++`. If `i` is your loop index (assuming 0 based), you can replace `j+=3` with `j = i*3`. That is, the value of `j` actually is a function of the value of `i`.


When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Lock Row

A lock row shows the source location of the lock annotation and its argument value.

To view the source code for this lock annotation, click the  icon.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Observe Uses Row

Use this special-purpose annotation to report the access operations to a memory location in the **Dependencies Report**. This information can help you understand what code accesses a memory location. When you have learned enough, remove the annotation from your source code.

To view the source code for this annotation, click the  icon.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Reduction Row

This special-purpose annotation marks a line that computes a reduction in a loop. Marking the line as a reduction causes the Dependencies tool to ignore the data race.

To view the source code for this annotation, click the  icon.

Reductions require special treatment when translating to parallel code (see the help topics [Special-purpose Annotations](#) and [About Replacing Annotations ...](#) for your parallel framework below).

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Re-enable Observations at End of Region Row

This special-purpose annotation enables reporting problems stopped by a previous `ANNOTATE_DISABLE_OBSERVATION_PUSH;` annotation.

To view the source code for this annotation, click the  icon.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Resume Collection Row

This special-purpose annotation resumes the analysis previously stopped by a previous `Pause Collection (disable-collection-push)` annotation. This annotation is recognized by the Dependencies, Survey, and Suitability tools.

To view the source code for this annotation, click the  icon.


When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Site Row

A site row shows the source location of the site annotation and the label of the site.

To view the source code for this site annotation, click the  icon.

When converting annotations to parallel code:

- For Intel® oneAPI Threading Building Blocks (oneTBB), you need to add a scheduler initialization call in each thread before you create any tasks.
- For OpenMP*, it depends on the pragmas/directives used.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, Task Row

A task row shows the source location of the task annotation and the label of the task. A task identifies time-consuming code whose work can be efficiently done by multiple cores.

To view the source code for this task annotation, click the  icon.

When the task is translated to parallel code and you remove or comment out the task annotation(s), this entry is removed from the table.

There are two types of task annotations. If the loop code changes, you can modify the type of task annotation.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, User Memory Allocator Use Row

This row shows a source location where memory is being allocated using a non-standard or user-defined memory deallocation. The Dependencies tool uses this as a hint about the lifetime of memory accesses, so memory that is allocated will not cause conflicts to be reported if the non-standard or user-defined memory allocation occurs within the span of this annotation's execution.

To view the source code for this annotation, click the  icon.

When translating annotations to parallel code, this special-purpose `record_allocation` annotation can be removed or commented out.


When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Annotation Report, User Memory Deallocator Use Row

This row shows a source location where memory is being freed using a non-standard or user-defined memory deallocation. The Dependencies tool uses this as a hint about the lifetime of memory accesses, so memory that is freed and then allocated again will not cause conflicts to be reported if the non-standard or user-defined memory free occurs with the span of this annotation's execution.

To view the source code for this annotation, click the  icon.

When translating annotations to parallel code, this special-purpose record_deallocation annotation can be removed or commented out.

When to View the Annotation Report

Use the **Annotation Report** window to view the types of annotations present in your project sources and access their locations. You can view the **Annotation Report** at any time to check the annotations. In addition:

- Intel recommends that you view annotations during [Adding Parallelism to Your Program](#) workflow step to help you locate site, task, lock, and other annotations that should be replaced by parallel framework code.
- When using the Intel Advisor product GUI, you can use this report to verify that the correct sources have been defined in the **Project Properties > Source Search** tab.

Explore Threading Results

Intel® Advisor provides several ways to work with the Threading results.

View Results in CLI

If you run the Threading perspective from command line, you can print the results collected in the CLI and save them to a .txt, .csv, or .xml file.

For example, to generate the Suitability report for the OpenMP* threading model:

```
advisor --report=suitability --project-dir=./advi_results --threading-model=openmp
```

You should see a similar result:

```
Target CPU Count: 8      Threading Model: OpenMP*
Maximum gain for all sites: 6.10998

All Sites
Site Label      Source Location      Impact to      Total Serial Time      Total Parallel Time
Site Gain      Average Serial Time ...
Gain                                                    Program
Gain                                                    ...

...
solve      nqueens_serial.cpp:154      6.11x      4.080s      0.631s
6.47x      4.080s ...
```

Site Details					
Annotation Instance	Annotation Average	Label Serial	Source Location	Number of Instances	Maximum
Time	Time		...		
...					
Selected Site		solve	nqueens_serial.cpp:154	1	
4.080s	4.080s		...		
Task		setQueen	nqueens_serial.cpp:156	14	
0.477s	0.267s		...		
Lock				?	365596
0.001s	< 0.001s		...		<
Site Options					
Site	Option	Done?	Benefit If Done	Loss If Not Done	Recommended
solve	Reduce Site Overhead				No
solve	Reduce Task Overhead				No
solve	Reduce Lock Overhead				No
solve	Reduce Lock Contention		0.16x		No
solve	Enable Task Chunking				No

The result is also saved into a text file `advisor-suitability.txt` located at `./advi_results/e<NNN>/st<NNN>`.

You can generate a report for any analysis you run. The *generic* report command looks as follows:

```
advisor --report=<analysis-type> --project-dir=<project-dir> --format=<format>
```

where:

- `<analysis-type>` is the analysis you want to generate the results for. For example, `survey` for the Survey report, `suitability` for the Suitability report, or `dependencies` for the Dependencies report.
- `--format=<format>` is a file format to save the results to. `<format>` is `text` (default), `csv`, `xml`.

If you generate the Suitability report, you can use additional options to control the result view:

- `--target-system=[cpu | xeon-phi | offload-to-xeon-phi]` is a platform to model parallelization on.
- `--threading-model=[tbb | cilk | openmp | tpl | other]` is a threading model to use.
- `--reduce-site-overhead=<string>` is a list of annotated loops/functions to check if you can reduce overhead.

You can also generate a report with the data from all analyses run and save it to a CSV file with the `--report=joined` action as follows:

```
advisor --report=joined --report-output=<path-to-csv>
```

where `--report-output=<path-to-csv>` is a path and a name for a `.csv` file to save the report to. For example, `/home/report.csv`. This option is required to generate a joined report.

See [advisor Command Option Reference](#) for more options.

View Results in GUI

If you run the Threading perspective from command line, a project is created automatically in the directory specified with `--project-dir`. All the collected results and analysis configurations are stored in the `.advixeproj` project, that you can view in the Intel Advisor.

To open the project in GUI, you can run the following command:

```
advisor-gui <project-dir>
```

NOTE If the report does not open, click **Show Result** on the Welcome pane.

If you run the Threading perspective from GUI, the result is opened automatically after the collection finishes.

You first see a Summary report that includes the overall information about loops/functions performance in your code and the annotated parallel sites:

- Performance metrics of your program and top five time-consuming loops/functions
- Optimization recommendations for the whole application
- Estimated performance gain for annotated loops/functions when parallelized

Summary

Survey & Roofline

Refinement Reports

Annotation Report

Suitability Report

Threading Perspective

Threading Perspective lets you analyze, design, tune, and check threading options without disrupting your development.

Program Metrics

Elapsed Time5.39s

Number of CPU Threads1

Vector Instruction SetNone

Performance Characteristics

Metrics	Total		
Total CPU time	5.00s	<div></div>	100%
Time in scalar code	5.00s	<div></div>	100%

> Vectorization Gain/Efficiency (Not Available)

Per Program Recommendations

Higher instruction set architecture (ISA) available

Consider recompiling your application using a higher ISA. [Show more](#)

Top Time-Consuming Loops

Consider adding parallel site and task annotations around these time-consuming loops found during Survey analysis.

Loop	Self Time	Total Time	Trip Counts
loop in setQueen at nqueens_serial.cpp:132	0.449s	4.996s	14
loop in solve at nqueens_serial.cpp:156	<0.001s	4.996s	14
loop in setQueen at nqueens_serial.cpp:103	1.934s	1.934s	4

Suitability And Dependencies Analysis Data

These annotated parallel sites were detected:

Site Location	Maximum Site Gain	Dependencies
loop in solve at nqueens_serial.cpp:154	6.4685634666754614	WAR:1 WAW:1

Recommendations


[align_loop_title](#) loop in [setQueen](#) at [nqueens_serial.cpp:103](#)

[align_loop_title](#) loop in [setQueen](#) at [nqueens_serial.cpp:132](#)

Save a Read-only Snapshot

A snapshot is a read-only copy of a project result, which you can view at any time using the Intel Advisor GUI. You can save a snapshot for a project using Intel Advisor GUI or CLI.



To save an active project result as a read-only snapshot from GUI: Click the  button in the top ribbon of the report. In the **Create a Result Snapshot** dialog box, enter the snapshot details and save it.

To save an active project result as a read-only snapshot from CLI:

```
advisor --snapshot --project-dir=<project-dir> [--cache-sources] [--cache-binaries] --<snapshot-path>
```

where:

- `--cache-sources` is an option to add application source code to the snapshot.
- `--cache-binaries` is an option to add application binaries to the snapshot.
- `<snapshot-path>` is a path and a name for the snapshot. For example, if you specify `/tmp/new_snapshot`, a snapshot is saved in a `tmp` directory as `new_snapshot.advixeexpz`. You can skip this and save the snapshot to a current directory as `snapshotXXX.advixeexpz`.

To open the result snapshot in the Intel Advisor GUI, you can run the following command:

```
advisor-gui <snapshot-path>
```

You can visually compare the saved snapshot against the current active result or other snapshot results.

See [Create a Read-only Result Snapshot](#) for details.

Result Interpretation

When you run the Threading perspective from GUI, you can examine the results and try different threading designs:

- [Model Threading Parallelism](#)
- [Check for Dependencies Issues](#)
- [Add Parallelism to Your Program](#)

See Also

[Run Threading Perspective from GUI](#) Steps to run the Threading perspective.

[Run Threading Perspective from Command Line](#)

[CPU Metrics](#) This reference section describes the contents of data columns in **Survey** and **Refinement Reports** of the Vectorization and Code Insights, CPU / Memory Roofline Insights, and Threading perspectives.

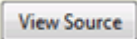




Model Threading Parallelism

The Suitability analysis examines your running serial program to provide approximate estimated performance characteristics of your annotated parallel sites. This shows you both the performance gain from running your parallel program on multiple CPUs and the likely impact of parallel overhead.

To choose the best places to add parallelism, locate the parallel sites that contribute the most to the overall program's gain. Because of the overhead of parallel execution - such as starting threads - certain parallel sites and tasks may not contribute to the overall program's gain, or may slow down its performance. After you identify such parallel sites or tasks that do not improve performance, either modify or eliminate their annotations.

Use the Suitability Report Window

After you run the Suitability tool, view its data in the **Suitability Report** window. This window contains multiple areas:

Location in Window	Description
Upper	<p>Any annotation-related error the Suitability tool detects appears at the top of the Suitability Report window. If you see such errors, the displayed Suitability data may not be reliable. To view the source location associated with an error, click the  button. To fix the error, read the displayed error message, modify your source code to fix the problem, rebuild your target executable, and run Suitability tool analysis again.</p>
Upper-left	<p>The upper-left area shows the Maximum Program Gain for All Sites in the program. Your overall goal of adding parallelism is to increase the Maximum Program Gain for All Sites so the parallel program will execute as fast as possible. The measured serial execution runtime, predicted parallel runtime, and any measured paused time are displayed below Maximum Program Gain for All Sites. Use the predicted Suitability gain values to help you make informed decisions about where to add parallelism.</p>
Upper-right	<p>Use the upper-right row of modeling parameters to model performance. Choose a hardware configuration and threading model (parallel framework) values from the drop-down lists. If you select a Target System for Intel® Xeon Phi™ processors, an additional value for total Coprocessor Threads appears.</p> <p>Below this row is a grid of data that shows the estimated performance of each parallel site detected during program execution. The Site Label shows the argument to the site annotation. Examine the predicted Site Gain and Impact to Program Gain (higher values are better) to estimate how much each site contributes to the Maximum Program Gain for All Sites for all sites (described above). To expand the data under Combined Site Metrics or Site Instance Metrics, click the  icon to the right of that heading; to collapse data, click  to the right of that heading.</p> <p>To show or hide the side command toolbar, click the  or  icon.</p>
Middle-left	<p>If you choose a Target System of CPU, to view detailed characteristics of the selected site as well as its tasks and locks, click the Site Details tab.</p> <p>The Scalability of Maximum Site Gain graph summarizes performance for the selected site. The number of CPU processors or total number of coprocessor threads appears on the horizontal X axis and the target's predicted performance gain appears on the Y axis. To change the default CPU Count and the Maximum CPU Count, set the Options value.</p>
Lower-left	<p>Below the graph is a list of issues that might be preventing better <i>predicted</i> performance gains as well as a summary of serial and predicted parallel time. To expand a line, click the down arrow to the right of the item's name. Most issues are related to the Runtime Modeling modeling parameters. Later, you can use other Analyzer tools like Intel® VTune™ Profiler to measure <i>actual</i> performance of your parallel program.</p>
Lower-middle	<p>Use the Loop Iterations (Tasks) Modeling (or Tasks Modeling) modeling parameters to experiment with different loop structures, iteration counts, and instance durations that might improve the predicted parallel performance.</p> <p>Click Apply to view the impact on the predicted performance.</p>

Location in Window	Description
Lower-right	Use the Runtime Modeling modeling parameters to learn which parallel overhead categories might have an impact on parallel overhead. If you agree to address a category later by using the chosen parallel framework's capabilities or by tuning the parallel code after you have implemented parallelism, check that category.
Bottom-right	If the chosen Target System is Intel Xeon Phi or Offload to Intel Xeon Phi , additional Intel® Xeon Phi™ Advanced Modeling options appear below the Runtime Modeling area. To expand this area, click the down arrow to the right of Intel Xeon Phi Advanced Modeling .
Lower, after clicking Site Details tab	If you chose a Target System of CPU , the Site Details tab shows details about the selected parallel site, as well as details for each task and lock executed in that site.

When using an active result (not a read-only result), you can change the modeling parameters. Changing modeling parameters updates the displayed data, except for **Loop Iterations (Tasks) Modeling** or **Tasks Modeling** (click **Apply**). These modeling parameters help you understand the sensitivity of your annotation choices so you can choose the best places to add parallelism, but the displayed data summary is not an accurate estimate of final execution time on any specific parallel hardware (general processor characteristics are used).

Later, before you add parallel code, you must choose one parallel framework (threading model) for your application.

To view the source code associated with a site, locate the list of sites (upper-right area) and either:

- Double-click a row (or right-click and select **View Source** from the context menu) to display the **Suitability Source** window. Later, to return to the **Suitability Report** window, click **Suitability Report**.
- Right-click a row and select **Edit Source** from the context menu to display the corresponding source file in a code editor. When using the Intel® Advisor GUI on Linux* OS, the editor defined by the **Options > Editor dialog box** Options > Editor dialog box appears with the file open at the corresponding location. When using the Intel® Advisor GUI on Windows* OS, the file type association (or **Open With** dialog box) determines the editor used. When using Microsoft Visual Studio*, the Visual Studio code editor appears with the file open at the corresponding location. Later, to return to the **Suitability Report** or **Suitability Source** window:

1. Click the **Result** tab.
2. Click either **Suitability Report** or **Suitability Source**.

Use the Suitability Source Window

Within the **Suitability Source** window, you can:

- Use the **Call Stack** pane to view different source locations in the call stack.
- Double-click a line (or right-click and select **Edit Source**) to open the corresponding source file in a code editor. When using the Intel® Advisor GUI on Linux* OS, the editor defined by the **Options > Editor dialog box** Options > Editor dialog box appears with the file open at the corresponding location. When using the Intel Advisor GUI on Windows* OS, the file type association (or **Open With** dialog box) determines the editor used. When using Microsoft Visual Studio*, the Visual Studio code editor appears with the file open at the corresponding location. Later, to return to the **Result** tab, click **Result**.
- Return to the **Suitability Report** window by clicking **Suitability Report**.

The **Suitability Report**, **Suitability Source**, and other Intel Advisor windows appear within the **Result** tab. There is one **Result** tab for each project.

Understand the Scalability Graph in the Suitability Report

One of two different graphs appear depending on the chosen **Target System**. For an explanation of the Scalability Graph, see Suitability Report Overview.

Tips on Understanding the Performance Data

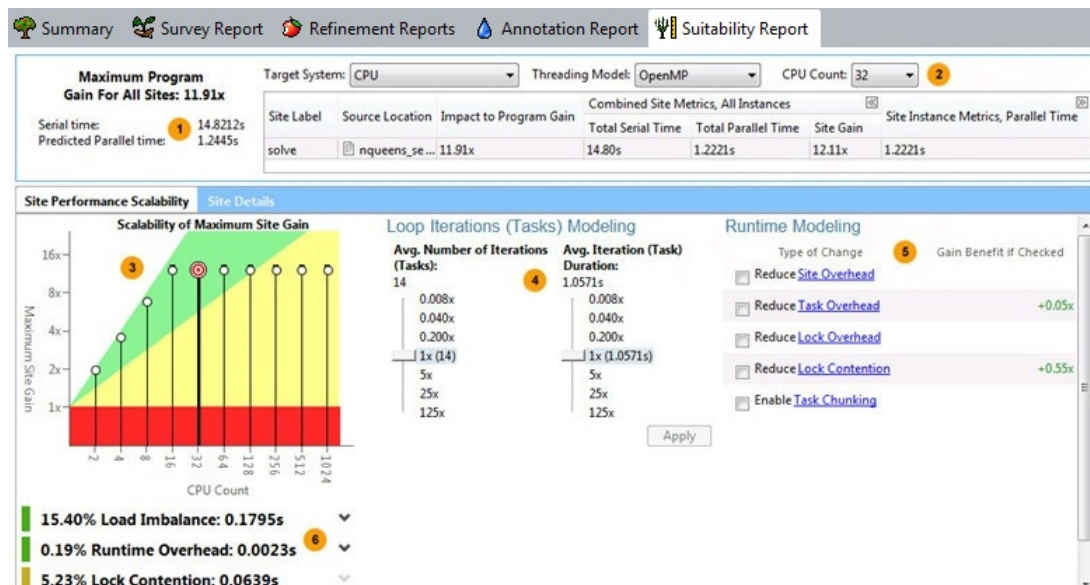
In the **Suitability Report** window, you start at the top, select a site, look at its details in the **Suitability Report** window, and examine its source code. You repeat this process to investigate each annotated site. View this information, and if needed, modify the annotations by using your code editor.

Use the following guidelines to evaluate the feasibility of each site:

- If the **Site Gain** values for the selected site shows an estimated performance gain of 1.0 or less, the overhead of parallel thread execution exceeds the potential performance gains. Modify or remove the annotations for the task(s) and its enclosing site. Repeat this for each parallel site.
- If the **Site Gain** values for the selected site shows a performance gain greater than 1.0, look at the site's contribution to the **Maximum Program Gain for All Sites**, which applies to all parallel sites. For sites that do not contribute significantly to the **Maximum Program Gain for All Sites**, modify or remove the annotations for the task(s) and its enclosing site. For sites that only contribute slightly to the **Maximum Program Gain for All Sites**, examine more closely the annotations and the assumptions about fixing the various overhead costs of parallel thread execution. In some cases, you may be able to adjust the annotations to improve the performance gain or reduce the overhead. Repeat this for each parallel site.
- When the **Maximum Program Gain for All Sites** for all sites and the **Site Gain** values for all the sites show a moderate or significant performance gain, proceed to the next workflow step that uses the Dependencies tool to check your remaining annotated sites for data sharing problems.

Suitability Report Overview





After the Suitability tool runs your program's target executable to collect data, the **Suitability Report** window appears. It displays the approximate predicted performance based on its analysis of the annotated parallel sites and tasks.



This screen shows data based on a **Target System** of **CPU**. The screen shown on your system will differ.

1

The upper-left area shows the **Maximum Program Gain for All Sites** in the program. Your overall goal of adding parallelism is to increase the **Maximum Program Gain for All Sites** so the parallel program will execute as fast as possible. The measured serial execution runtime, predicted parallel

	<p>runtime, and any measured paused time are displayed below Maximum Program Gain for All Sites. Use the predicted Suitability gain values to help you make informed decisions about where to add parallelism.</p> <p>If the Suitability tool detects any annotation-related errors, they appear at the top of the Suitability Report window. If you see this type of error, the displayed Suitability data may not be reliable. Annotation-related errors may be caused when the correct sequence of annotations do not occur because of missing annotations, when unexpected execution paths occur, or if Suitability data collection was paused while the target was executing.</p>
2	<p>Use the upper-right row of modeling parameters to model performance. Choose a hardware configuration and threading model (parallel framework) values from the drop-down lists. If you select a Target System for Intel® Xeon Phi™ processors, an additional value for total Coprocessor Threads appears.</p> <p>Below this row is a grid of data that shows the estimated performance of each parallel site detected during program execution. The Site Label shows the argument to the site annotation. Examine the predicted Site Gain and Impact to Program Gain (higher values are better) to estimate how much each site contributes to the Maximum Program Gain for All Sites for all sites (described above). To expand the data under Combined Site Metrics or Site Instance Metrics, click the  icon to the right of that heading; to collapse data, click  to the right of that heading.</p> <p>To view source code for a selected parallel site, click its row to display the Suitability Source window.</p> <p>To show or hide the side command toolbar, click the  or  icon.</p>
3	<p>The Scalability of Maximum Site Gain graph summarizes performance for the selected site. The number of CPU processors or total number of coprocessor threads appears on the horizontal X axis and the target's predicted performance gain appears on the Y axis. To change the default CPU Count and the Maximum CPU Count, set the Options value.</p> <p>If you choose a Target System of CPU, to view detailed characteristics of the selected site as well as its tasks and locks, click the Site Details tab.</p>
4	<p>Use the Loop Iterations (Tasks) Modeling (or Tasks Modeling) modeling parameters to experiment with different loop structures, iteration counts, and instance durations that might improve the predicted parallel performance.</p> <p>For example, you might want to see the impact of modifying your nested change loop structure, modify the loop body code, or change number of iterations.</p> <p>If the task annotations indicate likely task parallelism, the title will appear as Task Modeling (instead of Loop Iterations (Task) Modeling for data parallelism).</p>
5	<p>Use the Runtime Modeling modeling parameters to learn which parallel overhead categories might have an impact on parallel overhead. If you agree to address a category later by using the chosen parallel framework's capabilities or by tuning the parallel code after you have implemented parallelism, check that category.</p> <p>If the chosen Target System is Intel Xeon Phi or Offload to Intel Xeon Phi, additional Intel® Xeon Phi™ Advanced Modeling options appear below the Runtime Modeling area. To expand this area, click the down arrow to the right of Intel Xeon Phi Advanced Modeling.</p>

6

Below the graph is a list of issues that might be preventing better *predicted* performance gains as well as a summary of serial and predicted parallel time. To expand a line, click the down arrow to the right of the item's name. Most issues are related to the **Runtime Modeling** modeling parameters. Later, you can use other Analyzer tools like Intel® VTune™ Profiler to measure *actual* performance of your parallel program.

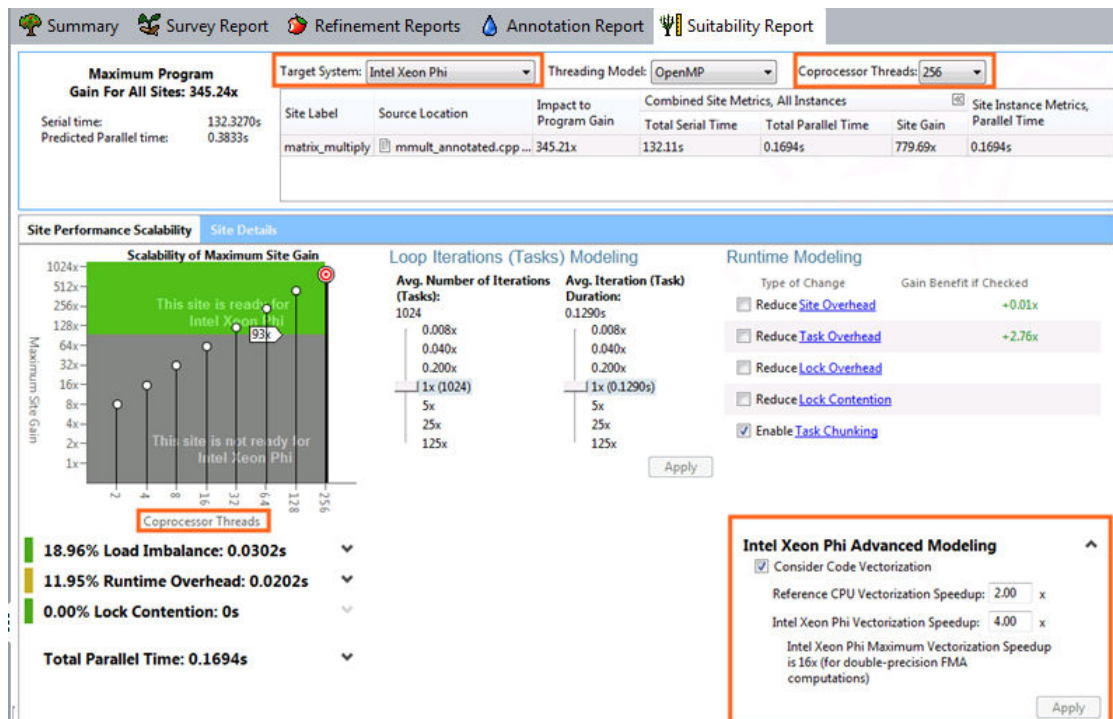
Target System Hardware Configurations

The **Target System** lets you select the type of hardware configuration to be analyzed. From this drop-down list, you can check each type to learn the likely predicted performance characteristics for each:

- **CPU** shows the predicted performance of only the CPU. Choose this item for Intel® Xeon® or similar processors that do not have significant *parallel coprocessors*. For an Intel® Xeon Phi™ processor, choose this setting to only model the host processor, such as an Intel Xeon processor. If you choose this configuration, you can specify the **CPU Count** modeling parameter.
- **Intel Xeon Phi** shows the predicted performance when using only the Intel Xeon Phi coprocessor cores, and not the host processor. This parameter does not account for data exchange amongst Intel Xeon Phi coprocessor cores and the host CPU. If you choose this configuration, you can specify the **Coprocessor Threads** modeling parameter.
- **Offload to Intel Xeon Phi** shows the predicted performance when using Intel Xeon Phi coprocessor manycores to execute parallel code after the host CPU starts the program and before execution resumes on the host CPU for program completion. If you choose this configuration, you can specify the **Coprocessor Threads** and **CPU Count** modeling parameters.

Data Displayed When the Target System is Intel® Xeon Phi™

A sample screen below shows changes in orange boxes when the **Target System** is **Intel Xeon Phi** (instead of **CPU**).



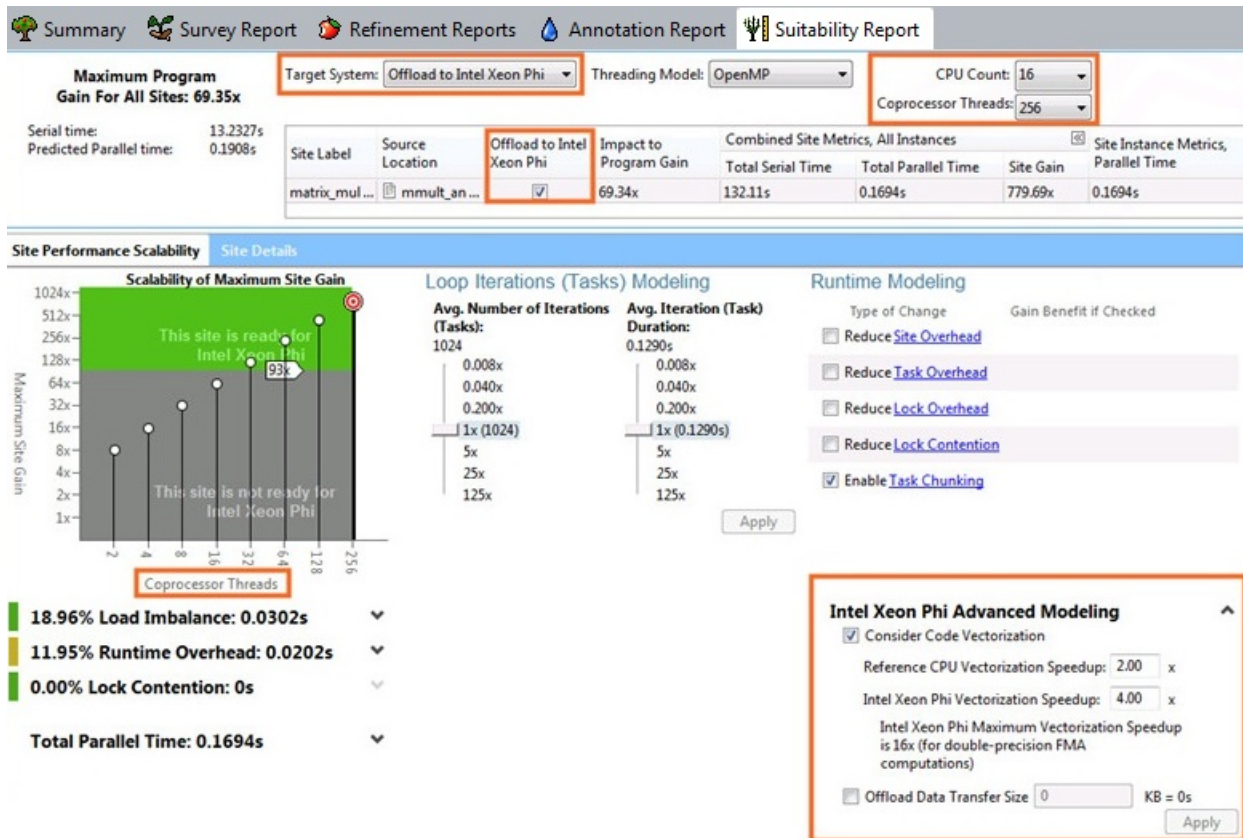
- The displayed data changes, such as the **Maximum Program Gain for All Sites** and the serial and predicted parallel time.

- The graph's appearance changes to a gray-green color and the X axis displays **Coprocessor Threads** (instead of **CPU Count**) to represent the predicted performance of the manycore parallel coprocessor. This graph shows the predicted parallel performance of the manycore parallel coprocessor without accounting for data exchange amongst Intel Xeon Phi coprocessor cores and the host CPU. For many applications, the number of task instances does not scale enough to fully utilize the many cores of the parallel coprocessor, as indicated by a hover tip. Applications that are not appropriate for a Intel Xeon Phi processing system have values that appears in the gray part of the graph; in this case, try modeling other types of the **Target System**.
- The lines between the graph's gray and green areas is a reference baseline, where the reference CPU chosen to calculate the Intel Xeon processor peak baseline is a dual-socket 8-core Intel Xeon processor E5-26xx product family (2.7 GHz, 16 cores total). When the **Maximum Site Gain** exceeds this baseline, you might consider using an Intel Xeon Phi coprocessor rather than an Intel Xeon or similar processor.

When the **Target System** is either **Intel Xeon Phi** or **Offload to Intel Xeon Phi**, the **Intel Xeon Phi Advanced Modeling** options appear. See Intel® Xeon Phi™ Advanced Modeling.

Data and Modeling Parameters When the Target System is Offload to Intel Xeon Phi

A sample screen below shows changes in orange boxes when the **Target System** is **Offload to Intel Xeon Phi** (instead of **CPU**) and the **Offload to Intel Xeon Phi** column is selected.



When you select a **Target System** of **Offload to Intel Xeon Phi** coprocessor:

- The displayed data changes, such as the **Maximum Program Gain for All Sites** and the serial and predicted parallel time.
- An additional modeling parameter appears as a new column for each site named **Offload to Intel Xeon Phi**. If selected, the **Scalability of Maximum Site Gain** graph displays **Coprocessor Threads** on the X axis. If unselected, the graph displays **CPU Count** on the X axis.

- In the upper-right corner, an additional modeling parameter appears. That is, both the total number of **Coprocessor Threads** and the **CPU Count** appear because both the number of CPUs and the coprocessor's total number of hardware threads should be considered to predict parallel execution.
- Additional modeling parameters appear below **Runtime Modeling** area under **Intel Xeon Phi Advanced Modeling** - see Intel® Xeon Phi™ Advanced Modeling.
- When the column named **Offload to Intel Xeon Phi** is selected, the graph's appearance changes to a gray-green color and the X axis displays **Coprocessor Threads** instead of **CPU Count**. This graph shows the predicted performance of the manycore parallel coprocessor and its host CPUs. For many applications, the number of task instances does not scale enough to fully utilize the many cores of the parallel coprocessor, as indicated by a hover tip. Applications that are not appropriate for an Intel Xeon Phi processing system have values that appear in the gray part of the graph; in this case, try modeling other types of the **Target System**. Applications that are appropriate for offload to an Intel Xeon Phi processing system have values that appear in the green part of the graph.

The lines between the graph's gray and green areas is a reference baseline, where the reference CPU chosen to calculate the Intel Xeon processor peak baseline is a dual-socket 8-core Intel Xeon processor E5-26xx product family (2.7 GHz, 16 cores total). When the **Maximum Site Gain** exceeds this baseline, you might consider using an Intel Xeon Phi coprocessor rather than an Intel Xeon or similar processor.

Site Details Tab

If you chose a **Target System** of **CPU**, after you click the **Site Details** tab (next to **Site Performance Scalability**), the lower part of the Suitability Report shows details about the selected site, as well as details about each task and lock within that site.

The screenshot displays the Intel Advisor Suitability Report interface. At the top, there are tabs for Summary, Survey Report, Refinement Reports, Annotation Report, and Suitability Report. The Suitability Report tab is active. Below the tabs, there are dropdown menus for Target System (CPU), Threading Model (OpenMP), and CPU Count (8). The main content area shows a table with columns for Site Label, Source Location, Impact to Program Gain, Combined Site Metrics (All Instances), and Site Instance Metrics (Parallel Time). The table lists a single site labeled 'matrix_multiply' with a source location of 'mmult_ann...' and an impact to program gain of 7.89x. The combined site metrics show a total serial time of 19.35s, a total parallel time of 2.4211s, and a site gain of 7.99x. The site instance metrics show a parallel time of 2.4211s. Below this, there is a section for Site Performance Scalability, which includes a table with columns for Annotation, Annotation Location, Source Location, Number of Instances, Maximum Instance Time, Average Serial Time, Minimum Instance Time, and Total Serial Time. The table lists a single site labeled 'matrix_multiply' with a source location of 'mmult_ann...' and a number of instances of 1. The maximum instance time is 19.35s, the average serial time is 19.35s, the minimum instance time is 19.35s, and the total serial time is 19.35s. Below this, there is a section for Task, which includes a table with columns for Task, Source Location, Number of Instances, Maximum Instance Time, Average Serial Time, Minimum Instance Time, and Total Serial Time. The table lists a single task labeled 'multiply_task' with a source location of 'mmult_ann...' and a number of instances of 1024. The maximum instance time is 0.0209s, the average serial time is 0.0189s, the minimum instance time is 0.0179s, and the total serial time is 19.35s.

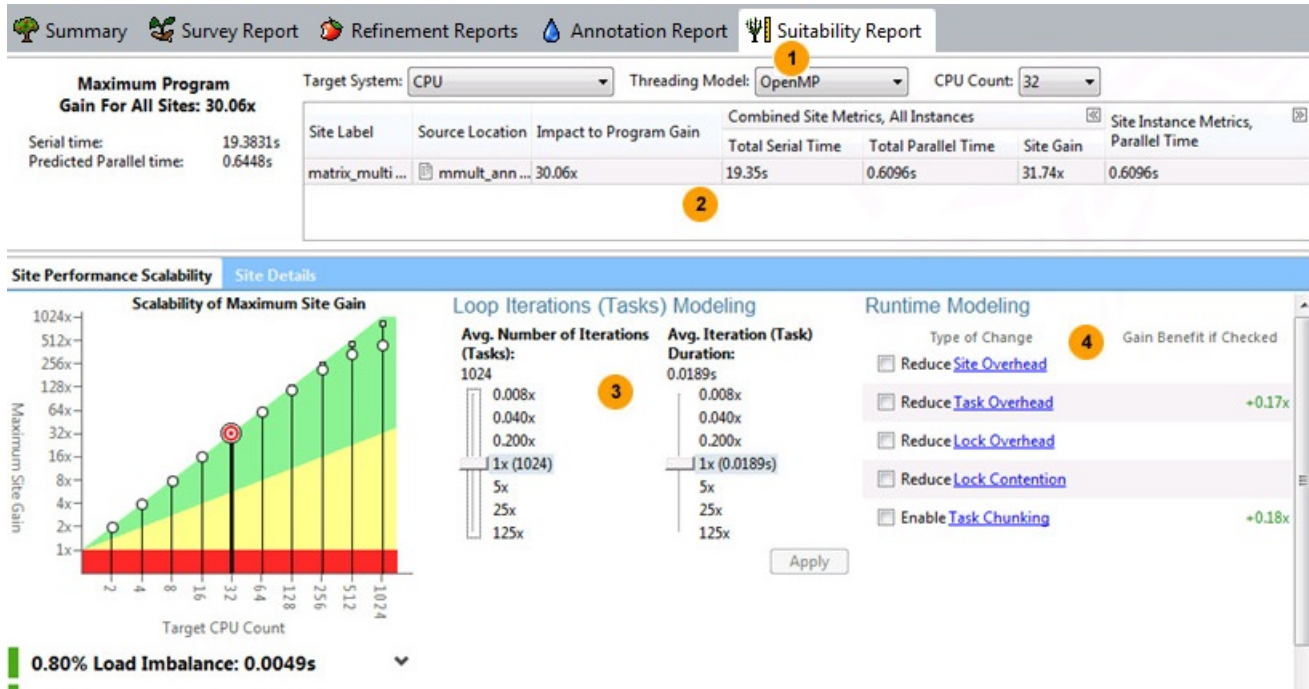
Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances		Site Instance Metrics, Parallel Time
			Total Serial Time	Total Parallel Time	Site Gain
matrix_m ...	mmult_...	7.89x	19.35s	2.4211s	7.99x

Annotation	Annotation L...	Source Location	Number of Instances	Maximum Instance Time	Average Serial Time	Minimum Instance Time	Total Serial Time
Selected S...	matrix_multiply	mmult_ann ...	1	19.35s	19.35s	19.35s	19.35s

Task	Source Location	Number of Instances	Maximum Instance Time	Average Serial Time	Minimum Instance Time	Total Serial Time
multiply_task	mmult_ann ...	1024	0.0209s	0.0189s	0.0179s	19.35s

Choose Modeling Parameters in the Suitability Report

The Suitability Report lets you adjust modeling parameters based on possible application needs. When using an active result, you can adjust modeling parameters and quickly view the likely impact on the predicted performance interactively.

**NOTE**

This screen shows data based on a **Target System** of **CPU**. The screen shown on your system may differ. If you use other **Target System** values for the Intel® Xeon Phi™ processor, additional modeling values appear. See [Suitability Report Overview](#).

- 1** The top row of modeling parameters provides drop-down lists that let you define the likely hardware configuration of target systems as well as the high-level parallel framework. These values let you predict the likely performance characteristics for the selected parallel site.

 - Use the **Target System** to select the type of hardware configuration to be analyzed: **CPU**, **Intel Xeon Phi**, or **Offload to Intel Xeon Phi**. The latter two apply to the Intel Xeon Phi processor system.
 - Use the **Threading Model** to choose the high-level parallel framework to be used, such as OpenMP* or Intel® oneAPI Threading Building Blocks (oneTBB).
 - Use **CPU Count** to specify the number of CPUs to model. To specify the default **CPU** count by setting the Options value.
 - If you choose a **Target System** of **Intel Xeon Phi**, or **Offload to Intel Xeon Phi**, use the **Coprocessor Threads** to choose the number of Intel Xeon Phi coprocessor threads.

As you modify these modeling parameters, the predicted performance estimates are updated automatically. Repeat as needed.
- 2** If your target app contains multiple parallel sites, select each parallel site you wish to examine. When you select a different parallel site, the predicted performance estimates for that site are updated automatically. Repeat as needed for each site.
- 3** Use the **Loop Iterations (Tasks) Modeling** or **Tasks Modeling** area to view the impact of changing the number of iterations and the iteration duration on the predicted performance for the selected parallel site (the label displayed depends on whether iteration loop annotations or general task annotations were detected). For example, you might want to see the impact of modifying your nested change loop structure, modify the loop body code, or change number of iterations. After you slide the **Avg. Number of Iterations (Tasks):** or **Avg. Number of Tasks:** and the **Avg. Duration** values, click the **Apply** button to view the predicted performance estimates. Repeat as needed.

4

Use the **Runtime Modeling** area to view the predicted impact of adjusting run-time parallel characteristics *after* you add parallelism for the selected parallel site, including using parallel framework capabilities to minimize parallel overhead or tuning your parallel code.

If you agree to later check and modify runtime performance aspects for a category, check the box to the left of that category name. For example, you can also examine and tune actual parallel code performance characteristics using tools like Intel® VTune™ Profiler and implement the runtime capabilities of high-level parallel frameworks to limit parallel overhead, such as task chunking. As you check or uncheck different categories, the predicted performance estimates are updated automatically. Repeat as needed.

If you choose **Target System** as **Intel Xeon Phi** or **Offload to Intel Xeon Phi**, additional **Intel Xeon Phi Advanced Modeling** options (not shown) appear below **Runtime Modeling** (see [Advanced Modeling](#)).

NOTE

The Intel® Advisor Suitability tool predicts the general performance characteristics of CPUs. For example, it does not consider CPU clock frequency, cache characteristics, versions of processors, and so on.

See Also

[Suitability Tool Overview](#)

[Suitability Report Overview](#)

[Advanced Modeling](#)

[Fixing Annotation-related Errors Detected by the Suitability Tool](#)

Fix Annotation-related Errors Detected by the Suitability Tool

As the Suitability tool executes your target executable, it scans for a proper sequence of Intel® Advisor annotations. If it detects an annotation-related or an error related to very small task sizes, it displays a message at the top of the **Suitability Report** window.

If you see such messages, investigate the cause and fix the error. The messages displayed are generally self-explanatory.

After you modify your source code and rebuild your application, run the Suitability tool again. When no errors appear near the top of the **Suitability Report** window, you can carefully examine the Suitability data to help you make decisions about the proposed parallel sites and tasks.

Tools to Help You Fix Annotation Errors

Use the **Suitability Source** window to view source code related to a specific site or task. You can also use the **Annotation Report** window to view a list of your annotations and display their code snippets.

When resolving annotation-related errors, consider the execution paths your program follows. If necessary, investigate the execution paths using a debugger.

In addition to annotation sequence messages, messages about task size may also appear. For example, if the CPU time used by a task per loop cycle is so small that it does not exceed the task overhead time, consider modifying the task annotation(s) after you examine the loop structure. In some cases, the message may suggest that you use a different type of task annotation (see the help topics under See Also below).

Proper Sequence of Annotations

The rules about a proper sequence of Intel® Advisor annotations include the following:

- **Sites:** A site-begin annotation is followed by annotations that mark one or more tasks. It is eventually terminated by a site-end annotation. For example, if a site-begin annotation is not followed by a task annotation or is not terminated by a site-end annotation, an error occurs.

- **Tasks:** A task may be marked either with one iterative-task annotation or a pair of task-begin and task-end annotations. When used, an iterative-task annotation must be the only task within a site. Only a task-begin and task-end pair allows task nesting.
- **Locks:** A lock-acquire annotation must be immediately followed by a lock-release annotation, and must occur within a task.

See Also

[Reducing Parallel Overhead, Lock Contention, and Enabling Chunking](#)

[Annotation Types Summary](#)

[Task Organization and Annotations](#)

[Troubleshooting Sources Not Available](#)

[Troubleshooting Debug Information Not Available](#)

[Site and Task Annotations for Simple Loops With One Task](#)

[Site and Task Annotations for Loops with Multiple Tasks](#)

Advanced Modeling Options

When you select a **Target System** of **Intel® Xeon Phi™** or **Offload to Intel Xeon Phi** coprocessor, additional modeling parameters appear below **Runtime Modeling** area under **Intel Xeon Phi Advanced Modeling**:

- Select **Consider Code Vectorization** if you agree to modify your parallel code later to improve vector parallel execution. If checked, you can specify:
 - **Reference CPU Vectorization Speedup** you expect can be achieved. This value indicates the speedup multiplier gain for the current site by using vectorization techniques with the reference CPU. When providing this estimate, base your estimates on target device characteristics and your expertise of *how much* and *how well* this part of code can be vectorized.
 - **Intel Xeon Phi Vectorization Speedup** you expect can be achieved. This value indicates the speedup multiplier gain for current site by using vectorization techniques with an Intel® Xeon Phi™ processor. When providing this estimate, base your estimates on target device characteristics and your expertise of *how much* and *how well* this part of code can be vectorized.
- When you choose **Target System** as **Offload to Intel Xeon Phi**, you can select the **Offload Transfer Data Size** to specify data transfer size value you expect can be achieved (unit is KB).
- Click **Apply** after modifying any of these values.

In some cases, you can restructure your code to enable more efficient vector operations. Loop vectorization allows hardware to process data independently in smaller units (usually 64-byte), such as operations on data arrays.

One way to enable more efficient vector operations is to modify a *single* loop to create a new outer loop where the two loops cover the same iteration space. A technique called strip-mining allows the innermost loop to use vector operations in small chunks.

Other ways to enable more efficient vector operations include examining outermost loops where threading parallelism might already be used, and consider vectorizing its innermost loops and/or callee functions.

Certain innermost loops may benefit from OpenMP 4 constructs. That is, under certain conditions you can use both an `omp parallel for` threading pragma and a `omp simd` (or similar) `simd` vectorization pragma (see the compiler vectorization report and descriptions at <http://openmp.org>).

The processor microarchitecture determines the type of vector instructions that will be supported and thus the size of data the hardware can process efficiently.

See Also

[Dependencies Analysis](#)

[Suitability Tool Overview](#)

[Suitability Report Overview](#)

Reduce Parallel Overhead, Lock Contention, and Enable Chunking

The data collected and analyzed in the **Suitability Report** window shows data for the selected site. The text that appears below **Runtime impact for this site** (lower-right area) may recommend that you consider reducing several types of parallel overhead, lock contention, and enable chunking in your parallel program, as explained in [Suitability Report Overview](#). If you agree to address a category later by using the chosen parallel framework's capabilities or by tuning the parallel code after you have implemented parallelism, check that category.

This group of topics explain site, task, and lock overhead, lock contention, and task chunking.

Reduce Site Overhead

Site overhead is the time spent starting up (and shutting down) parallel execution. This overhead includes creating threads, scheduling those threads onto cores, and waiting for the threads to begin executing. In some parallel framework implementations, real threads are only created once - rather than destroying them at the end of a parallel site; the implementation suspends the threads. In this case, the full site overhead will be experienced only the first time a site is entered.

Site overhead is proportional to the number of times a site is executed. If you have a site that is executed too frequently or where the average time per instance is too small, you should choose a location for your site that encloses a larger amount of computation.

If the Suitability tool recommends that you reduce site overhead, the parallel site is probably too small.

To reduce Site overhead, have the site do more work during its execution. You might be able to combine multiple site executions into one. For example, consider putting a site outside a loop instead of inside a loop.

Reduce Task Overhead

Task overhead is the time spent creating a task and getting it assigned to a thread, and also the time spent stopping or pausing the thread when the task is complete.

Task overhead is proportional to the number of times a task is executed. If you have a task that is executed too frequently or where the average time per task instance is too small, modify your task so it encloses a larger amount of computation. Alternatively, consider using the task chunking feature, which is supported by several parallel frameworks. In this case, the parallel framework groups multiple task executions at run-time.

If the Suitability analysis recommends that you reduce task overhead, the parallel task is probably too small. Often this is because you have chosen an inner loop in a leaf function as the location of your parallel site, where you instead should have chosen a function farther up the call tree.

There are two ways to reduce task overhead:

- Restructure your program to reduce the number of tasks you create. For example, restructure your task annotations and/or code to increase the amount of work that occurs during each task's execution.
- If available for the selected parallel framework, enable the task chunking feature.

You can reduce task overhead by combining multiple task executions into a single task execution. For example, by merging two tasks into one.

Reduce Lock Overhead

Lock overhead is the time spent in creating, destroying, acquiring, and releasing locks. Lock overhead does not include the time spent waiting for a lock held by another task - that is called lock contention. You can think of lock overhead as the cost of the lock operations themselves assuming the lock is always available.

If possible, restructure your code to reduce Lock overhead by creating a private copy of an object for each task to avoid the need to acquire a lock - see the help topic Problem Solving Strategies.

Reduce Lock Contention

Lock contention is the time spent waiting in one thread for a lock to be released while another thread is holding that lock.

You can reduce Lock contention by using different locks for unrelated data when you convert to a parallel framework.

Enable Task Chunking

Chunking means that the parallel framework will merge several tasks into a single task, with little or no overhead between them. For instance, if tasks are loop iterations, chunking would mean that several iterations are executed together (as a chunk) before heavyweight task control is performed.

Chunking is typically implemented when you convert to a parallel framework:

- With Intel® oneAPI Threading Building Blocks, by using a `parallel_for()` instance.
- With OpenMP*, by using the C/C++ `#pragma omp parallel for` or the Fortran directive `!$omp parallel do`.

You can also restructure your code to enable chunking. This can be done by modifying a single loop to create a new outer loop where the two loops cover the same iteration space. A technique called strip-mining allows the inner loop to use vector operations in small chunks. Loop vectorization allows hardware to process data independently in smaller units (usually 64-byte), such as operations on data arrays.

Once these two loops exist, move the inner loop inside the task annotations so the task begin and end annotations encapsulate the inner loop. The outer loop strides by some chunk size, and the inner loop iterates sequentially through each chunk.

In cases where the CPU time and the elapsed time are about the same, the **Suitability Report** window under **Runtime impact for this site** may recommend that you enable task chunking.

If you check an item under to the right of the **Scalability of Maximum Site Gain** graph (such as **Enable Task Chunking**), its value will be added to the **Site Gain** and possibly the **Maximum Site Gain for All Sites** values.

See Also

[Dependencies Analysis](#)

[Parallelize Functions - Intel® oneAPI Threading Building Blocks Tasks](#)

[Suitability Tool Overview](#)

[Reducing Task Overhead](#)

[Suitability Report Overview](#)



Check for Dependencies Issues

Purpose

View any predicted data sharing problems and informational remark messages.

Report Regions and Purpose

In the **Dependencies Report** tab at the bottom of the **Refinement Report**:

- [Problems and Messages](#) pane - Select the problems that you want to analyze by viewing their associated observations.
- [Code Locations](#) pane - View details about the code locations for the selected problem in the **Dependencies Report** window. Icons identify the focus code location  and related code location .
- **Filters** pane - Filter contents of the report tab.

Associated [Dependencies Source](#) window, from top left to bottom right:

- [Focus Code Location](#) pane - Use this pane to explore source code associated with focus code location in the **Dependencies Source** window.
- [Focus Code Location Call Stack](#) pane - Use this pane to select which source code appears in the **Focus Code Location** pane in the **Dependencies Source** window.
- [Related Code Locations](#) pane - Use this pane to explore source code associated with related code locations (related to the focus code location) in the **Dependencies Source** window.
- [Related Code Location Call Stack](#) pane - Use this pane to select which source code appears in the **Related Code Location** pane.

- [Code Locations](#) pane - Use this pane to view the details about the code location for the selected problem in the **Dependencies Report** window.
- [Relationship Diagram](#) pane - Use this pane to view the relationships among code locations for the selected problem.

Use Dependencies Data

Use the **Dependencies Report** to view each reported problem and its associated code locations. Use the **Dependencies Source** window to view the focus and related source code regions to help you understand the cause of the reported problem.



To learn about a reported problem, right-click its name in the **Dependencies Report, Problems and Messages** pane and select **What Should I Do Next?**. This displays the help topic for that problem type.

See Also

- [Dependencies Problem and Message Types](#)

Code Locations Pane






Purpose



View details about the code locations for the selected problem in the **Dependencies Report** window. Icons identify the focus code location  and related code location .

Location

Bottom of [Dependencies Report](#) tab

Controls

Use This	To Do This
Title bar	View the problem type.
Code Location data row(s)	Review related code locations: <ul style="list-style-type: none"> • ID - Code location identifier • Description - What happens at this code location. • Source - The source file for this code location. • Function - Function name. • Modules - The executable associated with this problem. • State - Indicates whether the problem has been fixed or not. To change the state, use the context menu.
Click  to the left of a code location name	Display a code snippet associated with the selected code location.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> • Whether this is a related code location. • If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> • Whether this is the focus code location. • If code location source code is available for viewing and editing.

Use This	To Do This
 icon,  icon, or no icon in the Source column	Shows if code location source code is available for viewing and editing.
Double-click a code location data row or source line, or right-click and select the View Source context menu item	Display the Dependencies Source window.
Right-click and select the Edit Source context menu item	Display a code editor with the corresponding source file. <ul style="list-style-type: none"> On Windows* OS: <ul style="list-style-type: none"> When using Visual Studio, the Visual Studio code editor appears with the file open at the corresponding location. When using the Intel Advisor GUI, the file type association (or Open With dialog box) determines the editor used. On Linux* OS: When using the Intel Advisor GUI, the editor defined by the Options > Editor dialog box appears with the file open at the corresponding location.
Column labels	Click a column heading to sort the data grid rows in either ascending or descending order.
Pane border	Resize the pane (drag).
Right click a row to display a context menu	Display a context menu to: expand or collapse all code snippets, open the Dependencies Source window, edit sources in the code editor, copy the selected data row(s) to the clipboard, mark the state as fixed or not fixed, or display context-sensitive help.

Filter Pane (Dependencies Report)

Purpose


Filter contents of the report tab.

Location

Right side of [Dependencies Report](#) tab

Controls

Use This	To Do This
Category column	Review categories that you can filter, such as Severity , Type , Site Name , Source , and so on.

Use This	To Do This
	<p>NOTE</p> <p>You can apply only one filter criterion per category; however, you can filter the listed problems and messages by multiple categories simultaneously.</p>
Click a filter criterion, such as: <ul style="list-style-type: none"> Error under the Severity category Memory Reuse under the Type category 	View only problems and messages of a specific type, and hide other types of problems and messages in the same category.
All button to the right of the category's name	To deselect all filter criteria and display all problems and messages in that category.
 button	To deselect all filter criteria in all categories.

See Also

- [Dependencies Problem and Message Types](#)

Problems and Messages Pane






Purpose

Select the problems that you want to analyze by viewing their associated observations.

Location

Top of [Dependencies Report](#) tab.

Controls

Use This	To Do This
Column labels	Click a column label to sort the data grid data rows in either ascending or descending order.
Selected data row	Review the characteristics of each data row in the grid. The columns are: <ul style="list-style-type: none"> ID - Identifier for the problem.  (severity) - The severity of the problem, such as error , warning , or an informational remark message . For example, the location of parallel sites executed are indicated by the message  Parallel Site. Type - The problem type or message type. For more information about a problem, right click to display the context menu. Site Name- The name of the site associated with this problem. Sources - The source file associated with this problem. Modules - The modules (executable) associated with this problem. State - Indicates whether the problem has been fixed or not. To change the state, use the context menu in this pane.

Use This	To Do This
Pane border	Resize the pane (drag).
Right click a row to display a context menu	Display a context menu to: open the code editor to the corresponding source line, display the Dependencies Source window, copy the selected data row(s) to the clipboard, or display context-sensitive help for that problem or message.

Dependencies Source Window

Purpose

Use this window to examine the source code for a selected **Problem**, **Message**, or **Code Location**. To modify your source code, double-click a source line or use the **Edit Source** context menu item to display that file in a code editor.

- On Windows* OS:
 - When using Visual Studio, the Visual Studio code editor appears with that file open at the corresponding location.
 - When using the Intel Advisor GUI, the file type association (or Open With dialog box) determines the editor used.
- On Linux* OS: When using the Intel Advisor GUI, the editor defined by the **Options > Editor dialog box** appears with the file open at the corresponding location.

Use This	To Do This
Workflow Tab	Run a tool of your choice and see results in the Result tab.
Result Tab	Select between available reports.
Focus Code Location pane	Explore the source code associated with the focus code location.
Focus Code Location Call Stack pane	Select the source code to appear in the Focus Code Location pane.
Related Code Location pane	Explore source code associated with the code locations. This pane does not appear if the Focus Code Location does not have a Related Code Location.
Related Code Location Call Stack pane	Select the source code to appear in the Related Code Location pane. This pane does not appear if the Focus Code Location does not have a Related Code Location.
Code Locations pane	View details about the code locations for the selected problem in the Dependencies Source window.
Relationship Diagram pane	View the relationships among code locations for the selected problem.

Access

To access this window in the **Refinement Reports**, double-click a data row or use the corresponding context menu item to view the source code associated with a **Problem**, **Message**, or **Code Location**.

Regions

From top left to bottom right:

- **Focus Code Location** pane - Use this pane to explore source code associated with focus code location in the **Dependencies Source** window.
- **Focus Code Location Call Stack** pane - Use this pane to select which source code appears in the **Focus Code Location** pane in the **Dependencies Source** window.
- **Related Code Locations** pane - Use this pane to explore source code associated with related code locations (related to the focus code location) in the **Dependencies Source** window.
- **Related Code Location Call Stack** pane - Use this pane to select which source code appears in the **Related Code Location** pane.
- **Code Locations** pane - Use this pane to view the details about the code location for the selected problem in the **Dependencies Report** window.
- **Relationship Diagram** pane - Use this pane to view the relationships among code locations for the selected problem.

Code Locations Pane (Dependencies Source Window)







Purpose

Use this pane to view the details about the code location for the selected problem in the **Dependencies Report** window.

Location

Bottom left of Dependencies Source window

Controls

Use This	To Do This
Title bar	View the problem type.
Code location data row(s)	Review related code locations: <ul style="list-style-type: none"> • ID - Code location identifier • Description - What happens at this code location • Source - The source file associated with this code location. • Function - Function name. • Modules - The executable associated with this problem. • State - Indicates whether the problem has been fixed or not. To change the state, use the context menu.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> • Whether this is a related code location. • If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> • Whether this is the focus code location. • If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	Shows if code location source code is available for viewing and editing.
Column labels	Click a column heading to sort the data grid rows in either ascending or descending order.

Use This	To Do This
Pane border	Resize the pane (drag).
Right click a row to display a context menu	Display a context menu to: set this code location as the focus or related code location, copy the selected data row(s) to the clipboard, mark the state as fixed or not fixed, or display context-sensitive help.

Focus Code Location Pane



Purpose

Use this pane to explore source code associated with focus code location in the **Dependencies Source** window.

Location

Top left of [Dependencies Source](#) window

Controls

Use This	To Do This
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> Whether this is the focus code location. If code location source code is available for viewing and editing.
Pane border	Resize the pane (drag).
Source code	<ul style="list-style-type: none"> Explore source code associated with the focus code location Display the code editor at the corresponding source file by double-clicking a data row or by using the corresponding context menu item.
Right click a row to display a context menu	Display a context menu to: open the code editor to the corresponding source line, copy the selected data row(s) to the clipboard, or display context-sensitive help.

Focus Code Location Call Stack Pane



Purpose

Use this pane to select which source code appears in the **Focus Code Location** pane in the **Dependencies Source** window.

Location

Top right of [Dependencies Source](#) window

Controls

Use This	To Do This
 or  icon	View whether: <ul style="list-style-type: none"> Source code is available for viewing and editing. An  icon indicates that source code is not available.

Use This	To Do This
Click a row in the Call Stack pane	Displays source code for the specified call stack entry.
Pane border	Resize the pane (drag).
Right click a row in the Call Stack pane	Customize the call stack presentation by using the Call Stack context menu.

Related Code Locations Pane



Purpose

Use this pane to explore source code associated with related code locations (related to the focus code location) in the **Dependencies Source** window.

Location

Middle left of [Dependencies Source](#) window

Controls

Use This	To Do This
 icon,  icon, or no icon in the Source column	View: <ul style="list-style-type: none"> Whether this is a related code location. If code location source code is available for viewing and editing.
Pane border	Resize the pane (drag).
Source code	<ul style="list-style-type: none"> Explore source code associated with the focus code location Display the code editor at the corresponding source file by double-clicking a data row or by using the corresponding context menu item.
Right click a line to display a context menu	Display a context menu to: open the code editor to the corresponding source line, copy the selected data row(s) to the clipboard, or display context-sensitive help.

Related Code Location Call Stack Pane




Purpose

Use this pane to select which source code appears in the **Related Code Location** pane.

Location

Middle right of [Dependencies Source](#) window

Controls

Use This	To Do This
 or  icon	View whether: <ul style="list-style-type: none"> Source code is available for viewing and editing. An  icon indicates that source code is not available.

Use This	To Do This
Click a row in the Call Stack pane	Displays source code for the specified call stack entry.
Pane border	Resize the pane (drag).
Right click a row in the Call Stack pane	Customize the all stack presentation by using the Call Stack context menu.

Relationship Diagram Pane







Purpose

Use this pane to view the relationships among code locations for the selected problem.

Location

Bottom right of [Dependencies Source](#) window

Controls

Use This	To Do This
Title bar	View the problem type.
 icon,  icon, or no icon in the Source column	View: <ul style="list-style-type: none"> Whether this is a related code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	View: <ul style="list-style-type: none"> Whether this is the focus code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	View if code location source code is available for viewing and editing.
Pane border	Resize the pane (drag).
Diagram	View the relationship among code locations in a problem: <ul style="list-style-type: none"> Each box in a diagram represents a code location in a problem. A diagram with a single box is a trivial problem with no related code locations. Boxes arranged left-to-right with connecting arrows indicate a time ordering. Boxes with connecting lines indicate association.

Add Parallelism to Your Program

Once you have completed the previous steps in the Threading perspective workflow and have tested and approved a serial version of your application program, you can add parallelism to a selected parallel site. Before you add parallel framework code, complete developer/architect design and code reviews about the proposed parallel changes.

To add parallelism to your program, perform the following steps:

1. Choose one parallel programming framework (threading model) for your application, such as Intel® oneAPI Threading Building Blocks (oneTBB) , OpenMP*, Microsoft Task Parallel Library* (TPL) (on Windows* OS systems only), or some other parallel framework. To learn about the parallel framework(s) available for your application's language, see the help topic [Parallel Frameworks](#).
2. Add the parallel framework to your build environment.
3. Add parallel framework code to synchronize access to the shared data resources, such as oneTBB or OpenMP locks.
4. Add parallel framework code to create the parallel tasks.

In the last two steps, as you add the appropriate parallel code from the chosen parallel framework, you can keep, comment out, or replace the Intel Advisor annotations.

You should add the synchronization code - such as oneTBB or OpenMP locks or mutexes - before adding the parallelism. Synchronized code without parallelism works correctly. In contrast, parallel code without synchronization works incorrectly.

With the synchronization in place, introduce the parallelism. This will cause the operations of multiple tasks to execute in parallel. If you have any remaining bugs caused by data sharing and synchronization problems, they will begin to appear and must be debugged.

Before You Add Parallelism: Choose a Parallel Framework

After you decide on parallel sites and tasks, select a parallel framework so you can replace Intel® Advisor annotations with parallel framework code.

The available high-level parallel frameworks depend on the language whose code you will add parallelism to:

- For C/C++ native code, there are several choices as explained in [Parallel Frameworks](#)
- For Fortran native code, use [OpenMP](#).
- For managed code such as C#, use the [Microsoft Task Parallel Library* \(TPL\)](#).

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

If you are not familiar with high-level parallel frameworks, read [Parallel Frameworks](#)

To use a different framework, read [Other Parallel Frameworks](#).

Parallel Frameworks

Before you can add parallel code, you must first choose a parallel framework.

There are two popular mechanisms for using threads - either use high-level parallel frameworks or explicit threading APIs. Intel recommends using parallel frameworks for both ease of use and their ability to optimize for different situations.

For managed code such as C#, use the Microsoft Task Parallel Library* (TPL).

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

This document shows how to use the widely-used parallel frameworks for native code, which are included with Intel® oneAPI Toolkits and may be included with other compilers:

- Intel® oneAPI Threading Building Blocks (oneTBB)
- OpenMP*

Intel® oneAPI Threading Building Blocks (oneTBB) is a parallel programming framework for C++ code. oneTBB is structured as a traditional C++ library, consisting of header files and a run-time library, so it can be used with any C++ compiler. Intel recommends that you consider using oneTBB for introducing parallelism into C++ programs. oneTBB programs can be run on any platform (OS/architecture pair) to which the oneTBB library has been ported. For example, the Intel® oneAPI DPC++/C++ Compiler includes oneTBB .

OpenMP is a high-level framework that supports C, C++, and Fortran. OpenMP is provided by compiler support, so you modify your sources by using compiler directives rather than using types, variables, and calls. An OpenMP program can often be changed from parallel execution to serial execution by setting an

environment variable or omitting a compiler option so the compiler ignores the directives. OpenMP 2 is supported by the Microsoft, the Intel, and the GNU* C, C++ and Fortran compilers. The OpenMP 3.0 standard adds `TASK` support and is supported by the Intel compilers, which also support parts of OpenMP 4.0. For Microsoft and GNU compilers, consult your compiler documentation for the current level of OpenMP support.

You can also use a different parallel framework.

Windows* OS: Support for Parallel Frameworks by Microsoft and Intel Compilers

With a Fortran program, the only high-level parallel framework available is OpenMP. The following table summarizes the support by Microsoft and Intel Compilers for the recommended parallel frameworks for C/C++ programs on Windows OS systems.

Language and Compiler	oneTBB	OpenMP
C programs, Intel® C++ Compiler Classic		Supported
C++ programs, Intel® C++ Compiler Classic	Supported	Supported
C programs, Microsoft Visual C++* Compiler		Supported
C++ programs, Microsoft Visual C++ Compiler	Supported	Supported

For more information about oneTBB and OpenMP, see the corresponding sections in this Intel Advisor help system.

Linux* OS: Support for Parallel Frameworks by GNU* and Intel Compilers

With a Fortran program, the only high-level parallel framework available is OpenMP. The following table summarizes the support by GNU gcc* and Intel compilers for the recommended parallel frameworks for C/C++ programs on Linux OS systems.

Language and Compiler	oneTBB	OpenMP
C programs, Intel® C++ Compiler Classic (<code>icc</code>)		Supported
C++ programs, Intel® C++ Compiler Classic (<code>icc</code>)	Supported	Supported
C programs, GNU gcc Compiler (<code>gcc</code>)		Supported
C++ programs, GNU gcc Compiler (<code>gxx</code>)	Supported	Supported

For more information about oneTBB and OpenMP, see the following sections in this Intel Advisor help system. For detailed instructions, see your compiler documentation and the resources listed in Related Information.

Intel® oneAPI Threading Building Blocks (oneTBB)

Intel® oneAPI Threading Building Blocks (oneTBB) is a high-level parallel programming framework for C++ code that uses a template-based runtime library to help you harness the performance of multi-core processors. oneTBB lets you specify logical parallelism instead of threads. You specify potential parallelism - what can be run in parallel. The library decides the actual parallelism at run-time, matching it to the available hardware. The library has templates that simplify using high level parallel patterns such as parallel loops. oneTBB programs are implemented by a library that has been ported to multiple C++ compilers.

Use oneTBB to write scalable programs that:

- Specify parallel work instead of managing threads.
- Emphasize data parallel programming.
- Take advantage of high-level parallel patterns.

oneTBB consists of header files and shared libraries, so it can be used with any C++ compiler.

Intel recommends that you consider using oneTBB for introducing parallelism into C++ programs. It has a small cost of entry and provides excellent initial performance with a lot of additional capabilities that can be used for future refinements.

It also has many powerful features that can make it possible to easily parallelize more places in your application. These features include:

- Parallel algorithmic patterns
- Concurrency-friendly containers
- Scalable memory allocation
- Synchronization primitives
- Timing

OpenMP*

OpenMP* is a parallel programming framework for C, C++, or Fortran code. Using OpenMP requires few source changes and is supported by multiple compilers. Because OpenMP is supported by OpenMP libraries, you modify your source code with compiler directives rather than using types, variables, and calls. An OpenMP program can often be changed from parallel execution to serial execution by omitting a compiler option so the compiler ignores the OpenMP directives.

OpenMP 2 is very good at using several cores on loops that process arrays, but does not support irregular parallelism through general tasking. It is supported by the Microsoft, the Intel, and the GNU* C, C++ and Fortran compilers. It is difficult to use OpenMP version 2 for situations other than simple divisions of statement sequences or complete loop bodies.

The OpenMP 3.0 specification adds `TASK` support. The `TASK` directives enable performing arbitrary pieces of an algorithm in parallel. The Intel® C++ Compiler Classic and Intel® Fortran Compiler Classic support OpenMP 3.0 and some parts of OpenMP 4.0. For Intel, Microsoft, and GNU* compilers, consult your compiler documentation for the level of OpenMP support.

If your application is written in Fortran, OpenMP is the only high-level parallel framework available.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well your OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

Microsoft Task Parallel Library* (TPL)

Microsoft Task Parallel Library* (TPL) in the Microsoft .NET* Framework is a combination of public types and APIs that allow addition of parallelism and concurrency on Windows* OS systems. For Intel Advisor users, use Microsoft TPL for C# and managed C++ libraries.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

Microsoft TPL is a high-level parallel programming framework for .NET code to help you harness the performance of multi-core processors. It lets you specify logical parallelism instead of threads. That is, you specify potential parallelism - what can be run in parallel. The library decides the actual parallelism at run-time, matching it to the available hardware.

Microsoft TPL provides two main classes:

- `System.Threading.Tasks.Parallel`: includes `For` and `ForEach` loops.
- `System.Threading.Tasks.Task`: is the preferred way to express asynchronous operations.

Other classes are also available. For example, `System.Collections.Concurrent` provides for concurrent collections that do not require external locking.

You can use Microsoft TPL for introducing parallelism into either C# programs or managed C++ code.

Please refer to your Microsoft MSDN* help documentation for information about this parallel framework. For example: MSDN Library > .NET Development > .NET Framework 4 > .NET Framework Advanced Development > Parallel Programming > Task Parallel Library

Other Parallel Frameworks

Intel Advisor helps you prepare your program for adding parallelism, regardless of the parallel framework you choose. Intel Advisor provides the ability to predict the parallel behavior of your serial program and lets you determine the feasibility of possible parallel regions before you actually add parallelism.

Intel Advisor does not perform analysis of your parallel program, so you can use any parallel framework. You can use Intel Advisor with high-level parallel frameworks that use a fork-join model, or with low-level APIs that provide explicit thread control.

Intel recommends using the parallel frameworks Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP*, which are included with Intel® oneAPI Toolkits. These high-level frameworks provide parallel features well-suited for most multi-core computer systems.

If you decide to use a different parallel framework or a low-level threading API, please be aware of the following considerations:

- Some parallel frameworks have limited abilities to scale with different number of cores, handle load balancing, and handle loop scalability (chunking), and so on.
- As part of your planning, you might create a mapping of at least Intel Advisor Site, Task, and Lock annotations to the equivalent code constructs in the chosen parallel framework. That is, create a list that maps the Intel Advisor annotations to your parallel framework's features. Thus, you need to be aware of all annotations. For example, all Intel Advisor programs need to use parallel site and task annotations. Most programs will also use lock annotations. For a complete list of annotations, see the help topic [Summary of Annotation Types](#).
- Some parallel frameworks require that you use certain compilers that recognize the parallel framework's keywords, while others are libraries that can be used with multiple compilers.
- Some parallel frameworks may not correctly handle multi-program workloads.

In all cases, you need to learn how to use the parallel framework that you select. The current Add Parallelism workflow step involves replacing annotations with chosen parallel framework code.

Add the Parallel Framework to Your Build Environment

After you choose the parallel framework, you need to add the parallel framework to your build environment.

Adding the parallel framework to your build environment can require installing additional `www`, as well as modifying build scripts, modifying project properties or Microsoft Visual Studio* project properties (on Windows* OS systems), and so on.

Later, after you add the parallel framework to your build environment, you can begin to make source code changes that use the parallel framework to add synchronization (such as locks) or parallelism to your program.

The following sections describe adding the Intel® oneAPI Threading Building Blocks (oneTBB) and OpenMP* parallel frameworks to your build environment, including adding C++11 (formerly C++0x) Lambda Expression Support that simplifies the use of oneTBB .

Enable Intel® oneAPI Threading Building Blocks (oneTBB) in your Build Environment

If you use the Intel® C++ Compiler Classic or Intel® oneAPI DPC++/C++ Compiler from the command line, specify the following option when you build your program:

- For Windows* OS: `/Qtbb`
- For Linux* OS: `-tbb`

This option tells the compiler to link with the Intel® oneAPI Threading Building Blocks (oneTBB) libraries. If you use other compilers, please see your oneTBB or compiler documentation.

NOTE

With Intel Advisor samples, to use the oneTBB project (`_tbb`), you might need to define the `TBBROOT` environment variable (see the help topic [Define the TBBROOT Environment Variable](#)) and specify the `TBBROOT/include` directory as an additional include path when compiling (in build properties on Windows OS).

The following instructions are for using the Visual Studio* development environment on a Windows OS system.

Modify the project properties for each of your Visual Studio project build configurations (debug, release, and so on). You can set multiple properties by using the Configuration Properties with Visual Studio:

1. In Solution Explorer, select (click) the name of one or more projects. To select multiple projects, hold down the **Ctrl** key.
2. With Visual Studio:
 - Right-click the project name(s) and select **Configuration Properties > Intel Performance Libraries > Intel oneAPI Threading Building Blocks**.
 - On the **Use oneTBB** line, specify **Yes**.

NOTE

If you change the version of oneTBB or the Visual Studio version installed on your system, you may see build errors related to oneTBB libraries. In this case, reset the integration by repeating the above steps to uncheck, and then check, the **Use oneTBB** box. See the Intel Advisor release notes for more information.

3. Click **OK** to save the specified properties.
4. Repeat the steps above for other configurations.

This procedure defines multiple properties to set up your build environment to use oneTBB .

See Also

[Define the TBBROOT Environment Variable](#)

[Parallel Frameworks Overview](#)

Define the TBBROOT Environment Variable

With Intel® Advisor samples, to build the Intel® oneAPI Threading Building Blocks (oneTBB) project (`_tbb`), you need to define the `TBBROOT` environment variable.

To define this environment variable:

On Linux* OS:

1. Open a command line window.
2. Use the `export` command to set the `TBBROOT` environment variable, type: `export TBBROOT=<tbb-install-dir>`. If you used the default path during installation, the `<tbb-install-dir>` is inside:
 - For root users:


```
/opt/intel/
```
 - For non-root users:


```
$HOME/intel/
```

For example, if you installed the Intel® oneAPI Threading Building Blocks as a part of Intel® oneAPI Base Toolkit, the `<tbb-install-dir>` may be `/opt/intel/oneapi/tbb/<version>`.

3. To always set this variable on the current system, add this definition to your `.login` or similar shell initialization file.

On Windows* OS:

1. Open the control panel and access: **Control Panel > System and Security > System > Advanced system settings > Environment Variables...**
2. Locate any existing definition of the `TBBROOT` user or system environment variable. If present, verify that its value is correct if you encountered build errors and either click **Cancel** or **OK** as needed to exit the dialog box.

3. If it is not present, under **System variables** or **User variables**, click **New**.
4. Specify the **Variable name** as: `TBBROOT`.
5. Specify the **Variable value** as the path of the installed Intel® oneAPI Base Toolkit files, including the `\tbb` directory.

If you installed the product as part of a Intel® oneAPI Base Toolkit and used the default path, files are installed below: `C:\Program Files (x86)\Intel\oneAPI\`, for example `C:\Program Files (x86)\Intel\oneAPI\tbb\<version>`.

6. Click **OK** several times.
7. For the change to take effect:
 - If using Microsoft Visual Studio*: close and reopen Visual Studio.
 - If using command window: close and reopen your command window.

In some cases, you may need to log off and log on for this change to take effect.

8. If needed, you can test the definition by opening a command window and typing `set TBBROOT`.

You have defined the `TBBROOT` environment variable.

See Also

[Enable C++11 Lambda Expression Support with Intel® oneAPI Threading Building Blocks \(oneTBB\) Parallel Frameworks Overview](#)

Enable C++11 Lambda Expression Support with Intel® oneAPI Threading Building Blocks (oneTBB)

The C++11 (new standard for the C++ language, formerly C++0x) lambda expression support makes many Intel® oneAPI Threading Building Blocks (oneTBB) constructs easier to program because it avoids the need to introduce extra classes to encapsulate code as functions. If you decide to use this feature, you need a compiler that supports it, such as the Intel® C++ Compiler Classic or Intel® oneAPI DPC++/C++ Compiler. For more information about C++11 lambda expression support in the other compilers, please see your compiler documentation (online help).

When using the command line with the Intel® C++ Compiler Classic or Intel® oneAPI DPC++/C++ Compiler, specify the following option to enable lambda expression support:

- For Windows* OS: `/Qstd=c++0x`
- For Linux* OS: `-std=c++0x`

To enable the C++11 support in Visual Studio on a Windows* OS system:

1. In Solution Explorer, select (click) the name of one or more projects. To select multiple projects, hold down the **Ctrl** key.
2. Right-click the project name and select **Intel Compiler** > **Use Intel C++** from the context menu.
3. Select **Project** > **Properties**, or right-click the project name and select **Properties** from the context menu.
4. Specify the following Configuration Properties:

C++ > Language Under **Intel Specific**, select **Enable C++0x Support** as **Yes**

5. Click **OK** to save the specified properties.
6. Repeat the steps above for other configurations.

You have set up your environment to use the C++11 lambda expression support.

See Also

[Adding Intel® oneAPI Threading Building Blocks \(oneTBB\) to your Build Environment](#)

Enable OpenMP* in your Build Environment

OpenMP* is supported by certain versions of the Microsoft Visual C++* compiler, the GNU* compilers, the Intel® C++ Compiler Classic, Intel® Fortran Compiler Classic, and Intel® oneAPI DPC++/C++ Compiler:

- Most recent versions of the Microsoft Visual C++* compiler include OpenMP support.
- Certain editions of the Intel® C++ Compiler Classic and the Intel® Fortran Compiler Classic support the `TASK` feature introduced with OpenMP 3.0.

For information about OpenMP support for the Microsoft compilers, see your Microsoft Visual Studio help. For information about OpenMP support for the GNU compilers, see your compiler help or the appropriate man page, such as `gcc(1)`.

To enable OpenMP on the command line, specify the appropriate compiler option (see your compiler documentation), such as the `-openmp` (for Linux* OS) or `/Qopenmp` (for Windows* OS) option when using the Intel compilers.

To enable OpenMP on a **Windows OS system** using Microsoft Visual Studio*:

1. In Solution Explorer, select (click) the name of one or more projects. To select multiple projects, hold down the **Ctrl** key.
2. Select **Project > Properties** or right-click the project name and select **Properties** from the pop-up menu.
3. Specify the Configuration Properties for your C/C++ or Fortran project(s):

C/C++ > Language	Specify OpenMP Support as Yes
--------------------------------	---

Fortran > Language	Specify OpenMP Support as Yes
----------------------------------	---

4. Click **OK** to save the specified properties.
5. Repeat the steps above for other configurations.
6. You should check your startup project properties before starting a build.

You have set up your environment for OpenMP support on a Windows OS system.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

NOTE

Even if you are only using the `#pragma omp` pragmas within your source, Visual C++ sources compiled with the Microsoft compilers need to `#include <omp.h>`. Otherwise, running the application will be missing a `.dll` at load time.

To include the appropriate OpenMP environment when using the Intel® Fortran Compiler Classic, specify the `use omp_lib` statement.

See Also

[Annotation Report Overview](#)

Annotation Report

Annotation Report Overview

The **Annotation Report** window displays the annotations for your program. Intel Advisor updates the listed annotations when changes occur to the specified source directories. For example, when you save a source file with a code editor.

Annotation	Source Location	Annotation Label
Site	nqueens_annotated.cpp:113	solve
111	// int * queens = new int[size]; //array representing queens placed c	
112		
113	ANNOTATE_SITE_BEGIN(solve);	
114	for(int i=0; i<size; i++) {	
115	ANNOTATE_ITERATION_TASK(setQueen);	
Site End	nqueens_annotated.cpp:122	-
Task	nqueens_annotated.cpp:115	setQueen
Lock	nqueens_annotated.cpp:88	0
Lock Release	nqueens_annotated.cpp:91	0
Intel Advisor XE annotations defi...	nqueens_annotated.cpp:47	advisor-annotate.h

The first three columns show the Annotation type, the source location, and the annotation label. To view or hide a source code snippet, click the icon in the **Annotation** column (as shown for the **Site** annotation). To display the source code associated with each annotation, either double-click in these columns or right-click and select **View Source** or **Edit Source**.

See Also

[Locating Annotations with the Annotation Report](#)

[Troubleshooting No Annotations Found](#)

[Troubleshooting Sources Not Available](#)

[Troubleshooting Debug Information Not Available](#)

Locate Annotations with the Annotation Report

The **Annotation Report** window lists all the Intel Advisor annotations found in your project and their types. Each annotation appears as a separate row in a table-like grid.

To use the list of annotations in the **Annotation Report** window to find annotations as you replace annotations with parallel framework code:

1. To display the **Annotation Report** window, click the **Annotation Report** tab or - if you are using the **Advisor Workflow** tab - click the (**View Annotations**) button below **2. Annotate Sources** or **5. Add Parallel Framework**. The annotations associated with the selected start-up project appear. If you have run the Suitability and Dependencies tools for this start-up project, the most recent relevant data also appears in their respective columns.
2. To sort the annotations by type, click the column heading Annotations. The suggested way to replace annotations is to replace lock annotations first, and then site and task annotations (this is because synchronized code without parallelism works correctly, but parallel code without synchronization works incorrectly). To show or hide a code snippet showing an annotation, click the icon next to its name in the Annotations column.
3. To open the code editor with the corresponding source file, double-click an annotation type (data row) in the Annotations column or a line in its code snippet (or use the **Edit Source** context menu item). When using the Intel Advisor GUI on Linux* OS, the editor defined by the **Options > Editor** dialog box appears with the file open at the corresponding location. When using the Intel Advisor GUI on Windows* OS, the file type association (or **Open With** dialog box) determines the editor used. When using Visual Studio*, the Visual Studio code editor appears with the file open at the corresponding location.

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

4. Read the documentation associated with the parallel framework as well as the relevant information in Intel Advisor help so you understand what parallel framework code to insert. In many cases, you need to insert parallel framework declarations at the start of the source file, as well as parallel framework code that replaces the annotations.
5. Repeat the steps above for each lock annotation.
6. Repeat the steps above for each site and task annotation.

You have used the **Annotation Report** window to help you locate and replace the Intel Advisor annotations with parallel framework code.

Replace Annotations with Intel® oneAPI Threading Building Blocks (oneTBB) Code

This topic explains the steps needed to implement parallelism proposed by the Intel Advisor annotations by adding Intel® oneAPI Threading Building Blocks (oneTBB) parallel framework code.

- Add oneTBB code to add appropriate synchronization of shared resources, using the LOCK annotations as a guide. The following topics cover the oneTBB synchronization options:
 - Intel® oneAPI Threading Building Blocks (oneTBB) Mutexes
 - Intel® oneAPI Threading Building Blocks (oneTBB) Mutex - Example
- Add code to create oneTBB tasks, using the SITE/TASK annotations as a guide. The following topics cover the oneTBB task creation options:
 - Parallelize Functions - Intel® oneAPI Threading Building Blocks (oneTBB) Tasks
 - Parallelize Data - Intel® oneAPI Threading Building Blocks (oneTBB) Counted Loops
 - Parallelize Data - Intel® oneAPI Threading Building Blocks (oneTBB) Loops with Complex Iteration Control

This is the recommended order of tasks for replacing the annotations with oneTBB code:

1. Add appropriate synchronization of shared resources, using LOCK annotations as a guide.
2. Test to verify you did not break anything, before adding the possibility of non-deterministic behavior with parallel tasks.
3. Add code to create oneTBB tasks or loops, using the SITE/TASK annotations as a guide.
4. Test with one thread, to verify that your program still works correctly.
5. Test with more than one thread to see that the multithreading works as expected.

The oneTBB parallel framework creates worker threads automatically. In general, you should concern yourself only with the tasks, and leave it to the framework to create and destroy the worker threads.

If you do need some control over creation and destruction of worker threads, read about `task_scheduler_init` in the oneTBB Reference manual.

The table below shows the serial, annotated program code in the left column and the equivalent oneTBB parallel code in the right column for some typical code to which parallelism can be applied.

Serial Code with Intel Advisor Annotations	Parallel Code using oneTBB
<pre>// Locking ANNOTATE_LOCK_ACQUIRE(); Body(); ANNOTATE_LOCK_RELEASE();</pre>	<pre>// Locking can use various mutex types provided // by oneTBB. For example: #include <tbb/tbb.h> ... tbb::mutex g_Mutex; ... { tbb::mutex::scoped_lock lock(g_Mutex); Body(); }</pre>

Serial Code with Intel Advisor Annotations	Parallel Code using oneTBB
<pre>// Do-All Counted loops, one task ANNOTATE_SITE_BEGIN(site); For (I = 0; I < N; ++){ ANNOTATE_ITERATION_TASK(task); {statement;} } ANNOTATE_SITE_END();</pre>	<pre>// Do-All Counted loops, using lambda // expressions #include <tbb/tbb.h> ... tbb::parallel_for(0,N,[&](int I) { statement; });</pre>
<pre>// Create Multiple Tasks ANNOTATE_SITE_BEGIN(site); ANNOTATE_TASK_BEGIN(task1); statement-or-task1; ANNOTATE_TASK_END(); ANNOTATE_TASK_BEGIN(task2); statement-or-task2; ANNOTATE_TASK_END(); ANNOTATE_SITE_END();</pre>	<pre>// Create Multiple tasks, using lambda // expressions #include <tbb/tbb.h> ... tbb::parallel_invoke([&]{statement-or-task1;}, [&]{statement-or-task2;});</pre>

For information about common parallel programming patterns and how to implement them in oneTBB, see the oneTBB help topic Design Patterns.

Intel® oneAPI Threading Building Blocks (oneTBB) Mutexes

With Intel® oneAPI Threading Building Blocks (oneTBB) , you can associate a *mutex* with a shared object to enforce mutually exclusive access to that object. A mutex is either locked or unlocked. For a thread to safely access the object:

- The thread *acquires a lock* on the mutex.
- The thread accesses the associated shared object.
- The thread *releases its lock* on the mutex.

When a mutex is locked, if another thread tries to also acquire a lock on it, this second thread is stalled until the first thread releases its lock on the mutex. This functionality provided by a mutex is exactly the semantic function intended by the Intel Advisor annotations `ANNOTATE_LOCK_ACQUIRE()` and `ANNOTATE_LOCK_RELEASE()`.

With oneTBB , the annotation lock address becomes the mutex object. The `ANNOTATE_LOCK_ACQUIRE()` and `ANNOTATE_LOCK_RELEASE()` annotations become operations on this mutex.

oneTBB provides several classes for locking, each with different properties. For more information, refer to the oneTBB documentation. If you are not sure what type of a mutex is most appropriate, consider using `tbb::mutex` as your initial choice.

See Also

[Intel® oneAPI Threading Building Blocks \(oneTBB\) Simple Mutex - Example](#)

Intel® oneAPI Threading Building Blocks (oneTBB) Simple Mutex - Example

The following examples shows basic usage of a Intel® oneAPI Threading Building Blocks (oneTBB) mutex to protect a shared variable named `count` using simple mutexes and scoped locks:

Simple Mutex Example

```
#include <tbb/mutex.h>

int count;
```

```
tbb::mutex countMutex;

int IncrementCount() {
    int result;
    // Add oneTBB mutex
    countMutex.lock();      // Implements ANNOTATE_LOCK_ACQUIRE()
    result = count++;       // Save result until after unlock
    countMutex.unlock();    // Implements ANNOTATE_LOCK_RELEASE()
    return result;
}
```

The semantics of `countMutex.lock()` and `unlock()` on `countMutex` correspond directly to the annotations `ANNOTATE_LOCK_ACQUIRE()` and `ANNOTATE_LOCK_RELEASE()`. However, it is generally better to use the *scoped locking* pattern.

Scoped Lock Example

With a scoped lock, you construct a temporary *scoped_lock* object that represents acquisition of a lock. Destruction of the *scoped_lock* object releases the lock on the mutex.

The following code shows the previous example rewritten using scoped locking:

```
#include <tbb/mutex.h>
int      count;
tbb::mutex countMutex;

int IncrementCount() {
    int result;
    {
        // Add oneTBB scoped lock at location of ANNOTATE_LOCK annotations
        tbb::mutex::scoped_lock lock(countMutex); // Implements ANNOTATE_LOCK_ACQUIRE()
        result = count++;
        // Implicit ANNOTATE_LOCK_RELEASE() when leaving the scope below.
    } // scoped lock is automatically released here
    return result;
}
```

The *scoped_lock* pattern is preferred because it releases the lock no matter how control leaves the block. The scoped lock is released when destruction of the *scoped_lock* object occurs. In particular, it releases the lock even when control leaves because an exception was thrown.

oneTBB also has a `tbb::atomic` template class that can be used in simple cases such as managing a shared integer variable. Check the Related Information for details.

See Also

[Testing the Intel® oneAPI Threading Building Blocks \(oneTBB\) Synchronization Code](#)
[Related Information](#)

Test the Intel® oneAPI Threading Building Blocks (oneTBB) Synchronization Code

After you add Intel® oneAPI Threading Building Blocks (oneTBB) synchronization code (such as mutexes), but before adding the constructs that cause the program to use parallel execution, you should test your serial program. The synchronization code may introduce problems if you have inadvertently used a non-recursive mutex in a recursive context, or if your edits accidentally changed some other piece of program behavior.

It is much easier to find these problems in the serial version of your program than it will be in the parallel version.

See Also

[Parallelize Functions - Intel® oneAPI Threading Building Blocks \(oneTBB\) Tasks](#)

Parallelize Functions - Intel® oneAPI Threading Building Blocks (oneTBB) Tasks

The following sections describe various alternatives, depending on how the tasks fit within the surrounding parallel site.

Two or More Parallel Statements

When the outermost statements in the annotation site have been placed into tasks, as shown in this serial example, it is easy to execute them in parallel.

```
ANNOTATE_SITE_BEGIN(sitename);
    ANNOTATE_TASK_BEGIN(task1);
        statement_1
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(task2);
        statement_2
    ANNOTATE_TASK_END();
ANNOTATE_SITE_END();
```

Two or More Parallel Statements - Intel® oneAPI Threading Building Blocks (oneTBB)

The easiest way to cause several sequential statements to be executed as independent tasks is to change your program as follows using `parallel_invoke`.

Both of the following examples use the C++11 lambda expression feature - you need to use the Intel® C++ Compiler Classic or Intel® oneAPI DPC++/C++ Compiler and enable the C++11 support to compile it.

```
#include <tbb/tbb.h>

...
tbb::parallel_invoke(
    [&]{statement_1;},
    [&]{statement_2;}
)
```

A variable used inside a lambda expression but declared outside it is said to be *captured*. The `[&]` in the example specifies capture by reference. It is also possible to capture by value `[=]`, or even capture different variables different ways. See the compiler documentation on lambda expressions for details.

Using C++ structs Instead of Lambda Expressions

Any code that can be written with a lambda expression can be written without one - it is just more work. All a lambda expression does is:

1. Define a class with `operator()` defined to execute the body of the lambda expression.
2. Define a class constructor that captures variables into fields of the class.
3. Construct an instance of that class.

The constructor can capture any of the surrounding locals that are needed and save them in data members.

```
{ struct S1 { void operator()() { statement_1 } };
  struct S2 { void operator()() { statement_2 } };
  tbb::parallel_invoke(S1(), S2());
}
```

See Also

[Parallelize Data - Intel® oneAPI Threading Building Blocks \(oneTBB\) Counted Loops](#)

Parallelize Data - Intel® oneAPI Threading Building Blocks (oneTBB) Counted Loops

When tasks are loop iterations, and the iterations are over a range of values that are known before the loop starts, the loop is easily expressed in Intel® oneAPI Threading Building Blocks (oneTBB) .

Consider the following serial code and the need to add parallelism to this loop:

```
ANNOTATE_SITE_BEGIN(sitename);
    for (int i = lo; i < hi; ++i) {
        ANNOTATE_ITERATION_TASK(taskname);
        statement;
    }
ANNOTATE_SITE_END();
```

Here is the serial example converted to use oneTBB , after you remove the Intel Advisor annotations:

```
#include <tbb/tbb.h>
...
tbb::parallel_for( lo, hi,
    [&](int i) {statement;}
);
```

The first two parameters are the loop bounds. As is typical in C++ (especially STL) programming, the lower bound is inclusive and the upper bound is exclusive. The third parameter is the loop body, wrapped in a lambda expression. The loop body will be called in parallel by threads created by oneTBB . As described before in Create the Tasks, Using C++ structs Instead of Lambda Expressions, the lambda expressions can be replaced with instances of explicitly defined class objects.

See Also

[Parallelize Data - Intel® oneAPI Threading Building Blocks \(oneTBB\) Loops with Complex Iteration Control](#)

[Parallelize Functions - Intel® oneAPI Threading Building Blocks \(oneTBB\) Tasks](#) for information on using C++ structs instead of lambda functions

Parallelize Data - Intel® oneAPI Threading Building Blocks (oneTBB) Loops with Complex Iteration Control

Sometimes the loop control is spread across complex control flow. Using Intel® oneAPI Threading Building Blocks (oneTBB) in this situation requires more features than the simple loops. Note that the task body must not access any of the auto variables defined within the annotation site, because they may have been destroyed before or while the task is running. Consider this serial code:

```
extern char a[];
int previousEnd = -1;
ANNOTATE_SITE_BEGIN(sitename);
    for (int i=0; i<=100; i++) {
        if (!a[i] || i==100) {
            ANNOTATE_TASK_BEGIN(do_something);
            DoSomething(previousEnd+1,i);
            ANNOTATE_TASK_END();
            previousEnd=i;
        }
    }
ANNOTATE_SITE_END();
```

In general, counted loops have better scalability than loops with complex iteration control, because the complex control is inherently sequential. Consider reformulating your code as a counted loop if possible.

The prior example is easily converted to parallelism by using the `task_group` feature of oneTBB :

```
#include <tbb/tbb.h>
...
extern char a[];
```

```

int previousEnd = -1;
task_group g;
for (int i=0; i<=100; i++) {
    if (!a[i] || i==100) {
        g.run( [=]{DoSomething(previousEnd+1,i);} );
        previousEnd=i;
    }
}
g.wait(); // Wait until all tasks in the group finish

```

Here the lambda expression uses capture by value [=] because it is important for it to grab the values of `i` and `previousEnd` when the expression constructs its functor, because afterwards the value of `previousEnd` and `i` change.

For more information on `tbb::task_group`, see the [oneTBB documentation](#).

See Also

[Using Intel® Inspector and Intel® VTune™ Profiler](#)

Replace Annotations with OpenMP* Code

This topic explains the steps needed to implement parallelism proposed by the Intel Advisor annotations by adding OpenMP* parallel framework code.

- Add OpenMP code to provide appropriate synchronization of shared resources, using the LOCK annotations as a guide.
- Add code to create OpenMP tasks, using the SITE/TASK annotations as a guide.

The recommended order for replacing the annotations with OpenMP code:

1. Add appropriate synchronization of shared resources, using LOCK annotations as a guide.
2. Test to verify you did not break anything, before adding the possibility of non-deterministic behavior with parallel tasks.
3. Add code to create OpenMP parallel sections or equivalent, using the SITE/TASK annotations as a guide.
4. Test with one thread to verify that your program still works correctly. For example, set the environment variable `OMP_NUM_THREADS` to 1 before you run your program.
5. Test with more than one thread to see that the multithreading works as expected.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well your OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

OpenMP creates worker threads automatically. In general, you should concern yourself only with the tasks, and leave it to the parallel frameworks to create and destroy the worker threads.

If you do need some control over creation and destruction of worker threads, see the compiler documentation. For example, to limit the number of threads, set the `OMP_THREAD_LIMIT` or the `OMP_NUM_THREADS` environment variable.

The table below shows the serial, annotated program code in the left column and the equivalent OpenMP C/C++ and Fortran parallel code in the right column for some typical code to which parallelism can be applied.

Serial C/C++ and Fortran Code with Intel Advisor Annotations	Parallel C/C++ and Fortran Code using OpenMP
<pre> // Synchronization, C/C++ ANNOTATE_LOCK_ACQUIRE(0); Body(); ANNOTATE_LOCK_RELEASE(0); </pre>	<pre> // Synchronization can use OpenMP // critical sections, atomic operations, locks, // and reduction operations (shown later) </pre>

Serial C/C++ and Fortran Code with Intel Advisor Annotations	Parallel C/C++ and Fortran Code using OpenMP
<pre>! Synchronization, Fortran call annotate_lock_acquire(0) body call annotate_lock_release(0)</pre>	<pre>// Synchronization can use OpenMP // critical sections, atomic operations, locks, // and reduction operations (shown later)</pre>
<pre>// Parallelize data - one task within a // C/C++ counted loop ANNOTATE_SITE_BEGIN(site); for (i = lo; i < n; ++i) { ANNOTATE_ITERATION_TASK(task); statement; } ANNOTATE_SITE_END();</pre>	<pre>// Parallelize data - one task, C/C++ counted loops #pragma omp parallel for for (int i = lo; i < n; ++i) { statement; }</pre>
<pre>! Parallelize data - one task within a ! Fortran counted loop call annotate_site_begin("site1") do i = 1, N call annotate_iteration_task("task1") statement end do call annotate_site_end</pre>	<pre>! Parallelize data - one task with a ! Fortran counted loop !\$omp parallel do do i = 1, N statement end do !\$omp end parallel do</pre>
<pre>// Parallelize C/C++ functions ANNOTATE_SITE_BEGIN(site); ANNOTATE_TASK_BEGIN(task1); function_1(); ANNOTATE_TASK_END(); ANNOTATE_TASK_BEGIN(task2); function_2(); ANNOTATE_TASK_END(); ANNOTATE_SITE_END();</pre>	<pre>// Parallelize C/C++ functions #pragma omp parallel //start parallel region { #pragma omp sections { #pragma omp section function_1(); #pragma omp section function_2(); } } // end parallel region</pre>
<pre>! Parallelize Fortran functions call annotate_site_begin("site1") call annotate_task_begin("task1") call subroutine_1 call annotate_task_end call annotate_task_begin("task2") call subroutine_2 call annotate_task_end call annotate_site_end</pre>	<pre>! Parallelize Fortran functions !\$omp parallel ! start parallel region !\$omp sections !\$omp section call subroutine_1 !\$omp section call subroutine_2 !\$omp end sections !\$omp end parallel ! end parallel region</pre>

Add OpenMP Code to Synchronize the Shared Resources

OpenMP provides several forms of synchronization:

- A *critical section* prevents multiple threads from accessing the critical section's code at the same time, thus only one active thread can update the data referenced by the code. A critical section may consist of one or more statements. To implement a critical section:
 - With C/C++: `#pragma omp critical`
 - With Fortran: `!$omp critical` and `!$omp end critical`

Use the optional named form for a non-nested mutex, such as (C/C++) `#pragma omp critical(name)` or (Fortran) `!$omp critical(name)` and `!$omp end critical(name)`. If the optional (name) is omitted, it locks a single unnamed global mutex. The easiest approach is to use the unnamed form unless performance measurement shows this shared mutex is causing unacceptable delays.

- An *atomic operation* allows multiple threads to safely update a shared numeric variable on hardware platforms that support its use. An atomic operation applies to only one assignment statement that immediately follows it. To implement an atomic operation:

- With C/C++: insert a `#pragma omp atomic` before the statement to be protected.
- With Fortran: insert a `!$omp atomic` before the statement to be protected.

The statement to be protected must meet certain criteria (see your compiler or OpenMP documentation).

- *Locks* provide a low-level means of general-purpose locking. To implement a lock, use the OpenMP types, variables, and functions to provide more flexible and powerful use of locks. For example, use the `omp_lock_t` type in C/C++ or the `type=omp_lock_kind` in Fortran. These types and functions are easy to use and usually directly replace Intel Advisor lock annotations.
- *Reduction operations* can be used for simple cases, such as incrementing a shared numeric variable or summing an array into a shared numeric variable. To implement a reduction operation, add the `reduction` clause within a parallel region to instruct the compiler to perform the summation operation in parallel using the specified operation and variable.
- OpenMP provides other synchronization techniques, including specifying a *barrier* construct where threads will wait for each other, an *ordered* construct that ensures sequential execution of a structured block within a parallel loop, and *master* regions that can only be executed by the master thread. For more information, see your compiler or OpenMP documentation.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

The following topics briefly describe these forms of synchronization. Check your compiler documentation for details.

See Also

Testing the OpenMP Synchronization Code

OpenMP Critical Sections

Use OpenMP critical sections to prevent multiple threads from accessing the critical section's code at the same time, thus only one active thread can update the data referenced by the code. Critical sections are useful for a non-nested mutex.

Unlike [OpenMP atomic operations](#) that provide fine-grain synchronization for a single operation, critical sections can provide course-grain synchronization for multiple operations.

Use:

- `#pragma omp critical` with C/C++.
- `!$omp critical` and `!$omp end critical` with Fortran.

If the optional (name) is omitted, it locks an unnamed global mutex. The easiest approach is to use the unnamed form unless this shared mutex is causing unacceptable performance delays.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

For example, consider this annotated C/C++ serial program:

```
int count;
void Tick() {
    ANNOTATE_LOCK_ACQUIRE(0);
    count++;
    ANNOTATE_LOCK_RELEASE(0);
}
. . .
```

The parallel C/C++ code after adding `#include <omp.h>` and `#pragma omp critical`:

```
#include <omp.h> //prevents a load-time problem with a .dll not being found
int count;
void Tick() {
    // Replace Lock annotations
    #pragma omp critical
    count++;
}
. . .
```

Consider this annotated Fortran serial code:

```
program ABC
    integer(kind=4) :: count = 0
    . . .
contains
subroutine Tick
    call annotate_lock_acquire(0)
    count = count + 1
    call annotate_lock_release(0)
end subroutine Tick
. . .
end program ABC
```

The parallel Fortran code after adding `use omp_lib`, `!$omp critical`, and `!$omp end critical`:

```
program ABC
    use omp_lib
    integer(kind=4) :: count = 0
    . . .
contains
subroutine Tick
    !$omp critical
    count = count + 1
    !$omp end critical
end subroutine Tick
. . .
end program ABC
```

See Also

[Testing the OpenMP Synchronization Code](#)
[Related Information](#)

Basic OpenMP Atomic Operations

Use OpenMP atomic operations to allow multiple threads to safely update a shared numeric variable, such as on hardware platforms that support atomic operation use. An atomic operation applies only to the single assignment statement that immediately follows it, so atomic operations are useful for code that requires fine-grain synchronization.

Before the statement to be protected, use:

- `#pragma omp atomic` for C/C++.
- `!$omp atomic` for Fortran.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

For example, consider this annotated C/C++ serial code:

```
int count;
void Tick() {
    ANNOTATE_LOCK_ACQUIRE(0);
    count = count+1;
    ANNOTATE_LOCK_RELEASE(0);
}
. . .
```

The parallel C/C++ code after adding `#include <omp.h>` and `#pragma omp atomic`:

```
#include <omp.h> //prevents a load-time problem with a .dll not being found
int count;
void Tick() {
    // Replace lock annotations
    #pragma omp atomic
    count = count+1;
}
. . .
```

Consider this annotated Fortran serial code:

```
program ABC
    integer(kind=4) :: count = 0
    . . .
contains
    subroutine Tick
        call annotate_lock_acquire(0)
        count = count + 1
        call annotate_lock_release(0)
    end subroutine Tick
    . . .
end program ABC
```

The parallel Fortran code after adding `use omp_lib` and the `!$omp atomic` directive:

```
program ABC
    use omp_lib
    integer(kind=4) :: count = 0
    . . .
contains
```

```

subroutine Tick
  !$omp atomic
  count = count + 1
end subroutine Tick
. . .
end program ABC

```

The Intel Advisor Fortran sample `nqueens.f90` demonstrates the use of an atomic operation.

This topic introduces basic OpenMP atomic operations. For advanced atomic operations that use clauses after the `atomic` construct, see [Advanced OpenMP Atomic Operations](#).

See Also

[Advanced OpenMP Atomic Operations](#)

[Testing the OpenMP Synchronization Code](#)

[Related Information](#)

Advanced OpenMP Atomic Operations

This topic provides advanced examples of OpenMP* atomic operations.

These advanced atomic operations use clauses after the `atomic` construct, such as `read`, `write`, `update`, `capture`, and `seq_cst`. If you do not add a clause after `atomic`, the default is `update`.

Because these clauses are part of OpenMP 3.1 and 4.0 specification, you need a compiler that supports these advanced atomic clauses, such as the Intel® C++ Compiler Classic or the Intel® Fortran Compiler Classic.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

Example Using the `read` and `write` Clauses

The following C/C++ example uses separate `read` and `write` clauses:

```

int atomic_read(const int *x)
{
  int value;
  /* Ensure that the entire value of *x is read atomically. */
  /* No part of *x can change during the read operation. */
#pragma omp atomic read
  value = *x;
  return value;
}

void atomic_write(int *x, int value)
{
  /* Ensure that value is stored atomically into *x. */
  /* No part of *x can change until after the entire write operation has completed. */
#pragma omp atomic write
  *x = value;
}

```

The following Fortran example uses the `read` and `write` clauses:

```

function atomic_read(x)
  integer :: atomic_read
  integer, intent(in) :: x
! Ensure that the entire value of x is read atomically. No part of x can change during
! the read operation.

```

```

!$omp atomic read
  atomic_read = x
  return
end function atomic_read
subroutine atomic_write(x, value)
  integer, intent(out) :: x
  integer, intent(in) :: value
  ! Ensure that value is stored atomically into x. No part of x can change
  ! until after the entire write operation has completed.
  !$omp atomic write
  x = value
end subroutine atomic_write

```

Example Using the Basic `capture` Clause

The following C/C++ example uses the `capture` clause:

```

#pragma omp parallel for shared (pos)
  for (int i=0; i < size; i++) {

      if (isValid(data[i])) {
          int tmpPos;
          // Using omp atomic capture pragma
          #pragma omp atomic capture
          {
              tmpPos = pos;
              pos = pos+1;
          }
          //Pack all selected element' indices in index; exact order of indices values is
not important.
          index[tmpPos] = i;
      }
  }

```

Example Using the Swap Form of the `capture` Clause

The `capture` clause example above might be modified to use the following code snippet:

```

//with introduction of "atomic swap" you can also use forms like:
    newPos = foo();

    .
    .
    .
#pragma omp atomic capture
    {
        tmpPos = pos;
        pos = newPos;
    }

```

See Also

[Testing the OpenMP Synchronization Code](#)

[Basic OpenMP Atomic Operations](#)

[Related Information](#)

OpenMP Reduction Operations

OpenMP reduction operations can be used for simple cases, such as incrementing a shared numeric variable or the summation of an array into a shared numeric variable. To implement a reduction operation, add the `reduction` clause within a parallel region to instruct the compiler to perform the summation operation in parallel using the specified operation and variable.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

Consider this annotated C/C++ serial code:

```
int i, n=500000;
float *array, total=0.0;
...
for (i=0; i <n ; ++i
{
    ANNOTATE_LOCK_ACQUIRE(0);
    total+ = array[i];
    ANNOTATE_LOCK_RELEASE(0);
}
. . .
```

The parallel C/C++ code after adding `#include <omp.h>` and `#pragma omp parallel` for reduction:

```
#include <omp.h> //prevents a load-time problem with a .dll not being found
int i, n=500000;
float *array, total=0.0;
...
#pragma omp parallel for reduction(+:total)
    for (i=0; i <n ; ++i
    {
        total+ = array[i];
    }
. . .
```

Consider this annotated Fortran serial code:

```
integer(4) n
real(4) array(50000), total = 0.0
n = 500000
...
do i=1, n
    call annotate_lock_acquire(0)
    total = total + array(i)
    call annotate_lock_release(0)
. . .
end do
```

Consider this parallel Fortran code after adding `use omp_lib, !$omp parallel do reduction(+:total),` and `!$omp end parallel do`:

```
use omp_lib
integer(4) n
real(4) array(50000), total = 0.0
```

```

n = 500000
...

!$omp parallel do reduction(+:total)
  do i=1, n
    total = total + array(i)
!$omp end parallel do
. . .
  end do

```

See Also

[Testing the OpenMP Synchronization Code](#)

[Related Information](#)

OpenMP Locks

Consider the following annotated C/C++ serial code:

```

int count;
void Tick() {
  ANNOTATE_LOCK_ACQUIRE(0);
  count++;
  ANNOTATE_LOCK_RELEASE(0);
}

```

To implement a lock, use the OpenMP types, variables, and functions to provide more flexible and powerful use of locks. For example, for simple locks, use the `omp_lock_t` type in C/C++ or the `type=omp_lock_kind` in Fortran.

Locks can be wrapped inside C++ classes, as shown in the following parallel C/C++ code:

```

#include <omp.h>
int count;
omp_lock_t countMutex;

struct CountMutexInit {
  CountMutexInit() { omp_init_nest_lock (&countMutex); }
  ~CountMutexInit() { omp_destroy_nest_lock(&countMutex); }
} countMutexInit;

// The declaration of the above object causes countMutex to
// be initialized on program startup, and to be destroyed when
// the program completes, via the constructor and destructor.

struct CountMutexHold {
  CountMutexHold() { omp_set_nest_lock (&countMutex); }
  ~CountMutexHold() { omp_unset_nest_lock (&countMutex); }
};

void Tick() {
  // unlocks on scope exit
  CountMutexHold releaseAtEndOfScope;
  count++;
}
...

```

Consider the following annotated Fortran serial code:

```
program BBB
  integer(kind=4) :: count = 0
  . . .
contains
  subroutine Tick
    call annotate_lock_acquire(0)
    count = count + 1
    call annotate_lock_release(0)
  end subroutine Tick
  . . .
end program BBB
```

For simple locks with Fortran code, use the `type=omp_lock_kind`. The parallel Fortran code follows after adding `use omp_lib` and the integer declaration for `count`:

```
program BBB
  use omp_lib
  integer(kind=4) :: count = 0
  integer (kind=omp_lock_kind) countMutex
  call omp_nest_lock_init(countMutex)
  . . .

contains
  subroutine Tick
    call omp_set_nest_lock(countMutex)
    count = count + 1
    call omp_unset_nest_lock(countMutex)
  end subroutine Tick
  . . .
end program BBB
```

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

See Also

[Testing the OpenMP Synchronization Code](#)

[Related Information](#)

Test the OpenMP Synchronization Code

After you have added OpenMP synchronization code (such as locks, critical sections, or atomic operations), but before adding the constructs that cause the program to use parallel execution, you should test your serial program. The synchronization code may introduce problems if you have inadvertently used a non-recursive mutex in a recursive context, or if your edits accidentally changed some other piece of program behavior.

It is much easier to find these problems in the serial version of your program than it will be in the parallel version.

See Also

[Parallelize Functions - OpenMP Tasks](#)

Parallelize Functions - OpenMP Tasks

You can enable multiple function calls to run in parallel as two or more tasks. This is useful for functions in library code for which the source is not available. The statements to run in parallel are not limited to function calls (see the help topic [Data and Task Parallelism](#)).

When the outermost statements in the annotation site have been placed into tasks, as shown in the following serial example, it is easy to execute them in parallel.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

Consider the C/C++ annotated code:

```
ANNOTATE_SITE_BEGIN(sitename);
    ANNOTATE_TASK_BEGIN(task1);
        statement-1;
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(task2);
        statement-2;
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(task3);
        statement-3;
    ANNOTATE_TASK_END();
ANNOTATE_SITE_END();
```

For the C/C++ parallel code, OpenMP provides explicit support using `#pragma omp parallel` sections and related pragmas within a parallel code region:

```
#pragma omp parallel sections
{
    #pragma omp section
    {
        statement-1;
    }
    #pragma omp section
    {
        statement-2;
    }
    . . .
}
```

Consider the annotated Fortran code:

```
call annotate_site_begin("sitename")
call annotate_task_begin("task_1")
call subroutine_1
call annotate_task_end

call annotate_task_begin("task_2")
call subroutine_2
call annotate_task_end
call annotate_site_end
...
```

For the parallelized Fortran code, OpenMP provides the `!$omp sections` and related directives that can often replace the corresponding annotations within a parallel code region:

```
!$omp parallel
!$omp sections
!$omp section
    call subroutine_1
!$omp section
    call subroutine_2
!$omp end sections
!$omp end parallel
...
```

See Also

[Parallelize Data - OpenMP Counted Loops](#)
[Data and Task Parallelism](#)

Parallelize Data - OpenMP Counted Loops

When tasks are loop iterations, and the iterations are over a range of values that are known before the loop starts, the loop is easily expressed in OpenMP.

Consider the following annotated serial C/C++ loop:

```
ANNOTATE_SITE_BEGIN(sitename);
    for (int i = lo; i < hi; ++i) {
        ANNOTATE_ITERATION_TASK(taskname);
        statement;
    }
ANNOTATE_SITE_END();
```

OpenMP makes it easy to introduce parallelism into loops. With C or C++ programs, add the `omp parallel for pragma` immediately before the C/C++ `for` statement:

```
...
#pragma omp parallel for
    for (int i = lo; i < hi; ++i) {
        statement;
    }
```

Consider the following annotated Fortran serial loop:

```
call annotate_site_begin("sitename")

do i = 1, N
    call annotate_iteration_task("taskname")
    statement
end do

call annotate_site_end
```

With Fortran programs, add the `!$omp parallel do` directive immediately before the Fortran `do` statement:

```
...
!$omp parallel do
    do i = 1, N
        statement
    end do
!$omp end parallel do
```

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

The OpenMP compiler support encapsulates the parallel construct. The rules for capturing the locals can be defaulted or specified as part of the pragma or directive. The loop control variable defaults to being private so each iteration sees its allotted value.

See Also

[Parallelize Data - OpenMP Loops with Complex Iteration Control](#)

[Parallelize Functions - OpenMP Tasks](#)

Parallelize Data - OpenMP Loops with Complex Iteration Control

Sometimes the loop control is spread across complex control flow. Using OpenMP in this situation requires more features than the simple loops. The task body must not access any of the auto variables defined within the annotation site, because they may have been destroyed before or while the task is running. Also, note that variables referenced within the `task` construct default to `firstprivate`.

Consider this serial C/C++ code:

```
extern char a[];
int previousEnd = -1;
ANNOTATE_SITE_BEGIN(sitename);
for (int i=0; i<=100; i++) {
    if (!a[i] || i==100) {
        ANNOTATE_TASK_BEGIN(do_something);
        DoSomething(previousEnd+1,i);
        ANNOTATE_TASK_END();
        previousEnd=i;
    }
}
ANNOTATE_SITE_END();
```

This is done using the OpenMP `task` pragma. Without using this feature, such loops are extremely difficult to parallelize. One approach to the adding parallelism to the loop is to simply spawn each call to `DoSomething()`:

```
extern char a[];
int previousEnd = -1;
#pragma omp parallel
{
    #pragma omp single
    {
        ...
        for (int i=0; i<=100; i++) {

            if (!a[i] || i==100)
            {
                #pragma omp task
                DoSomething(previousEnd+1,i);
            }
        }
    }
}
```

It is important that the parameters to `DoSomething` be passed by value, not by reference, because `previousEnd` and `i` can change before or while the spawned task runs.

Consider this serial Fortran code:

```
. . .
logical(1) a(200)
integer(4) i, previousEnd
...
previousEnd=0
call annotate_site_begin(functions)
do i=1,101
  if a(.not. a(i)) .or. (i .eq. 101) then
    call annotate_task_begin(do_something)
    call DoSomething(previousEnd+1, i)
    call annotate_task_end
  endif
end do
call annotate_site_end
```

This is easily done using the OpenMP `task` directive. Without using this feature, such loops are extremely difficult to parallelize. One approach to the parallelize the above loop is simply to spawn each call to `DoSomething()`:

```
. . .
logical(1) a(200)
integer(4) i, previousEnd
...
previousEnd=0
!$omp parallel
!$omp single
do i=1,101
  if a(.not. a(i)) .or. (i .eq. 101) then
    !$omp task
    call DoSomething(previousEnd+1, i)
    !$omp end task
  endif
end do
!$omp end parallel
```

There is no requirement that the `omp task` pragma or directive be within the surrounding parallel directive's static extent.

Tip After you rewrite your code to use OpenMP* parallel framework, you can analyze its performance with Intel® Advisor perspectives. Use the [Vectorization and Code Insights](#) perspective to analyze how well you OpenMP code is vectorized or use the [Offload Modeling](#) perspective to model its performance on a GPU.

See Also

[Using Intel® Inspector and Intel® VTune™ Profiler](#)

Next Steps for the Parallel Program

After you add parallel framework code to your program, use the related Intel® www products to check for parallel thread errors and improve the performance of your parallel program. Tips for debugging parallel code are also provided.

Use Intel® Inspector and Intel® VTune™ Profiler

Intel® Advisor helps you:

- Discover where to add parallelism to your program by identifying where your program spends its time. You propose parallel code regions when you annotate the parallel sites and tasks.
- Predict the performance you might achieve with the proposed parallel code regions.
- Predict the data sharing problems that could occur in the proposed parallel code regions.

Intel Advisor does not catch all problems, and it cannot ensure that you have correctly implemented the parallelism. Before deploying your parallel program, you need to test it for Dependencies and verify its performance. To do this, you can use analyzer tools provided in the Intel® oneAPI Base Toolkit, Intel® oneAPI IoT Toolkit, and Intel® oneAPI HPC Toolkit.

The thread error analysis provided by the [Intel® Inspector](#) and the Dependencies analysis provided by the Intel Advisor use similar technology. Intel Inspector includes a data race and deadlock detection tool that works on the *parallel* code. It can find more errors because it operates on the parallel code instead of working on the annotated *serial* code analyzed by the Dependencies tool. Intel Inspector also can find problems with memory: memory leaks, references to freed storage, references to uninitialized memory, and so forth. The memory-checking tool works on serial or parallel code.

Similarly, the Intel Advisor Survey and Suitability tools provide features found in the [Intel® VTune™ Profiler](#). The Survey tool profiles your program to find hotspots and the Suitability tool makes predictions of approximate parallel performance including overhead costs based on the Intel Advisor annotations. When you have a working parallel program, you should use Intel VTune Profiler to measure the parallel program gain and core utilization, as well as check whether the parallel framework overhead is acceptable.

Once you have parallel code, you should:

- Measure the speedup.
- Make adjustments if locks are causing excessive delays, or if one task runs much longer than others.

Intel VTune Profiler has many features to help you find and fix performance problems in your parallel code. It also helps you check:

- Where are the hotspots now?
- Am I missing opportunities for more parallelism?
- Is my program spending a lot of time waiting?
- How does the performance compare to that of prior versions?

Another technique is to use a debugger to debug a serial version of your parallel program with the parallel constructs in reverse order (see [Debug Parallel Programs](#)).

See Also

Maintain Results

Debug Parallel Programs

Your program might have bugs that are now being exposed during parallel execution because of changes in order, memory allocation, uninitialized memory contents, and so on.

Such bugs are debugged in the same way as a serial (single-threaded) application, with the following challenges:

1. The program does not run in exactly the same order each time. Possible causes include:
 - Locks may be acquired first by different threads.
 - Pointers returned by `new` and `malloc` may differ from one run to the next.
 - Random number sequences observed by a thread may differ from those observed in the serial version, and from run to run.
 - Items removed from a shared list by a thread may differ from run to run
2. The debugger can interact in strange manners with the threads.
 - Breakpoints can appear to be hit multiple times by a thread, even though the thread only make progress through the breakpoint on last hit of the series.
3. Thread local storage can be difficult to examine.

To determine whether you can reproduce the bug with a single thread, run the parallel version of your program as a serial program by limiting the number of threads to 1. Use such methods as setting an environment variable before you run your program or by using the Intel® oneAPI Threading Building Blocks (oneTBB) `tbb::task_scheduler_init init(1);` object.

Before spending a significant amount of time debugging the parallel code, you should try running the parallel loops and other parallel constructs as serial code but in reverse order. This may expose the bugs caused by your program depending on the order of execution of these statements, without requiring you to debug a parallel program.

Debug the Remaining Sharing Problems

After your program works in serial mode, and in serial mode with the parallel constructs in reverse order, use the Intel® Inspector tool to find any remaining conflicts.

See Also

[Use Intel Inspector and Intel® VTune™ Profiler](#)

[Use Partially Parallel Programs with Intel Advisor Tools](#)

Model Offloading to a GPU

Find high-impact opportunities to offload/run your code and identify potential performance bottlenecks on a target graphics processing unit (GPU) by running the Offload Modeling perspective.

The Offload Modeling perspective can help you to do the following:

- For code running on a CPU, determine if you should offload it to a target device and estimate a potential speedup before getting a hardware.
- For code running on a GPU, estimate a potential speedup from running it on a different target device before getting a hardware.
- Identify loops that are recommended for offloading from a baseline CPU to a target GPU.
- Pinpoint potential performance bottlenecks on the target device to decide on optimization directions.
- Check how effectively data can be transferred between host and target devices.

With the Offload Modeling perspective, the following workflows are available:

- *CPU-to-GPU offload modeling:*
 - For C, C++, and Fortran applications: Analyze an application and model its performance on a target GPU device. Use this workflow to find offload opportunities and prepare your code for efficient offload to the GPU.
 - For SYCL, OpenMP* target, and OpenCL™ applications: Analyze an application *offloaded to a CPU* and model its performance on a target GPU device. Use this workflow to understand how you can improve performance of your application on the target GPU and check if your code has other offload opportunities. This workflow analyzes parts of your application running on host and offloaded to a CPU.
- *GPU-to-GPU offload modeling for SYCL, OpenMP target, and OpenCL applications:* Analyze an application *that runs on a GPU* and model its performance on a different GPU device. Use this workflow to understand how you can improve your application performance and check if you can get a higher speedup if you offload the application to a different GPU device.

NOTE You can model application performance only on Intel® GPUs.

How It Works

The Offload Modeling perspective runs the following steps:

1. Get the baseline performance data for your application by running a **Survey** analysis.

2. Identify the number of times kernels are invoked and executed and the number of floating-point and integer operations, estimate cache and memory traffics on target device memory subsystem by running the **Characterization** analysis.
3. Mark up loops of interest and identify loop-carried dependencies that might block parallel execution by running the **Dependencies** analysis (CPU-to-GPU modeling only).
4. Estimate the total program speedup on a target device and other performance metrics according to Amdahl's law, considering speedup from the most profitable regions by running **Performance Modeling**. A region is profitable if its execution time on the target is less than on a host.

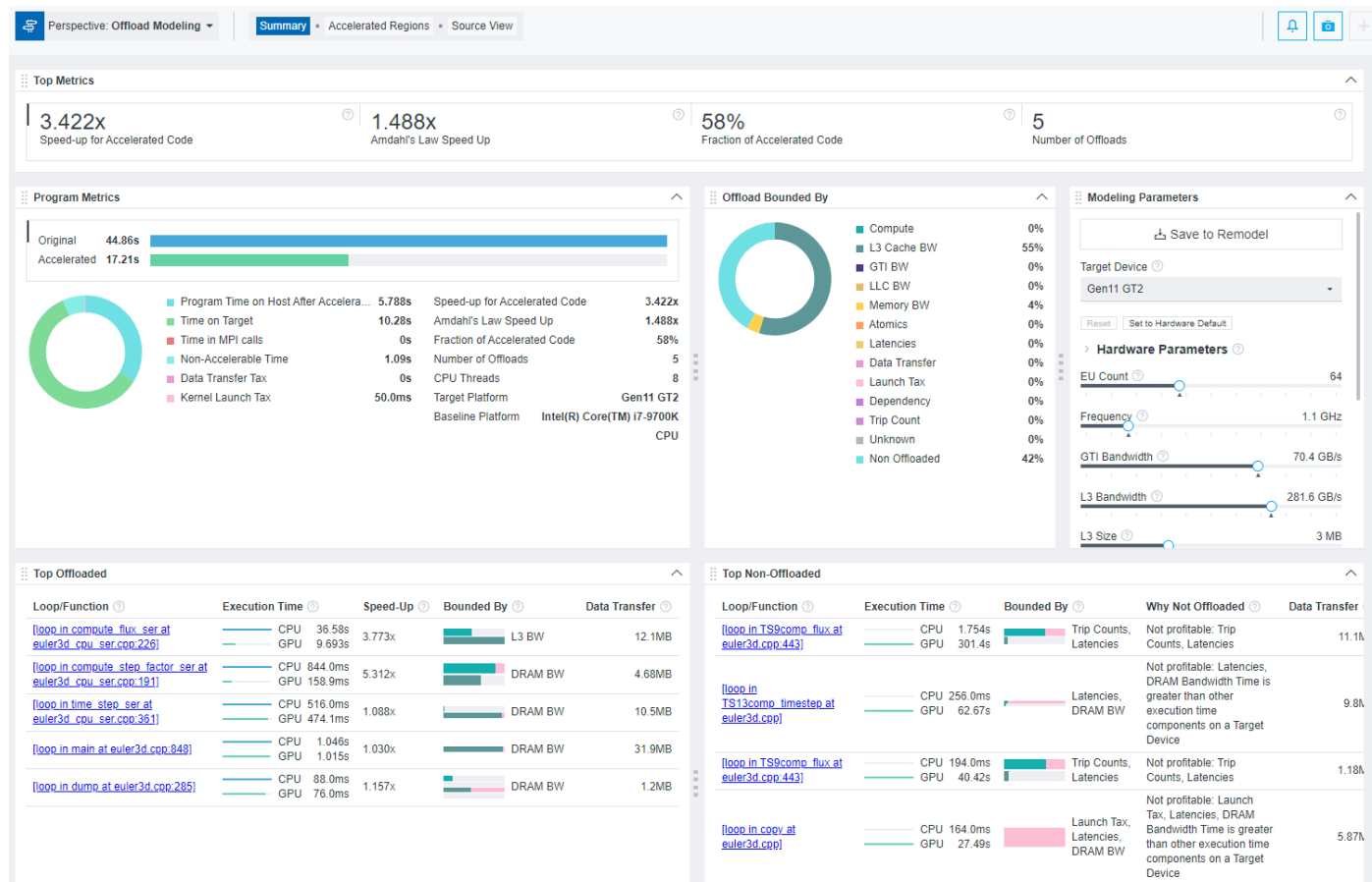
The CPU-to-GPU and GPU-to-GPU modeling workflows are based on different hardware configurations, compilers code-generation principles, and software implementation aspects to provide an accurate modeling results specific to the baseline device for your application. Review the following features of the workflows:

CPU-to-GPU modeling	GPU-to-GPU modeling
Only loops/functions executed or offloaded to a CPU are analyzed.	Only GPU compute kernels are analyzed.
Loop/function characteristics are measured using the CPU profiling capabilities.	Compute kernel characteristics are measured using the GPU profiling capabilities.
Only profitable loops/functions are recommended for offloading to a target GPU. Profitability is based on the estimated speedup.	All kernels executed on GPU are modeled one to one, even if they have low speedup estimated.
High-overhead features, such as call stack handling, cache and data transfer simulation, dependencies analysis, can be enabled. You might need to run the Dependencies analysis to check if loop-carried dependencies affect performance on a GPU.	High-overhead features, such as call stack handling, cache and data transfer simulation, dependencies analysis, are disabled. You <i>do not</i> need to run the Dependencies analysis.
Data transfer between baseline and target devices can be simulated in two different modes: footprint-based and memory object-based.	Memory objects transferred between host and device memory are traced.

Offload Modeling Summary

Offload Modeling perspective measures performance of your application and compares it with its modeled performance on a selected target GPU so that you can decide what parts of your application you can execute on the GPU and how you can optimize it to get a better performance after offloading.

- Main metrics for the modeled performance of your program indicating if you should offload your application hotspots to a target device or not
- Specific factors that prevent your code from achieving a better performance if executed on a target device (the factors that your code is bounded by)
- Top offloaded loops/functions that provide the highest benefit (up to five)
- For the CPU-to-GPU modeling: Top non-offloaded loops/functions (up to five) with reasons why a loop is not offloaded



See Also

[Run Offload Modeling Perspective from GUI](#)

[Run Offload Modeling Perspective from Command Line](#)

[Run GPU-to-GPU Performance Modeling from Command Line](#) With Intel® Advisor, you can model performance of SYCL, OpenCL™, or OpenMP* target application running on a graphics processing unit (GPU) for a different GPU device without its CPU version. For this, run the GPU-to-GPU modeling workflow of the Offload Modeling perspective.

[Offload Modeling Report Overview](#)

[Optimize Your GPU Application with Intel® oneAPI Base Toolkit](#)

Run Offload Modeling Perspective from GUI

Prerequisites:

- For a SYCL, OpenMP* target, or OpenCL™ application, do *one* of the following:
 - To analyze the application running on a GPU: [Configure your system](#) to analyze GPU kernels.
 - To analyze the application running on a CPU: [Set up environment variables](#) to offload it temporarily to a CPU.
- In the graphical-user interface (GUI): [Create a project](#) and specify an analysis target and target options.

To configure and run the Offload Modeling perspective from the GUI:

- Select a baseline device from the drop-down. This is the device that your application runs on for the Intel® Advisor to collect performance data.
 - To analyze an application running on a CPU (for example, C, C++, or Fortran), make sure **CPU** is selected.

- To analyze an application running on a GPU (for example, SYCL, OpenMP target, OpenCL), select the **GPU** baseline device.

NOTE If you select GPU, make sure the **GPU Profiling** checkbox is enabled under Survey, Characterization, and Performance Modeling analyses.

2. Configure the perspective and set analysis properties, depending on desired results.

- Select a collection accuracy level with analysis properties preset for a specific result:
 - Low:** Model your application performance for a target device and get the basic low-confidence information about potential speed-up and performance.
 - Medium:** Model your application performance and data transfers between host and target devices.
 - High:** Model your application performance, data transfers, and memory objects attribution to improve offload modeling accuracy. For application running on CPU, analyze loop-carried dependencies.
- Select the analyses and properties manually to adjust the perspective flow to your needs. The accuracy level is set to **Custom**.

The higher accuracy value you choose, the higher runtime overhead is added to your application. The **Overhead** indicator shows the overhead for the selected configuration. For the **Custom** accuracy, the overhead is calculated automatically for the selected analyses and properties.

The Dependencies analysis (included in the high accuracy for CPU baseline device) adds the highest overhead and is not required if your application is highly parallelized or vectorized on a CPU or if you know that key hotspots in your application do not have loop-carried dependencies. You may need to run it for a CPU application if it has scalar loops/functions or you are not sure about dependencies in your code. See [Check How Assumed Dependencies Affect Modeling](#) for a workflow to learn about potential dependencies in your code.

By default, accuracy is set to **Low**. See [Offload Modeling Accuracy Presets](#) for more details.

3. Select a target platform from the **Target Platform Model** drop-down. This is a platform that the Intel Advisor models your application performance on. The following target platforms are available:








Platform	Device
pvc_xt_448xve (default)	Intel® Data Center GPU Max 448
pvc_xt_512xve	Intel® Data Center GPU Max 512
XeHPG 512	Intel® Arc™ Graphics with 512 vector engines
XeHPG 256	Intel® Arc™ Graphics with 256 vector engines
Gen11 GT2	Intel® Iris® Plus Graphics
XeLP Max 96	Intel® Iris® Xe MAX Graphics
XeLP GT2	Intel® Iris® Xe Graphics
Gen9 GT2	Intel® HD Graphics 530
Gen9 GT3e	Intel® Iris® Graphics 550
Gen9 GT4e	Intel® Iris® Pro Graphics 580

NOTE Multi-tiler and multi-GPU analysis for pvc_xt_448xve and pvc_xt_512xve platforms is not supported at the moment.

4.

Click  **Run** to run the perspective.

While the perspective is running, you can do the following in the **Analysis Workflow** tab:

- Control the perspective execution:
 - Stop data collection and see the already collected data: Click the  button.
 - Pause data collection: Click the  button.
 - Cancel data collection and discard the collected data: Click the  button.
- Expand an analysis with  to control the analysis execution:
 - Pause the analysis: Click the  button.
 - Stop the currently running analysis and start the next analysis selected: Click the  button.
 - Interrupt execution of all selected analyses and see the already collected data: Click the  button.

After you run the Offload Modeling perspective, the collected Survey data becomes available for all other perspectives. If you switch to another perspective, you can skip the Survey step and run only perspective-specific analyses.


To run the CPU-to-GPU Offload Modeling perspective with the Medium accuracy from the command line interface:

```
advisor --collect=offload --project-dir=./advi_results -- ./myApplication
```

To run the GPU-to-GPU Offload Modeling perspective with the Medium accuracy from the command line interface:

```
advisor --collect=offload --gpu --project-dir=./advi_results -- ./myApplication
```

See [Run Offload Modeling Perspective from Command Line](#) for details. See [Run GPU-to-GPU Performance Modeling from Command Line](#) for details about the GPU-to-GPU Offload Modeling.

NOTE To [generate command lines](#) for selected perspective configuration, click the  **Command Line** button.

Once the Offload Modeling perspective collects data, the report opens showing a **Summary** tab with performance metrics estimated for the selected target platform, such as estimated speedup, potential performance bottlenecks, and top offloaded loops. Depending on the selected accuracy level and perspective properties, continue to investigate the results. See [Explore Offload Modeling Results](#)

Offload Modeling Accuracy Presets

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium	High
Overhead	5 - 10x	15 - 50x	50 - 80x

Comparison / Accuracy Level	Low	Medium	High
Goal	Model performance of an application that is mostly compute bound and does not have dependencies	Model application performance considering memory traffic for all cache and memory levels	Model application performance with all potential limitations for offload candidates
Analyses	Survey + Characterization (Trip Counts and FLOP) + Performance Modeling with no assumed dependencies	Survey + Characterization (Trip Counts and FLOP with cache simulation for the selected target device, callstacks, and light data transfer simulation) + Performance Modeling with no assumed dependencies	Survey + Characterization (Trip Counts and FLOP with cache simulation for the selected target device, callstacks, and medium data transfer simulation) + Dependencies + Performance Modeling with assumed dependencies
Result	Basic Offload Modeling report that shows potential speedup and performance metrics estimated on a target considering memory traffic from execution units to L1 cache only. The result might be inaccurate for memory-bound applications.	Offload Modeling report extended with data transfers estimated between host and device platforms considering memory traffic for all cache and memory levels	Offload Modeling report with detailed data transfer estimations and automated check for loop-carried dependencies for more accurate search for the most profitable regions to offload

You can choose custom accuracy and set a custom perspective flow for your application. For more information, see [Customize Offload Modeling Perspective](#).

NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).


NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).


Customize Offload Modeling Perspective

Customize the perspective flow to better fit your goal and your application.

If you change any of the analysis settings from the **Analysis Workflow** tab, the accuracy level changes to **Custom** automatically. With this accuracy level, you can customize the perspective flow and/or analysis properties.

To change the properties of a specific analysis:

1. Expand the analysis details on the **Analysis Workflow** pane with .
2. Select desired settings.

3. For more detailed customization, click the *gear*  icon. You will see the **Project Properties** dialog box open for the selected analysis.
4. Select desired properties and click **OK**.

For a full set of available properties, click the  icon on the left-side pane or go to **File > Project Properties**.

The following tables cover project properties applicable to the analyses in the Offload Modeling perspective.

Common Properties

Use This	To Do This
Inherit settings from Visual Studio project checkbox and field (Visual Studio* IDE only)	Inherit Intel Advisor project properties from the Visual Studio* startup project (enable). If enabled, the Application , Application parameters , and Working directory fields are pre-filled and cannot be modified.
Application field and Browse... button	Select an analysis target executable or script. If you specify a script in this field, consider specifying the executable in the Advanced > Child application field (required for Dependencies analysis).
Application parameters field and Modify... button	Specify runtime arguments to use when performing analysis (equivalent to command line arguments).
Use application directory as working directory checkbox	Automatically use the value in the Application directory to pre-fill the Working directory value (enable).
Working directory field and Browse... button	Select the working directory.
User-defined environment variables field and Modify... button	Specify environment variables to use during analysis.
Managed code profiling mode drop-down	<ul style="list-style-type: none"> Automatically detect the type of target executable as Native or Managed, and switch to that mode (choose Auto). Collect data for native code and do not attribute data to managed code (choose Native). Collect data for both native and managed code, and attribute data to managed code as appropriate (choose Mixed). Consider using this option when analyzing a native executable that makes calls to managed code. Collect data for both native and managed code, resolve samples attributed to native code, and attribute data to managed source only (choose Managed). The call stack in the analysis result displays data for managed code only.
Child application field	Analyze a file that is not the starting application. For example: Analyze an executable (identified in this field) called by a script (identified in the Application field). Invoking these properties could decrease analysis overhead.

Use This	To Do This
	<p>NOTE</p> <p>For the Dependencies Analysis Type: If you specify a script file in the Application field, you must specify the target executable in the Child application field.</p>
Modules radio buttons, field, and Modify... button	<ul style="list-style-type: none"> Analyze specific modules and disable analysis of all other modules (click the Include only the following module(s) radio button and choose the modules). Disable analysis of specific modules and analyze all other modules (click the Exclude only the following module(s) radio button and choose the modules). <p>Including/excluding modules could minimize analysis overhead.</p>
Use MPI launcher checkbox	<p>Generate a command line (enable) that appears in the Get command line field based on the following parameters:</p> <ul style="list-style-type: none"> Select MPI Launcher - Intel or another vendor Number of ranks - Number of instances of the application Profile ranks - All or a range of ranks to profile
Automatically stop collection after (sec) checkbox and field	<p>Stop collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could minimize analysis overhead.</p>

Survey Analysis Properties

Use This	To Do This
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <p>Tip</p> <p>The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p>
Sampling Interval selector	<p>Set the wait time between each analysis collection CPU sample while your target application is running.</p> <p>Increasing the wait time could decrease analysis overhead.</p>
Collection data limit, MB selector	<p>Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses.</p> <p>Decreasing the limit could decrease analysis overhead.</p>

Use This	To Do This
Callstack unwinding mode drop-down list	<p>Set to After collection if:</p> <ul style="list-style-type: none"> Survey analysis runtime overhead exceeds 1.1x. A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions. <p>Otherwise, set to During Collection. This mode improves stack accuracy but increases overhead.</p>
Stitch stacks checkbox	<p>Restore a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload (enable).</p> <p>Disable if Survey analysis runtime overhead exceeds 1.1x.</p>
Analyze MKL Loops and Functions checkbox	<p>Show Intel® oneAPI Math Kernel Library (oneMKL) loops and functions in Intel Advisor reports (enable).</p> <p>Enabling could increase analysis overhead.</p>
Analyze Python loops and functions checkbox	<p>Show Python* loops and functions in Intel Advisor reports (enable).</p> <p>Enabling could increase analysis overhead.</p>
Analyze loops that reside in non-executed code paths checkbox	<p>Collect a variety of data during analysis for loops that reside in non-executed code paths, including loop assembly code, instruction set architecture (ISA), and vector length (enable).</p> <p>Enabling could increase analysis overhead.</p> <hr/> <p>NOTE</p> <p>Analyzing non-executed code paths in binaries that target multiple ISAs (contain multiple code paths) is available only for binaries compiled using the <code>-ax</code> (Linux* OS) / <code>Qax</code> (Windows* OS) option with an Intel compiler.</p> <hr/>
Enable registry spill/fill analysis checkbox	<p>Calculate the number of consecutive load/store operations in registers and related memory traffic (enable).</p> <p>Enabling could increase analysis overhead.</p>
Enable static instruction mix analysis checkbox	<p>Statically calculate the number of specific instructions present in the binary (enable).</p> <p>Enabling could increase analysis overhead.</p>
Source caching drop-down list	<ul style="list-style-type: none"> Delete source code cache from a project with each analysis run (default; choose Clear cached files). Keep source code cache within the project (choose Keep cached files).

Trip Counts and FLOP Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds.
Trip Counts / Collect information about Loop Trip Counts checkbox	Measure loop invocation and execution (enable).
FLOP / Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage checkbox	Measure floating-point operations, integer operations, and memory traffic (enable).
Callstacks / Collect callstacks checkbox	Collect call stack information when performing analysis (enable). Enabling could increase analysis overhead.
Capture metrics for dynamic loops and functions checkbox	Collect metrics for dynamic Just-In-Time (JIT) generated code regions.
Capture metrics for stripped binaries checkbox	Collect metrics for stripped binaries. Enabling could increase analysis overhead.
Cache Simulation / Enable Memory-Level Roofline with cache simulation checkbox	Model multiple levels of cache for data, such as counts of loaded or stored bytes for each loop, to plot the Roofline chart for all memory levels (enable). Enabling could increase analysis overhead.
Cache simulator configuration field	Specify a cache hierarchy configuration to model (enable and specify hierarchy). The hierarchy configuration template is: [num_of_level1_caches]:[num_of_ways_level1_connected]: [level1_cache_size]:[level1_cacheline_size]/ [num_of_level2_caches]:[num_of_ways_level2_connected]: [level2_cache_size]:[level2_cacheline_size]/ [num_of_level3_caches]:[num_of_ways_level3_connected]: [level3_cache_size]:[level3_cacheline_size]

Use This	To Do This
	<p>For example: 4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l is the hierarchy configuration for:</p> <ul style="list-style-type: none"> • Four eight-way 32-KB level 1 caches with line size of 64 bytes • Four four-way 256-KB level 2 caches with line size of 64 bytes • One sixteen-way 6-MB level 3 cache with line size of 64 bytes
Data Transfer Simulation / Data transfer simulation mode drop-down	<p>Select a level of details for data transfer simulation:</p> <ul style="list-style-type: none"> • Off - Disable data transfer simulation analysis. • Light - Model data transfers between host and device memory. • Medium - Model data transfers, attribute memory objects to loops that accessed the objects, and track accesses to stack memory. • Full - Model data transfers, attribute memory objects, track accesses to stack memory, and identify where data can be potentially reused if transferred between host and target.

Dependencies Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	<p>Copy similar settings from Survey analysis properties (enable).</p> <p>When enabled, this option disables application parameters controls.</p>
Suppression mode radio buttons	<ul style="list-style-type: none"> • Report possible dependencies in system modules (choose the Show problems in system modules radio button). • Do not report possible dependencies in system modules (choose the Suppress problems in system modules radio button).
Loop call count limit selector	<p>Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances.</p> <p>Supplying a non-zero value could decrease analysis overhead.</p>
Instance of interest selector	<p>Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes.</p> <p>Supplying a non-zero value could decrease analysis overhead.</p>
Analyze stack variables checkbox	<p>Analyze parallel data sharing for stack variables (enable).</p> <p>Enabling could increase analysis overhead.</p>
Filter stack variables by scope checkbox	<p>Enable to report:</p> <ul style="list-style-type: none"> • Variables initiated inside the loop as potential dependencies (warning) • Variables initialized outside the loop as dependencies (error) <p>Enabling could increase analysis overhead.</p>
Reduction Detection / Filter reduction variables checkbox	<p>Mark all potential reductions by a specific diagnostic (enable).</p> <p>Enabling could increase analysis overhead.</p>
Markup type checkbox	<p>Select loops/functions by pre-defined markup algorithm. Supported algorithms are:</p>

Use This	To Do This
	<ul style="list-style-type: none"> • GPU generic - Select loops executed on a GPU. • OpenMP - Select OpenMP* loops. • SYCL - Select SYCL loops. • OpenCL - Select OpenCL™ loops. • DAAL - Select Intel® oneAPI Data Analytics Library loops. • TBB - Select Intel® oneAPI Threading Building Blocks loops. <hr/> <p>NOTE This option is available only from the Analysis Workflow pane for the Offload Modeling perspective.</p> <hr/>

Performance Modeling Properties

Use This	To Do This
Assume Dependencies checkbox	<p>Assume that loops have dependencies if their dependency type is unknown.</p> <hr/> <p>NOTE This option is available only from the Analysis Workflow pane.</p> <hr/>
Single Kernel Launch Tax checkbox	<p>Assume the invocation tax is paid only for the first kernel launch when estimating invocation taxes.</p> <hr/> <p>NOTE This option is available only from the Analysis Workflow pane.</p> <hr/>
Data Reuse Analysis checkbox	<p>Analyze potential data reuse between code regions for the data transferred between host and target platforms.</p> <hr/> <p>NOTE This option is available only from the Analysis Workflow pane.</p> <hr/> <p>NOTE Make sure to set the Data Transfer Analysis to the <i>Full</i> mode in the Characterization step to analyze data reuse.</p> <hr/>
Target Config drop-down	<p>Select a pre-defined hardware configurations from a drop-down list to model application performance on.</p>
Other parameters field	<p>Enter a space-separated list of command-line parameters. See Command Option Reference for available parameters.</p>

Use This	To Do This
Baseline Device drop-down	Select a baseline device that your application runs on for the Intel® Advisor to collect performance data.
Custom Device Configuration field	Specify the absolute path or name for a custom TOML configuration file with additional modeling parameters.

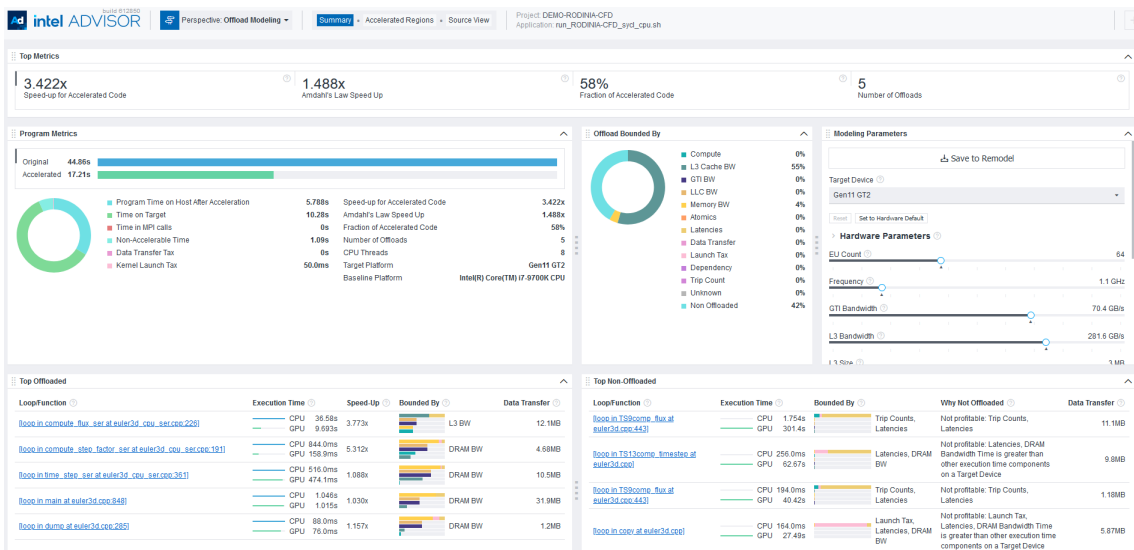
Run Offload Modeling Perspective from Command Line

Intel® Advisor provides several methods to run the Offload Modeling perspective from command line. Use one of the following:

- **Method 1.** [Run Offload Modeling with a command line collection presets.](#) Use this method if you want to use basic Intel Advisor analysis and modeling functionality, especially for the first-run analysis. This simple method allows you to run multiple analyses with a single command and control the modeling accuracy.
- **Method 2.** [Run Offload Modeling analyses separately.](#) Use this method if you want to analyze an MPI application or need more advanced analysis customization. This method allows you to select what performance data you want to collect for your application and configure each analysis separately.
- **Method 3.** [Run Offload Modeling with Python* scripts.](#) Use this method if you need more analysis customization. This method is moderately flexible and allows you to customize data collection and performance modeling steps.

Tip See [Intel Advisor cheat sheet](#) for quick reference on command line interface.

After you run the Offload Modeling with any method above, you can [view the results](#) in Intel Advisor graphical user interface (GUI), command line interface (CLI), or an interactive HTML report. For example, the interactive HTML report is similar to the following:



Prerequisites

1. Set Intel Advisor environment variables with an automated script.

The script enables the `advisor` command line interface (CLI), `advisor-python` command line tool, and the `APM` environment variable, which points to the directory with Offload Modeling scripts and simplifies their use.

2. For SYCL, OpenMP* target, OpenCL™ applications: [Set Intel Advisor environment variables](#) to offload temporarily your application to a CPU for the analysis.

NOTE You are recommended to run the [GPU-to-GPU performance modeling](#) to analyze SYCL, OpenMP target, and OpenCL application because it provides more accurate estimations.

Optional: Generate pre-Configured Command Lines

With the Intel Advisor, you can generate pre-configured command lines for your application and hardware. Use this feature if you want to:

- Analyze an MPI application
- Customize pre-set Offload Modeling commands

Offload Modeling perspective consists of multiple analysis steps executed for the same application and project. You can configure each step from scratch or use pre-configured command lines that do not require you to provide the paths to project directory and an application executable manually.

Option 1. Generate pre-configured command lines with `--collect=offload` and the `--dry-run` option. The option generates:

- Commands for the Intel Advisor CLI collection workflow
- Commands that correspond to a specified accuracy level
- Commands *not* configured to analyze an MPI application. You need to manually adjust the commands for MPI.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

The workflow includes the following steps:

1. Generate the command using the `--collect=offload` action with the `--dry-run` option. Specify accuracy level and paths to your project directory and application executable.

For example, to generate the low-accuracy commands for the `myApplication` application, run the following command:

- On Linux* OS:

```
advisor --collect=offload --accuracy=low --dry-run --project-dir=./advi_results -- ./myApplication
```

- On Windows* OS:

```
advisor --collect=offload --accuracy=low --dry-run --project-dir=.\advi_results -- .\myApplication.exe
```

You should see a list of commands for each analysis step to get the Offload Modeling result with the specified accuracy level (for the commands above, it is *low*).

2. **If you analyze an MPI application:** Copy the generated commands to your preferred text editor and modify each command to use an MPI tool. For details about the syntax, see [Analyze MPI Applications](#).
3. Run the generated commands one by one from a command prompt or a terminal.

Option 2. If you have an Intel Advisor graphical user interface (GUI) available on your system and you want to analyze an MPI application from command line, you can generate the pre-configured command lines from GUI.

The GUI generates:

- Commands for the Intel Advisor CLI collection workflow
- Commands for a selected accuracy level if you want to run a pre-defined accuracy level or commands for a custom project configuration if you want to enable/disable additional analysis options
- Command configured for MPI application with Intel® MPI Library. You do not need to manually modify the commands for the MPI application syntax.

For detailed instructions, see [Generate Pre-configured Command Lines](#).

Method 1. Use Collection Presets

For the Offload Modeling perspective, Intel Advisor has a special collection mode `--collect=offload` that allows you to run several analyses using *only one* Intel Advisor CLI command. When you run the collection, it sequentially runs data collection and performance modeling steps. The specific analyses and options depend on the *accuracy level* you specify for the collection.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

For example, to run the Offload Modeling perspective with the default (medium) accuracy level:

- On Linux* OS:

```
advisor --collect=offload --project-dir=./advi_results -- ./myApplication
```

- On Windows* OS:

```
advisor --collect=offload --project-dir=.\advi_results -- .\myApplication.exe
```

The collection progress and commands for each analysis executed will be printed to a terminal or a command prompt. By default, the performance is modeled for the Intel® Arc™ graphics code-named Alchemist (xe_hpg_512xve configuration). When the collection is finished, you will see the result summary.

Analysis Details

To change the analyses to run and their option, you can specify a different accuracy level with the `--accuracy=<level>` option. The default accuracy level is *medium*.

The following accuracy levels are available:

- *low* accuracy includes Survey, Characterization with Trip Counts and FLOP collections, and Performance Modeling analyses.
- *medium* (default) accuracy includes Survey, Characterization with Trip Counts and FLOP collections, cache and data transfer simulation, and Performance Modeling analyses.
- *high* accuracy includes Survey, Characterization with Trip Counts and FLOP collections, cache, data transfer, and memory object attribution simulation, and Performance Modeling analyses. For CPU applications, also includes the Dependencies analysis.

For CPU applications, this accuracy level adds a high collection overhead because it includes the Dependencies analysis. This analysis is not required if your application is highly parallelized or vectorized on a CPU or if you know that key hotspots in your application do not have loop-carried dependencies. Otherwise, to learn how dependencies might affect your application performance on a GPU, see [Check How Assumed Dependencies Affect Modeling](#).

For example, to run the *low* accuracy level:

```
advisor --collect=offload --accuracy=low --project-dir=./advi_results -- ./myApplication
```

To run the *high* accuracy level:

```
advisor --collect=offload --accuracy=high --project-dir=./advi_results -- ./myApplication
```

If you want to see the commands that are executed at each accuracy level, you can run the collection with the `--dry-run` option. The commands will be printed to a terminal or a command prompt.

For details about each accuracy level, see [Offload Modeling Accuracy Levels in Command Line](#).

Customize Collection

You can also specify additional options if you want to run the Offload Modeling with custom configuration. This collection accepts most options of the Performance Modeling analysis (`--collect=projection`) and some options of the Survey, Trip Counts, and Dependencies analyses that can be useful for the Offload Modeling.

Important Make sure to specify the additional options *after* the `--accuracy` option to make sure they take precedence over the accuracy level configurations.

Consider the following action options:

Option	Description
<code>--accuracy=<level></code>	<p>Set an accuracy level for a collection preset. Available accuracy levels:</p> <ul style="list-style-type: none"> • low • medium (default) • high <p>For details, see Offload Modeling Accuracy Levels in Command Line.</p>
<code>--config</code>	<p>Select a target GPU configuration to model performance for. For example, <code>xehpg_512xve</code> (default), <code>gen12_dg1</code>, or <code>gen9_gt3</code>.</p> <p>See config for a full list of possible values and mapping to device names.</p>
<code>--gpu</code>	<p>Analyze a SYCL, OpenCL™, or OpenMP* target application on a graphics processing unit (GPU) device. This option automatically adds all related options to each analysis included in the preset.</p> <p>If you use this option, the high accuracy does not include the Dependencies analysis.</p> <p>For details about this workflow, see Run GPU-to-GPU Performance Modeling from Command Line.</p>
<code>--data-reuse-analysis</code>	<p>Analyze potential data reuse between code regions. This option automatically adds all related options to each analysis included in the preset.</p>
<code>--enforce-fallback</code>	<p>Emulate data distribution over stacks if stacks collection is disabled. This option automatically adds all related options to each analysis included in the preset.</p>

For details about other available options, see [collect](#).

Method 2. Use per-Analysis Collection

You can collect data and model performance for your application by running each Offload Modeling analysis in a separate command using Intel Advisor CLI. This option allows you to:

- Control what analyses you want to run to profile your application and what data you want to collect
- Modify behavior of each analysis you run with an extensive set of options
- Remodel application performance without re-collecting performance data. This can save time if you want to see how the performance of your application might change with different modeling parameters using the same performance data as baseline
- Profile and model performance of [an MPI application](#)

Consider the following workflow example. Using this example, you can run the Survey, Trip Counts, and FLOP analyses to profile an application and the Performance Modeling to model its performance on a selected target device.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

On Linux OS:

1. Run the Survey analysis.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

2. Run the Trip Counts and FLOP analyses with data transfer simulation for the default Intel® Arc™ graphics code-named Alchemist (xe_hpg_512xve configuration).

```
advisor --collect=tripcounts --flop --stacks --cache-simulation=single --target-device=xe_hpg_512xve --data-transfer=light --project-dir=./advi_results -- ./myApplication
```

3. *Optional:* Run the Dependencies analysis to check for loop-carried dependencies.

```
advisor --collect=dependencies --loop-call-count-limit=16 --select markup=gpu_generic --filter-reductions --project-dir=./advi_results -- ./myApplication
```

The Dependencies analysis adds a high collection overhead. *You can skip it* if your application is highly parallelized or vectorized on a CPU or if you know that key hotspots in your application do not have loop-carried dependencies. If you are not sure, see [Check How Assumed Dependencies Affect Modeling](#) to learn how dependencies affect your application performance on a GPU.

4. Run the Performance Modeling analysis to model application performance on the default Intel® Arc™ graphics code-named Alchemist(xe_hpg_512xve configuration).

```
advisor --collect=projection --project-dir=./advi_results
```

You will see the result summary printed to the command prompt.

Tip: If you already have an analysis result saved as a [snapshot](#) or a result for an MPI rank, you can use the [exp-dir](#) option instead of `project-dir` to model performance for the result.

On Windows OS:

1. Run the Survey analysis.

```
advisor --collect=survey --static-instruction-mix --project-dir=.\advi_results -- .\myApplication.exe
```

2. Run the Trip Counts and FLOP analyses with cache simulation for the default Intel® Arc™ graphics code-named Alchemist (xe_hpg_512xve configuration).

```
advisor --collect=tripcounts --flop --stacks --cache-simulation=single --target-device=xe_hpg_512xve --data-transfer=light --project-dir=.\advi_results -- .\myApplication.exe
```

3. *Optional:* Run the Dependencies analysis to check for loop-carried dependencies.

```
advisor --collect=dependencies --loop-call-count-limit=16 --select markup=gpu_generic --filter-reductions --project-dir=.\advi_results -- myApplication.exe
```

The Dependencies analysis adds a high collection overhead. *You can skip it* if your application is highly parallelized or vectorized on a CPU or if you know that key hotspots in your application do not have loop-carried dependencies.. If you are not sure, see [Check How Assumed Dependencies Affect Modeling](#) to learn how dependencies affect your application performance on a GPU.

4. Run the Performance Modeling analysis to model application performance on the default Intel® Arc™ graphics code-named Alchemist (xe_hpg_512xve configuration).

```
advisor --collect=projection --project-dir=.advi_results
```

You will see the result summary printed to the command prompt.

Tip: If you already have a collected analysis result saved as a snapshot or result for an MPI rank, you can use the `exp-dir` option instead of `project-dir` to model performance for the result.

For more useful options, see the **Analysis Details** section below.

Analysis Details

The Offload Modeling workflow includes the following analyses:

1. Survey to collect initial performance data.
2. Characterization with trip counts and FLOP to collect performance details.
3. Dependencies (optional) to identify loop-carried dependencies that might limit offloading.
4. Performance Modeling to model performance on a selected target device.

Each analysis has a set of additional options that modify its behavior and collect additional performance data. The more analyses you run and option you use, the higher the modeling accuracy.

Consider the following options:

Survey Options

To run the Survey analysis, use the following command line action: `--collect=survey`.

Recommended action options:

Options	Description
<code>--static-instruction-mix</code>	Collect static instruction mix data. This option is recommended for the Offload Modeling perspective.
<code>--profile-gpu</code>	Analyze a SYCL, OpenCL, or OpenMP target application on a GPU device. If you use this option, skip the Dependencies analysis. For details about this workflow, see Run GPU-to-GPU Performance Modeling from Command Line .

Characterization Options

To run the Characterization analysis, use the following command line action: `--collect=tripcounts`.

Recommended action options:

Options	Description
<code>--flop</code>	Collect data about floating-point and integer operations, memory traffic, and mask utilization metrics for AVX-512 platforms.
<code>--stacks</code>	Enable advanced collection of call stack data.

Options	Description
<code>--cache-simulation=<mode></code>	<p>Simulate cache behavior for a target device. Available modes:</p> <ul style="list-style-type: none"> • <code>off</code> - disable cache simulation. • <code>single</code> - simulate cache behavior only for the selected target device. Make sure to use with the <code>--target-device=<target></code> option. • <code>multi</code> - simulate cache behavior for all available target devices to enable performance remodeling without running the Characterization analysis.
<code>--target-device=<target></code>	<p>Specify a target graphics processing unit (GPU) to model cache for. For example, <code>xehpg_512xve</code> (default), <code>gen12_dg1</code>, or <code>gen9_gt3</code>. See target-device for a full list of possible values and mapping to device names.</p> <p>Use with the <code>--cache-simulation=single</code> option.</p> <hr/> <p>Important Make sure to specify the same target device as for the <code>--collect=projection --config=<config></code>.</p> <hr/>
<code>--data-transfer=<mode></code>	<p>Enable modeling data transfers between host and target devices. The following modes are available:</p> <ul style="list-style-type: none"> • Use <code>off</code> (default) to disable data transfer modeling. • Use <code>light</code> to model only data transfers. • Use <code>medium</code> to model data transfers, attributes memory objects, and tracks accesses to stack memory. • Use <code>full</code> to model data transfers, attributes memory objects, tracks accesses to stack memory, and enables data reuse analysis as well.
<code>--profile-gpu</code>	<p>Analyze a SYCL, OpenCL, or OpenMP target application on a GPU device.</p> <p>If you use this option, skip the Dependencies analysis.</p> <p>For details about this workflow, see Run GPU-to-GPU Performance Modeling from Command Line.</p>

Dependencies Options

The Dependencies analysis is optional because it adds a high overhead and is mostly necessary if you have scalar loops/functions in your application or if you do not know about loop-carried dependencies in key hotspots. For details about when you need to run the Dependencies analysis, see [Check How Assumed Dependencies Affect Modeling](#).

To run the Dependencies analysis, use the following command line action: `--collect=dependencies`.

Recommended action options:

Options	Description
<code>--select=<string></code>	<p>Select loops to run the analysis for.</p> <p>For the Offload Modeling, the recommended value is <code>--select markup=gpu_generic</code>, which selects only loops/functions profitable for offloading to a target device to reduce the analysis overhead.</p> <p>For more information about markup options, see Loop Markup to Minimize Analysis Overhead.</p> <hr/> <p>NOTE The generic markup strategy is recommended if you want to run the Dependencies analysis for an application that does not use SYCL, C++/Fortran with OpenMP target, or OpenCL.</p>
<code>--loop-call-count-limit=<num></code>	<p>Set the maximum number of call instances to analyze assuming similar runtime properties over different call instances.</p> <p>The recommended value is 16.</p>
<code>--filter-reductions</code>	<p>Mark all potential reductions with a specific diagnostic.</p>

Performance Modeling Options

To run the Performance Modeling analysis, use the following command line action: `--collect=projection`.

Recommended action options:

Options	Description
<code>--exp-dir=<path></code>	<p>Specify a path to an unpacked result snapshot or an MPI rank result to model performance. Use this option instead of <code>project-dir</code> if you already have an analysis result ready.</p>
<code>--config=<config></code>	<p>Select a target GPU configuration to model performance for. For example, <code>xehpg_512xve</code> (default), <code>gen12_dg1</code>, or <code>gen9_gt3</code>.</p> <hr/> <p>Important Make sure to specify the same target device as for the <code>--collect=tripcounts --target-device=<target></code>.</p> <hr/> <p>For details about configuration files, see config.</p>
<code>--no-assume-dependencies</code>	<p>Assume that a loop <i>does not</i> have dependencies if a loop dependency type is unknown.</p>

Options	Description
<code>--data-reuse-analysis</code>	<p>Use this option if your application contains parallel and/or vectorized loops and you did not run the Dependencies analysis.</p> <p>Analyze potential data reuse between code regions when offloaded to a target GPU.</p> <hr/> <p>Important Make sure to use <code>--data-transfer=full</code> with <code>--collect=tripcounts</code> for this option to work correctly.</p> <hr/>
<code>--assume-hide-taxes</code>	<p>Assume that an invocation tax is paid only for the <i>first</i> time a kernel is launched.</p>
<code>--set-parameter</code>	<p>Specify a single-line configuration parameter to modify in a format "<code><group>.<parameter>=<new-value></code>". For example, <code>"min_required_speed_up=0"</code>.</p> <p>For details about the option, see set-parameter. For details about some of the possible modifications, see Advanced Modeling Configuration.</p>
<code>--profile-gpu</code>	<p>Analyze a SYCL, OpenCL, or OpenMP target application on a GPU device.</p> <p>If you use this option, skip the Dependencies analysis.</p> <p>For details about this workflow, see Run GPU-to-GPU Performance Modeling from Command Line.</p>

See [advisor Command Option Reference](#) for more options.

Method 3. Use Python* Scripts

Intel Advisor has three scripts that use the Intel Advisor Python* API to run the Offload Modeling. You can run the scripts with the `advisor-python` command line tool or with your local Python 3.6 or 3.7.

The scripts vary in functionality and run different sets of Intel Advisor analyses. Depending on what you want to run, use one or several of the following scripts:

- `run_oa.py` is the simplest script with limited modification flexibility. Use this script to run the collection and modeling steps with a single command. This script is the equivalent of the Intel Advisor command line collection preset.
- `collect.py` is a moderately flexible script that runs *only* the collection step.
- `analyze.py` is a moderately flexible script that runs *only* the performance modeling step.

NOTE The scripts do not support the analysis of MPI applications. For an MPI application, use the per-analysis collection with the Intel Advisor CLI.

You can run the Offload Modeling using different combinations of the scripts and/or the Intel Advisor CLI. For example:

- Run `run_oa.py` to profile application and model its performance.
- Run the `collect.py` to profile application and `analyze.py` to model its performance. Re-run `analyze.py` to remodel with a different configuration.
- Run the Intel Advisor CLI to collect performance data and `analyze.py` to model performance. Re-run `analyze.py` to remodel with a different configuration.
- Run `run_oa.py` to collect data and model performance for the first time and run `analyze.py` to remodel with a different configuration.

Consider the following examples of some typical scenarios with Python scripts.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Example 1. Run the `run_oa.py` script to profile an application and model its performance for the Intel® graphics code-named Alchemist (xe_hpg_512xve configuration).

- On Linux OS:

```
adviser-python $APM/run_oa.py ./advi_results --collect=basic --config=xe_hpg_512xve -- ./myApplication
```

- On Windows OS:

```
adviser-python %APM%\run_oa.py .\advi_results --collect=basic --config=xe_hpg_512xve -- .\myApplication.exe
```

You will see the result summary printed to the command prompt.

For more useful options, see the **Analysis Details** section below.

Example 2. Run the `collect.py` to profile an application and run the `analyze.py` to model its performance for the Intel® Arc™ graphics code-named Alchemist (xe_hpg_512xve configuration).

- On Linux OS:

1. Collect performance data.

```
adviser-python $APM/collect.py ./advi_results --collect=basic --config=xe_hpg_512xve -- ./myApplication
```

2. Model application performance.

```
adviser-python $APM/analyze.py ./advi_results --config=xe_hpg_512xve
```

You will see the result summary printed to the command prompt.

- On Windows OS:

1. Collect performance data.

```
adviser-python %APM%\collect.py .\advi_results --collect=basic --config=xe_hpg_512xve -- .\myApplication.exe
```

2. Model application performance.

```
adviser-python %APM%\analyze.py .\advi_results --config=xe_hpg_512xve
```

For more useful options, see the **Analysis Details** section below.

Analysis Details

Each script has a set of additional options that modify its behavior and collect additional performance data. The more analyses you run and options you use, the higher the modeling accuracy.

Collection Options

The following options are applicable to the `run_oa.py` and `collect.py` scripts.

Option	Description
<code>--collect=<mode></code>	<p>Specify data to collect for your application:</p> <ul style="list-style-type: none"> Use <code>basic</code> to run only Survey, Trip Counts and FLOP analyses, analyze data transfer between host and device memory, attribute memory objects to loops, and track accesses to stack memory. This value corresponds to the Medium accuracy. Use <code>refinement</code> to run only Dependencies analysis. Do not analyze data transfers. Use <code>full</code> (default) to run Survey, Trip Counts, FLOP, and Dependencies analyses, analyze data transfer between host and device memory and potential data reuse, attribute memory objects to loops, and track accesses to stack memory. This value corresponds to the High accuracy. <p>See Check How Assumed Dependencies Affect Modeling to learn when you need to collect dependency data.</p>
<code>--config=<config></code>	<p>Select a target GPU configuration to model performance for. For example, <code>xehpg_512xve</code> (default), <code>gen12_dg1</code>, or <code>gen9_gt3</code>.</p> <hr/> <p>Important For <code>collect.py</code>, make sure to specify the same value of the <code>--config</code> option for the <code>analyze.py</code>.</p> <hr/> <p>For details about configuration files, see config.</p>
<code>--markup=<markup-mode></code>	<p>Select loops to collect Trip Counts and FLOP and/or Dependencies data for with a pre-defined markup algorithm. This option decreases collection overhead.</p> <p>By default, it is set to <code>generic</code> to analyze all loops profitable for offloading.</p>
<code>--gpu</code>	<p>Analyze a SYCL, OpenCL, or OpenMP target application on a GPU device.</p> <p>For details about this workflow, see Run GPU-to-GPU Performance Modeling from Command Line.</p>

For a full list of available options, see:

- [run_oa.py Options](#)
- [collect.py Options](#)

Performance Modeling Options

The following options are applicable to the `run_oa.py` and `analyze.py` scripts.

Option	Description
<code>--config=<config></code>	<p>Select a target GPU configuration to model performance for. For example, <code>xehpg_512xve</code> (default), <code>gen12_dg1</code>, or <code>gen9_gt3</code>.</p> <hr/> <p>Important For <code>analyze.py</code>, make sure to specify the same value of the <code>--config</code> option for the <code>collect.py</code>.</p> <hr/> <p>For details about configuration files, see config.</p>
<code>--assume-parallel</code>	<p>Assume that a loop does not have dependencies if there is no information about the loop dependency type and you did not run the Dependencies analysis.</p>
<code>--data-reuse-analysis</code>	<p>Analyze potential data reuse between code regions when offloaded to a target GPU.</p> <hr/> <p>Important Make sure to use <code>--collect=full</code> when running the analyses with <code>collect.py</code> or use the <code>--data-transfer=full</code> when running the Trip Counts analysis with Intel Advisor CLI.</p> <hr/>
<code>--gpu</code>	<p>Analyze a SYCL, OpenCL, or OpenMP target application on a GPU device.</p> <p>For details about this workflow, see Run GPU-to-GPU Performance Modeling from Command Line.</p>

For a full list of available options, see:

- [run_oa.py Options](#)
- [analyze.py Options](#)

Next Steps

Continue to [explore the Offload Modeling results](#) with a preferred method. For details about the metrics reported, see [Accelerator Metrics](#).

See Also

Model Offloading to a GPU Find high-impact opportunities to offload/run your code and identify potential performance bottlenecks on a target graphics processing unit (GPU) by running the Offload Modeling perspective.

Command Line Interface This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

Minimize Analysis Overhead

Model MPI Application Performance on GPU You can model your MPI application performance on a target graphics processing unit (GPU) device to determine whether you can get a performance speedup from offloading the application to the GPU.

Offload Modeling Accuracy Levels in Command Line

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

You can generate commands for a desired accuracy level from the Intel Advisor GUI. See [Generate Command Lines from GUI](#) for details.

NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).

CPU-to-GPU Modeling

For the CPU-to-GPU modeling, the following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium	High
Overhead	5 - 10x	15 - 50x	50 - 80x
Goal	Model performance of an application that is mostly compute bound and does not have dependencies	Model application performance considering memory traffic for all cache and memory levels	Model application performance with all potential limitations for offload candidates
Analyses	Survey + Characterization (Trip Counts and FLOP) + Performance Modeling with no assumed dependencies	Survey + Characterization (Trip Counts and FLOP with cache simulation for the selected target device, callstacks, and light data transfer simulation) + Performance Modeling with no assumed dependencies	Survey + Characterization (Trip Counts and FLOP with cache simulation for the selected target device, callstacks, and medium data transfer simulation) + Dependencies + Performance Modeling with assumed dependencies
Result	Basic Offload Modeling report that shows potential speedup and performance metrics estimated on a target considering memory traffic from execution units to L1 cache only. The result might be inaccurate for memory-bound applications.	Offload Modeling report extended with data transfers estimated between host and device platforms considering memory traffic for all cache and memory levels	Offload Modeling report with detailed data transfer estimations and automated check for loop-carried dependencies for more accurate search for the most profitable regions to offload

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

NOTE Families of Intel® X^e graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® X^e Graphics](#).

Low Accuracy

To model application performance for with low accuracy for a default target device, run the following command:

```
advisor --collect=offload --accuracy=low --project-dir=./advi_results -- ./myApplication
```

This command runs the following analyses one by one:

1. Survey analysis:

```
advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

2. Characterization analysis to collect trip count and FLOP data

```
advisor --collect=tripcounts --flop --auto-finalize --target-device=xehpg_512xve --project-dir=./advi_results -- ./myApplication
```

3. Performance modeling:

```
advisor --collect=projection --no-assume-dependencies --config=xehpg_512xve --project-dir=./advi_results
```

Medium Accuracy

This accuracy is set by default. To model application performance with medium accuracy for a default target device, run the following command:

```
advisor --collect=offload --project-dir=./advi_results -- ./myApplication
```

This command runs the following analyses one by one:

1. Survey analysis:

```
advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

2. Characterization analysis to collect trip count and FLOP data

```
advisor --collect=tripcounts --flop --stacks --auto-finalize --cache-simulation=single --data-transfer=light --target-device=xehpg_512xve --project-dir=./advi_results -- ./myApplication
```

3. Performance modeling:

```
advisor --collect=projection --no-assume-dependencies --config=xehpg_512xve --project-dir=./advi_results
```

High Accuracy

To model application performance with high accuracy for a default target device, run the following command:

```
advisor --collect=offload --accuracy=high --project-dir=./advi_results -- ./myApplication
```

This command runs the following analyses one by one:

1. Survey analysis:

```
advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

2. Characterization analysis to collect trip count and FLOP data

```
advisor --collect=tripcounts --flop --stacks --auto-finalize --cache-simulation=single --target-device=xehpg_512xve --data-transfer=medium --project-dir=./advi_results -- ./myApplication
```

3. Dependencies analysis:

```
advisor --collect=dependencies --filter-reductions --loop-call-count-limit=16 --select
markup=gpu_generic --project-dir=./advi_results -- ./myApplication
```

4. Performance modeling:

```
advisor --collect=projection --config=xehpg_512xve --project-dir=./advi_results
```

See [Check How Dependencies Affect Modeling](#) for a recommended strategy to check for loop-carried dependencies.

GPU-to-GPU Modeling

For the GPU-to-GPU modeling, the following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium	High
Overhead	5 - 10x	15 - 50x	15 - 50x
Goal	Model performance of an application that is mostly compute bound	Model application performance considering memory traffic for all cache and memory levels	Model application performance with all potential limitations for offload candidates
Analyses	Survey + Characterization (Trip Counts and FLOP) + Performance Modeling	Survey + Characterization (Trip Counts and FLOP with light data transfer simulation) + Performance Modeling	Survey + Characterization (Trip Counts and FLOP with medium data transfer simulation) + Performance Modeling
Result	Basic Offload Modeling report that shows potential speedup and performance metrics estimated on a target considering memory traffic from execution units to L1 cache only. The result might be inaccurate for memory-bound applications.	Offload Modeling report extended with data transfers estimated between host and device platforms	Offload Modeling report with detailed data transfer estimations for more accurate search for the most profitable regions to offload

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Low Accuracy

To model application performance for with low accuracy for a default target device, run the following command:

```
advisor --collect=offload --accuracy=low --gpu --project-dir=./advi_results -- ./myApplication
```

This command runs the following analyses one by one:

1. Survey analysis:

```
advisor --collect=survey --auto-finalize --static-instruction-mix --profile-gpu --project-dir=./advi_results -- ./myApplication
```

2. Characterization analysis to collect trip count and FLOP data

```
advisor --collect=tripcounts --flop --auto-finalize --target-device=xehpg_512xve --profile-gpu --project-dir=./advi_results -- ./myApplication
```

3. Performance modeling:

```
advisor --collect=projection --no-assume-dependencies --config=xehpg_512xve --profile-gpu --project-dir=./advi_results
```

Medium Accuracy

This accuracy is set by default. To model application performance with medium accuracy for a default target device, run the following command:

```
advisor --collect=offload --gpu --project-dir=./advi_results -- ./myApplication
```

This command runs the following analyses one by one:

1. Survey analysis:

```
advisor --collect=survey --auto-finalize --static-instruction-mix --profile-gpu --project-dir=./advi_results -- ./myApplication
```

2. Characterization analysis to collect trip count and FLOP data

```
advisor --collect=tripcounts --flop --auto-finalize --data-transfer=light --target-device=xehpg_512xve --profile-gpu --project-dir=./advi_results -- ./myApplication
```

3. Performance modeling:

```
advisor --collect=projection --no-assume-dependencies --config=xehpg_512xve --profile-gpu --project-dir=./advi_results
```

High Accuracy

To model application performance with high accuracy for a default target device, run the following command:

```
advisor --collect=offload --accuracy=high --gpu --project-dir=./advi_results -- ./myApplication
```

This command runs the following analyses one by one:

1. Survey analysis:

```
advisor --collect=survey --auto-finalize --static-instruction-mix --profile-gpu --project-dir=./advi_results -- ./myApplication
```

2. Characterization analysis to collect trip count and FLOP data

```
advisor --collect=tripcounts --flop --auto-finalize --target-device=xehpg_512xve --profile-gpu --data-transfer=medium --project-dir=./advi_results -- ./myApplication
```

3. Performance modeling:

```
advisor --collect=projection --config=xehpg_512xve --profile-gpu --project-dir=./advi_results
```

After you run the perspective, you can [view the results](#) in the Intel Advisor GUI, in CLI, or an interactive HTML report.

See Also

[Run Offload Modeling Perspective from Command Line](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

Run GPU-to-GPU Performance Modeling from Command Line

With Intel® Advisor, you can model performance of SYCL, OpenCL™, or OpenMP target application running on a graphics processing unit (GPU) for a different GPU device without its CPU version. For this, run the GPU-to-GPU modeling workflow of the Offload Modeling perspective.*

The GPU-to-GPU modeling analyzes only GPU compute kernels and ignores the application parts executed on a CPU. As a result, there are several changes in the modeling flow:

- Compute kernels characteristics are collected with the Intel Advisor GPU profiling capabilities.
- High-overhead features, such as call stack handling, cache simulation, data transfer simulation, dependencies analysis, are disabled.
- Instead of CPU-to-GPU data transfer simulation, memory objects transferred between host and device memory are traced.

Workflow

The GPU-to-GPU performance modeling workflow is similar to the CPU-to-GPU modeling and includes the following steps:

1. Survey analysis measures execution time, cache, and GTI traffic using hardware counters for GPU-enabled kernels running on an Intel® Graphics.
2. Characterization analysis measures the number of compute operations counting different GPU instructions separately for kernels running on a Intel Graphics. For example, it implements separate counters for hardware-implemented 32-bit math functions, such as SQRT, EXP, DIV.
3. Performance Modeling analysis models performance of all kernels on a target GPU device, whether they are profitable or not.

NOTE For correct memory object tracing, GPU kernels should run with the oneAPI Level Zero back end.

Prerequisites

1. [Configure your system](#) to analyze GPU kernels.
2. [Set Intel Advisor environment variables](#) with an automated script to enable Intel Advisor command line interface.

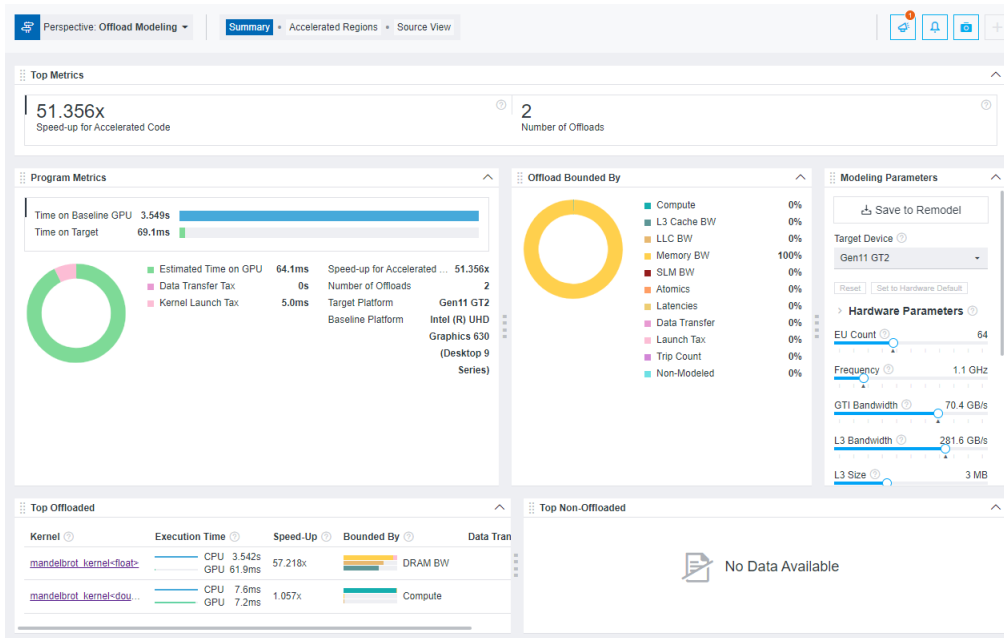
Run the GPU-to-GPU Performance Modeling

To run the GPU-to-GPU performance modeling from command line, you can use *one* of the following methods:

- **Method 1.** [Run a collection preset](#) using Intel Advisor command line interface (CLI) to execute multiple analyses with a single command and control modeling accuracy.
- **Method 2.** [Run analyses separately](#) using Intel Advisor CLI if you need more advanced customization for each analysis.
- **Method 3.** [Run Python* scripts](#) if you need more customization for collection and modeling steps.

You can also run the GPU-to-GPU Offload Modeling from Intel Advisor graphical user interface (GUI). See [Run Offload Modeling Perspective from GUI](#).

After you run the Offload Modeling with any method above, you can [view the results](#) in Intel Advisor graphical user interface (GUI), command line interface (CLI), or an interactive HTML report. For example, the interactive HTML report is similar to the following:



Tip If you want to analyze an MPI application, you can generate pre-configured command lines, copy them, and run one by one. For details, see [Generate Pre-configured Command Lines](#).

Method 1. Use Collection Preset

To run the collection preset for the GPU-to-GPU modeling, use the `--gpu` option with the `--collect=offload` action. When you run the collection, it sequentially runs data collection on a GPU and performance modeling steps. The specific analyses and options depend on the *accuracy level* you specify for the collection.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

For example, to run the GPU-to-GPU modeling with the default (medium) accuracy level:

- On Linux* OS:

```
advisor --collect=offload --gpu --project-dir=./advi_results -- ./myApplication
```

- On Windows* OS:

```
advisor --collect=offload --gpu --project-dir=.\\advi_results -- myApplication.exe
```

The collection progress and commands for each analysis executed will be printed to a terminal or a command prompt. When the collection is finished, you will see the result summary.

You can also specify a different accuracy level to change analyses to run and their options. Available accuracy levels are `low`, `medium` (default), and `high`.

For example, run the high accuracy level:

```
advisor --collect=offload --accuracy=high --gpu --project-dir=./advi_results -- ./myApplication
```

If you want to see the commands that are executed at each accuracy level, you can run the collection with the `--dry-run` and `--gpu` options. The commands will be printed to a terminal or a command prompt.

For details about each accuracy level, see [Offload Modeling Accuracy Levels in Command Line](#).

Method 2. Use per-Analysis Collection

You can collect data and model performance for your application by running each Offload Modeling analysis in a separate command for more advanced customization. To enable the GPU-to-GPU modeling, use the `--profile-gpu` option for each analysis you run.

Consider the following workflow example. Using this example, you can run the Survey, Trip Counts, and FLOP analyses to profile an application running on a GPU and the Performance Modeling to model its performance on Intel® Iris® Xe MAX graphics (gen12_dg1 configuration).

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

On Linux OS:

1. Run the Survey analysis.

```
advisor --collect=survey --profile-gpu --project-dir=./advi_results -- ./myApplication
```

2. Run the Trip Counts and FLOP analyses with data transfer estimation.

```
advisor --collect=tripcounts --profile-gpu --flop --target-device=gen12_dg1 --data-transfer=light --project-dir=./advi_results -- ./myApplication
```

3. Run the Performance Modeling analysis.

```
advisor --collect=projection --profile-gpu --config=gen12_dg1 --project-dir=./advi_results
```

You will see the result summary printed to the command prompt.

On Windows OS:

1. Run the Survey analysis.

```
advisor --collect=survey --project-dir=.\advi_results -- myApplication.exe
```

2. Run the Trip Counts and FLOP analyses with data transfer.

```
advisor --collect=tripcounts --profile-gpu --flop --stacks --target-device=gen12_dg1 --data-transfer=light --project-dir=.\advi_results -- myApplication.exe
```

3. Run the Performance Modeling analysis.

```
advisor --collect=projection --profile-gpu --config=gen12_dg1 --project-dir=.\advi_results
```

You will see the result summary printed to the command prompt.

Method 3. Use Python* Scripts

Intel Advisor has three scripts that use the Intel Advisor Python* API to run the Offload Modeling - `run_oa.py`, `collect.py`, `analyze.py`. You can run the scripts with the `advisor-python` command line interface of the Intel Advisor or with your local Python 3.6 or 3.7.

To enable the GPU-to-GPU modeling, use the `--gpu` option for each script you run.

NOTE The scripts *do not* support MPI applications. To analyze an MPI application, use the per-analysis collection with the Intel Advisor CLI.

You can run the Offload Modeling using different combinations of the scripts and/or the Intel Advisor CLI. For example:

- Run `run_oa.py` to profile application and model its performance.
- Run the `collect.py` to profile application and `analyze.py` to model its performance. Re-run `analyze.py` to remodel with a different configuration.

- Run the Intel Advisor CLI to collect performance data and `analyze.py` to model performance. Re-run `analyze.py` to remodel with a different configuration.
- Run `run_oa.py` to collect data and model performance for the first time and run `analyze.py` to remodel with a different configuration.

Consider the following examples of some typical scenarios with Python scripts.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Example 1. Run the `run_oa.py` script to profile an application on GPU and model its performance for Intel® Iris® Xe MAX graphics (`gen12_dg1` configuration).

- On Linux OS:

```
adviser-python $APM/run_oa.py ./advi_results --gpu --config=gen12_dg1 -- ./myApplication
```

- On Windows OS:

```
adviser-python %APM%\run_oa.py .\advi_results --gpu --config=gen12_dg1 -- myApplication.exe
```

You will see the result summary printed to the command prompt.

Example 2. Run the `collect.py` to profile an application on GPU and run the `analyze.py` to model its performance for Intel® Iris® Xe MAX graphics (`gen12_dg1` configuration).

- On Linux OS:

1. Collect performance data.

```
adviser-python $APM/collect.py ./advi_results --gpu --config=gen12_dg1 -- ./myApplication
```

2. Model application performance.

```
adviser-python $APM/analyze.py ./advi --gpu --config=gen12_dg1
```

You will see the result summary printed to the command prompt.

- On Windows OS:

1. Collect performance data.

```
adviser-python %APM%\collect.py .\advi_results --collect=basic --gpu --config=gen12_dg1 -- myApplication.exe
```

2. Model application performance.

```
adviser-python %APM%\analyze.py .\advi_results --gpu --config=gen12_dg1
```

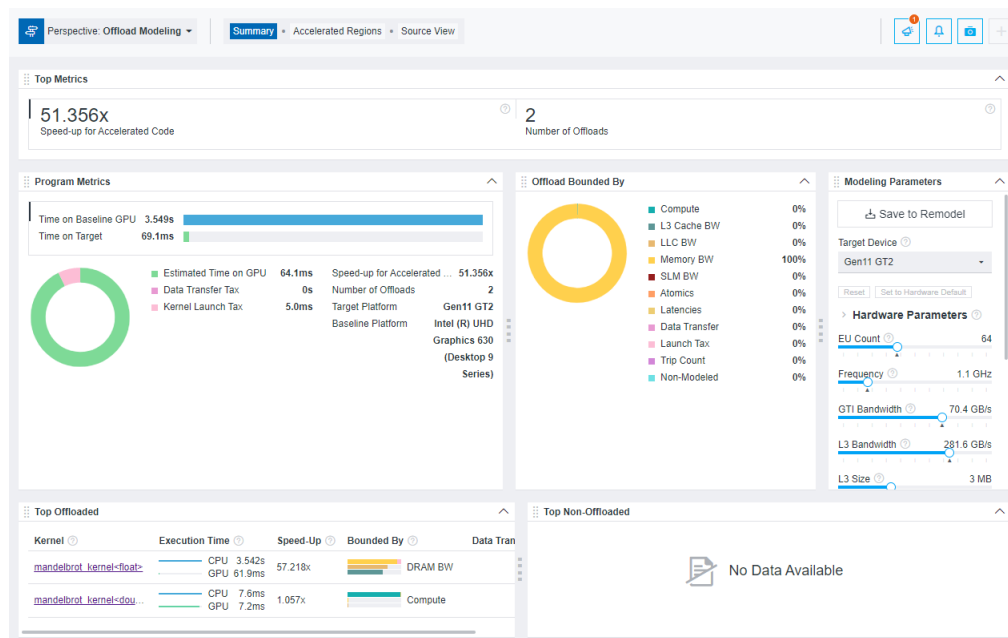
You will see the result summary printed to the command prompt.

View the Results

Once the Intel Advisor finishes the analyses, the result is available in the following formats:

- Review the result summary and a result file location printed to a command prompt or a terminal.
- Review the project result in the Intel Advisor GUI generated to the project directory.
- Review HTML reports generated to the `<project-dir>/e<NNN>/report` directory. View the detailed information about HTML reports in [Work with Standalone HTML Reports](#).
- Review a set of reports generated to the `<project-dir>/e<NNN>/pp<NNN>/data.0` directory. The directory includes the main report in HTML format named as `report.html` and a set of CSV files with detailed metric tables.

For example, the result in the Intel Advisor GUI looks as follows:



To explore the interactive HTML report, you can [download precollected Offload Modeling reports](#) and examine the results and structure.

See [Explore Offload Modeling Results](#) for details about available result formats and [Explore Performance Gain from GPU-to-GPU Modeling](#) for details about the GPU-to-GPU modeling result.

See Also

[advisor Command Option Reference](#)

[Offload Modeling Command Line Reference](#) This reference section describes the command line options available for each of the Python* scripts that you can use to run the Offload Modeling perspective.

Explore Offload Modeling Results

Intel® Advisor provides several ways to work with the Offload Modeling results generated from the command line.

View Results in CLI

When you run the Offload Modeling perspective from command line,, the *result summary* is printed in a terminal or a command prompt. In this summary report, you can view:

- Description of a *baseline* device where application performance was measured and a target device for which the application performance was modeled
- Executive binary name
- Top metrics for measured and estimated (accelerated) application performance
- Top regions recommended for offloading to the target and performance metrics per region

For example:

```
Info: Selected accelerator to analyze: Intel(R) Gen11 Integrated Graphics Accelerator 64EU.
Info: Baseline Host: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz, GPU: Intel (R) .
Info: Binary Name: 'CFD'.
Info: An unknown atomic access pattern is specified: partial_sums_16. Possible values are same, sequential. sequential will be used.
```

```
Measured CPU Time: 44.858s    Accelerated CPU+GPU Time: 16.265s
```

Speedup for Accelerated Code: 3.5x Number of Offloads: 7 Fraction of Accelerated Code: 60%

Top Offloaded Regions

Location		CPU	GPU	Estimated
Speedup	Bounded By	Data Transferred		
[loop in compute_flux_ser at euler3d_cpu_ser.cpp:226]		36.576s	9.340s	
3.92x L3_BW	12.091MB			
[loop in compute_step_factor_ser at euler3d_cpu_ser....]		0.844s	0.101s	
8.37x LLC_BW	4.682MB			
[loop in time_step_ser at euler3d_cpu_ser.cpp:361]		0.516s	0.278s	
1.86x L3_BW	10.506MB			
[loop in time_step_ser at euler3d_cpu_ser.cpp:361]		0.456s	0.278s	
1.64x L3_BW	10.506MB			
[loop in time_step_ser at euler3d_cpu_ser.cpp:361]		0.432s	0.278s	
1.55x L3_BW	10.506MB			

See [Accelerator Metrics](#) reference for more information about the metrics reported.

View Results in GUI

If you run the *Offload Modeling perspective from command line*, an `.advixeproj` project is created automatically in the directory specified with `--project-dir`. This project is interactive and stores all the collected results and analysis configurations. You can view it in the Intel Advisor GUI.

To open the project in GUI, you can run the following command from a command prompt:

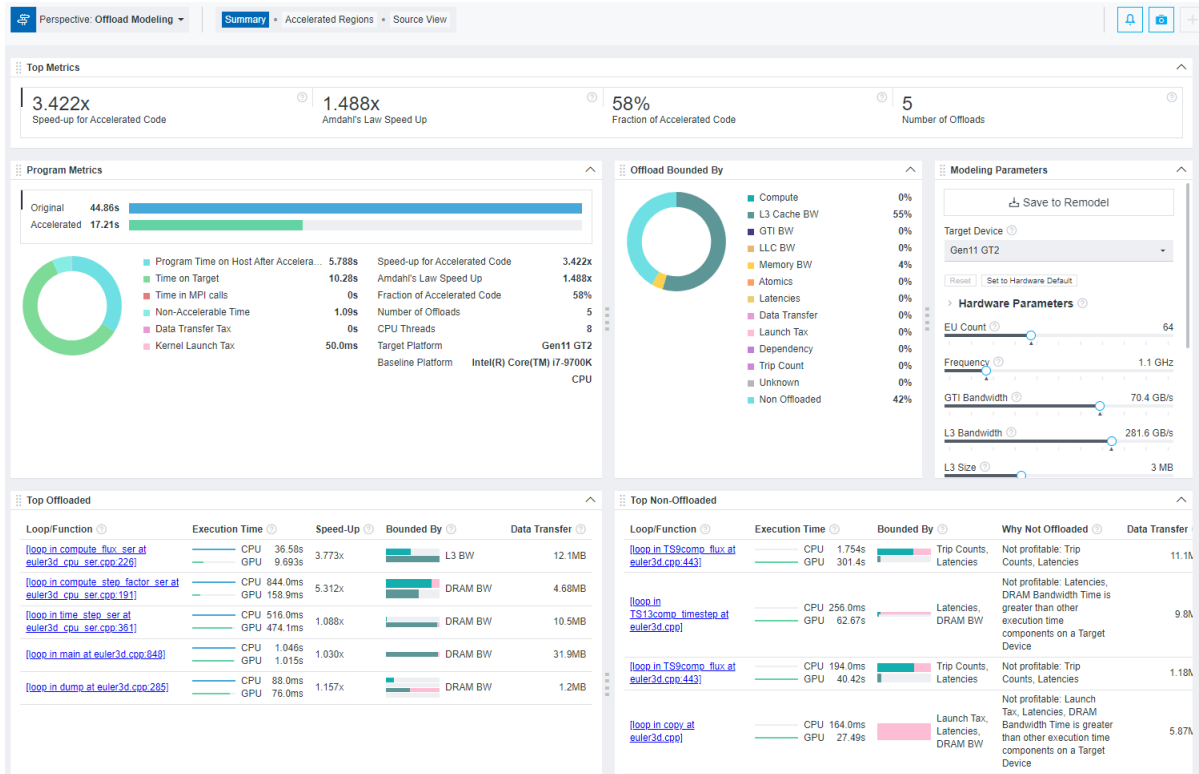
```
advisor-gui <project-dir>
```

NOTE If the report does not open, click **Show Result** on the Welcome pane.

If you run the *Offload Modeling perspective from GUI*, the result is opened automatically after the collection finishes.

You first see a **Summary** report that includes the most important information about measured performance on a baseline device and modeled performance on a target device, including:

- Main metrics for the modeled performance of your program that indicates if you should offload your application to a target device.
- Specific factors that prevent your code from achieving a better performance if executed on a target device in the Offload Bounded by.
- Top five offloaded loops/functions that provide the highest benefit and top five not offloaded loops/functions with the reason why they were not offloaded.



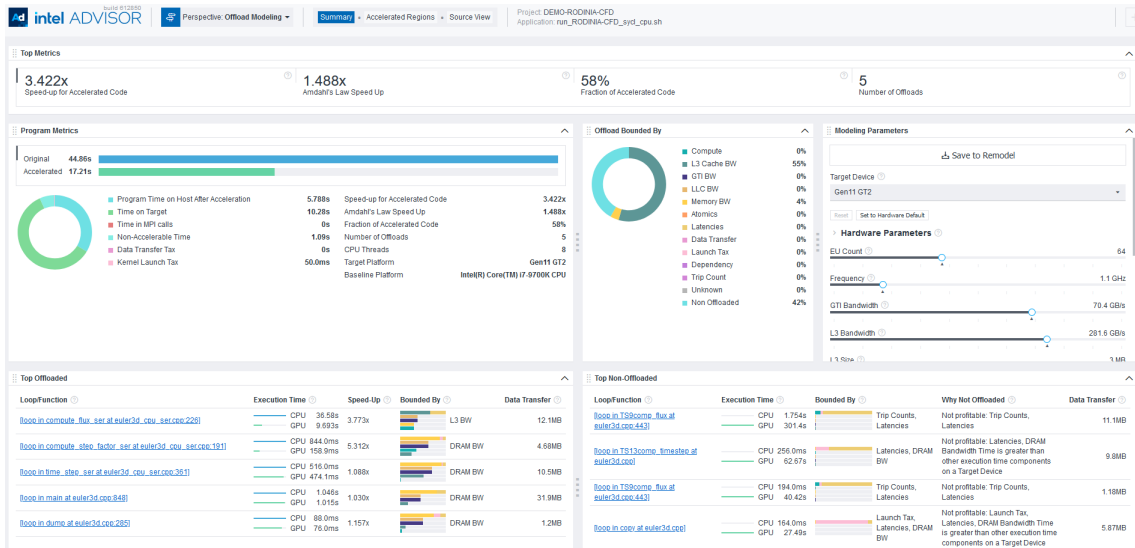
View an Interactive HTML Report

When you execute Offload Modeling from CLI, Intel Advisor automatically saves two types of HTML reports in the `<project-dir>/e<NNN>/report` directory:

- Interactive HTML report that represents results in the similar way as GUI and enables you to view key estimated metrics for your application: `advisor-report.html`

Tip Collect GPU Roofline data to view results for Offload Modeling and GPU Roofline Insights perspectives in a single interactive HTML report.

- Legacy HTML report that enables you to get the detailed information about functions in a call tree, download a configuration file for a target accelerator, and view perspective execution logs: `report.html`.



For details about HTML reports and instructions on exporting them if you run the Offload Modeling from GUI, see [Work with Standalone HTML Reports](#).

To explore the interactive HTML report, you can [download precollected Offload Modeling reports](#) and examine the results and structure.

An additional set of reports is generated in the `<project-dir>/e<NNN>/pp<NNN>/data0` directory, including:


- Multiple CSV reports for different metric groups, such as `report.csv`, `whole_app_metrics.csv`, `bounded_by_times.csv`, `latencies.csv`.
- A graphical representation of the call tree showing the offloadable and accelerated regions named as `program_tree.dot`.
- A graphical representation of the call tree named as `program_tree.pdf`, which is generated if a DOT* utility is installed on your system.
- LOG files, which can be used for debugging and reporting bugs and issues.

These reports are light-weighted and can be easily shared as they do not require Intel Advisor GUI.

Save a Read-only Result Snapshot

A snapshot is a read-only copy of a project result, which you can view at any time using the Intel Advisor GUI. You can save a snapshot for a project using Intel Advisor GUI or CLI.



To save an active project result as a read-only snapshot from GUI: Click the  button in the top ribbon of the report. In the **Create a Result Snapshot** dialog box, enter the snapshot details and save it.

To save an active project result as a read-only snapshot from CLI:

```
advisor --snapshot --project-dir=<project-dir> [--cache-sources] [--cache-binaries] --<snapshot-path>
```

where:

- `--cache-sources` is an option to add application source code to the snapshot.
- `--cache-binaries` is an option to add application binaries to the snapshot.
- `<snapshot-path>` is a path and a name for the snapshot. For example, if you specify `/tmp/new_snapshot`, a snapshot is saved in a `tmp` directory as `new_snapshot.advixeexpz`. You can skip this and save the snapshot to a current directory as `snapshotXXX.advixeexpz`.

To open the result snapshot in the Intel Advisor GUI, you can run the following command:

```
advisor-gui <snapshot-path>
```

You can visually compare the saved snapshot against the current active result or other snapshot results.

See [Create a Read-only Result Snapshot](#) for details.

Result Interpretation

When you run the Offload Modeling perspective, depending on a configuration chosen, the report shows a different level of details:

- [Examine regions recommended for offloading](#) and view estimated performance of your application after offloading to a target platform assuming it is mostly bounded by compute limitations. You need to run at least the Survey, Trip Counts and FLOP (Characterization), and Performance Modeling analyses (**Low** accuracy) to collect this data.
- [Examine data transfers estimated for modeled regions](#) and view estimated performance with data transfer estimations between host and target platforms for all memory levels and total data for loop/function. You need to run at least the Survey, Trip Counts and FLOP with callstacks, light data transfer simulation, and cache simulation (Characterization), and Performance Modeling analyses (**Medium** accuracy) to collect this data.
- [Check for dependencies issues](#) and view a more accurate performance estimated considering loop/function dependencies. You need to run at least the Survey, Trip Counts and FLOP with callstacks, cache simulation, and medium data transfer simulation (Characterization), Dependencies, and Performance Modeling analyses (**High** accuracy) to collect this data.
- [Explore performance gain from GPU-to-GPU modeling](#) to see how your SYCL, OpenMP* target, or OpenCL™ application can have a better performance if you run it on a different graphics processing unit (GPU) device.
- [Investigate non-offloaded code regions](#) and understand why they are not profitable to run on a target platform. The higher accuracy level you run, the more accurate offload recommendations and non-offloaded reasons are.

For a general overview of the report, see [Offload Modeling Report Overview](#).

See Also

[Run Offload Modeling Perspective from GUI](#)

[Run Offload Modeling Perspective from Command Line](#)

[Accelerator Metrics](#) This reference section describes the contents of data columns in reports of the Offload Modeling and GPU Roofline Insights perspectives.

Offload Modeling Report Overview

Review the controls available in the main report of the Offload Modeling perspective. You can view the interactive HTML report or a graphical user interface.

In the **Accelerated Regions** and **Summary** reports, you can drag and drop, close/open, collapse/expand panes to change the report layout.

The screenshot displays the Intel Advisor interface with several key components:

- Top Bar:** Includes tabs for Perspective, Offload Modeling, Summary, Accelerated Regions, and Source View. It also shows performance metrics like 3.5x speedup, 1.5x Amdahl's Law, and 60% fraction of accelerated code.
- Code Regions (5):** A table listing code regions with columns for Loop/Function, Performance Issues, Time, Baseline Device, Dependency Type, Programming Model, Iteration Space, and Measured. The first row shows a loop in `compute_flux_ser` at `euler3d_cpu_ser.cpp` with a time of 36.58s and a speedup of 3.916x.
- Data Transfer Estimations (6):** A panel showing estimated and measured times, bounded by L3 BW, and various performance metrics like Compute, DRAM BW, L3 BW, LLC BW, Atomic Throughput, Data Transfer Tax, Kernel Launch Tax, Load Latency, and Estimated Time.
- Source (7):** A code editor showing the source code for the selected region, with a 'Top-Down' view and 'Recommendations' tab. The code is for a loop in `compute_flux_ser` at `euler3d_cpu_ser.cpp`.

1

Switch between perspectives or different Offload Modeling report tabs.

2



Review the summary of *estimated* offload characteristics for your application. The pane highlights total speedup, number of loops and functions offloaded, and a fraction of code accelerated.



3

Select code regions to show in the report based on offload type:

- Show *all* code regions in your code.
- Show only code regions *recommended* for offloading.
- Show only code regions *not recommended* for offloading.

4

-  Click the  button to show per-program recommendations, which you can collapse/expand.

-  Click the  button to see the collection log including featured events separated by analyses, full collection log, and application output.

5

- Create a snapshot for the current project results. For details, see [Create a Read-only Result Snapshot](#).
- Click a + button to open previously closed panes.

Review application performance measured on a host platform and its performance modeled on a target platform. For details about metrics reported, see [Accelerator Metrics](#).

Depending on a perspective configuration and accuracy level, you might see different metrics reported and some metrics might be not accurate. Refer to the following topics for interpretation details:

- Low accuracy: [Examine Regions Recommended for Offloading](#)
- Medium accuracy: [Examine Data Transfers for Modeled Regions](#)
- High accuracy: [Check for Dependency Issues](#)

6

Switch between **Data Transfer Estimations** and **Details** tabs for more information about loops selected in **Code Regions**:

- Examine details about estimated data transfers in the loop and memory objects tracked and review data transfer recommendations. For details about how to read this pane, see [Examine Data Transfers for Modeled Regions](#).

NOTE You need to enable light or full data transfer analysis before running the perspective to see metrics in this pane,

7

- Examine offload details about the loop, such as loop type, measured and estimated execution time, estimated offload speed-up (for offloaded loops), and reason for not offloading, time for several bounded-by factors. If the loop is not recommended for offloading, this tab also reports the reason for it. For details, see [Investigate Non-Offloaded Code Regions](#).

Switch between **Source** and **Top-Down** tabs for more information about loops selected in **Code Regions**:

- Examine the source code and offload details for each source line. Select a loop in the **Code Regions** table to focus on the corresponding part of the source code.
- View the loop/function hierarchy in a stack and its metrics.

Examine Regions Recommended for Offloading

Accuracy Level

Low

Enabled Analyses

Survey + Characterization (Trip Counts and FLOP) + Performance Modeling with no assumed dependencies

Result Interpretation

After running the Offload Modeling perspective with **Low** accuracy, you get a basic Offload Modeling report, which shows you estimated performance of your application after offloading to a target platform assuming it is mostly bounded by compute limitations.

Offload Modeling with **Low** accuracy assumes that:

- There is no tax for transferring data between baseline and target platforms.
- All data goes to L1/L3 cache level only. L1/L3 cache traffic estimation might be inaccurate.
- A loop is parallel if the loop dependency type is unknown (**Assume Dependencies** checkbox is disabled). This happens when there is no information about a loop dependency type from a compiler or the loop is not explicitly marked as parallel, for example, with a programming model (OpenMP*, SYCL, Intel® oneAPI Threading Building Blocks (oneTBB))

NOTE This topic describes data as it is shown in the Offload Modeling report in the Intel Advisor GUI and an interactive HTML report.

In the Offload Modeling report:

1. Review the metrics for the whole application in the **Summary** tab.
 - Check if your application is profitable to offload to a target device or if it has a better performance on a baseline platform in the **Program Metrics** panes.
 - See what prevents your code from achieving a better performance if executed on a target device in the **Offload Bounded by** pane.

NOTE If you enable **Assume Dependencies** option for the **Performance Modeling** analysis, you might see high percentage of dependency-bound code regions. You are recommended run the Dependencies analysis and rerun Performance Modeling to get more accurate results.

2. If the estimated speed-up is high enough and other metrics in the **Summary** pane suggest that your application can benefit from offloading to a selected target platform, you can start offloading your code.
3. If you want to investigate the results reported for each region in more detail, go to the **Accelerated Regions** tab and select a code region:

Code Regions									
Loop/Function	Performance Issues	Measured			Basic Estimated Metrics				
		Time	Iteration Space	Speed-Up	Time	Offload Summary	Why Not Offloaded		
► [loop in compute_flux_ser at euler3d_cpu_ser.cp	Code re...	36.58s	Call Count 6000 Trip Count 506	3.916x	9.34s	Offloaded			L3 BV
► [loop in time_step_ser at euler3d_cpu_ser.cpp:3i	Code re...	516.0ms	Call Count 2000 Trip Count 506	1.859x	277.5ms	Offloaded			L3 BV
► [loop in time_step_ser at euler3d_cpu_ser.cpp:3i	Code re...	456.0ms	Call Count 2000 Trip Count 506	1.643x	277.5ms	Offloaded			L3 BV
► [loop in time_step_ser at euler3d_cpu_ser.cpp:3i	Code re...	432.0ms	Call Count 2000 Trip Count 506	1.554x	277.9ms	Offloaded			L3 BV
► [loop in compute_step_factor_ser at euler3d_cpu...	Code re...	844.0ms	Call Count 2000 Trip Count 506	8.370x	100.8ms	Offloaded			LLC E
► [loop in verify_ser at euler3d_cpu_ser.cpp:437]		252.0ms	Call Count 1 Trip Count 97046	0.999x	252.2ms	Not offloaded	Not profitable: L3 B...		L3 BV
► [loop in TS9comp_flux at euler3d.cpp:443]		1.754s	Call Count 2742825 Trip Count 22	0.007x	255.4s	Not offloaded	Not profitable: The ...		Trip C
► [loop in TS13comp_timestep at euler3d.cpp]		256.0ms	Call Count 2799013 Trip Count 22	0.005x	47.78s	Not offloaded	Not profitable: The ...		Trip C
► [loop in TS13comp_step_fac at euler3d.cpp]		84.0ms	Call Count 881423 Trip Count 21 AVX2	0.016x	5.185s	Not offloaded	Not profitable: Lau...		Launc
► [loop in TS9comp_flux at euler3d.cpp:443]		194.0ms	Call Count 293176 Trip Count 3	0.006x	34.04s	Not offloaded	Not profitable: The ...		Trip C
			Call Count 236988						

Data Transfer Estimations Details

Why Not Offloaded: Not profitable: L3 Bandwidth Time is greater than other execution time components on a Target Device

ESTIMATED SPEED-UP: 0.999X

Estimated Time 252.2ms Measured Time 252.0ms

BOUNDED BY: L3 BW

Compute < 0.1ms
 DRAM BW 0.1ms
 L3 BW 0.2ms
 LLC BW 0.2ms
 Atomic Throughput 0ms
 Data Transfer Tax 0ms
 Kernel Launch Tax < 0.1ms
 Load Latency < 0.1ms
 Total Time In System Calls 252.0ms
 Estimated Time 252.2ms

- Check whether your target code region is recommended for offloading to a selected platform. In the **Basic Estimated Metrics** column group, review the **Offload Summary** column. The code region is considered profitable for offloading if estimated speed-up is more than 1, that is, estimated time execution on a target device is smaller than on a host platform.

If your code region of interest is not recommended for offloading, consider re-running the perspective with a higher accuracy or refer to [Investigate Not Offloaded Code Regions](#) for recommendations on how to model offloading for this code region.

- Examine the **Bounded By** column of the **Estimated Bounded By** group in the code region pane to identify the main bottleneck that limits the performance of the code region (see the [Bounded by section](#)). The metrics show one or more bottleneck(s) in a code region.
 - In the **Throughput** column of the **Estimated Bounded By** group, review time spent for compute- and L3 cache bandwidth-bound parts of your code. If the value is high, consider optimizing compute and/or L3 cache usage in your application.
 - Review the metrics in the **Compute Estimates** column group to see the details about instructions and number of threads used in each code region.
- Get guidance for offloading your code to a target device and optimizing it so that your code benefits the most in the **Recommendations** tab. If the code region has room for optimization or underutilizes the capacity of the target device, Intel Advisor provides you with hints and code snippets that may be helpful to you for further code improvement.
 - View the offload summary and details for the selected code region in the **Details** pane.

For details about metrics reported, see [Accelerator Metrics](#).

Next Steps

- If you think that the estimated speedup is enough and the application is ready to be offloaded, rewrite your code to offload profitable code regions to a target platform and measure performance of GPU kernels with [GPU Roofline Insights](#) perspective.
- Consider running the Offload Modeling perspective with a [higher accuracy level](#) to get a more detailed report.

See Also

- [Optimize Your GPU Application with the Intel® oneAPI Base Toolkit](#)
- [Intel® oneAPI Programming Guide](#)

Examine Data Transfers for Modeled Regions

Accuracy Level

Medium

Enabled Analyses

Survey + Characterization (Trip Counts and FLOP with cache simulation and light data transfer simulation) + Performance Modeling with no assumed dependencies

Result Interpretation

After running the Offload Modeling perspective with **Medium** accuracy, you get an extended Offload Modeling report, which provides information about memory and cache usage and taxes of your offloaded application. In addition to the basic data, the result includes:

- More accurate estimations of traffic and time for all cache and memory levels.
- Measured data transfer and estimated data transfer between host and device memory.
- Total data for the loop/function from different callees.

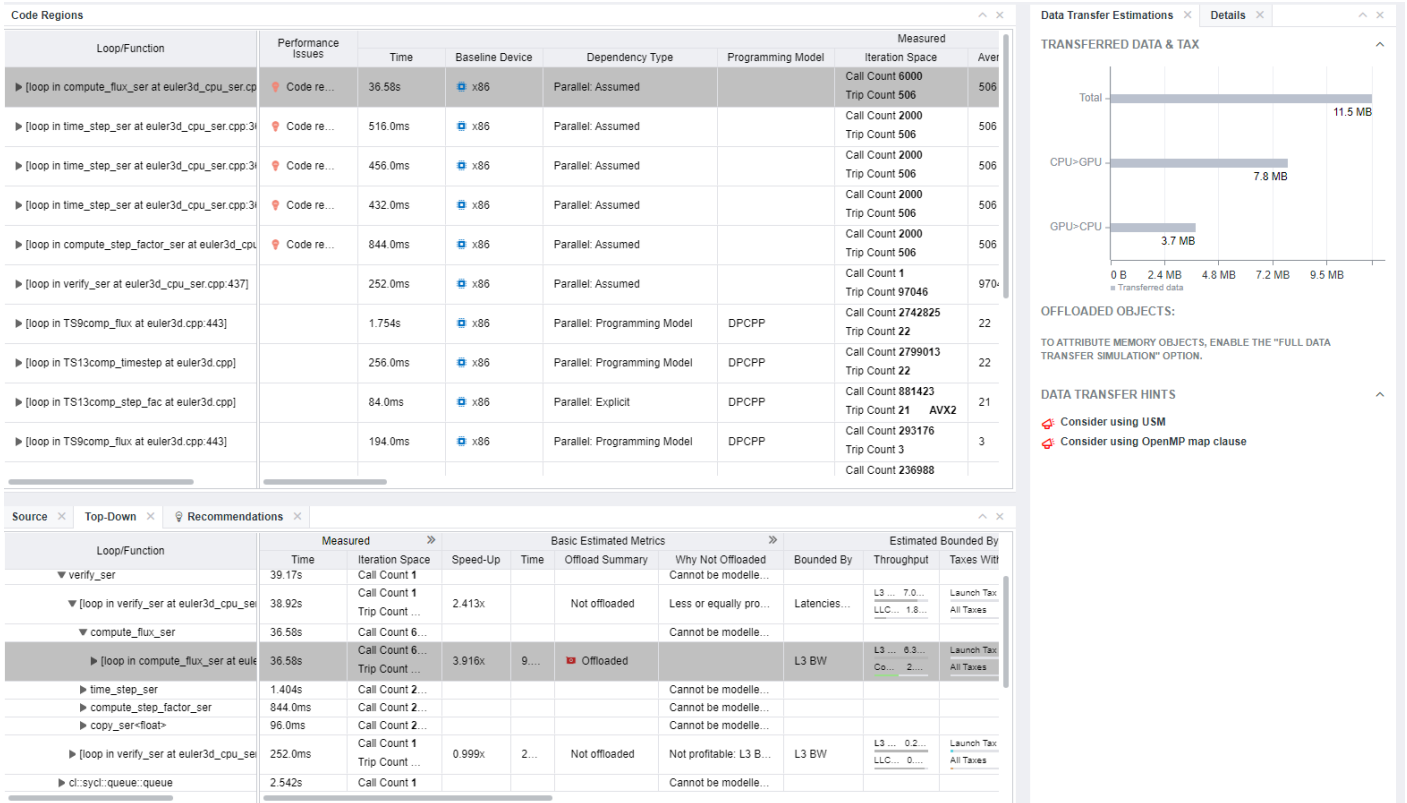
NOTE When profiling a GPU application with **Light** data transfer simulation mode, you will get memory traffic estimation only for CPU code.

Offload Modeling perspective assumes a loop is parallel if its dependency type is unknown. It means that there is no information about a loop from a compiler or the loop is not explicitly marked as parallel, for example, with a programming model (OpenMP*, SYCL, Intel® oneAPI Threading Building Blocks).

If you had a report generated for a lower accuracy, all offload recommendations, metrics, and speed-up will be updated to be more precise taking into account new data.

NOTE This topic describes data as it is shown in the Offload Modeling report in the Intel Advisor GUI and an interactive HTML report.

In the **Accelerated Regions** tab of the Offload Modeling report, review the metrics about memory usage and data transfers.



- In the **Code Regions** metrics table:

- In the **Estimated Bounded By** column, review how much time is spent to transfer data (data transfer tax). In the **Taxes with Reuse** column, see the biggest and total time taxes paid for offloading a code regions to a target platform.

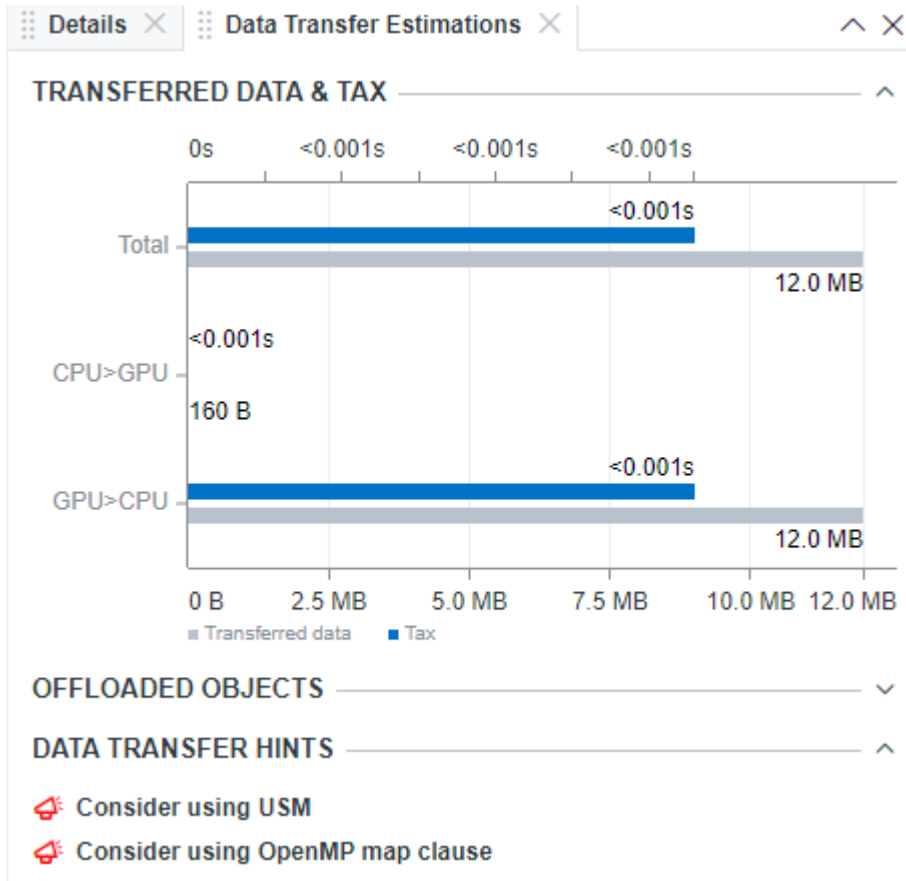
Expand the **Estimated Bounded By** group to see a full picture of all time taxes paid for offloading the region to the target platform.

- In the **Estimated Data Transfer with Reuse** column, review how much data is transferred per kernel in different directions (from host to device, from device to host). Expand the column to see data per memory level.
- In the **Memory Estimations** column, see how well your application uses resources of all memory levels. Expand the group to see more detailed and accurate metrics for different memory levels.

Code Regions | **Roofline**

Loop/Function	Estimated Bounded By			Estimated Data Transfer With Reuse
	Throughput	Taxes With Reuse	Latencies	
[loop in _Z16group_by_clusterP5P]	L3 BW 26.1ms Compute 16.7ms	DT Tax 824.2us All Taxes 1.3ms	Load 13.4ms	Read 180B Write 12MB
[loop in _Z18sum_points_clusterP5P]	L3 BW 5.7ms Compute 957.1us	DT Tax 840.7us All Taxes 1.3ms	Load 2.1ms	Read 240kB Write 12MB
[loop in func@0x46b0]	Compute 0s L3 BW 0s	Launch Tax 5.0us All Taxes 5.0us	Load < 0.1us	Read 0B Write 0B
[loop in func@0x4f00]	Compute 0s L3 BW 0s	Launch Tax 5.1us All Taxes 5.1us	Load < 0.1us	Read 0B Write 0B
[loop in main at main.cpp:106]	L3 BW 44.2us Compute 1.6us	DT Tax 824.2us All Taxes 829.3us	Load 5.2us	Read 12MB Write 0B

- Select a code region from the table and review the details about data transferred between host and device memory in the **Data Transfer Estimations** pane.
 - In the **Transferred Data & Tax** histogram, see the distribution of data transferred between the host and target devices in each direction.
 - See hints about optimizing data transfers in the selected code region.



- In the **Recommendations** tab, get guidance for offloading your code to a target device and optimizing it so that your code benefits the most. If the code region has room for optimization or underutilizes the capacity of the target device, Intel Advisor provides you with hints and code snippets that might be helpful to you for further code improvement.

NOTE For details about metrics reported, see [Accelerator Metrics](#).

Next Steps

To learn more about data transfers estimated between host and target device for your application, run Offload Modeling with one of the following properties:

- Set the data transfer simulation under the characterization analysis to **Medium** and run the perspective. The result should have the **Data Transfer Estimations** pane extended with new data reporting information about memory objects in each code region.

The **Offloaded Objects** pane shows a list of memory objects with data about each object aggregated between different instances of one region.

OFFLOADED OBJECTS

Source ... ?	Type ?	Direction ?	Size ?
main.cpp:90	heap	↻	12.0 MB
main.cpp:93	heap	↻	160 B

Analytics histogram shows the number of memory objects that the selected region accessed distributed by their size.

ANALYTICS



- Set the data transfer simulation under the characterization analysis to **High** and enable the **Data Reuse Analysis** checkbox under the Performance Modeling analysis. With data reuse analysis, Intel Advisor detects groups of parallel code regions that can reuse memory objects transferred to a target GPU device. Such memory objects can be transferred to GPU only once and reused, which can improve data transfer efficiency.

The result should have data transfer metrics in the **Code Regions** pane estimated with and without data reuse for each code region. Examine the metrics in the **Estimated Bounded By** and **Estimated Data Transfer with Reuse** columns to check if a code region can benefit from applying data reuse.

For code regions that can benefit from data reuse, you should see *Apply Data Reuse* guidance in the **Recommendations** tab. The guidance shows the data transfer estimated with and without data reuse and the performance gain from applying the data reuse. It also explains how you can apply the data reuse technique to your code.

Inefficient data transfer present

Confidence level: high

The kernel is mostly bounded by **Data Transfer, L3 BW**.

When this kernel is offloaded, the data transfer will not be efficient and will decrease performance. For computations that run on a GPU, you should minimize data transfers between host and target devices.



Apply data reuse

Confidence level: high

Apply the **data reuse** technique.

With this technique, two sequential kernels can share data without additional data transfer, which can improve data transfer efficiency. For variables used by multiple target constructs, use the `target_enter_data` and `target_exit_data` directives to minimize data transfers between host and target devices.

The estimated default data transfer without data reuse for this kernel is **0.02 GB** (the data transfer read from the target device is **0.01 GB**, the data transfer written to the target device is **0.01 GB**).

The estimated data transfer with data reuse applied is **0.01 GB** (the data transfer read from the target device is **0.01 GB**, the data transfer written to the target device is **0 GB**).

With the data reuse used, you may get the data transfer gain (speed-up): **1670.28%**.

- If you think that the estimated speedup is enough and the application is ready to be offloaded, rewrite your code to offload profitable code regions to a target platform and measure performance of GPU kernels with [GPU Roofline Insights](#) perspective.

See Also

- [Optimize Your GPU Application with the Intel® oneAPI Base Toolkit](#)
- [Intel® oneAPI Programming Guide](#)

Check for Dependency Issues

Accuracy Level

High

Enabled Analyses

Survey + Characterization (Trip Counts and FLOP with cache simulation and medium data transfer simulation) + Dependencies + Performance Modeling

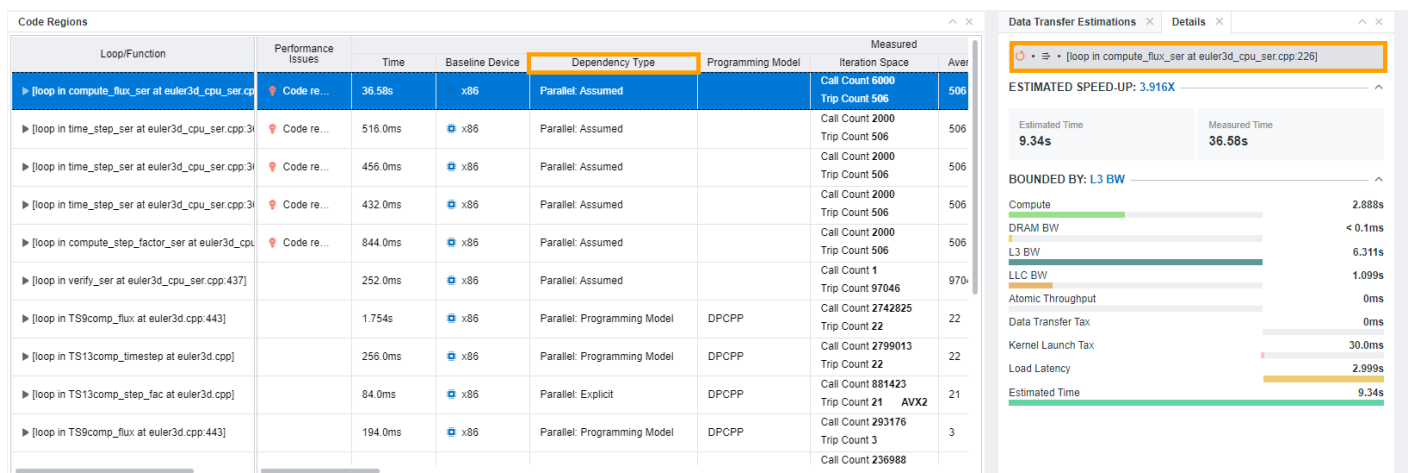
Result Interpretation

Without the Dependencies analysis, if a loop is not explicitly marked as parallel with pragmas or if a compiler assumes dependencies present, Intel® Advisor assumes the loop is not recommended for offloading because they have high compute time. In this case, you can see high percentage of dependency-bound code regions. To get accurate information about dependencies, run the Dependencies analysis.

After running the Offload Modeling perspective with **High** accuracy, you will get a complete Offload Modeling report extended with detailed information about loops that have and do not have dependencies and a full data transfer report.

If you had a report generated for a lower accuracy, all offload recommendations, metrics, and speedup will be updated to be more precise taking into account new data.



NOTE This topic describes data as it is shown in the Offload Modeling report in the Intel Advisor GUI and an interactive HTML report.



In the metrics table of the **Accelerated Regions** tab:

- Expand the **Measured** column group and see the **Dependency Type** column. It indicates if the loop has dependencies and if yes, reports dependency types.

In the **Details** tab, see an icon indicating loop dependency type:

-  - code region is parallel or can be parallelized.
-  - code region has dependencies.
- In the **Throughput** column of the **Estimated Bound-by** group, review time spent for dependencies-bound parts of your code. If the value is high, fix the dependencies.
- Intel Advisor might detect that some of the loops do not have dependencies and can be offload candidates, even though they were previously assumed as having dependencies. Review the list of loops/functions considered profitable for offloading for new candidates.

Review the **Data Transfer Estimations** pane with detailed information about data transferred between host and device and memory objects. In addition to [basic data transfer report](#), it includes:

- Offloaded memory objects with size and transfer direction.
- The histogram distribution of objects that the selected region accessed by size.

Get guidance for offloading your code to a target device and optimizing it so that your code benefits the most in the **Recommendations** tab. If the code region has room for optimization or underutilizes the capacity of the target device, Intel Advisor provides you with hints and code snippets that may be helpful to you for further code improvement.

Next Steps

If you think that the estimated speedup is enough and the application is ready to be offloaded, rewrite your code to offload profitable code regions to a target platform and measure performance of GPU kernels with [GPU Roofline Insights](#) perspective.

See Also

- [Optimize Your GPU Application with the Intel® oneAPI Base Toolkit](#)
- [Intel® oneAPI Programming Guide](#)

Explore Performance Gain from GPU-to-GPU Modeling

Enabled Analyses

Performance collection for GPU kernels only (Survey, Characterization) + Performance modeling for GPU kernels only

Result Interpretation

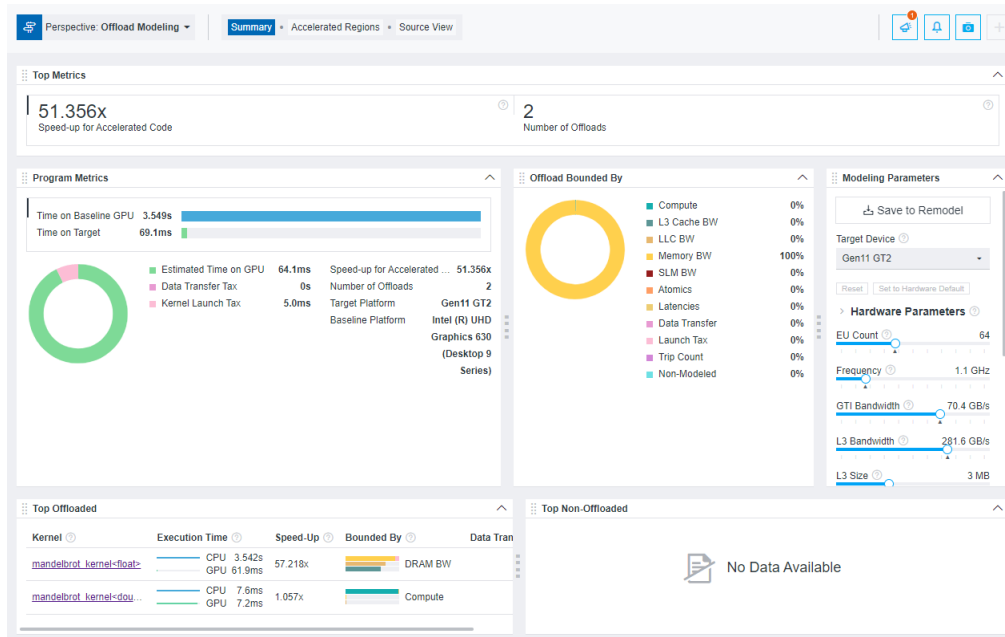
You can view the result generated in the following:

- Review the result summary and a result file location printed to a command prompt or a terminal.
- Review the project result in Intel® Advisor graphical user interface (GUI) generated to the project directory.
- Review HTML reports generated to the `<project-dir>/e<NNN>/report` directory.
- Review a set of CSV reports with detailed metric tables generated to the `<project-dir>/e<NNN>/pp<NNN>/data.0` directory.

This topic describes data as it is shown in the Offload Modeling report in the Intel Advisor GUI. You can also view the results using an HTML report, but data arrangement and some metric names may vary.

The structure and controls of the GPU-to-GPU performance modeling report generated for the are similar to the CPU-to-GPU offload modeling report, but the content is different because for the GPU-to-GPU modeling, Intel Advisor models performance *only* for GPU-enabled parts of your application.

When you open the report, it first shows the **Summary** tab. In this tab, you can review the summary of the modeling results and estimated performance metrics for some GPU kernels in your application.



- In the **Program Metrics** pane, compare the **Time on Baseline GPU** and **Time on Target GPU** and examine the **Speedup for Accelerated Code** to understand if the GPU kernels in your application have a better performance on a target GPU. **Time on Baseline GPU** includes *only* execution time of the GPU kernels and ignores the CPU parts of your application. **Time on Target GPU** includes estimated execution time for GPU kernels on the target GPU and offload taxes.

In the pie chart, review ratio of GPU execution time and offload taxes (kernel launch tax and data transfer tax) and see where the GPU kernels spend most of the time.

- In the **Offloads Bounded by** pane, examine what the GPU kernels are potentially bounded by on the target GPU. The parameters with the highest percentage mean that this is where the GPU kernels spend the most time. Review the detailed metrics for these parameters in other tabs to understand if you need to optimize your application for this.
- In the **Top offloaded** pane, review the top five GPU kernels with the highest absolute offload gain (in seconds) estimated on the target GPU. The gain is calculated as (*Time measured on the baseline GPU - Time estimated on the target GPU*).

For each kernel in the pane, you can review the speedup, time on the baseline and the target GPUs, main bounded-by parameters, and estimated amount of data transferred. Intel Advisor models kernels one-to-one and does not filter out kernels with estimated speedup less than 1.

NOTE The **Top non offloaded** pane shows only GPU kernels that cannot be modeled. If all kernels are modeled, the pane is empty. For the GPU-to-GPU modeling, estimated speedup lower than 1 is not a reason for not offloading a kernel.

To see the details about GPU kernel performance, go to the **Accelerated Regions** tab.

Kernel	Measured			Basic Estimated Metrics				
	Time	Instances	Iteration Space	Speed-Up	Time	Offload Summary	Bounded By	Throu
mandelbrot_kernel<double>	7.6ms	2	Global 16 x 50 x 600 Local 16 x 2 x 8 SIMD 16	1.057x	7.2ms	Offloaded	Compute	Compute DRAM BW
mandelbrot_kernel<float>	3.542s	1000	Global 16 x 50 x 600 Local 16 x 2 x 8 SIMD 16	57.218x	61.9ms	Offloaded	DRAM BW	DRAM BW LLC BW

- In the **Code Regions** table, examine the detailed performance metrics for the GPU kernels. The **Measured** column group shows metrics measured on the baseline GPU. Other column groups show metrics estimated for the target GPU. You can expand column groups to see more metrics.

You can also select a kernel in the table and examine the highlight measured and estimated metrics for it in the Details tab of the right-side pane to identify what you need to focus on.

For example, to find a potential bottleneck:

1. Examine the **Estimated Bounded by** column group focusing on the **Bounded by** and **Throughput** columns. In the **Bounded by** column, you can see the main bottleneck and secondary bottlenecks. The **Throughput** expands the bottlenecks with time by compute or memory throughput, latencies, and offload taxes shown as a chart. See [Bounded By](#) for bottleneck details.
2. For details about the bounding factors, expand the column group and find the columns corresponding to these bounding factors, for example, **L3 Cache BW**, **DRAM BW**, or **LLC BW**.
3. Scroll to the right, expand the **Memory Estimations** column group, and examine the columns corresponding to the bottleneck identified. For example, the bandwidth utilization is calculated as a relation of average memory level bandwidth to its peak bandwidth. High value means that the kernel does not use well this memory level and it is the potential bottleneck.

You can also review the following data to find bottlenecks:

- If you see high cache or memory bandwidth utilization (for example, in the **L3 Cache**, **SLM**, **LLC** column groups), consider optimizing cache/memory traffic to improve performance.
- If you see high latency in the **Estimated Bounded By** column group, consider optimizing cache/memory latency by scheduling enough parallel work for this kernel to increase thread occupancy.
- If you see high data transfer tax in the **Estimated Data Transfer with Reuse**, consider optimizing data transfer taxes or using unified shared memory (USM).
- If you see a high data transfer tax for a kernel, select the kernel in the Code Regions table and examine the details about memory objects transferred between the host device and a target GPU for a kernel in the right-side **Data Transfer Estimations** pane. Review the following data:
 - The histogram for the transferred data that shows amount of data transferred in each direction and the corresponding offload taxes.
 - The memory object table that lists all memory objects accessed by the kernel with details about each object, such as size, transfer direction (only to the host, only to the target, from the host to the target and back), object type. If you see a lot of small-sized objects, this may result in high latency for the kernel. High latency might cause a high data transfer tax.
 - Hints about optimizing data transfers in the selected code region.

Intel Advisor uses this data to estimate data transfer traffic and data transfers for each kernel.

Next Steps

- Based on collected data, rewrite your code to offload it to a different target GPU to improve performance and measure its performance with the [GPU Roofline Insights](#) perspective.
- See optimization tips for oneAPI applications running on GPU in the [oneAPI GPU Optimization Guide](#).

See Also

[Run GPU-to-GPU Performance Modeling from Command Line](#) With Intel® Advisor, you can model performance of SYCL, OpenCL™, or OpenMP* target application running on a graphics processing unit (GPU) for a different GPU device without its CPU version. For this, run the GPU-to-GPU modeling workflow of the Offload Modeling perspective.

[Work with Standalone HTML Reports](#) Export the interactive Intel® Advisor HTML reports that you can share or open on a remote machine using your web browser.

Investigate Non-Offloaded Code Regions

The modeling step analyzes code regions profitability for offloading to a target device. Some regions might be not profitable for offloading or cannot be modeled.

If you view a result in the *Intel® Advisor GUI*: To see details why your code region of interest is reported as not recommended for offloading, select a loop in a **Code Regions** pane and see the **Details** tab in the right-side pane for detailed loop information, including the reason why the loop is not recommended for offloading.

By default, the report shows all code regions. You can apply filters to see only regions recommended or not



recommended for offloading: open the drop-down list and select the desired filter option.

If you view the result in an *Offload Modeling HTML report*: Go to the **Non-Offloaded Regions** tab and examine the **Why Not Offloaded** column in the **Offload Information** group to the reason why a code region is not recommended for offloading to a selected target platform.

Tip For each region not recommended for offloading, you can force offload modeling. See the [Enforce Offloading for Specific Loops](#).

Cannot Be Modeled

Message	Cause and Details	Solution
Cannot be modeled: Outside of Marked Region	Intel® Advisor cannot model performance for a code region because it is not marked up for analysis.	Make sure a code region satisfies all markup rules or use a different markup strategy: <ul style="list-style-type: none"> It is not a system module or a system function. It has instruction mixes. It is executed. Its execution time is not less than 0.02 seconds.
Cannot be modeled: Not Executed	A code region is in the call tree, but the Intel Advisor detected no calls to it for a dataset used during Survey.	This can happen if execution time of the loop is very small and close to the sampling interval of the Intel Advisor. Such loops can have significant inaccuracies in time measurement. By default, the sampling interval for Survey is 0.01 seconds. You can try to decrease the sampling interval of the Intel Advisor:

Message	Cause and Details	Solution
Cannot be modeled: Internal Error	<i>Internal Error</i> means incorrect data or lack of data because the Intel Advisor encountered issues when collecting or processing data.	<ul style="list-style-type: none"> From GUI: <ol style="list-style-type: none"> Go to Project Properties > Survey Hotspots Analysis > Advanced. Set the Sampling Interval to less than 10ms. Re-run Offload Modeling. From CLI: Use the <code>--interval</code> option when running the <code>--collect=survey</code>. <p>Try to re-run the Offload Modeling perspective to fix the metrics attribution problem. If this does not help, use the Analyzers Community forum for technical support.</p>
Cannot be modeled: System Module	This code region is a system function/loop.	This is not an issue. If this code region is inside an offload region, or a runtime call, its execution time is added to execution time of offloaded regions.
Cannot be modeled: No Execution Count	The Intel Advisor detected no calls to a loop during Trip Count step of the Characterization analysis and no information about execution counts is available for this loop.	<p>Check what loop is executed at this branch.</p> <p>If you see a wrong loop, try re-running the Offload Modeling to fix the metrics attribution problem.</p>

Less or Equally Profitable Than Children/Parent Offload

This message is not an issue. It means that Intel Advisor has found a more profitable code region to offload. If you still want to see offload estimations for the original code region, use the solutions described in the table below.

Message	Cause and Details	Solution
Less or equally profitable than children offloads	Offloading child loops/functions of this code region is more profitable than offloading the whole region with all its children. This means that the execution Time estimated on a target platform for the region of interest is greater than or equal to the sum of Time values estimated on a target platform for <i>its child regions</i> profitable for offloading.	Model offloading for specific code regions even if they are not profitable. See Enforce Offloading for Specific Loops for details.

Message	Cause and Details	Solution
Less or equally profitable than parent offload	<p>The following reasons might prevent offloading: total execution time, taxes, trip counts, dependencies.</p> <p>Offloading a whole parent code region of the region of interest is more profitable than offloading any of its child regions separately. This means that the Time estimated on a target platform for the region of interest is greater than or equal to the Time estimated on the target platform for <i>its parent region</i>.</p> <p>Offloading a child code region might be limited by high offload taxes.</p>	<p>Solution 1</p> <p>If you assume the kernel execution should overlap offload taxes, use the <code>--assume-hide-taxes</code> option with <code>--collect=projection</code> action option or the <code>analyze.py</code> script. See Manage Invocation Taxes for details.</p> <p>Solution 2</p> <p>Model offloading for only specific code regions even if they are not profitable. See Enforce Offloading for Specific Loops for details.</p>

Not Profitable

Message	Cause and Details	Solution
Not profitable: Parallel execution efficiency is limited due to Dependencies	Dependencies limit parallel execution and the code region cannot benefit from offloading to a target device. The estimated execution time after acceleration is greater than or equal to the original execution time.	<p>Solution 1</p> <p>Ignore assumed dependencies and model offloading for all or selected code regions:</p> <ul style="list-style-type: none"> From GUI: <ul style="list-style-type: none"> 1Go to Project Properties > Performance Modeling. 2Enter one of the options in the Other Parameters field: <ul style="list-style-type: none"> <code>--no-assume-dependencies</code> to assume <i>all</i> code regions that do not have information about their dependency are parallel <code>--set-parallel=[<loop-IDs/source</code> to ignore dependencies for specified code regions 3Re-run Performance Modeling. From CLI: When running <code>--collect=projection</code> or <code>analyze.py</code>, use one of the following:

Message	Cause and Details	Solution
Not profitable: The Number of Loop Iterations is not enough to fully utilize Target Platform capabilities	The loop cannot benefit from offloading to a target platform as it has a low number of iterations.	<ul style="list-style-type: none"> • <code>--no-assume-dependencies</code> to ignore dependencies for <i>all</i> code regions • <code>--set-parallel=[<loop-IDs/source-</code> to ignore dependencies for specified code regions <p>For details, see Check How Dependencies Affect Modeling.</p> <p>Solution 2</p> <p>If you did not enable the Dependencies analysis when collecting data, run the analysis as follows to get detailed information about real dependencies in your code:</p> <ul style="list-style-type: none"> • From GUI: Enable the Dependencies and Performance Modeling analyses from the Analysis Workflow pane and re-run the perspective. • From CLI: Run the Dependencies analysis with <code>--collect=dependencies</code> and re-run the Performance Modeling with <code>--collect=projection</code> or <code>analyze.py</code>. <p>See Dependency Type metric description for details.</p> <p>In most cases, such code regions cannot benefit from offloading. If you assume that during code migration, the amount of parallel work grows and a loop is broken down into several chunks by a compiler or a program model, use the following workaround:</p> <ul style="list-style-type: none"> • From GUI: <ol style="list-style-type: none"> 1Go to Project Properties > Performance Modeling. 2Enter <code>--batching</code> or <code>--threads=<target-threads></code> in the Other Parameters field. <code><target-threads></code> is the

Message	Cause and Details	Solution
Not profitable: Data Transfer Tax is greater than Computation Time and Memory Bandwidth Time	Time spent on transferring data to a target device is greater than <i>compute time</i> and <i>memory bandwidth time</i> . The resulting time estimated on a target platform with data transfer tax is greater than or equal to the time measured on a host platform.	<p>number of parallel threads equal to the target device capacity.</p> <p>Re-run Performance Modeling.</p> <ul style="list-style-type: none"> From CLI: When running <code>--collect=projection</code> or <code>analyze.py</code>, use one of the following: <ul style="list-style-type: none"> <code>--batching</code> to model batching-like techniques <code>--threads=<target-threads></code>, where <i><target-threads></i> is the number of parallel threads equal to the target device capacity <p>If you enable batching, the kernel invocation tax might grow. You can use the <code>--assume-hide-taxes</code> option to reduce the task. See Manage Invocation Taxes for details.</p> <p>Check the Bounded By and Data Transfer Tax columns in the Estimated Bounded By column group and the Estimated Data Transfer with Reuse column group. Large value means that this code region cannot benefit from offloading.</p> <p>See Bounded By for details about metric interpretation.</p> <p>If you still want to offload such regions, disable data transfer analysis with the <code>--data-transfer=off</code> to use only estimated execution time for speedup and profitability calculation.</p> <hr/> <p>NOTE This option disables data transfer analysis for all loops. You might get different performance modeling results for all loops.</p>

Message	Cause and Details	Solution
Not profitable: Computation Time is high despite the full use of Target Platform capabilities	The code region uses full target platform capabilities, but time spent for compute operations is still high. As a result, the execution time estimated on a target platform is greater than or equal to the time measured on a host platform.	<p>If you already collected data transfer metrics, you can turn off modeling data transfer tax with the command line option <code>--hide-data-transfer-tax</code>.</p> <p>Check the value in the <i>Compute</i> column in the <i>Estimated Bound-by</i> column group. Unexpectedly high value means one of the following:</p> <ul style="list-style-type: none"> • There is a problem with a programming model used. • Target GPU compute capabilities are lower than baseline CPU compute capabilities. • Internal Intel Advisor error happened caused by incorrect compute time estimation.
Not profitable: <i>Cache/Memory Bandwidth Time</i> is greater than other execution time components on Target Device	<p>The time spent in <i>cache or memory bandwidth</i> takes a big part of the time estimated on a target platform. As a result, it is greater than or equal to the time measured on a host platform.</p> <p>In the report, the <i>Cache/Memory</i> is replaced with a specific cache or memory level that prevents offloading, for example, L3 or LLC. See the Throughput column for details about the highest bandwidth time.</p>	<ol style="list-style-type: none"> 1. Examine code region children to identify which part takes most of the time and prevents offloading. 2. Optimize the part of your code that takes most of the time measured on a baseline platform and rerun the perspective.
Not profitable because of offload overhead (taxes)	Total time of offload taxes, which includes Kernel Launch Tax , Data Transfer Tax , takes a big part of the time estimated on a target platform. As a result, it is greater than or equal to the time measured on a host platform.	Examine the Taxes with Reuse column in the Estimated Bounded by group for the biggest and total time taxes paid for offloading the code region to a target platform. Expand the Estimated Bounded by group to see a full picture of time taxes paid for offloading the region to the target platform. Big value in any of the columns means that this code region cannot benefit from offloading because the cost of offloading is high.

Message	Cause and Details	Solution
Not profitable: Kernel Launch Tax is greater than Kernel Execution Time and Data Transfer Time	Time spent on launching a kernel is greater than execution time estimated on a target platform and estimated data transfer time. The resulting time estimated on the target platform with data transfer tax is greater than or equal to the time measured on a host platform.	<p>If kernel launch tax is large and you assume the kernel execution should overlap the launch tax, model hiding the launch taxes as follows:</p> <ul style="list-style-type: none"> From GUI: Enabled the Single Kernel Launch Tax option from the Analysis Workflow pane and rerun the Performance Modeling analysis. From CLI: Use the <code>--assume-hide-taxes</code> option with the <code>--collect=projection</code> or <code>analyze.py</code> <p>See Manage Invocation Taxes for details.</p> <p>Examine the Bounded By and Kernel Launch Tax columns in the Estimated Bounded By column group.</p> <p>See Bounded By for details about metric interpretation.</p> <p>High value in Kernel Launch Tax means that the Intel Advisor detects high call count for a potentially profitable code region and assumes that the kernel invocation tax is paid as many times as the kernel is launched. For this reason, it assumes that the code region cannot benefit from offloading.</p> <p>If you assume the kernel execution should overlap the launch tax, model hiding the launch taxes as follows:</p> <ul style="list-style-type: none"> From GUI: Select the Single Kernel Launch Tax checkbox for the Performance Modeling analysis. From CLI: Use the <code>--assume-hide-taxes</code> option with the <code>--collect=projection</code> action option or <code>analyze.py</code>. <p>For details, see Manage Invocation Taxes.</p>

Message	Cause and Details	Solution
Not profitable: Atomic Throughput Time is greater than other execution time components on a Target Device	Atomic operations include loading, changing, and storing data to make sure it is not affected by other threads between the calls. When modeling atomic operations, Intel Advisor assumes that <i>all</i> threads wait for each other, so Atomic Throughput time might be high and can be one of the main hotspots.	Go to the Analyzers Community forum for technical support and advice.
Not profitable: Instruction Latency is greater than Compute Time and Memory Bandwidth Time	Each memory read instruction produces a GPU thread stall. The stall is called a <i>memory latency</i> . Usually, execution of other threads can overlap it. However, sometimes the amount of non-overlapped latency has a big impact on performance. Intel Advisor can estimate the non-overlapped memory latency and add it to the kernel estimated execution time. If you reduce thread occupancy, it can increase the amount of non-overlapped memory latency	Examine the Latency column to see how much time spent for load latency and the Thread Occupancy column to understand the reason for this. Low occupancy means that this is the reason for a high load latency. In this case, when offloading the code, increase the kernel parallelism or cover latency with other instructions. If you are sure that the load latency is overlapped with compute instructions in your code, you can enable latency hiding mode with the following: <ul style="list-style-type: none"> From GUI: <ol style="list-style-type: none"> Go to Project Properties > Performance Modeling. Enter <code>--count-send-latency=first</code> in the Other Parameters field. Re-run Performance Modeling. From CLI: Use the <code>--count-send-latency=first</code> option with the <code>--collect=projection</code> action option or <code>analyze.py</code>.

N/A - Part of Offload

This means that offloading a code region is less profitable than offloading its outer loop.

This is not an issue. The code region of interest is located inside of an offloaded loop.

Total Time Is Too Small for Reliable Modeling

This means the execution time of a code region or a whole loop nest is less than 0.02 seconds. In this case, Intel Advisor cannot estimate the speedup correctly and say if it is worth to offload the code regions because its execution time is close to the sampling interval of the Intel Advisor.

Possible Solution

If you want to check the profitability of offloading code regions with total time less than 0.02 seconds:

- From GUI:
 1. Go to **Project Properties > Performance Modeling**.
 2. Enter the `--loop-filter-threshold=0` option to the **Other parameters** field to model such small offloads.
 3. Re-run Performance Modeling.
- From CLI: Use the `--loop-filter-threshold=0` option with the `--collect=projection` or `analyze.py`.

Advanced Modeling Configuration

In some cases, pre-configured accuracy levels of the Intel Advisor are not enough to accurately model performance of your application. You might want to use some advanced custom modeling strategies or fine-tune offload modeling parameters to better adjust the model to your application and for your goal.

- [Model performance for a custom target GPU](#) with configuration parameters adjusted using the remodeling pane.
- [Check how assumed dependencies affect modeling](#) to decide if you need to run the Dependencies analysis to get more accurate modeling results.
- [Manage invocation taxes](#) to control how to model kernel launch for your application,
- [Enforce offloading for specific loops](#) to get modeled performance results on a target for specific loops even if the estimated speedup for them is low.

Model Application Performance on a Custom Target GPU Device

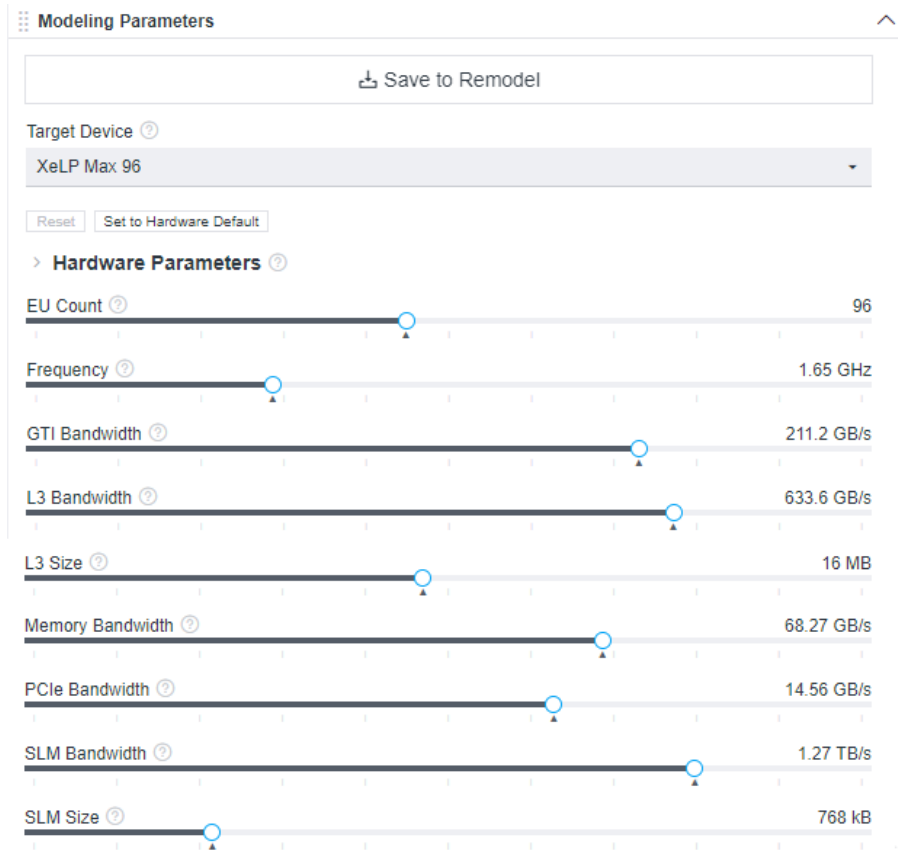
You can change GPU parameters to model performance of future or custom graphics processing units (GPU) and see how your application performance changes.

Intel® Advisor has several predefined GPU device configurations that you can use to model application performance. If you want to estimate performance on a future GPU device or experiment with hardware parameters to see how they can change application performance, you can modify target hardware parameters for the Offload Modeling perspective in one of the following ways:

- Customize hardware parameters using sliders in an *interactive remodeling pane* in the Intel Advisor graphical user interface (GUI) or an interactive HTML report and remodel performance. This is currently available for GPU-to-GPU modeling only.
- Generate a TOML configuration file that defines customized hardware parameters and use the file to remodel performance with Intel Advisor command line interface (CLI). You can generate the file using the interactive modeling pane in the Intel Advisor GUI or the interactive HTML report. You can reuse the file for multiple analysis executions.
- Provide a command-line option with one or more modified target hardware parameters when running the analysis with Intel Advisor CLI. This is a one-time change.

Use the Modeling Parameters Pane

When you open the **Summary** tab of the Offload Modeling report in the Intel Advisor GUI or the [interactive HTML report](#), you should see the **Modeling Parameters** pane, which shows the current modeled device and its parameters. Each parameter is a slider you can move to change its value.



You can use this pane to:

- Examine device parameters that the application performance was modeled on to understand how they affect the estimated performance.
- Change the target device to compare the new configuration with the current modeled device.
- Adjust the parameters and remodel performance for a custom device. You can experiment with parameters to see how they affect the application performance or adjust the configuration to model performance for a future or a specific device not listed in the target devices. See the sections below for a full workflow.

For CPU-to-GPU modeling, you can remodel performance using Intel Advisor CLI only.

For details about pane controls, see [Window: Offload Modeling Summary](#).

NOTE The parameter list might change depending on the target device selected. This might be due to differences between GPU architecture or terminology specifics. For example, the **Gen11 GT2** configuration has the LLC bandwidth and LLC size parameters, while the **XeLP Max 96** does not because of architecture differences.

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

Remodel Performance from GUI

This workflow is currently available for remodeling from a baseline GPU to a different target GPU device using Intel Advisor GUI. You can remodel application performance for the custom device from the Offload Modeling report.

Prerequisites:

1. [Set up system](#) to analyze GPU kernels.
2. Run Offload Modeling with your preferred method: [from graphical user interface](#) or [from command line interface](#).
3. [Open the result](#) in the Intel Advisor GUI.

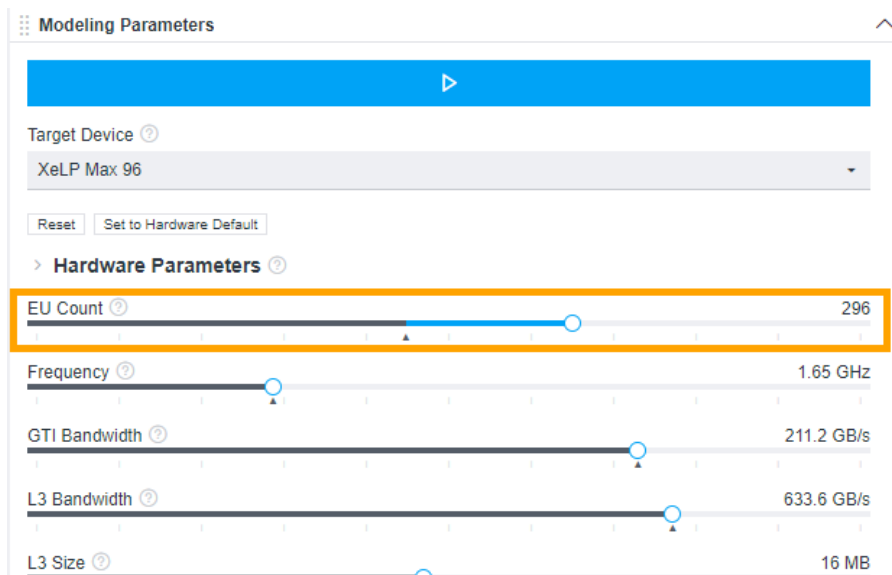
To customize the hardware parameters and remodel application performance:


1. In the **Analysis Workflow** pane, make sure **GPU** device is selected in the **Baseline Device** drop-down.
2. *Optional:* In the **Modeling Parameters** pane of the Summary report, select a device from the **Target Device** drop-down to use as a baseline for further changes.

If you do not change the device, the current modeled target device will be used as a baseline.

3. Move the sliders under **Hardware Parameters** to the desired values. The black line indicates the baseline parameter value, and the blue line indicates the difference between the new value and the baseline value.

For example, you can increase the number of execution units **EU Count** to enable more compute operations to be executed at once. This can be useful for compute-bound applications, which is indicated in the **Offload Bounded By** pane.



4. Click  button at the top of the pane to run the Performance Modeling analysis for your target device configuration.
 5. Examine the performance changes for the new target GPU.
- For example, if you increased the EU count value, you may see the compute time and compute bound percentage decreased and compute estimate metrics changed.

Remodel Performance Using a Configuration File

This workflow is currently available for remodeling performance:

- From a baseline CPU to a custom GPU using Intel Advisor GUI or an interactive HTML report
- From a baseline GPU to a custom GPU using an interactive HTML report

For these cases, you can modify the parameters using Offload Modeling report and remodel performance using Intel Advisor CLI only.

Prerequisites:

1. [Set up environment variables.](#)
2. Run Offload Modeling with your preferred method: [from graphical user interface](#) or [from command line interface](#).
3. [Open the result](#) in the Intel Advisor GUI on an interactive HTML report.

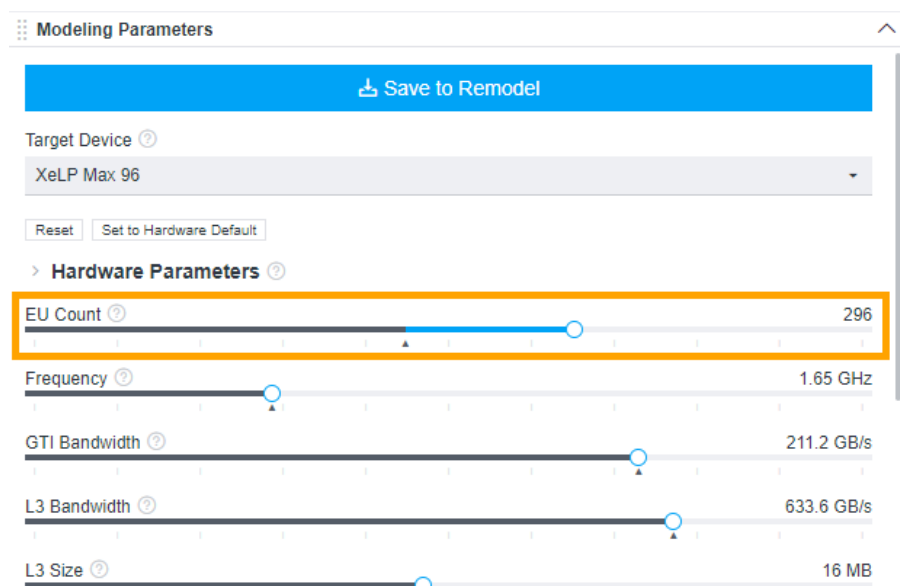
To customize the hardware parameters and remodel application performance:

1. *Optional:* In the **Modeling Parameters** pane of the Summary report, select a device from the **Target Device** drop-down to use as a baseline for further changes.

If you do not change the device, the current modeled target device will be used as a baseline.

2. Move the sliders under **Hardware Parameters** to the desired values. The black line indicates the baseline parameter value, and the blue line indicates the difference between the new value and the baseline value.

For example, you can increase the number of execution units **EU Count** to enable more compute operations to be executed at once. This can be useful for compute-bound applications, which is indicated in the **Offload Bounded By** pane.



After you move a slider, the **Save to Remodel** button activates, enabling you to save your custom configuration.

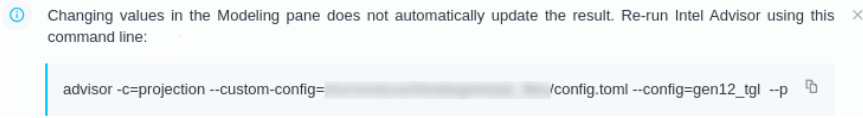
NOTE Currently, for the CPU-to-GPU modeling, if you change the memory-related parameters, such as bandwidth or size, run the Characterization analysis *first* with Trip Counts and multi-device cache simulation (for CLI, run Trip Counts collection with `--cache-simulation=multi` option) *before* running the Performance Modeling with the custom configuration. Otherwise, the results may not be accurate because they require updating cache simulation for the new device.


3. Click **Save to Remodel** to save the generated configuration file.

The **Save Configuration** dialog box opens.

4. In the opened dialog box, navigate to a location to save the TOML file, change the file name if needed, and click **Save**. By default, the file is saved as `config.toml`.

After you save the custom configuration file, a command line for the Performance Modeling analysis appears under the hardware parameter sliders in the **Modeling Parameters** pane.



5. Click  to copy the command line generated under the hardware parameters to a clipboard.
Notice that the command line has a `--custom-config` option with a full path to the custom configuration file you saved. The command line has all required options, and you can copy and paste it without modifications.
6. Paste the copied command to a terminal or a command prompt and run it.
After the analysis execution completes, the result in your project directory will be updated for the new target device configuration.
7. Open the updated results with your preferred method and examine the performance changes for the new target GPU.
For example, if you increased the EU count value, it you may see the compute time and compute bound percentage decreased and compute estimate metrics changed.

Remodel Performance Using a Command Line Option

When you run the Offload Modeling perspective from the command line, you can use the `--set-parameter=<parameter-to-change>` option to change target parameters. You can use this option with the Offload Modeling collection preset or the Performance Modeling analysis. This is a one-time change applied only for the current execution. You can specify more than one parameter as a comma-separated list.

For example, you can model performance for a target device with 1.4 GHz frequency, 224 execution units, and other parameters corresponding to the `gen12_tgl` device configuration as following:

```
advisor --collect=offload --config=gen12_tgl --set-parameter="EU_count=224,Frequency=1.4e+9" --project-dir=./advi_results -- ./myApplication
```

You can open the generated results with your preferred method and examine the performance changes for the new target GPU.

To see what parameters you can change, you can save a configuration file for a selected device from Intel Advisor GUI or HTML report and examine the parameters listed.

[Run Offload Modeling Perspective from GUI](#)

[Run Offload Modeling Perspective from Command Line](#)

[Explore Offload Modeling Results](#)

[advisor Command Option Reference](#)

Check How Assumed Dependencies Affect Modeling

If a loop has dependencies, it cannot be run in parallel and in most cases cannot be offloaded to the GPU. Intel Advisor can get the information about loop-carried dependencies from the following resources:

- Using Intel® Compiler diagnostics. The dependencies are found at the compile time for some loops and the diagnostics are passed to the Intel Advisor using the integration with Intel Compilers.
- Parsing the application call stack tree. If a loop is parallelized or vectorized on a CPU or is already offloaded to a GPU but executed on a CPU, Intel Advisor assumes that you resolved the loop-carried dependencies before parallelizing or offloading the loop.

- Using the Dependencies analysis results. This analysis detects dependencies for most loops at run time, but a result might depend on an application workload. It also adds a high overhead making the application execute 5 - 100 times slower during the analysis. To [reduce overhead](#), you can use various techniques, for example, mark up loops of interest.

For the Offload Modeling perspective, the Dependencies analysis is optional, but it might add important information about loop-carried dependencies Intel® Advisor to decide if a loop can be profitable to run on a graphics processing unit (GPU).

This topic describes a workflow that you can follow to understand if there are potential loop-carried dependencies in your code that might affect its performance on a target GPU.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Verify Assumed Dependencies

If you do not know what dependency types there are present in your application, run the Offload Modeling without the Dependencies analysis first to check if potential dependencies affect modeling results and to decide if you need to run the Dependencies analysis:

- Run the Offload Modeling without the Dependencies analysis.

- From GUI: Select Medium accuracy level and enable the **Assume Dependencies** option for the Performance Modeling in the **Analysis Workflow** tab. Run the perspective.
- From CLI: Run the following analyses, for example, using the `advisor` command line interface:

```
advisor --collect=survey --project-dir=./advi_results --static-instruction-mix -- ./myApplication
```

```
advisor --collect=tripcounts --project-dir=./advi_results --flop --stacks --enable-cache-simulation --target-device=xehpg_512xve --data-transfer=light -- ./myApplication
```

```
advisor --collect=projection --project-dir=./advi_results
```

- Open the generated report and go to the **Accelerated Regions** tab.
- In the **Code Regions** pane, expand the **Measured** column group and examine the **Dependency Type** column.
 - You *do not* need to run the Dependencies analysis for loops with the following dependency types:
 - Parallel: Programming Model* dependency type means that the loop is uses SYCL, OpenCL™ or OpenMP* target programming model.
 - Parallel: Explicit* dependency type means that the loop is threaded and vectorized on CPU (for example, with OpenMP `parallel for` or Intel® oneAPI Threading Building Blocks `parallel for`).
 - Parallel: Proven* dependency type means that an Intel Compiler found no dependencies at the compile time.
 - You *might* need to run the Dependencies analysis for loops that have the *Dependency: Assumed* dependency type. It means that the Intel Advisor does not have information about loop-carried dependencies for these loops and do not consider them as offload candidates.
- If you see many *Dependency: Assumed* types, rerun the performance modeling with assumed dependencies ignored, as follows:
 - From GUI: Select *only* the Performance Modeling step in the **Analysis Workflow** tab and disable the **Assume Dependencies** option. Run the perspective.
 - From CLI: Run the Performance Modeling with *one* of the following options
 - Use `--no-assume-dependencies` to ignore assumed dependencies for *all* loops/functions. For example:

```
advisor --collect=projection --project-dir=./advi_results --no-assume-dependencies
```

- Use `--set-parallel=[<loop-ID1>|<file-name1>:<line1>, <loop-ID2>|<file-name2>:<line2>, ...]` to ignore assumed dependencies for *specific* loops/functions only. Use this option if you know that some loops/functions have dependencies and you do not want to model them as parallel. For example:

```
adviser --collect=projection --project-dir=./advi_results --set-parallel=foo.cpp:34,bar.cpp:192
```

5. Review the results generated to check if the potential dependencies might block offloading to GPU.

Loops that previously had *Dependency: Assumed* dependency type are now marked as *Parallel: Assumed*. Intel Advisor models their performance on the target GPU and checks potential offload profitability and speedup.

6. Compare the program metrics calculated with and without assumed dependencies, such as speedup, number of offloads, and estimated accelerated time.
 - If the difference is small, for example, 1.5x speedup with assumed dependencies and 1.6x speedup without assumed dependencies, you can *skip the Dependencies analysis* and rely on the current estimations. In this case, most loops with potential dependencies are not profitable to be offloaded and do not add much speedup to the application on the target GPU.
 - If the difference is big, for example, 2x speedup with assumed dependencies and 40x speedup without assumed dependencies, you should *run the Dependencies analysis*. In this case, the information about loop-carried dependencies is critical for correct performance estimation.

Run the Dependencies Analysis

To check for real dependencies in your code, run the Dependencies analysis and rerun the Performance Modeling to get more accurate estimations of your application performance on GPU:

- From GUI:
 1. Enable only the Dependencies and Performance Modeling analyses in the **Analysis Workflow** tab.
By default, the generic markup strategy is applied to select only potentially profitable loops to run the Dependencies analysis.
 2. Rerun the perspective with only these two analyses enabled.
- From CLI:

1. Run the Dependencies analysis for potentially profitable loops only:

```
adviser --collect=dependencies --select markup=gpu_generic --loop-call-count-limit=16 --filter-reductions --project-dir=./advi_results -- ./myApplication
```

2. Run the Performance Modeling analysis:

```
adviser --collect=projection --project-dir=./advi_results
```

Open the result in the Intel Advisor, view the interactive HTML report, or print it to the command line. Continue to investigate the results and [identify code regions to offload](#).

See Also

[Run Offload Modeling Perspective from GUI](#)

[Run Offload Modeling from Command Line](#)

[Loop Markup to Minimize Analysis Overhead](#)

Manage Invocation Taxes

You can control how to model invocation taxes for your application.

When Intel® Advisor detects high call count value for a potentially profitable code region, it assumes that the kernel invocation tax is paid as many times as the kernel is launched. The result is high invocation tax and cost of offloading, which means that this code region cannot benefit from offloading. This is a *pessimistic* assumption.

However, for simple applications where there is no need to wait for a kernel instance to finish, this cost can be hidden every time except the very first one.

In the Offload Modeling report, the kernel invocation tax is reported in the **Estimated Bounded By > Kernel Launch Tax** column. This metric includes time spent for kernel configuration (first launch), each kernel launch, and kernel code transfers.

You can tell Intel Advisor how to handle invocation taxes for your application when modeling its performance on a target device.

NOTE In the commands below, `<APM>` is the environment variable that points to script directory. Replace it with `$APM` on Linux* OS or `%APM%` on Windows* OS.

Hide All Taxes

For simple applications, you are recommended to enable the *optimistic* approach for estimating invocation taxes. In this approach, Offload Modeling assumes the invocation tax is paid only for the first time the kernel executes.

To enable this approach:

- From GUI: Enable **Single Kernel Launch Tax** checkbox from the **Analysis Workflow** and re-run the **Performance Modeling** analysis.
- From command line: Run Performance Modeling with the `--assume-hide-taxes` option:

```
advisor --collect=projection --assume-hide-taxes --project-dir=./advi_results
```

where `./advi_results` is a path to your project directory. Make sure to replace it with your actual project directory where you collected results to before running the command.

NOTE You can specify a comma-separated list of loop IDs and source locations to the `--assume-hide-taxes` option to hide taxes only for those loops/functions. If you do not provide a list, taxes are hidden for all loops.

Check how the **Kernel Launch Tax** column value changed with the option used.

Do Not Hide Taxes

By default, Offload Modeling estimates invocation taxes using the *pessimistic* approach and assumes the invocation tax is paid each time the kernel is launched.

- From GUI: Disable **Single Kernel Launch Tax** checkbox from the **Analysis Workflow** and re-run the **Performance Modeling**.
- From command line: Run Performance Modeling with the `--no-assume-hide-taxes` option (default):

```
advisor --collect=projection --no-assume-hide-taxes --project-dir=./advi_results
```

where `./advi_results` is a path to your project directory. Make sure to replace it with your actual project directory where you collected results to before running the command.

Check how the **Kernel Launch Tax** column value changed with the option used.

Fine-Tune the Number of Hidden Taxes

You can fine-tune the number of invocation taxes to hide by specifying the `Invoke_tax_ratio` parameter and a fraction of invocation taxes to hide in a TOML configuration file.

1. Create a TOML file, for example, `my_config.toml`. Copy and paste the following text there:

```
[scale]
# Fraction of invocation taxes to hide.
# Note: The invocation tax of the first kernel instance is not scaled.
# Possible values: 0.0--1.0
# Default value: 0.0
Invoke_tax_hidden_ratio = <float>
```

where `<float>` is a fraction of invocation taxes to hide, for example, `Invoke_tax_hidden_ratio = 0.95`, which means that 95% of invocation taxes will be hidden and only 5% of the taxes will be estimated.

2. Save and close the file.
3. Run the performance projection with the new TOML file as additional custom configuration:

```
advisor --collect=projection --custom-config=my_config.toml --project-dir=./advi_results
```

where `./advi_results` is a path to your project directory. Make sure to replace it with your actual project directory where you collected results to before running the command.

Important If you use the configuration parameter to control the number of taxes to hide, *do not* use the `--assume-hide-taxes` or `--assume-never-hide-taxes` option. These options overwrite the value of the configuration parameter.

You can also use the `--set-parameter="scale.Invoke_tax_hidden_ratio = <float>"` for the Performance Modeling to set the number of invocation taxes to hide only for the current execution. In this case, you do not need to create a custom configuration file.

Check how the **Kernel Launch Tax** column value changed with the parameter modified.

Run Offload Modeling Perspective from Command Line

Enforce Offloading for Specific Loops

Model performance on a target device for specific loops only even if they are not profitable.

If you want to check offload profitability only for specific loops or if your loop of interest is reported as not recommended for offloading to an accelerator, you can model performance only for these loops and ignore all other loops.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To do this:

1. Collect baseline performance data. For example, run the Survey and Characterization analyses:

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --stacks --enable-cache-simulation --target-device=xehpg_512xe --project-dir=./advi_results -- ./myApplication
```

2. Rerun performance modeling for selected loops/functions only. For example:

```
advisor --collect=projection --select=foo.cpp:34,bar.cpp:192 --enforce-offload --project-dir=./advi_results
```

where:

- `--select` lists loops/functions to analyze by file and line number, loop/function ID, or criteria.
- `--enforce-offloadsmodels` performance for all selected loops/functions even if offloading their child loops/functions is more profitable.

View the [Offload Modeling results](#) in the Intel® Advisor GUI or view an HTML report. The results will show performance estimated for the selected loops/functions only.

Run Offload Modeling Perspective from Command Line

[advisor Command Option Reference](#)

[Offload Modeling Command Line Reference](#) This reference section describes the command line options available for each of the Python* scripts that you can use to run the Offload Modeling perspective.

Analyze GPU Roofline

Measure and visualize the actual performance of GPU kernels using benchmarks and hardware metric profiling against hardware-imposed performance ceilings, as well as determine the main limiting factor, by running the GPU Roofline Insights perspective.

Use the **Roofline** chart to answer the following questions:

- What is the maximum achievable performance with your current hardware resources?
- Does your application work optimally on current hardware resources?
- If not, what are the best candidates for optimization?
- Is memory bandwidth or compute capacity limiting performance for each optimization candidate?

Run the GPU Roofline Insights to measure performance of SYCL, C++/Fortran with OpenMP* pragmas, Intel® oneAPI Level Zero (Level Zero), or OpenCL™ applications enabled to run on a GPU.

How It Works

The GPU Roofline Insights perspective includes the following steps:

1. Collect OpenCL™ kernels timings and memory data using the Survey analysis with GPU profiling.
2. Measure the hardware limitations and collect floating-point and integer operations data using the Characterization analysis with GPU profiling.

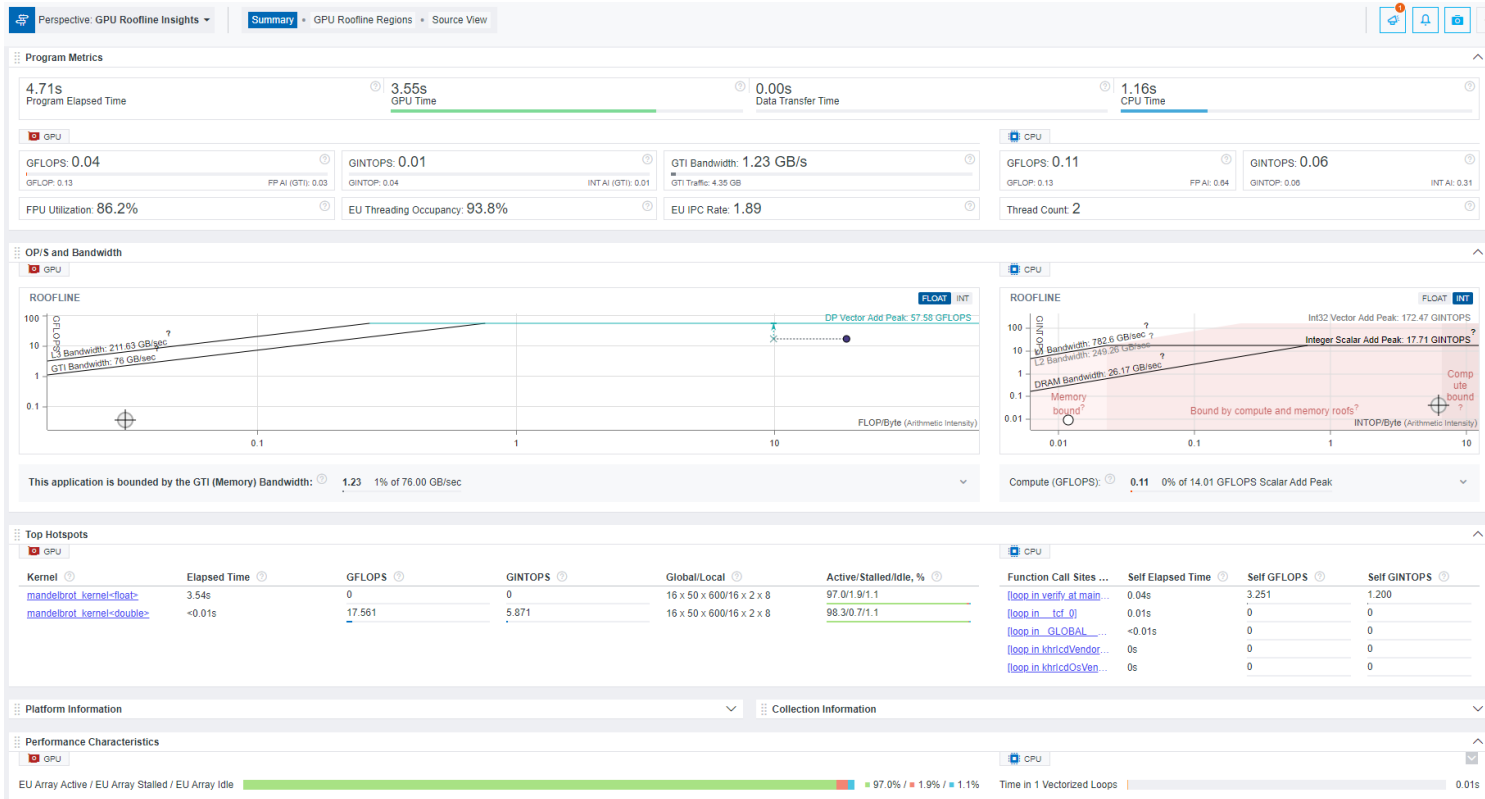
Intel® Advisor calculates compute operations (FLOP and INTOP) as a weighted sum of the following groups of instructions: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Intel Advisor automatically determines data type in the collected operations using the `dst` register.

GPU Roofline Summary

GPU Roofline Insights perspective measures performance of kernels executed on GPU and loops/functions executed on CPU and shows what you should optimize your application for. Examine the following performance data:

- See application execution time on GPU and CPU, time spent to transfer data between the CPU and GPU, and how well your application uses the GPU resources.
- Review the Roofline charts for CPU and GPU parts of your application.
- View the execution time details and various performance metrics on GPU- and CPU-executed parts of your application.
- View top five time-consuming loops on GPU and on CPU sorted by self time with performance metrics. You are recommended to start with these loops when checking for performance issues.



See the **Summary** section to examine the performance summary of your application, and continue to GPU Roofline Insights Regions tab to examine the performance in more detail.

See Also

[Run GPU Roofline Insights Perspective from GUI](#)

[Run GPU Roofline Insights Perspective from Command Line](#)

[Optimize Your GPU Application with Intel® oneAPI Base Toolkit](#)

Run GPU Roofline Insights Perspective from GUI

Prerequisites:

1. [Configure system](#) to analyze GPU kernels.
2. In the graphical-user interface (GUI): [Create a project](#) and specify an analysis target and target options.

To configure and run the GPU Roofline Insights perspective from the GUI:


1. Configure the perspective and set analysis properties, depending on desired results:
 - Select a collection accuracy level with analysis properties preset for a specific result:
 - **Low:** Analyze performance of kernels executed on GPU and plot a GPU Roofline chart for all memory levels. Plot a basic CPU Roofline chart for loops/functions executed on CPU.
 - **Medium:** Analyze performance of kernels executed on GPU, plot a GPU Roofline chart for all memory levels, and model the application performance to get more optimization recommendations. Plot a basic CPU Roofline chart for loops/functions executed on CPU.
 - **High:** Analyze performance of kernels executed on GPU, plot a GPU Roofline chart for all memory levels, and model the application performance to get more optimization recommendations. Plot an extended CPU Roofline chart for loops/functions executed on CPU for all memory levels.
 - Select the analyses and properties manually to adjust the perspective flow to your needs. The accuracy level is set to **Custom**.

By default, accuracy is set to **Low**. The higher accuracy value you choose, the higher runtime overhead is added to your application. The **Overhead** indicator shows the overhead for the selected configuration. For the **Custom** accuracy, the overhead is calculated automatically for the selected analyses and properties.








For more information, see [GPU Roofline Accuracy Presets](#).

NOTE If you want to analyze only code regions executed on GPU, select the **Low** or **Medium** accuracy. This decreases analysis overhead. For the GPU Roofline Insights perspective, the **High** accuracy level controls the complexity of the *CPU Roofline* chart generated for loops/functions in your code executed on CPU.

2. If you have multiple GPUs connected to your system, select a target GPU to collect data for from the **Target GPU** drop-down.
The drop-down shows an adapter address and a name for each GPU available. The address is in the format `<domain>:<bus>:<device-number>.<function-number>`.

3. Run the perspective: click  button.


While the perspective is running, you can do the following in the **Analysis Workflow** tab:

- Control the perspective execution:
 - Stop data collection and see the already collected data: Click the  button.
 - Pause data collection: Click the  button.
 - Cancel data collection and discard the collected data: Click the  button.
- Expand an analysis with  to control the analysis execution:
 - Pause the analysis: Click the  button.
 - Stop the currently running analysis and start the next analysis selected: Click the  button.
 - Interrupt execution of all selected analyses and see the already collected data: Click the  button.

To run the GPU Roofline Insights perspective with the Low accuracy from the command line interface:

```
advisor --collect=roofline --profile-gpu --project-dir=./advi_results -- ./myApplication
```

See [Run GPU Roofline Insights from Command Line](#) for details.

NOTE To [generate command lines](#) for selected perspective configuration, click the  **Command Line** button.

Once the GPU Roofline Insights perspective collects data, the report opens showing a **Summary** with performance metrics measured for CPU- and GPU-executed parts of your application and preview Roofline charts. Continue to [examine GPU bottlenecks on the Roofline chart](#) to investigate the results.

GPU Roofline Accuracy Presets

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium	High
Overhead	5 - 10x	15 - 20x	20 - 50x
Goal	Analyze kernels in your application running on GPU	Analyze kernels running on GPU and loops/functions running on CPU in more details	Analyze kernels running on GPU and loops/functions running on CPU in more details
Analyses	Survey with GPU profiling + Characterization (FLOP)	Survey with GPU profiling + Characterization (FLOP, memory object analysis with light data transfer simulation between host and target device memory) + Performance Modeling for a baseline GPU	Survey with GPU profiling + Characterization (Trip Counts and FLOP with call stacks for CPU, CPU cache simulation, memory object analysis with medium data transfer simulation between host and target device memory) + Performance Modeling for a baseline GPU
Result for kernels on GPU	Memory-level GPU Roofline (for CARM, L3, SLM, GTI) with basic set of recommendations for performance optimization	Memory-level GPU Roofline (for CARM, L3, SLM, GTI) with extended set of recommendations for performance optimization	Memory-level GPU Roofline (for CARM, L3, SLM, GTI) with extended set of recommendations for performance optimization
Result for loops/functions on CPU	Cache-aware CPU Roofline for L1 cache	Memory-level Roofline with call stacks (for L1, L2, L3, DRAM)	Memory-level Roofline with call stacks (for L1, L2, L3, DRAM)

You can choose custom accuracy and set a custom perspective flow for your application. For more information, see [Customize GPU Roofline Insights Perspective](#).



NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).

Customize GPU Roofline Insights Perspective

Customize the perspective flow to better fit your goal and your application.

If you change any of the analysis settings from the **Analysis Workflow** tab, the accuracy level changes to **Custom** automatically. With this accuracy level, you can customize the perspective flow and/or analysis properties.

To change the properties of a specific analysis:

1. Expand the analysis details on the **Analysis Workflow** pane with .
2. Select desired settings.
3. For more detailed customization, click the *gear*  icon. You will see the **Project Properties** dialog box open for the selected analysis.
4. Select desired properties and click **OK**.

For a full set of available properties, click the  icon on the left-side pane or go to **File > Project Properties**.

The following tables cover project properties applicable to the analyses in the GPU Roofline Insights perspective.

Common Properties

Use This	To Do This
Target type drop-down	<ul style="list-style-type: none"> Analyze an executable or script (choose Launch Application). Analyze a process (choose Attach to Process). <p>If you choose Attach to Process, you can either inherit settings from the Survey Hotspots Analysis Type or specify the needed settings.</p>
Inherit settings from Visual Studio project checkbox and field (Visual Studio* IDE only)	<p>Inherit Intel Advisor project properties from the Visual Studio* startup project (enable).</p> <p>If enabled, the Application, Application parameters, and Working directory fields are pre-filled and cannot be modified.</p> <hr/> <p>NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.</p> <hr/>
Application field and Browse... button	<p>Select an analysis target executable or script.</p> <p>If you specify a script in this field, consider specifying the executable in the Advanced > Child application field (required for Dependencies analysis).</p>
Application parameters field and Modify... button	Specify runtime arguments to use when performing analysis (equivalent to command line arguments).
Use application directory as working directory checkbox	Automatically use the value in the Application directory to pre-fill the Working directory value (enable).
Working directory field and Browse... button	Select the working directory.
User-defined environment variables field and Modify... button	Specify environment variables to use during analysis.
Managed code profiling mode drop-down	<ul style="list-style-type: none"> Automatically detect the type of target executable as Native or Managed, and switch to that mode (choose Auto). Collect data for native code and do not attribute data to managed code (choose Native). Collect data for both native and managed code, and attribute data to managed code as appropriate (choose Mixed). Consider using this option when analyzing a native executable that makes calls to managed code.

Use This	To Do This
	<ul style="list-style-type: none"> Collect data for both native and managed code, resolve samples attributed to native code, and attribute data to managed source only (choose Managed). The call stack in the analysis result displays data for managed code only.
Child application field	<p>Analyze a file that is not the starting application. For example: Analyze an executable (identified in this field) called by a script (identified in the Application field).</p> <p>Invoking these properties could decrease analysis overhead.</p> <hr/> <p>NOTE For the Dependencies Analysis Type: If you specify a script file in the Application field, you must specify the target executable in the Child application field.</p> <hr/>
Modules radio buttons, field, and Modify... button	<ul style="list-style-type: none"> Analyze specific modules and disable analysis of all other modules (click the Include only the following module(s) radio button and choose the modules). Disable analysis of specific modules and analyze all other modules (click the Exclude only the following module(s) radio button and choose the modules). <p>Including/excluding modules could minimize analysis overhead.</p>
Use MPI launcher checkbox	<p>Generate a command line (enable) that appears in the Get command line field based on the following parameters:</p> <ul style="list-style-type: none"> Select MPI Launcher - Intel or another vendor Number of ranks - Number of instances of the application Profile ranks - All or a range of ranks to profile
Automatically stop collection after (sec) checkbox and field	<p>Stop collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could minimize analysis overhead.</p>

Survey Analysis Properties

Use This	To Do This
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <hr/> <p>Tip The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p> <hr/>

Use This	To Do This
Sampling Interval selector	Set the wait time between each analysis collection CPU sample while your target application is running. Increasing the wait time could decrease analysis overhead.
Collection data limit, MB selector	Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses. Decreasing the limit could decrease analysis overhead.
Callstack unwinding mode drop-down list	Set to After collection if: <ul style="list-style-type: none"> Survey analysis runtime overhead exceeds 1.1x. A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions. Otherwise, set to During Collection . This mode improves stack accuracy but increases overhead.
Stitch stacks checkbox	Restore a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload (enable). Disable if Survey analysis runtime overhead exceeds 1.1x.
Analyze MKL Loops and Functions checkbox	Show Intel® oneAPI Math Kernel Library (oneMKL) loops and functions in Intel Advisor reports (enable). Enabling could increase analysis overhead.
Analyze Python loops and functions checkbox	Show Python* loops and functions in Intel Advisor reports (enable). Enabling could increase analysis overhead.
Analyze loops that reside in non-executed code paths checkbox	Collect a variety of data during analysis for loops that reside in non-executed code paths, including loop assembly code, instruction set architecture (ISA), and vector length (enable). Enabling could increase analysis overhead. NOTE Analyzing non-executed code paths in binaries that target multiple ISAs (contain multiple code paths) is available only for binaries compiled using the <code>-ax</code> (Linux* OS) / <code>Qax</code> (Windows* OS) option with an Intel compiler.
Enable registry spill/fill analysis checkbox	Calculate the number of consecutive load/store operations in registers and related memory traffic (enable). Enabling could increase analysis overhead.
Enable static instruction mix analysis checkbox	Statically calculate the number of specific instructions present in the binary (enable). Enabling could increase analysis overhead.

Use This	To Do This
Source caching drop-down list	<ul style="list-style-type: none"> Delete source code cache from a project with each analysis run (default; choose Clear cached files). Keep source code cache within the project (choose Keep cached files).

Trip Counts and FLOP Analysis Properties

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <hr/> <p>Tip The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p> <hr/>
Collect information about Loop Trip Counts checkbox	Measure loop invocation and execution (enable).
Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage checkbox	Measure floating-point operations, integer operations, and memory traffic (enable).
Collect stacks checkbox	Collect call stack information when performing analysis (enable). Enabling could increase analysis overhead.
Capture metrics for dynamic loops and functions checkbox	Collect metrics for dynamic Just-In-Time (JIT) generated code regions.
Enable Memory-Level Roofline with cache simulation checkbox	<p>Model multiple levels of cache for data, such as counts of loaded or stored bytes for each loop, to plot the Roofline chart for all memory levels (enable).</p> <p>Enabling could increase analysis overhead.</p> <hr/> <p>NOTE This option is applicable to CPU Roofline only.</p> <hr/>
Cache simulator configuration field	<p>Specify a cache hierarchy configuration to model (enable and specify hierarchy).</p> <hr/> <p>NOTE This option is applicable to CPU Roofline only.</p> <hr/> <p>The hierarchy configuration template is:</p>

Use This	To Do This
	<pre>[num_of_level1_caches]:[num_of_ways_level1_connected]: [level1_cache_size]:[level1_cacheline_size]/ [num_of_level2_caches]:[num_of_ways_level2_connected]: [level2_cache_size]:[level2_cacheline_size]/ [num_of_level3_caches]:[num_of_ways_level3_connected]: [level3_cache_size]:[level3_cacheline_size]</pre> <p>For example: 4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l is the hierarchy configuration for:</p> <ul style="list-style-type: none"> • Four eight-way 32-KB level 1 caches with line size of 64 bytes • Four four-way 256-KB level 2 caches with line size of 64 bytes • One sixteen-way 6-MB level 3 cache with line size of 64 bytes

Run GPU Roofline Insights Perspective from Command Line

To plot a Roofline chart, the Intel® Advisor runs two steps:

1. Collect OpenCL™ kernels timings and memory data using the Survey analysis with GPU profiling.
2. Measure the hardware limitations and collect floating-point and integer operations data using the Characterization analysis with GPU profiling.

Intel® Advisor calculates compute operations (FLOP and INTOP) as a weighted sum of the following groups of instructions: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Intel Advisor automatically determines data type in the collected operations using the `dst` register.

For convenience, Intel Advisor has the shortcut `--collect=roofline` command line action, which you can use to run both Survey and Characterization analyses with a single command. This shortcut command is recommended to run the GPU Roofline Insights perspective.

Tip See [Intel Advisor cheat sheet](#) for quick reference on command line interface.

Prerequisites

1. [Configure your system](#) to analyze GPU kernels.
2. [Set Intel Advisor environment variables](#) with an automated script to enable the `advisor` command line interface (CLI).

Run the GPU Roofline Insights Perspective

There are two methods to run the GPU Roofline analysis. Use *one* of the following:

- Run the shortcut `--collect=roofline` command line action to execute the Survey and Characterization analyses for GPU kernels with a single command. This method is recommended to run the CPU / Memory Roofline Insights perspective, but it does not support MPI applications.
- Run the Survey and Characterization analyses for GPU kernels with the `--collect=survey` and `--collect=tripcounts` command actions separately one by one. This method is recommended if you want to analyze an MPI application.

Optionally, you can also run the Performance Modeling analysis as part of the GPU Roofline Insights perspective. If you select this analysis, it models your application performance on a baseline GPU device as a target to compare it with the actual application performance. This data is used to suggest more recommendations for performance optimization.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Method 1. Run the Shortcut Command

1. Collect data for a GPU Roofline chart with a shortcut.

```
advisor --collect=roofline --profile-gpu --project-dir=./advi_results -- ./myApplication
```

This command collects data both for GPU kernels and CPU loops/functions in your application. For kernels running on GPU, it generates a Memory-Level Roofline.

2. Run Performance Modeling for the GPU that the application runs on.

```
advisor --collect=projection --profile-gpu --model-baseline-gpu --project-dir=./advi_results
```

Important Make sure to use the `--model-baseline-gpu` option for Performance Modeling to work correctly.

This command models your application potential performance on a baseline GPU as a target to determine additional optimization recommendations.

Method 2. Run the Analyses Separately

Use this method if you want to analyze an MPI application.

1. Run the Survey analysis.

```
advisor --collect=survey --profile-gpu --project-dir=./advi_results -- ./myApplication
```

2. Run the Characterization analysis to collect trip counts and FLOP data:

```
advisor --collect=tripcounts --flop --profile-gpu --project-dir=./advi_results -- ./myApplication
```

These commands collect data both for GPU kernels and CPU loops/functions in your application. For kernels running on GPU, it generates a Memory-Level Roofline.

3. Run Performance Modeling for the GPU that the application runs on.

```
advisor --collect=projection --profile-gpu --model-baseline-gpu --project-dir=./advi_results
```

Important Make sure to use the `--model-baseline-gpu` option for Performance Modeling to work correctly.

This command models your application potential performance on a baseline GPU as a target to determine additional optimization recommendations.

You can view the results in the Intel Advisor graphical user interface (GUI) or in CLI, or generate an interactive HTML report. See View the Results below for details.

Analysis Details

The CPU / Memory Roofline Insights workflow includes only the Roofline analysis, which sequentially runs the Survey and Characterization (trip counts and FLOP) analyses.

The analysis has a set of additional options that modify its behavior and collect additional performance data.

Consider the following options:

Roofline Options

To run the Roofline analysis, use the following command line action: `--collect=roofline`.

NOTE You can also use these options with `--collect=survey` and `--collect=tripcounts` if you want to run the analyses separately.

Recommended action options:

Options	Description
<code>--profile-gpu</code>	Analyze GPU kernels. This option is <i>required</i> for each command.
<code>--target-gpu</code>	<p>Select a target GPU adapter to collect profiling data. The adapter configuration should be in the following format <code><domain>:<bus>:<device-number>.<function-number></code>. Only decimal numbers are accepted. Use this option if you have more than one GPU adapter on your system. The default is the latest GPU architecture version found on your system.</p> <hr/> <p>Tip To see a list of GPU adapters available on your system, run <code>advisor --help target-gpu</code> and see the option description.</p> <hr/>
<code>--gpu-sampling-interval=<double></code>	Set an interval (in milliseconds) between GPU samples. By default, it is set to 1.
<code>--enable-data-transfer-analysis</code>	Model data transfer between host memory and device memory. Use this option if you want to run the Performance Modeling analysis.
<code>--track-memory-objects</code>	Attribute memory objects to the analyzed loops that accessed the objects. Use this option if you want to run the Performance Modeling analysis.
<code>--data-transfer=<level></code>	<p>Set the level of details for modeling data transfers during Characterization. Use this option if you want to run the Performance Modeling analysis.</p> <p>Use one of the following values:</p> <ul style="list-style-type: none"> Use <code>light</code> to model only data transfer between host and device memory. Use <code>medium</code> to model data transfers, attribute memory object, and track accesses to stack memory. Use <code>high</code> to model data transfers, attribute memory objects, track accesses to stack memory, and identify where data can be reused.

See [advisor Command Option Reference](#) for more options.

Performance Modeling Options

To run the Performance Modeling analysis, use the following command line action: `--collect=projection`.

The action options in the table below are *required* to use when you run the Performance Modeling analysis as part of the GPU Roofline Insights perspective:

Options	Description
<code>--profile-gpu</code>	Analyze GPU kernels. This option is <i>required</i> for each command.
<code>--enforce-baseline-decomposition</code>	Use the same local size and SIMD width as measured on the baseline. This option is <i>required</i> .
<code>--model-baseline-gpu</code>	Use the baseline GPU configuration as a target device for modeling. This option is <i>required</i> . This option automatically enables the <code>--enforce-baseline-decomposition</code> option, so you can use only <code>--model-baseline-gpu</code> .

See [advisor Command Option Reference](#) for more options.

Next Steps

Continue to [explore GPU Roofline results](#). For details about the metrics reported, see [Accelerator Metrics](#).

See Also

[GPU Roofline Insights Perspective](#) Measure and visualize the actual performance of GPU kernels using benchmarks and hardware metric profiling against hardware-imposed performance ceilings, as well as determine the main limiting factor, by running the GPU Roofline Insights perspective.

[Command Line Interface](#) This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

[Minimize Analysis Overhead](#)

GPU Roofline Accuracy Levels in Command Line

For each perspective, Intel® Advisor has several levels of collection accuracy. Each accuracy level is a set of analyses and properties that control what data is collected and the level of collection details. The higher accuracy value you choose, the higher runtime overhead is added.

In CLI, each accuracy level corresponds to a set of commands with specific options that you should run one by one to get a desired result.

The following accuracy levels are available:

Comparison / Accuracy Level	Low	Medium	High
Overhead	5 - 10x	15 - 20x	20 - 50x
Goal	Analyze kernels in your application running on GPU	Analyze kernels running on GPU and loops/functions running on CPU in more details	Analyze kernels running on GPU and loops/functions running on CPU in more details
Analyses	Survey with GPU profiling + Characterization (FLOP)	Survey with GPU profiling + Characterization (FLOP, memory object analysis with light data transfer	Survey with GPU profiling + Characterization (Trip Counts and FLOP with call stacks for CPU, CPU cache simulation, memory object analysis with medium data transfer

Comparison / Accuracy Level	Low	Medium	High
		simulation between host and target device memory) + Performance Modeling for a baseline GPU	simulation between host and target device memory) + Performance Modeling for a baseline GPU
Result for kernels on GPU	Memory-level GPU Roofline (for CARM, L3, SLM, GTI) with basic set of recommendations for performance optimization	Memory-level GPU Roofline (for CARM, L3, SLM, GTI) with extended set of recommendations for performance optimization	Memory-level GPU Roofline (for CARM, L3, SLM, GTI) with extended set of recommendations for performance optimization
Result for loops/functions on CPU	Cache-aware CPU Roofline for L1 cache	Memory-level Roofline with call stacks (for L1, L2, L3, DRAM)	Memory-level Roofline with call stacks (for L1, L2, L3, DRAM)

You can generate commands for a desired accuracy level from the Intel Advisor GUI. See [Generate Command Lines from GUI](#) for details.

NOTE There is a variety of techniques available to minimize data collection, result size, and execution overhead. Check [Minimize Analysis Overhead](#).

Consider the following command examples.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Low Accuracy

To run the GPU Roofline Insights perspective with the low accuracy:

```
advisor --collect=roofline --profile-gpu --enable-data-transfer-analysis --project-dir=./advi_results -- ./myApplication
```

Medium Accuracy

1. Run the GPU Roofline.

```
advisor --collect=roofline --profile-gpu --enable-data-transfer-analysis --track-memory-objects --data-transfer=light --project-dir=./advi_results -- ./myApplication
```

2. Run Performance Modeling for the GPU that the application runs on.

```
advisor --collect=projection --profile-gpu --enforce-baseline-decomposition --model-baseline-gpu --project-dir=./advi_results
```

NOTE The `--model-baseline-gpu` option automatically enables `--enforce-baseline-decomposition`. To simplify the command, you can skip the `--enforce-baseline-decomposition` option and use only `--model-baseline-gpu`.

High Accuracy

1. Run the GPU Roofline.

```
advisor --collect=roofline --profile-gpu --stacks --enable-cache-simulation --enable-data-transfer-analysis --track-memory-objects --data-transfer=medium --project-dir=./advi_results -- ./myApplication
```

2. Run Performance Modeling for the GPU that the application runs on.

```
advisor --collect=projection --profile-gpu --enforce-baseline-decomposition --model-baseline-gpu --project-dir=./advi_results
```

NOTE The `--model-baseline-gpu` option automatically enables `--enforce-baseline-decomposition`. To simplify the command, you can skip the `--enforce-baseline-decomposition` option and use only `--model-baseline-gpu`.

You can view the results in the Intel Advisor GUI or generate an interactive HTML report.

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[Run GPU Roofline Insights Perspective from Command Line](#)

[Minimize Analysis Overhead](#)

Explore GPU Roofline Results

Intel® Advisor provides several ways to work with the GPU Roofline results.

View Results in GUI

If you run the GPU Roofline Insights perspective from command line, a project is created automatically in the directory specified with `--project-dir`. All the collected results and analysis configurations are stored in the `.advixeproj` project, that you can view in the Intel Advisor.

To open the project in GUI, you can run the following command:

```
advisor-gui <project-dir>
```

NOTE If the report does not open, click **Show Result** on the Welcome pane.

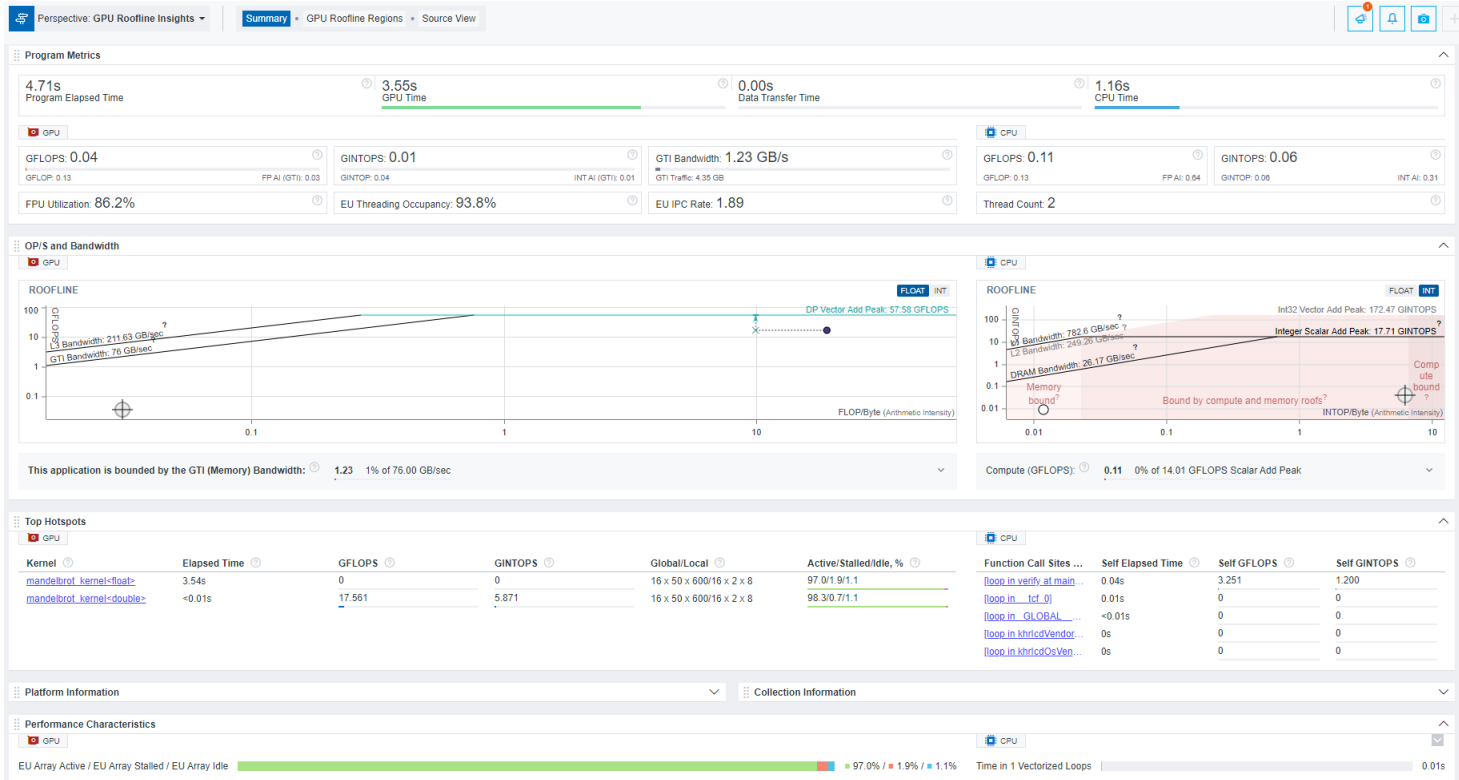
If you run the GPU Roofline Insights perspective from GUI, the result is opened automatically after the collection finishes.

You first see a Summary report that includes performance characteristics for code regions in your code. The left side of the report shows metrics for code regions that run on a GPU, the right side of the report shows metrics for code regions that run on a CPU. The report shows the following data:

- Program metrics for all code regions executed on the GPU and loops/functions executed on the CPU, including total execution time, GPU usage effectiveness, and the number of executed operations.
- Preview Roofline charts for CPU and GPU parts of your code. The charts plot an application's achieved performance and arithmetic intensity against the maximum achievable performance for top three dots and total dot, which combines all loops/functions (for CPU) and kernels (for GPU). By default, it shows Roofline for a dominating operations data type (INT or FLOAT). You can switch to a different data type using the **FLOAT/INT** toggle.

This pane also reports the number of operations transferred per second, bandwidth for different memory levels, and an instruction mix histogram (for GPU only).

- Top five hotspots on CPU and GPU sorted by elapsed time.
- Performance characteristics of how well the application uses hardware resources.
- Information about the analyses executed and platforms that the data was collected on.



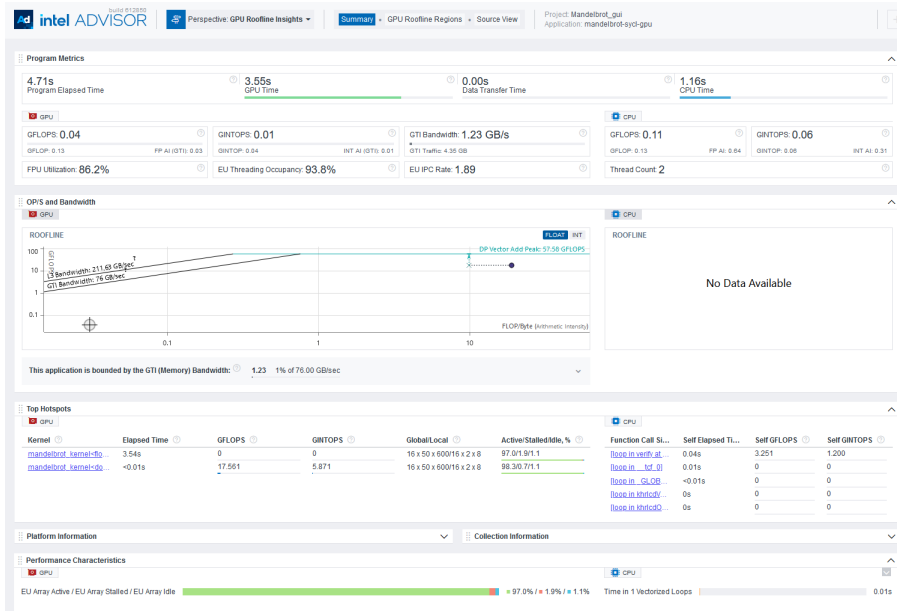
View an Interactive HTML Report

Intel Advisor enables you to export two types of HTML reports, which you can open in your preferred browser and share:

- Interactive HTML report that represents results in the similar way as in GUI and comprises GPU metrics, operations and memory information, a roofline chart, a source view, and grid data.

Tip Collect offload modeling data to view results for Offload Modeling and GPU Roofline Insights perspectives in a single interactive HTML report.

- HTML Roofline report that contains a GPU Roofline chart and enables you to customize your hardware configuration to view how your application executes with given compute and memory parameters.




For details on exporting the HTML reports, see [Work with Standalone HTML Reports](#).

To explore the interactive HTML report, you can [download a precollected GPU Roofline report](#) and examine the results and structure.

Save a Read-only Snapshot

A snapshot is a read-only copy of a project result, which you can view at any time using the Intel Advisor GUI. You can save a snapshot for a project using Intel Advisor GUI or CLI.



To save an active project result as a read-only snapshot from GUI: Click the  button in the top ribbon of the report. In the **Create a Result Snapshot** dialog box, enter the snapshot details and save it.

To save an active project result as a read-only snapshot from CLI:

```
advisor --snapshot --project-dir=<project-dir> [--cache-sources] [--cache-binaries] --<snapshot-path>
```

where:

- `--cache-sources` is an option to add application source code to the snapshot.
- `--cache-binaries` is an option to add application binaries to the snapshot.
- `<snapshot-path>` is a path and a name for the snapshot. For example, if you specify `/tmp/new_snapshot`, a snapshot is saved in a `tmp` directory as `new_snapshot.adviceexpz`. You can skip this and save the snapshot to a current directory as `snapshotXXX.adviceexpz`.

To open the result snapshot in the Intel Advisor GUI, you can run the following command:

```
advisor-gui <snapshot-path>
```

You can visually compare the saved snapshot against the current active result or other snapshot results.

See [Create a Read-only Result Snapshot](#) for details.

Result Interpretation

When you run the GPU Roofline Insights perspective, analyze performance of your application running on GPU and identify headroom for optimization:

- Explore the basic performance metrics and identify top hotspots for optimization using the [GPU Roofline Summary](#)
- Visualize performance of your kernels against hardware-imposed performance ceilings and explore the relationships between your kernels and different memory levels using the [GPU Roofline chart](#)
- Analyze performance and memory metrics for specific kernels, identify headroom for optimization, and get actionable recommendations helping you optimize your application performance using the [GPU Details](#) tab
- Compare results of different optimization iterations using [Roofline Compare](#) functionality

See Also

[Run GPU Roofline Insights Perspective from GUI](#)

[Run GPU Roofline Insights Perspective from Command Line](#)

[Accelerator Metrics](#) This reference section describes the contents of data columns in reports of the Offload Modeling and GPU Roofline Insights perspectives.

Examine GPU Roofline Summary

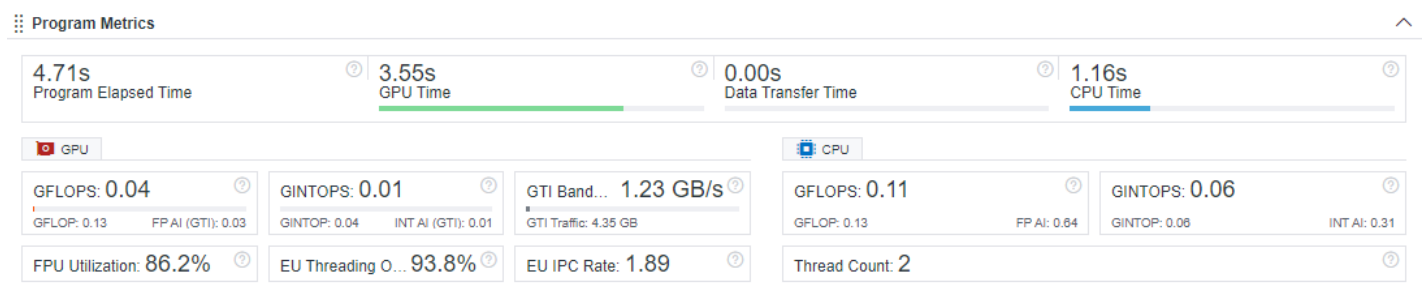
*Explore the overview of program metrics and operations and memory data for your application using the **Summary** report of GPU Roofline Insights.*

NOTE Families of Intel® X^e graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® X^e Graphics](#).

Explore Program Metrics for Code Regions Executed on GPU

Get the insight into performance of your entire application and evaluate the following using the **Program Metrics** pane:

- How much time your application spends on CPU and on GPU in relation to the total time of the application to understand if your application is CPU-bound or GPU-bound
- How much time your application spends on transferring data between CPU and GPU
- How well your application utilizes the floating-point units (FPUs) for parallel execution of operations
- How many threads in each execution unit your application occupies to execute compute operations
- How your application utilizes FPU pipelines and how many instructions it executes per cycle



NOTE For discrete GPUs, FPU Utilization and EU IPC Rate metrics are unavailable.

Identify Dominating Data Types and Hotspots

Intel Advisor profiles your application during its execution and identifies the dominating data type in operations and top hotspots for optimization.

- Explore the operations and identify the dominating data type in the **OP/S and Bandwidth** pane. Use this data to see if the compiler generates integer operations (INTOP) or floating-point operations (FLOP) that are not obvious.



- View the list of top hotspots on the GPU in the **Top Hotspots** pane and examine their performance in relation to compute performance and memory bandwidth using the Roofline chart in the **OP/S and Bandwidth** pane. These hotspots are the best candidates for optimization as they have the greatest impact on the application total time. To view detailed information about the performance of each kernel and visualize it against hardware limitations, double-click a hotspot in the pane or a dot on a roofline chart.

Top Hotspots

GPU

Compute Task	Elapsed Time	GFLOPS	GINTOPS	Global/Local	Active/Stalled/Idle, % ...
GEMM	1.89s	4.546	14.901	1024 x 1024/1 x 1, 256 ...	63.3/36.7/<0.1

CPU

Function Call Sites and Loops	Self Elapsed Time	Self GFLOPS	Self GINTOPS
[loop in func@0x140002270]	0s	0	0

Other analyses and properties are for a CPU Roofline part of the result, which shows metrics for loops/functions executed on CPU. For details about CPU Roofline data, see [CPU / Memory Roofline Insights](#).

Next Steps

Examine Bottlenecks on [GPU Roofline Chart](#).

See Also

- [Accelerator Metrics](#)
- [oneAPI GPU Optimization Guide](#)
- [Optimize Your GPU Application with the Intel® oneAPI Base Toolkit](#)

Examine Bottlenecks on GPU Roofline Chart

*GPU Roofline Insights perspective enables you to view your application performance in relation to the maximum capabilities of your hardware plotted on a Roofline chart, which is available in the **GPU Roofline Regions** view.*

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

Explore Performance-Limiting Factors


Intel® Advisor visualizes the maximum compute capacity and maximum memory bandwidth of your hardware on a Roofline chart:

- Horizontal lines indicate *compute capacity limitations* preventing kernels from achieving better performance without some form of optimization.
- Diagonal lines indicate *memory bandwidth limitations* preventing kernels from achieving better performance without some form of optimization:
 - **L3 cache roof:** Represents the maximal bandwidth of the L3 cache for your current graphics hardware. Measured using an optimized sequence of load operations, iterating over an array that fits entirely into L3 cache.
 - **SLM cache roof:** Shared local memory (SLM). Represents the maximal bandwidth of the SLM for your current graphics hardware. Measured using an optimized sequence of load and store operations that work only with SLM.
 - **GTI roof:** Graphics technology interface (GTI). Represents the maximum bandwidth between the GPU and the rest of the system on a chip (SoC). This estimate is calculated via analytical formula based on the maximum frequency of your current graphics hardware.
 - **DRAM roof:** Dynamic random-access memory (DRAM). Represents the maximal bandwidth of the DRAM memory available to your current graphics hardware. Measured using an optimized sequence of load operations, iterating over an array that does not fit in GPU caches.
 - **HBM roof:** High bandwidth memory (HBM). Represents the maximum bandwidth of HBM memory available to your current graphics hardware. Measured using an optimized sequence of load operations iterating over an array that does not fit in **discrete GPU** caches.

Identify Hotspots and Estimate Room for Optimization

According to Amdahl's law, optimizing kernels that take the largest portion of the total program time leads to greater speedups than optimizing kernels that take the smaller portion of the total time. Intel Advisor enables you to identify kernels taking the largest portion of the total time as hotspots. To find the best candidates for optimization, notice the dots on the Roofline chart. The dots on the chart correspond to kernels running on GPU. Size and color of the dots depends on a dot, or point weight, which is the percentage of the dot time to the program total time and is calculated as *dot self-elapsed time / program total elapsed time * 100*. By default, the size and color of dots is the following:

- Small green dots represent kernels with relatively small execution time (0-1% of the total time).
- Medium-sized yellow dots represent kernels with medium-range execution time (1-20% of the total time).
- Large red dots represent kernels with the largest execution time (20-100% of the total time).

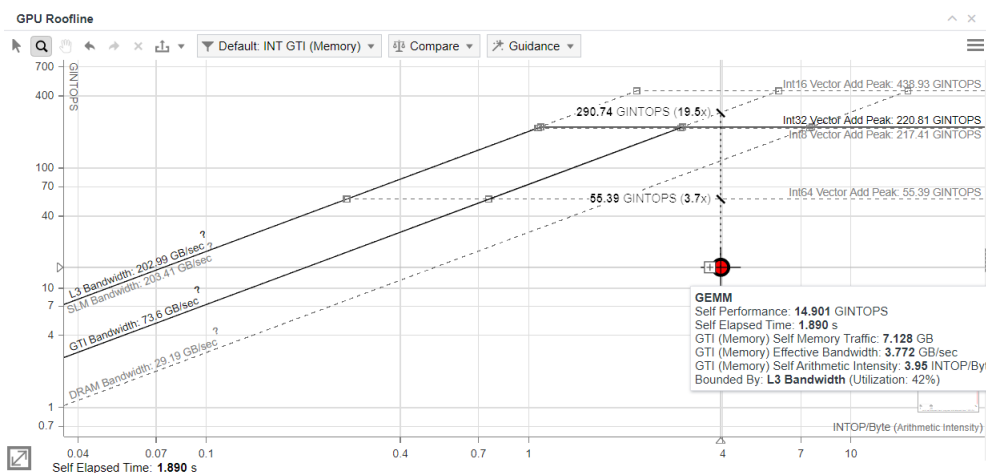
NOTE To customize the dot execution time range, size and color, click the  button on the Roofline chart to open the **Loop Weight Representation** menu.

The best candidates for optimization are the largest dots (red ones by default) located far below the topmost rooflines because:

- Their size clearly shows that improving self elapsed time for these kernels has a significant impact on the total time of the program.
- Their location shows that there is a significant headroom for optimization.

To identify optimization headroom for a specific kernel, double-click a dot on the chart to highlight the roof that limits its performance. The roofs above the dot represent the restrictions preventing it from achieving a higher performance. The dot cannot exceed the topmost rooflines, as they represent the maximum capabilities of the hardware. The farther the dot is from the topmost roofs, the more room for improvement there is.

Hover over the selected dot to view its projection on the limiting roof and the estimated speedup that can be achieved by optimizing this kernel.

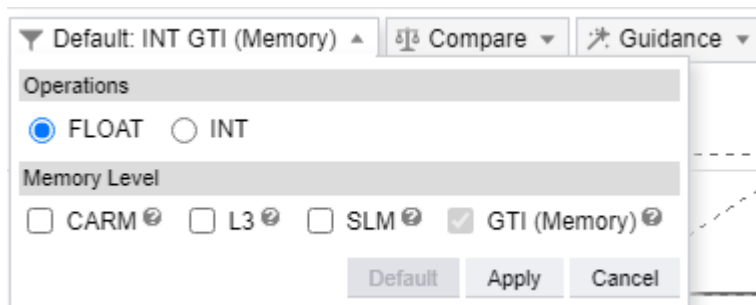


Explore Kernel Performance at Different Memory Levels

By default, Intel Advisor collects data for all memory levels. This enables you to examine each kernel at different cache levels and arithmetic intensities and provides precise insights into which cache level causes the performance bottlenecks.

Configure Memory-Level Roofline Chart

1. Expand the filter pane in the GPU Roofline chart toolbar.
2. In the **Memory Level** section, select the memory levels you want to see metrics for.



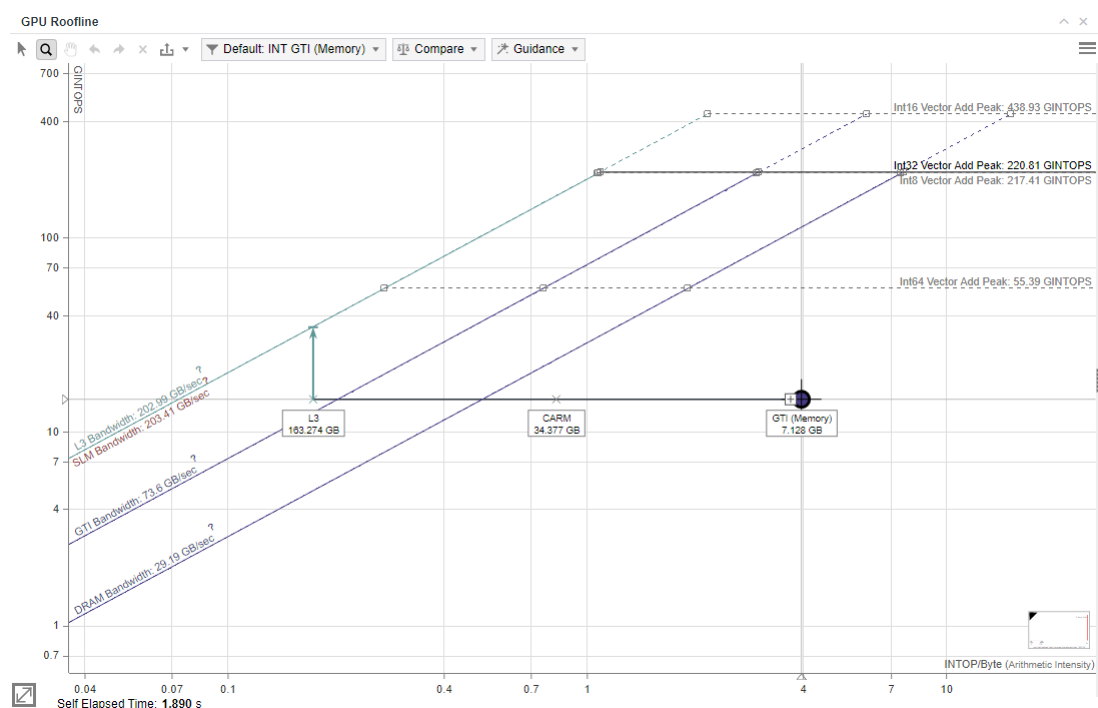
NOTE By default, GPU Roofline reports data for GTI memory level (for integrated graphics) and HBM/DRAM memory level (for discrete graphics).

3. Click **Apply**.

Interpret Memory-Level GPU Roofline Data

Double-click a dot on the chart to review and compare the changes in traffic between the memory levels displayed, identify a memory hierarchy bottleneck, and highlight the roof that limits your kernel performance the most. You can use this information to determine optimization steps. Labeled dots and/or X marks are displayed, representing memory levels with arithmetic intensity for the selected kernel at the following memory levels:

- **CARM**: Memory traffic generated by all execution units (EUs). Includes traffic between EUs and corresponding GPU cache or direct traffic to main memory. For each retired instruction with memory arguments, the size of each memory operand in bytes is added to this metric.
- **L3**: Data transferred directly between execution units and L3 cache.
- **SLM**: Memory access to/from Shared Local Memory (SLM), a dedicated structure within the L3 cache.
- **HBM**: the maximum bandwidth of HBM memory available to your current graphics hardware. The HBM roof is measured using an optimized sequence of load operations iterating over an array that does not fit in **discrete GPU** caches.
- **GTI**: GPU memory read bandwidth, which is the accesses between the GPU, chip uncore (LLC), and main memory on **integrated GPUs**. Use this to understand external memory traffic.
- **DRAM**: Maximum DRAM memory bandwidth available to your current GPU. The DRAM roof is measured using an optimized sequence of load operations iterating over an array that does not fit in GPU caches. This roof represents the maximum bandwidth between the GPU, chip uncore (LLC), and main memory on **discrete GPUs**.



The *vertical distance* between memory dots and their respective roofline shows how much you are limited by a given memory subsystem. If a dot is close to its roof line, it means that the kernel is limited by the bandwidth of this memory level.

The *horizontal distance* between memory dots indicates how efficiently the kernel uses cache. For example, if L3 and GTI dots are very close on the horizontal axis for a single kernel, the kernel uses L3 and GTI similarly. This means that it does not use L3 and GTI efficiently. Improve re-usage of data in the code to improve application performance.

Arithmetic intensity on the x axis determines the order in which dots are plotted, which can provide some insight into your code's performance. For example, the CARM dot is typically far to the right of the L3 dot, as read/write access by cache lines and CARM traffic is the sum of actual bytes used in operations. To identify room for optimization, check L3 cache line utilization metric for a given kernel. If the L3 cache line is not utilized well enough, check memory access patterns in your kernel to improve its elapsed time.

Ideally, the CARM and the L3 dots should be located close to each other, and the GTI dot should be far to the right from them. In this case, the kernel has good memory access patterns and mostly utilizes the L3 cache. If the kernel utilizes the L3 cache line well, it:

- Spends less time on transferring data between L3 and CARM memory levels
- Uses as much data as possible for actual calculations
- Enhances the elapsed time of the kernel and of the entire application

Determine If Your Kernel Is Compute or Memory Bound

To determine if your selected kernel is compute or memory bound, examine the Roofline chart for the selected kernel with the following data in the **Roofline Guidance** section in the **GPU Details** tab:

- Guidance on possible optimization steps depending on the factor limiting performance. Click the bounding factor to expand the hint.
- Amount of data transferred for each cache memory level.
- The exact roof that limits the kernel performance. The arrow points to what you should optimize the kernel for and shows the potential speedup after the optimization in the callout.

If the arrow points to a diagonal line, the kernel is mostly memory bound. If the arrow points to a horizontal line, the kernel is mostly compute bound. Intel® Advisor displays a compute roof limiting the performance of your kernel based on the instruction mix used.

The chart is plotted for a dominant type of operations in a code (FLOAT or INT) and shows only roofs with cache memory levels, data types, and instructions mix used in the kernel. If there is no FLOP or INTOP in the kernel, the single-kernel Roofline chart is not shown.

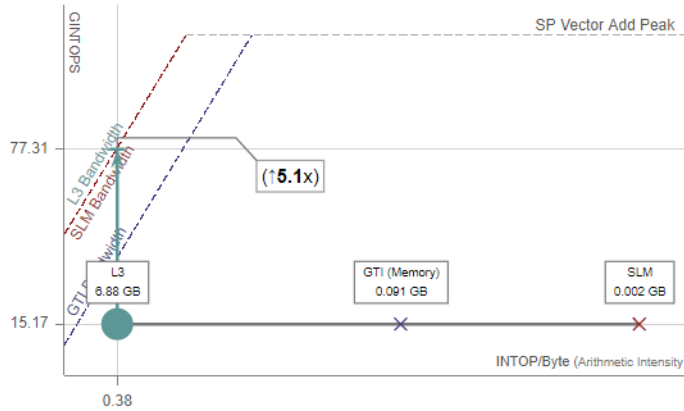
For example, in the screenshot below, the kernel is memory bound. Its performance is limited by the L3 Bandwidth because the kernel uses this memory level to transfer the largest amount of data (6.88 GB) compared to other memory levels. If you optimize the memory access patterns in the kernel, it gets up to 5.1x speedup.

ROOFLINE GUIDANCE

**This kernel is bounded by the L3 Bandwidth**

Improve cache line utilization by optimizing memory access pattern.

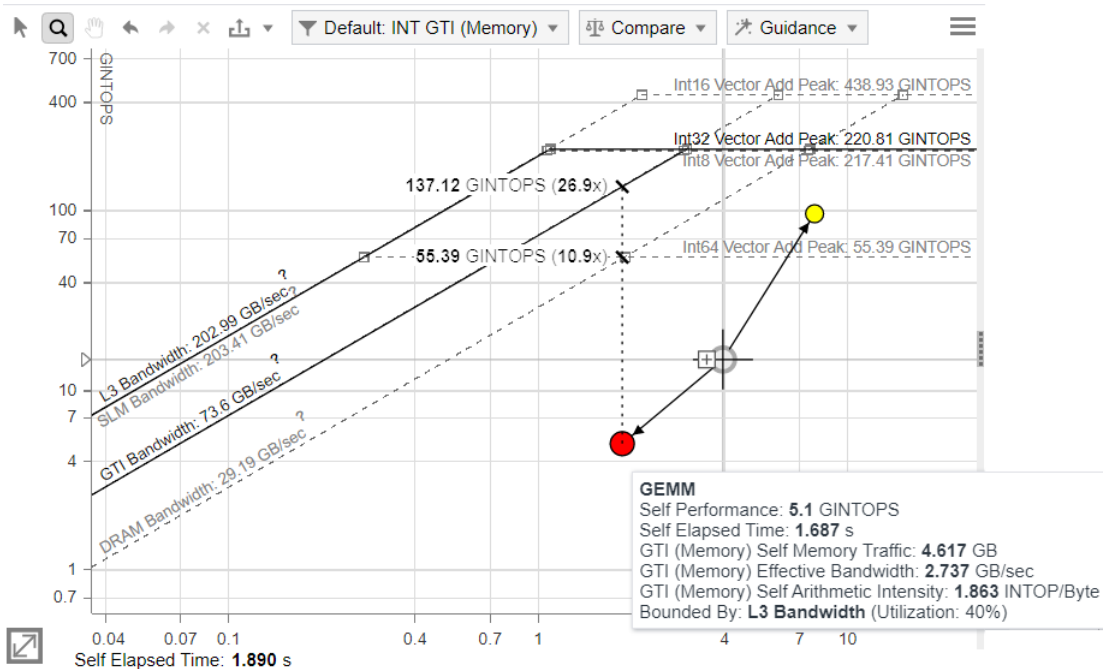
Current cache line utilization is 44%. It means that the number of bytes used by an execution unit is less than the number of bytes transferred to the L3 cache.



Investigate Performance of Compute Tasks

Initialization of the same kernel with different global and local work size is called a compute task. For kernels initialized with different global and local work size, you can review and compare the performance of its compute tasks.


- In the GPU Roofline chart:
 - Click a dot on a Roofline chart and click the **+** button that appears next to the dot. The dot expands into several dots representing the corresponding compute tasks.
 - Click a dot representing a compute task and view details about its global and local work size in the **GPU Details** pane.
 - Hover over a dot representing a compute task to review and compare its performance metrics. Double-click the dot to highlight a roofline limiting the performance of a given instance.



- In the **GPU** pane grid:
 - Expand a kernel in the **Kernels** column.
 - View the information about the work size of a compute task by expanding the **Work Size** column in the grid. To view the number of compute tasks with a given global/local size, expand the **Kernel Details** column in the grid and examine the **Instances** metric.
 - Compare performance metrics for different compute tasks using the grid and the **GPU Details** pane.

GPU Kernels × Recommendations × GPU Source × GPU Assembly ×									
Kernel	Work Size		Kernel Type	Data Tra...		Total Time	Average Time	Instances	SIMD Width
	Global	Local		Total Size					
▼ GEMM	1024 x 1024	1 x 1, 256 x 1	Compute			1.890s	0.472s	4	32
GEMM	1024 x 1024	1 x 1	Compute			1.687s	1.687s	1	32
GEMM	1024 x 1024	256 x 1	Compute			0.203s	0.068s	3	32
zeCommandListAppendMemoryCopy			Host-to-De...	50.3 MB		0.003s	0.000s	12	

Selecting a dot on a chart automatically highlights the respective kernel in the grid and vice versa.

NOTE You can add the CPU Roofline panes to the main view using the  button on the top pane. For details about CPU Roofline data, see [CPU / Memory Roofline Insights Perspective](#)

Next Steps

Explore detailed information about each kernel and get actionable recommendations for optimization of the kernel code using the [GPU Details](#) tab of the GPU Roofline Insights report.

See Also

- [Accelerator Metrics](#)
- [oneAPI GPU Optimization Guide](#)
- [Optimize Your GPU Application with the Intel® oneAPI Base Toolkit](#)

Examine Kernel Details

*After identifying hotspots, use the GPU Roofline Insights perspective to analyze their performance deeper. Select a dot on the chart and use **GPU Details** and **Recommendations** tabs in the right-side pane to examine code analytics for a specific kernel in more details and view actionable recommendations for code optimization.*

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

Get Recommendations

Check the Performance Issues column of the **GPU** pane to see if Intel® Advisor identifies any recommendations for a kernel.

Select a kernel on a Roofline chart and switch to **Recommendations** tab to view actionable recommendations helping you optimize your code for compute and memory bound applications running on GPU. Expand a recommendation to access a full description and a code sample containing a possible solution of the problem.

GPU Details × Recommendations × GPU Source ×

All Advisor-detectable issues: [C++](#) | [Fortran](#)

SIMD divergence present
 Confidence level: low
 The kernel is mostly bounded by GPU **Trip Counts**. This kernel contains SIMD divergence. It decreases the efficiency of vector instruction performance: a part of SIMD lanes will be inactive waiting for another part to finish work. The percentage of inactive SIMD lanes (**in average**) in this kernel is **36.97%**.

Reduce SIMD width
 Confidence level: low
 Current SIMD width: **32**. Reduce SIMD width to **16** to decrease SIMD divergence impact on vector instruction performance.
Note: As a result, the number of threads scheduled on Execution Unit in parallel might grow and improve performance. Current number of parallel threads is **19** (**1** per EU).
 In [Data Parallel C++ \(DPC++\)](#), you can set SIMD width with the attribute

```
[[intel::reqd_sub_group_size(<SIMD_width>)]]
```

 as specified in the documentation for [sub-groups](#).

Review Compute and Memory Bandwidth Utilization

Review how well your kernel uses the compute and memory bandwidth of your hardware in the **OP/S and Bandwidth** pane. It indicates the following metrics:

- The total number of floating-point and integer operations transferred by the kernel per second as a percentage of the maximum compute capacity of your hardware. The red bar represents the dominant operation data type used in the kernel.
- The amount of data transferred by the kernel at each cache memory level per second as a percentage of the memory level bandwidth. Cache memory level bandwidth utilization (in per cent) is a ratio of effective bandwidth and maximum bandwidth of a given memory level. This metric shows how well the kernel uses the capability of your hardware and can help you identify bottlenecks for your kernel.

For example, in the screenshot below, the dominating data type is FLOP. The kernel utilizes 19% of L3 Bandwidth. Considering these data and compared to utilization metrics for other memory levels and compute capacity, the Roofline chart displays the L3 Bandwidth as the main factor limiting the performance of the kernel.

OP/S AND BANDWIDTH ^

▶ GINTOPS: ?	10.57	4% of 220.82 Int32 GINTOPS
▶ GFLOPS: ?	15.17	6% of 220.81 SP GFLOPS
▶ CARM Bandwidth: ?	25.72	GB/sec
▶ L3 Bandwidth: ?	39.95	19% of 203.55 GB/sec
▶ SLM Bandwidth: ?	0.01	0% of 203.46 GB/sec
▶ GTI (Memory) Bandwidth: ?	0.53	0% of 73.60 GB/sec

Review how your application uses memory levels using the **Memory Metrics** pane:

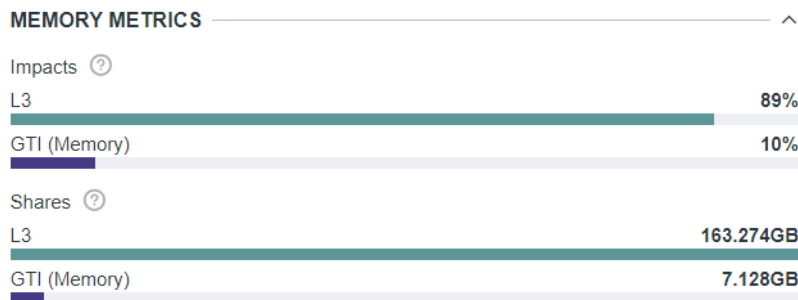
- Review how much time the kernel spends processing requests for each memory level in relation to the total time, in perspective, reported in the Impacts histogram.

A big value indicates a memory level that bounds the selected kernel. Examine the difference between the two largest bars to see how much throughput you can gain if you reduce the impact on your main bottleneck. It also gives you a long-time plan to reduce your memory bound limitations as once you will solve the problems coming from the widest bar, your next issue will come from the second biggest bar and so on.

Ideally, you should see the L3 or SLM as the most impactful memory.

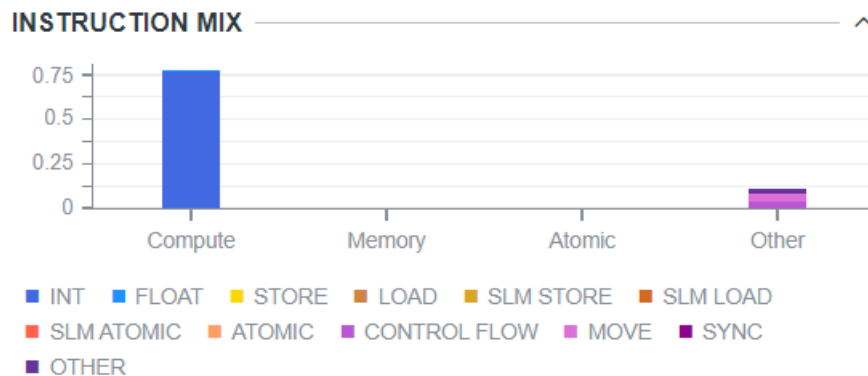
- Review an amount of data that passes through each memory level reported in the **Shares** histogram.

NOTE Data in the Memory Metrics pane is based on a dominant type of operations in your code (FLOAT or INT).



Explore Operation Types Used During Application Execution

Examine instruction types that the kernel executes in the **Instruction Mix** pane. For example, in a screenshot below, the kernel mostly executes compute instructions with integer operations, which means that the kernel is mostly compute bound.



Intel Advisor automatically determines the data type used in operations and groups the instructions collected during Characterization analysis by the following categories:

Category	Instruction Types
Compute (FLOP and INTOP)	<ul style="list-style-type: none"> BASIC COMPUTE: add, addc, mul, rndu, rndd, rnde, rndz, subb, avg, frc, lzd, fbh, fbl, cbit

Category	Instruction Types
	<ul style="list-style-type: none"> • BIT: and, not, or, xor, asr, shr, shl, bfreve, bfe, bfil, bfi2, ror, rol • FMA: mac, mach, mad, madm (weight 2) • DIV: INT_DIV_BOTH, INT_DIV_QUOTIENT, INT_DIV_REMAINDER, and FDIV types of extended math function • POW extended math function • MATH: other function types performed by math instruction • VECTOR: add3 (weight 2), line (weight 2), sad2 (weight 3), dp2 (weight 3), sada2 (weight 4), lrp (weight 4), pln (weight 4), dp3 (weight 5), dph (weight 6), dp4 (weight 7), dp4a (weight 8)
Memory	LOAD, STORE, SLM_LOAD, SLM_STORE types depending on the argument: send, sendc, sends, sendsc
Other	<ul style="list-style-type: none"> • MOVE: mov, sel, movi, smov, csel • CONTROL FLOW: if, else, endif, while, break, cont, call, calla, ret, goto, jmp, brd, brc, join, halt • SYNC: wait, sync • OTHER: cmp, cmpn, nop, f32to16, f16to32, dim
Atomic	SEND

Get more insights about instructions used in your kernel using **Instruction Mix Details** pane:

- Examine instruction count for each category as well as its percentage in overall instruction mix to determine the dominating category of instructions in the kernel.
- Examine instruction count for each type of compute, memory, atomics, and other instructions.
- For compute instructions, view the dominating data type for each type of instructions.

NOTE The data type dominating in the entire kernel is highlighted blue.

INSTRUCTION MIX DETAILS

▼ Atomic	0.02	1%
<div>Instruction ▾Count</div> <div>ATOMIC0.02</div>		
▼ Compute	0.41	30%
<div>Instruction ▾Data Type ▾Count</div> <div>BITINT0.07</div> <div>BASICINT0.09</div> <div>BASICFLOAT0.11</div> <div>FMAFLOAT0.13</div> <div>MATHFLOAT0.01</div>		
▼ Memory	0.04	3%
<div>Instruction ▾Count</div> <div>STORE<0.01</div> <div>LOAD0.04</div>		
▼ Other	0.88	66%
<div>Instruction ▾Count</div> <div>CONTROL FLOW0.12</div> <div>MOVE0.72</div> <div>OTHER0.04</div>		

In the **Performance Characteristics**, review how effectively the kernel uses the GPU resources: activity of all execution units, percentage of time when both FPUs are used, percentage of cycles with a thread scheduled. Ideally, you should see a higher percentage of active execution units and other effectiveness metrics to use more GPU resources.

PERFORMANCE CHARACTERISTICS



Active: ?	63.3%
Stalled: ?	36.7%
Idle: ?	0.0%

SIMD Width: ?	32
EU Threading Occupancy: ?	99.8%
2 FPUs Active: ?	16.2%

See Also

- [Accelerator Metrics](#)
- [Cookbook: Identify Code Regions to Offload to GPU and Visualize GPU Usage](#)
- [oneAPI GPU Optimization Guide](#)
- [Optimize Your GPU Application with the Intel® oneAPI Base Toolkit](#)

Compare GPU Roofline Results

Use the *Roofline Compare* functionality to display Roofline chart data from other Intel® Advisor results or non-archived snapshots for comparison purposes to track optimization progress.

Prerequisites

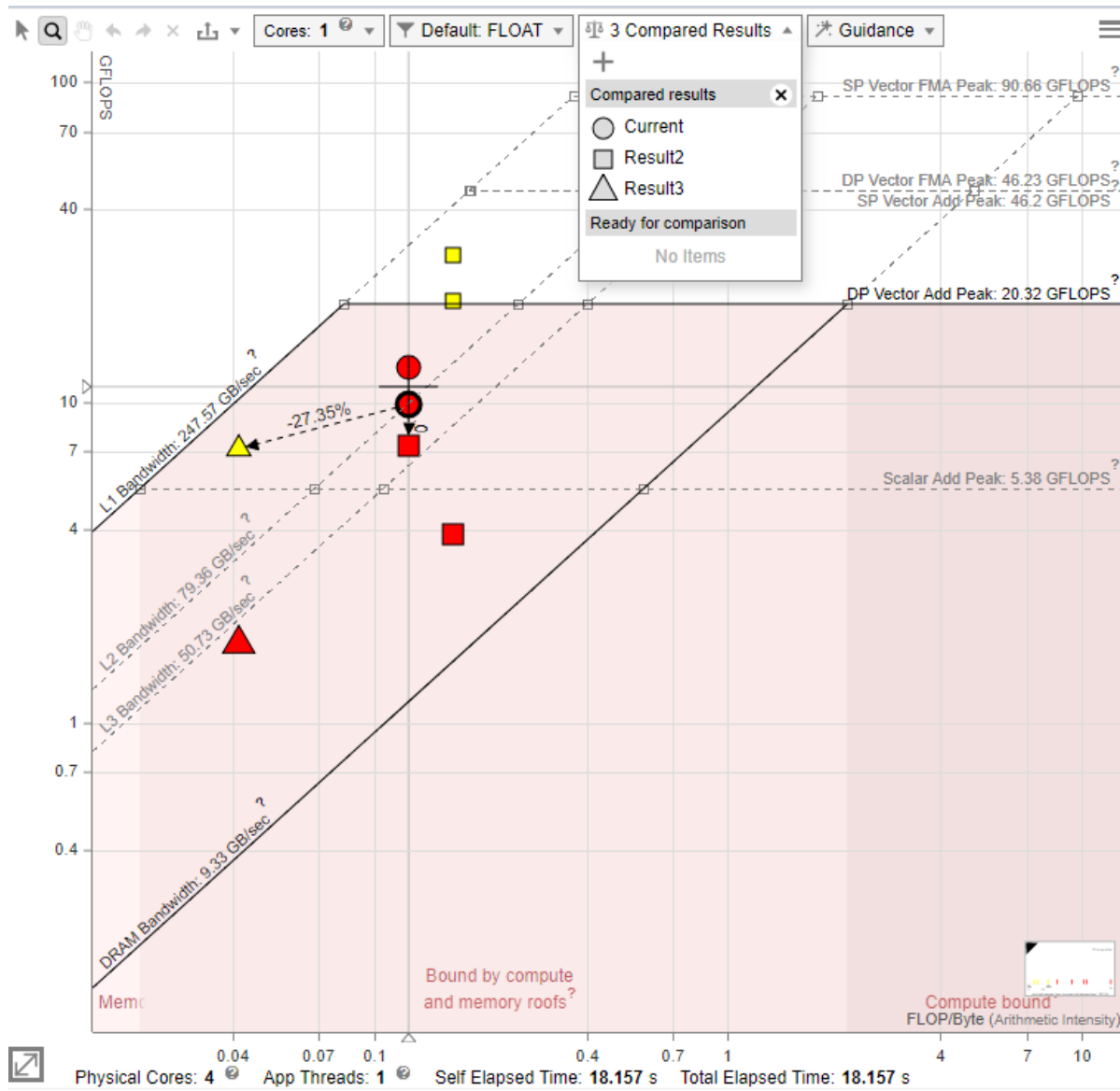
To compare the GPU Roofline results, make sure to get the following:

- A baseline GPU Roofline *result* or *snapshot*
- One or more GPU Roofline results or snapshots of the *same* application with an optimization applied

Important You can only compare Roofline results of the same type: CPU Roofline or GPU Roofline.

To compare the results:

1. Open a baseline GPU Roofline result/snapshot.
2. From the **Compare** drop-down toolbar, click **+** to load a comparison result/snapshot. You can load multiple results/snapshots for comparison one by one.



When the comparison is uploaded:

- The filenames for uploaded results/snapshots are displayed in the **Compared Results** region.
- Similar kernels from all compared results are recognized automatically. They are connected with a dashed arrow line. The performance improvement between the kernels is shown above the line, in per cent. The improvement is calculated as the *difference* in FLOPS, INTOPS, or OPS and Total Time.

NOTE The arrows showing the relationship among kernels do not reappear if you upload a new comparison file.

- Kernels from different compared results are shown as different icons on the chart. For example, for three , the baseline kernels can be shown as circles and comparison kernels can be triangles and diamonds.
- To highlight all dots from a specific compared result, open the **Compare** drop-down and hover over the result name.
- Each time you change the Roofline configuration or filter the dots on the chart, the comparison is updated automatically.
- You can remove a selected result from **Compared Results** by hovering over it and clicking the **X** icon. The result is removed from the chart and appears in the **Ready for comparison** region. Click a name in the **Ready for comparison** region to reload the result back to the chart.
- You can save the comparison itself to a file using the export feature.

NOTE To find the same kernels among the results, Intel Advisor compares several kernel features, such as their type, nesting level, source code file name and line, and function name. When a certain threshold of similar or equal features is reached, the two kernels are considered a match and connected with a dashed line. However, this method still has few limitations. Sometimes, there can be no match for the same kernel if one is optimized or moved in the source code to four or more lines from the original place. Intel Advisor tries to ensure some balance between matching source code changes and false positives.

Design and Analyze Flow Graphs

This section explains how to use Intel® Advisor – Flow Graph Analyzer (FGA), a graphical tool for constructing, analyzing, and visualizing applications that use the Intel® oneAPI Threading Building Blocks flow graph interfaces. Through a graphical interface, the Flow Graph Analyzer lets you:

- Start with a blank canvas and construct a flow graph application by interactively adding nodes and edges.
- Collect events during the execution of an existing application. These events allow you to explore the topology and performance of the flow graphs used by that application.

Where to Find the Flow Graph Analyzer

The Flow Graph Analyzer package consist of a *Flow Graph Analyzer tool* and an associated *collector* to capture traces from Intel® oneAPI Threading Building Blocks flow graph and OpenMP* applications.

This package is part of Intel® Advisor and is installed in the following directory when the Intel® Advisor is installed: <advisor-install-dir>/fga

Under this directory, you can find the Flow Graph Analyzer tool, the collector, and supporting documentation.

- Flow Graph Analyzer: <advisor-install-dir>/fga/fga
- Flow Graph Collector: <advisor-install-dir>/fga/fgt
- Documentation: <advisor-install-dir>/fga/doc
- Samples: <advisor-install-dir>/fga/samples
- License: <advisor-install-dir>/fga/EULA.pdf

Launching the Flow Graph Analyzer

The Flow Graph Analyzer executable and all its supporting files are located in the directory <advisor-install-dir>/fga/fga. To launch it, go to this directory and do the following:

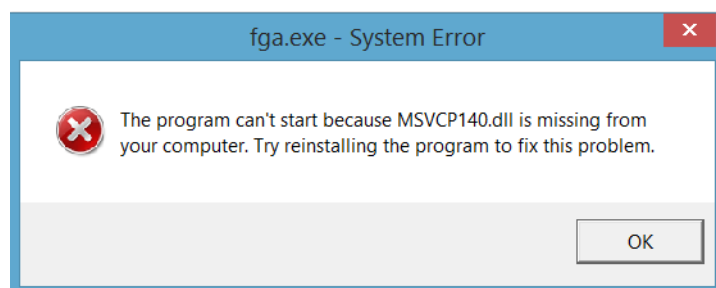
Operating System	To run from an executable	To run from command line
Windows* OS	Double-click fga.exe.	

Operating System	To run from an executable	To run from command line
Linux* platforms	<ol style="list-style-type: none"> 1. Add . to your LD_LIBRARY_PATH environment variable. 2. Double-click fga in the <advisor-install-dir>/fga/fga. 	<ol style="list-style-type: none"> 1. Open a command prompt application from <advisor-install-dir>/fga/fga. 2. Run the run_fga.sh from the command line, which automatically sets the LD_LIBRARY_PATH and then starts the executable.
macOS*	<ol style="list-style-type: none"> 1. Add . to your DYLD_LIBRARY_PATH environment variable. 2. Add ./Frameworks to the DYLD_FRAMEWORK_PATH environment variable. 3. Run the fga executable in the <advisor-install-dir>/fga/fga. 	<ol style="list-style-type: none"> 1. Open a command prompt application from <advisor-install-dir>/fga/fga. 2. Run run_fga.sh from the command line.

Warning

Moving the executable to a different location may cause the application to fail.

On Windows* OS without the Microsoft Visual Studio* runtime components, you may encounter the following error when launching the Flow Graph Analyzer application:



Solution: Download the Microsoft Visual Studio* runtime components and install them before running the application.

Flow Graph Collector

The Flow Graph Collector allows you to capture a topology and execution trace information of a running flow graph application. You can load this captured data to the Flow Graph Analyzer for closer inspection of the graph described by the application and the performance data for the graph. See the [Scalability Analysis](#) section for the steps to build your application with enabled traces and to capture the trace information. See the [Collecting Traces from Applications](#) for instructions on how to use the Flow Graph Analyzer for analyzing the performance of flow graph applications after a trace has been collected.

Flow Graph Analyzer GUI Overview

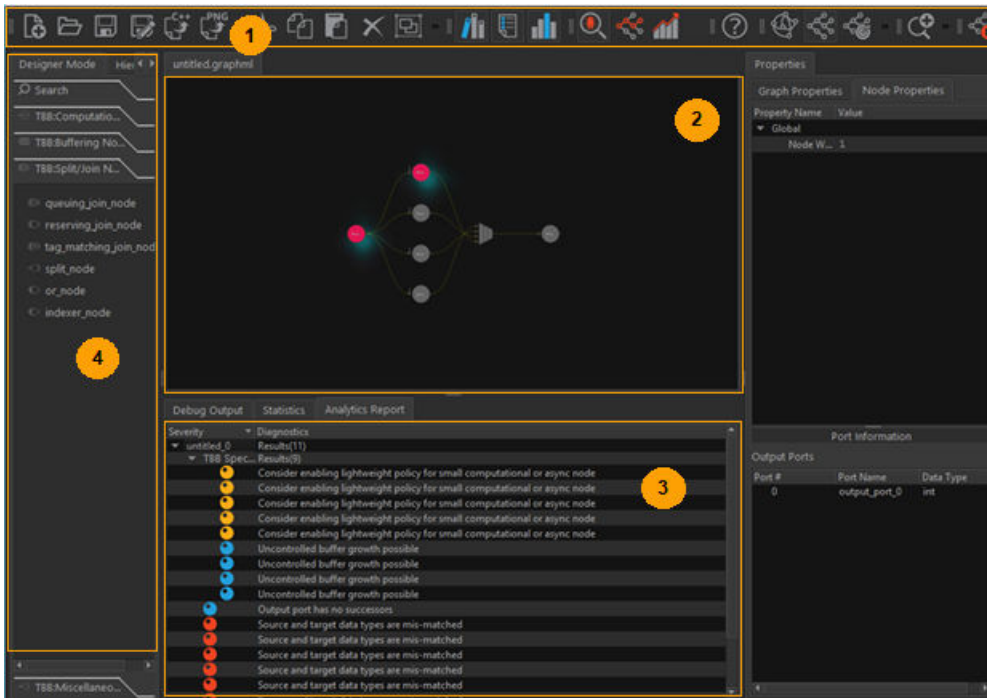
This section describes the Flow Graph Analyzer tool and the various features it offers to speed up the development of new flow graph applications. There are two main tools represented by corresponding workflows:

- A *designer workflow* enhances productivity during development.
- An *analyzer workflow* supports performance-tuning tasks.

Basic GUI Layout

The Flow Graph Analyzer GUI allows you to create new graphs visually and load previously created or application-generated graphs.

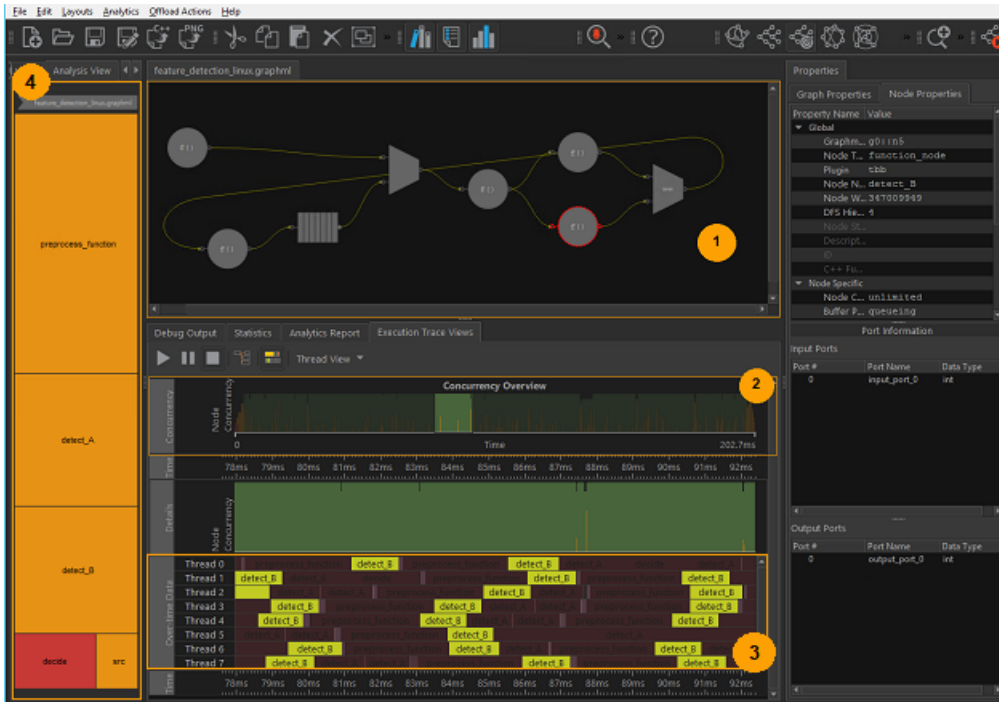
The following figure describes the basic GUI layout and the key elements necessary for constructing graphs visually. These graphs can be saved for later use and, as described in the [Generating C++ Stubs](#) section, used to generate C++ framework code.



- | | |
|---|--|
| 1 | Toolbar supporting basic file and editing operations, visualization and analytics that operate on the graph or performance traces. |
| 2 | Canvas for visualizing and drawing flow graphs. |
| 3 | Output generated by custom analytics. You can interact with the output. |
| 4 | Palette of supported Intel® oneAPI Threading Building Blocks node types organized in groups. |

GUI Layout with Trace Data

When analyzing an existing application's performance, the graph topology and performance data are captured from a running application and saved for a post-mortem analysis. If performance traces are available when the graph is loaded, they are displayed in a timeline window below the canvas area. You can interact with the trace data in many ways, from cursory inspection of the trace data to detailed inspection of specific tasks and the nodes they map to. The following figure shows the timeline charts created when trace data is available.

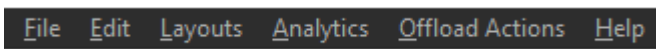


- 1 Selection on the timeline highlights the nodes executing at that point in time.
- 2 The concurrency histogram shows the parallelism achieved by the graph over time. You can interact with this chart by zooming in to a region of time, for example, during low concurrency.

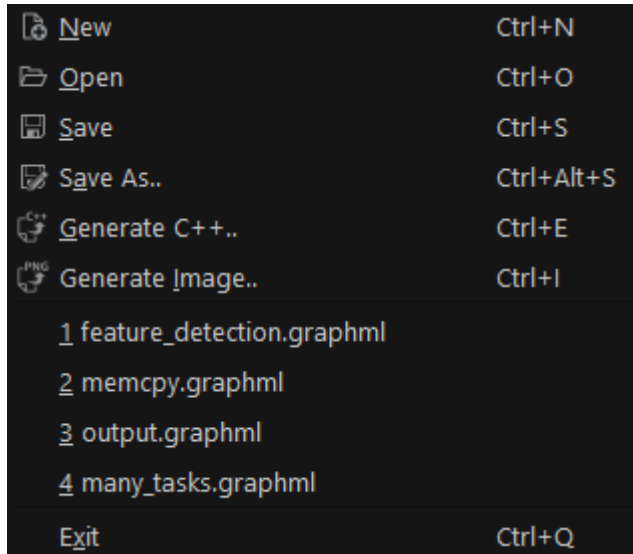
The concurrency histogram remains at the initial zoom level, and the zoomed-in region is displayed below it.
- 3 The per-thread task view shows the tasks executed by each thread, along with the task durations.
- 4 Treemap view provides the general health of the graph's performance, along with the ability to dive to the node level.

Menus

The menus in the menu bar have fixed components such as **File**, **Edit** and **Help**, and dynamic components such as **Layouts**, **Analytics**, and **Trace Data Collection**. The fixed components are always available, and the dynamic components may change depending on the plugins registered with the tool.

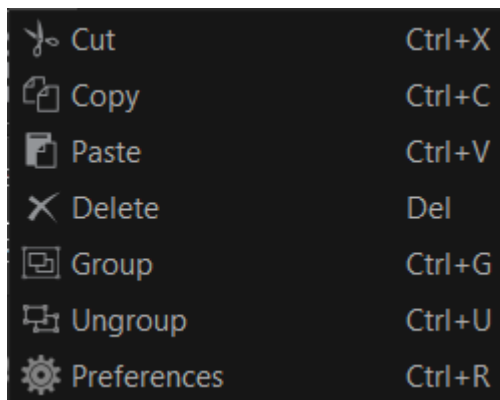


- **File** menu: Allows you to create a new graph, load an existing graph, save the current graph on the canvas to a GraphML* file, or export it as C++ source files.

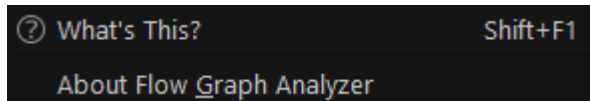


The menu also keeps a list of recently used files for quick access. Print support is currently unavailable. The **Generate Image** option enables printing the graph displayed in the canvas as a PNG file.

- **Edit** menu: Allows you to edit the graph displayed on the canvas and supports common edit actions, such as **Cut**, **Copy**, **Paste**, **Delete**, **Group**, **Ungroup**, and **Preferences**. These actions support the common keyboard shortcuts.



- **Help** menu: Allows you to switch to the **What's This?** mode, which provides help information for various GUI elements. In this mode, you can click any GUI element that has supporting help information to view more information about the element and what it helps you accomplish.

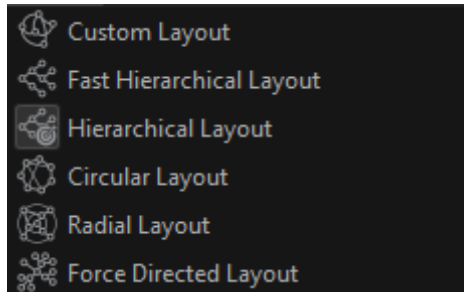


You can also get into the **What's This?** mode using the keyboard shortcut Shift+F1.

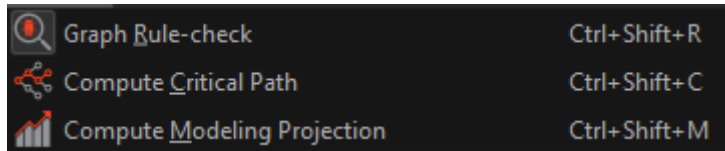
- **Layout** menu: Allows you to visualize a graph on the canvas in different ways. Currently supported layout types are **Hierarchical**, **Radial**, **Force-Directed**, **New Hierarchical**, and **Circular**. For most graphs, the **Hierarchical Layout** is enough and is set as the default layout. If the **Hierarchical Layout** does not work properly for a graph model, you can use the **New Hierarchical Layout**.

NOTE The **New Hierarchical Layout** is 3x slower than the default **Hierarchical Layout**.

If you cannot get a visually pleasing layout using the hierarchical layouts, use the **Radial**, **Circular**, or **Force-Directed** layout. The **Circular Layout** and **Force-Directed Layout** use the Boost* Graph library. The cost of running the **Force-Directed Layout** is high compared to other layouts, but it provides better graph layout visuals.



- **Analytics** menu: Allows you to choose an analytical algorithm from available plugins. This menu changes as new plugins are added.



These analytical algorithms are available for Intel® oneAPI Threading Building Blocks flow graphs. More algorithms may be added in the future.

- **Compute Critical Path** computes one or more critical paths for a graph and lists them in the Analytics Report tab. You can interact with these critical paths to see which nodes are part of them.
- **Graph Rule-check** performs basic rule checks on a graph and highlights potential performance and correctness problems.
- **Compute Modeling Projection** projects the speedup of a graph with varying numbers of threads. The speedup with the corresponding number of threads is shown in the **Analytics Reports** window, while a chart showing the ideal versus actual speedup is shown in the chart area.

Toolbars

The menu items are shown in the toolbar area as shortcuts to frequently used operations. Hover the mouse over an icon on a toolbar to see a tooltip with description.

- **File** toolbar: Provides access to the functionality from the **File** menu.



- **Edit** toolbar: Provides access to the editing operations from the **Edit** menu.



- **Window** toolbar: Provides the show/hide toggle capability for configurable GUI elements. From left to right, these icons represent the **Toolbox**, **Reports**, and **Charts** tabs.



- Hide the **Toolbox** tab, which contains the **Designer Mode**, **Analysis Mode**, and **Hierarchical View** tabs to increase available screen space for large graphs.
- Hide the **Reports** tab, which shows information such as node properties or output of analytics algorithms, to get more screen space for visualizing large graphs.
- By default, the **Charts** tab is hidden if the graph does not have associated execution trace data. When the trace data is available, the **Charts** tab is displayed.
- **Analytics** toolbar: Provides a subset of plugins from the **Analytics** menu. A plugin can be hidden in the toolbar, but it is always registered in the **Analytics** menu.



The analytics supported for Intel® oneAPI Threading Building Blocks (oneTBB) flow graphs are, from left to right in the toolbar, **Compute Critical Path**, **Graph Rule-Check**, and **Compute Modeling Projection**. Use these algorithms to design new and tune existing graphs.

- **Layout** toolbar: Provides a subset of the layouts from the **Layout** menu.



- **Zoom** toolbar: Allows you to zoom in or out the canvas area that displays the graph. The reset-zoom button resets the zoom factor, so the entire graph is visible in the canvas area.

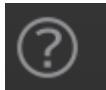


NOTE You can also zoom in or out using the mouse-wheel when the mouse is in the canvas area.

- **Trace Data Collector** toolbar: Opens a dialog box to configure a data collection run on a oneTBB flow graph application. This dialog box also allows you to configure the environment before launching the application.



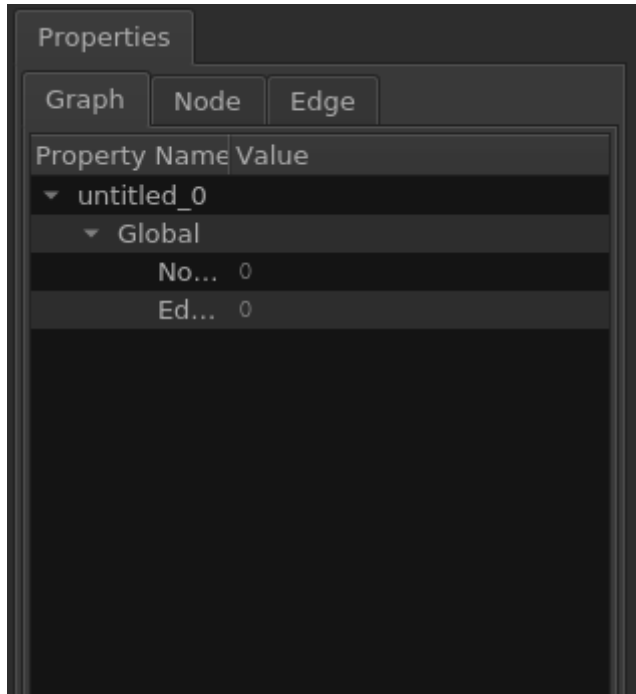
- **What's This?** toolbar: Switches to the **What's This?** mode.



NOTE You can also switch to this mode using the **Shift+F1** keyboard shortcut.

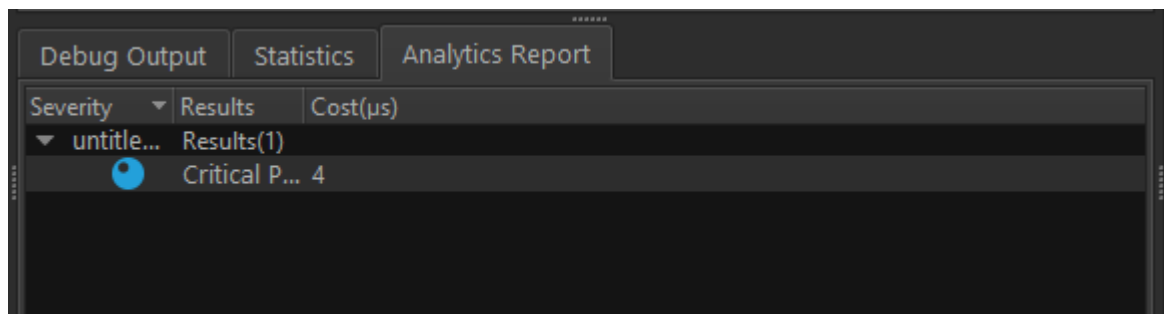
Tabs

- **Properties** tab: Displays information about a selected graph in the **Graph** tab, a node in the **Node** tab, and an edge in the **Edge** tab.



The **Node** tab displays all properties supported for a given node type and allows you to interactively edit them. Some properties might be tied to the node type and are marked as read-only. See the [Designer Workflow](#) section for instructions on editing the properties of a node.

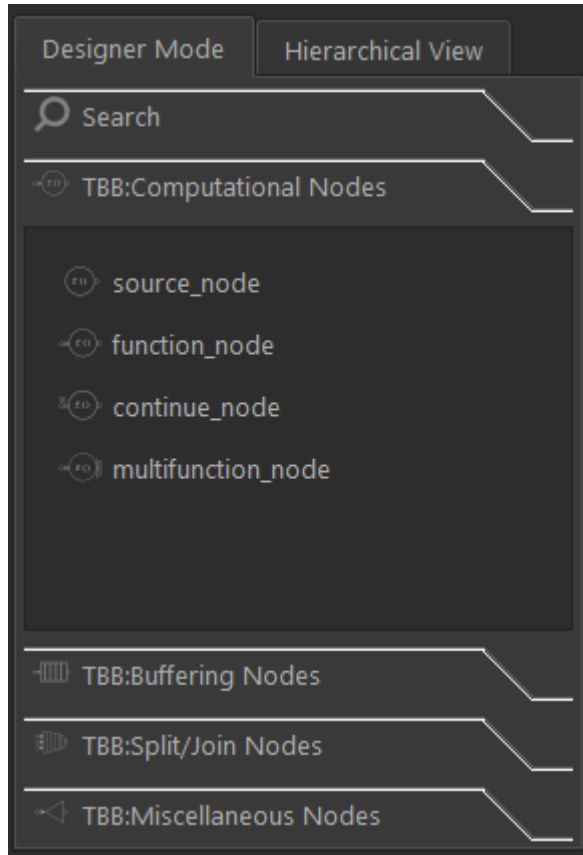
- **Analytics Report** tab: Displays the output generated by any invoked analytics algorithms. Because analytics algorithms generate different outputs, this view changes when depending on an algorithm you run on the graph. The screenshot below shows a sample output for **Compute Critical Path**. The columns in this tab are sortable and enable efficient data manipulation during the performance tuning workflow.



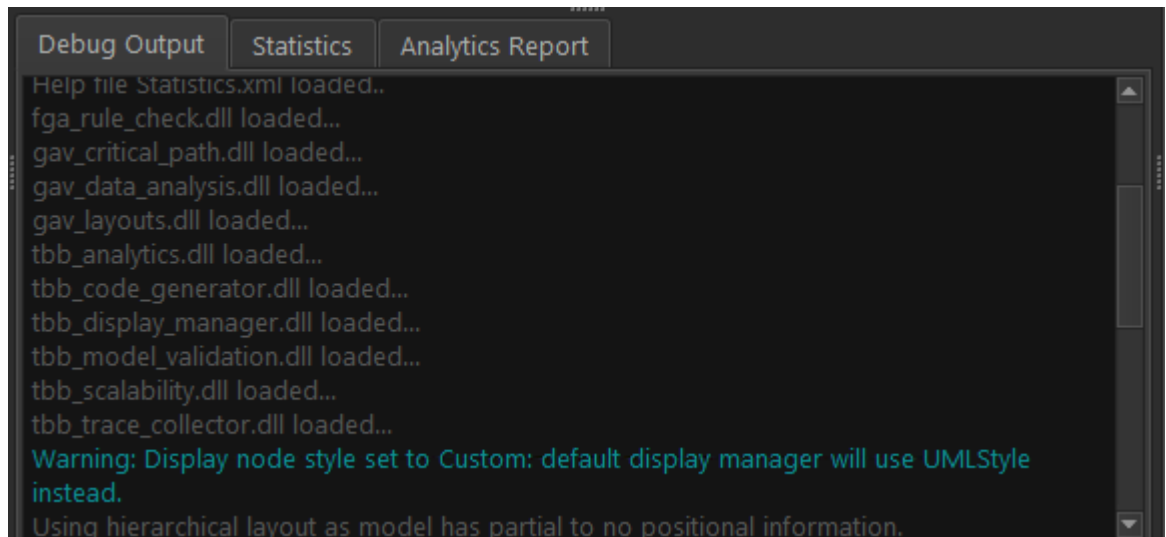
- **Designer Mode** tab: Shows the available node types you can use to construct a graph. This tab might change when new nodes are added to a Intel® oneAPI Threading Building Blocks (oneTBB) flow graph interface. For more information about each node type, use the **What's This?** functionality on the node-type buttons.

After you select a node type to insert into the graph, the mouse cursor takes the shape of the selected node type. You may add as many nodes of the same type as you want to by placing the cursor at a location on the canvas and clicking a mouse. To switch to a different node type, select the node type of interest in the tab.

To exit the **Insert Node** mode, press the **ESC** key if the mouse is placed in the canvas area, or select the **Insert Edge** or **Move Node** modes from the **Toolbar** menu.



- **Debug Output** tab: Displays messages output by the tool. Most of the messages are informational, but warning or error messages may appear. **Warning** messages are green, and **Error** messages are yellow.

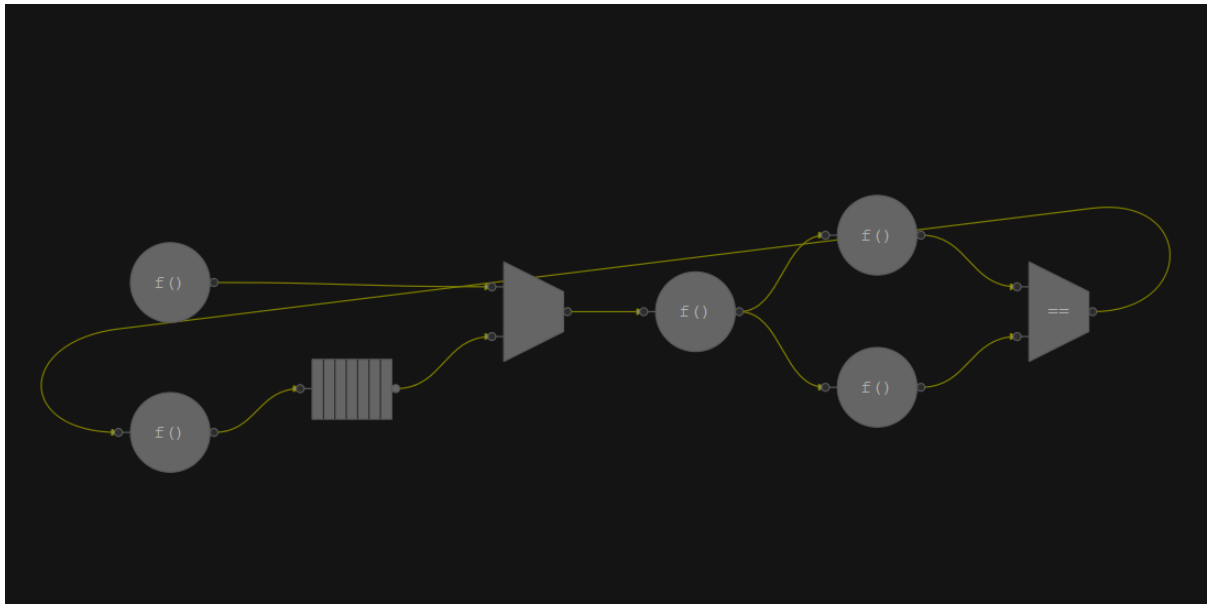


- **Source View** tab: Displays the source mapping of a selected node if symbol resolution information has been captured during the collection.

NOTE Currently, this feature is only supported on Linux* OS. To enable data collection with symbol resolution information, please refer to [Building on a Linux* Operating System](#) and [Collecting Trace Files with an fgtrun.sh Script](#).

Main Canvas

The main canvas shows created or loaded graphs.

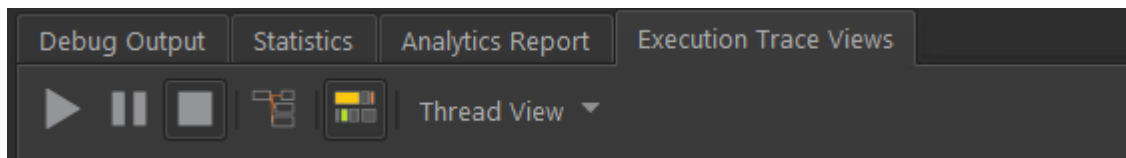


The following controls are available in this area:

- Zoom in or zoom out of this area using the mouse wheel or the zoom buttons in the toolbar.
- Open a context menu with node-specific options by right-clicking a node.
- Select and move a node by dragging it when in the **Move Node** mode.
- Insert new edges between two nodes when in the **Insert Edge** mode.

Charts

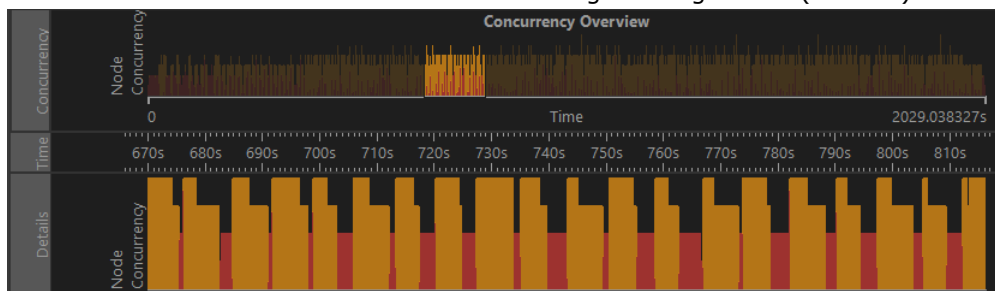
This area displays the task execution trace data available for a graph. You can zoom in or zoom out of all charts in this area to inspect them at various resolutions. Use the mouse wheel or the buttons in the **Zoom** toolbar above the charts to zoom in or zoom out.



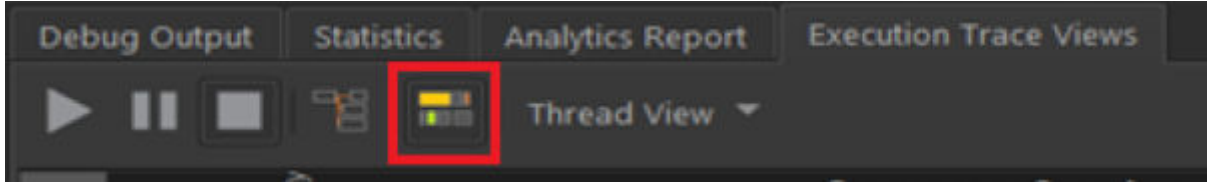
The execution traces are displayed in two different forms:

- **Node Concurrency** display: This form includes two charts.
 - The top chart shows the concurrency for the entire length of the application run.
 - The bottom chart shows the details of the zoomed region in the top chart.

Both charts plot the node concurrency over time. The maximum node concurrency is limited to the maximum number of threads in the Intel® oneAPI Threading Building Blocks (oneTBB) thread pool.



- **Per-Thread Task** display: This chart shows tasks executed by each thread and their duration. To see tasks associated with a particular node, enable the **Show/Hide Selected tasks** button.



You can zoom in or out the data in both views using the specific buttons in the chart toolbar or a mouse wheel. Use the drop-down box in the toolbar to switch between a **Thread View** and a **Node View**.

- In the **Thread View**, the vertical axis is a set of threads that participated in executing the flow graph, and the horizontal axis is time. Tasks with short durations are displayed with a lighter color than those with a longer duration. The lighter color highlights tasks that are small relative to the cost of scheduling the task.



- In the **Node View**, a set of thread timelines is created for each node in the graph. In each set, the vertical axis is a set of threads that participated in executing the flow graph, and the horizontal axis is time.

NOTE

In the **Node View**, a node's set of timelines only displays tasks related to that node, while in the **Thread View**, a single set of timelines shows the tasks related to all nodes.



In some cases, the trace data can contain additional information about the logical core on which a task executes and the data ID it is processing with the help of user-APIs supported by oneTBB and the Flow Graph Analyzer. When this information is available, you can visualize the **Thread View** data and color it by core information or by the data being processed.

Flow Graph Analyzer Workflows

To design flow graph applications using the Intel® oneAPI Threading Building Blocks (oneTBB) library, you need to understand the various node types supported, how to map them to end-user concepts or computational entities, and link them all together to form the flow graph. However, it is difficult to visualize and map such computational blocks when the count of such blocks goes beyond a handful.

To help you solve this visualization problem, the Flow Graph Analyzer tool supports two workflows:

- **Designer workflow** – Enables expressing the relationships between nodes using oneTBB flow graph node types.
- **Analyzer workflow** – Enables capturing and viewing graph topologies and related performance data captured from executed applications.

Designer Workflow

The Flow Graph Analyzer *designer workflow* allows you to describe your graphs visually, set meaningful properties, and generate the flow graph framework code in C++. Before generating the framework code, you can also perform rule-checks to make sure the described graph does not have any potential issues that could lead to incorrect execution or a poorly performing graph. The generated code can be compiled and run without modification in many cases. Sometimes, the generated code may have to be modified to provide meaningful inputs or outputs.

Specifically, the tool supports the following capabilities necessary for visual design of graphs:

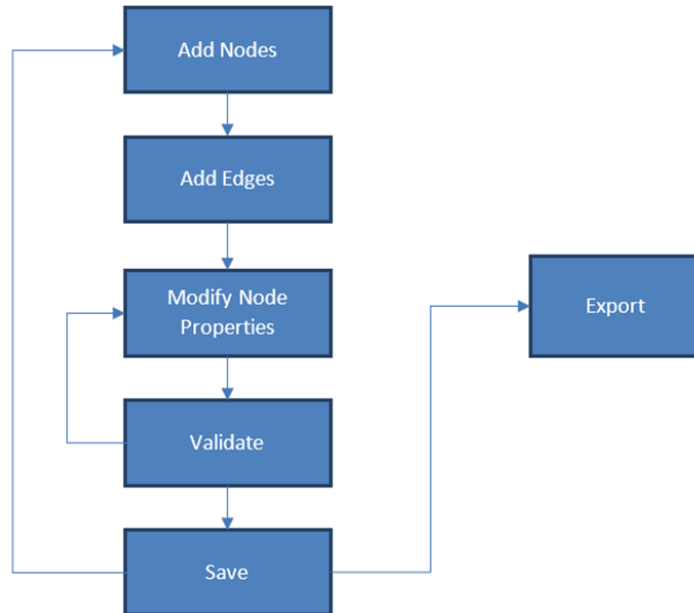
- Choose from a variety of available node types to build a graph.
- Express the explicit relationships with edges.
- Edit properties of these nodes.
- Perform common editing operations.
- Save the described graph and reload it later.

In addition to these basic capabilities, the tool provides the means to:

- Validate each node to ensure that flow graph rules are not broken.
- Perform basic rule checks on a graph to identify potential performance problems.

- Export the graph as a C++ framework code that uses the Intel® oneAPI Threading Building Blocks (oneTBB) flow graph API.
- Export the graph as a PNG image.

This section walks through the design workflow and the capabilities that support it. The following figure shows the simple flow of the design process.



Adding Nodes, Edges, and Ports

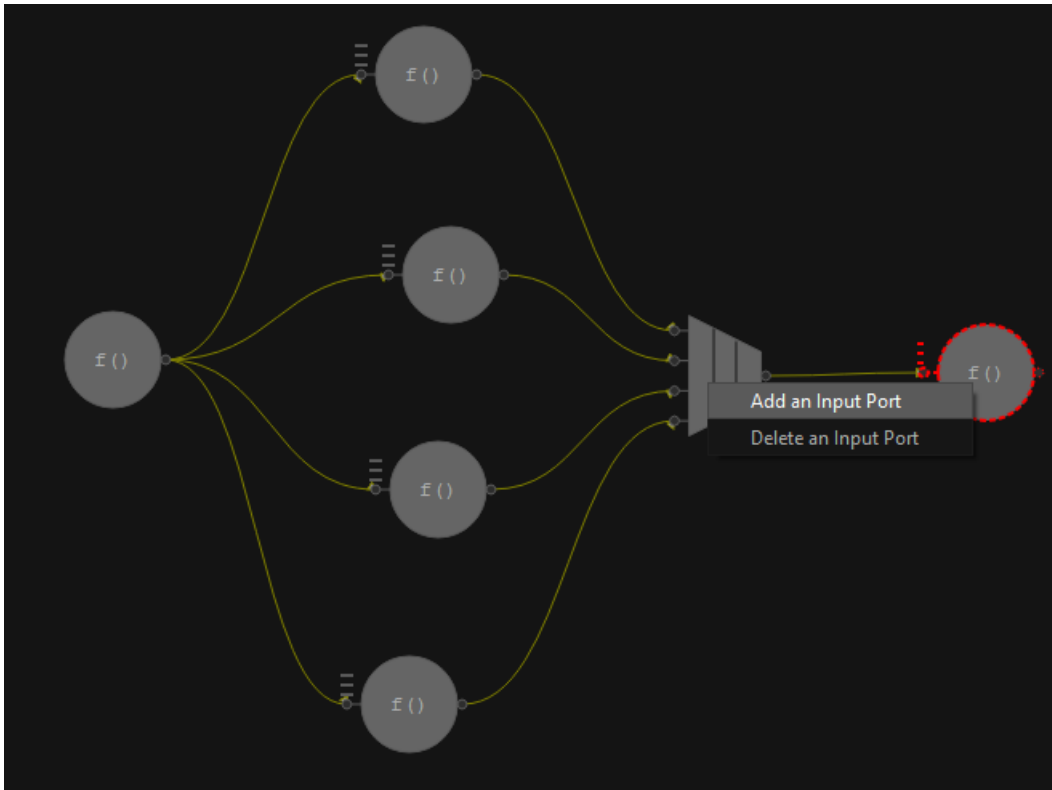
When the tool starts up, you are presented with a blank canvas to which you can add nodes from the list of nodes under **Designer Mode** pane on the left side of the tool.

To add new *nodes*:

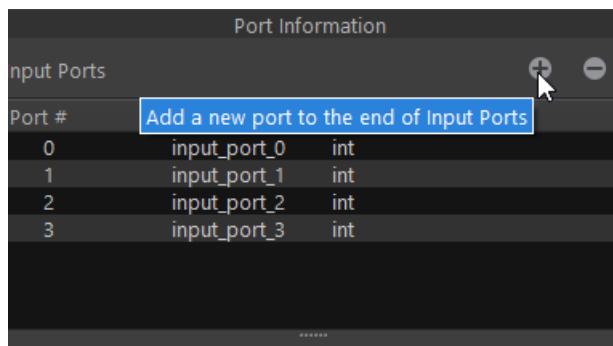
1. Expand a required node group in the **Designer Mode** pane.
2. Drag the required nodes to the canvas.
3. Add the dependencies, or *edges*, between the nodes by clicking an output port of a node and dragging to an input port of another node.

To add new *ports* to a node or delete ports:

1. Right-click a node to open a context menu.
2. Choose **Add an Input/Output Port** or **Delete an Input/Output Port**.



You also can add or remove ports from the **Port Information** tab in the **Reports** area:



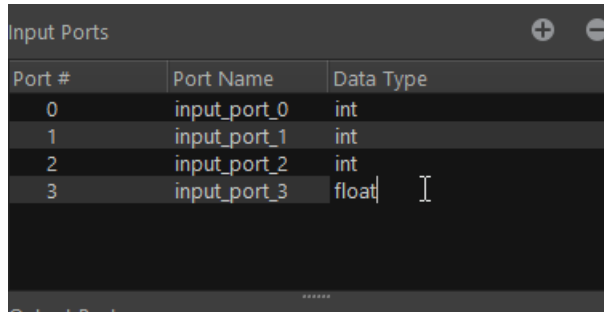
New ports are added to the end of a port list and deleted from the end of the list.

Modifying Node Properties

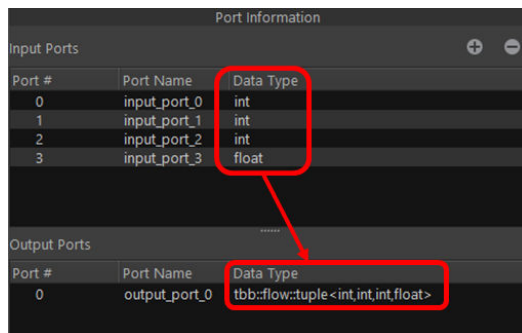
After you add nodes to a graph and connect them with edges, inspect the nodes to ensure the data flowing through the graph has correct types. Some data types are dictated by the Intel® oneAPI Threading Building Blocks (oneTBB) flow graph node types themselves or by the logic the graph represents. Because the data flows through nodes and edges are connected to ports, the data types are managed at the port level. The default node data types are *int* for most ports and *continue_msg* for nodes that expect this type of data.

To edit node properties:

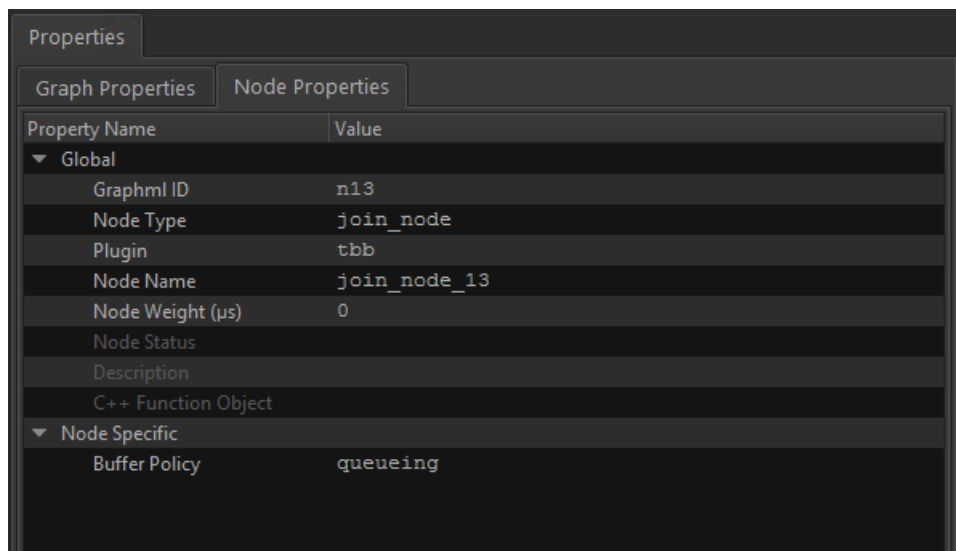
1. Select a node.
2. In the **Port Information** pane of the **Node** tab, change the **Data Type** of a port by selecting its type and editing the field.



For certain nodes, such as a *join_node*, only the input port data types can be modified, and the output data type is automatically generated when you update the input port data types.



3. Select a node on the canvas to see its properties in the **Node** tab. This property pane displays all properties for a given node type. The properties that are not set for the selected node type are shown in a darker color. In the figure below, you can see that the **Description** property is not set for the *join_node*.

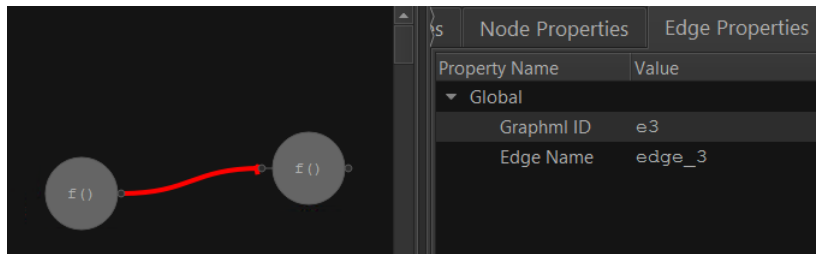


Some of the properties for the nodes are set automatically and tied to the node type. Such properties are not available for editing and the **Node Properties** tab enforces these rules. For example, you cannot change the **Node Type**, but you can edit the **Node Weight** and the **Node Name**.

- The **Node Weight** is a placeholder that indicates the computational complexity of a node. The larger the number, the more computationally intensive the node is with respect to the other nodes in the graph. This number is also used by the C++ code generator to create a busy loop in the empty body that is created for each node. See the [Generating C++ Stubs](#) section for more details.
- The **Node Name** is a unique name automatically assigned to each new node. You can change it to something meaningful. This name is a variable name of the object generated for the node by the C++ code generator.

Viewing Edge Properties

Select an edge on the canvas to see the edge properties. This opens the edge properties in the **Edge** tab, as shown below.

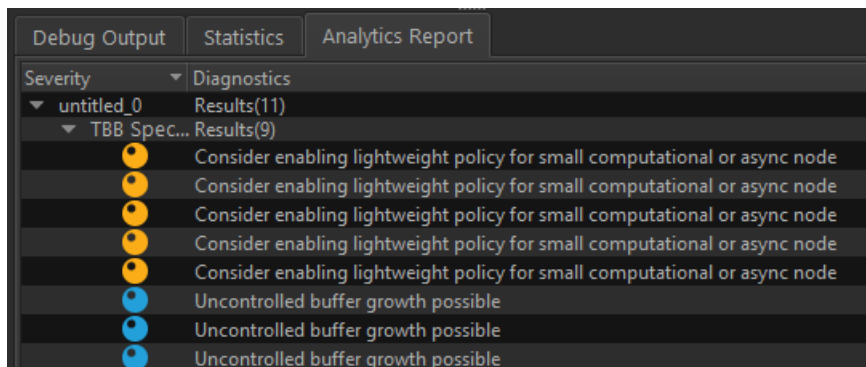


Validating a Graph

After you add nodes to a graph, connect them with edges, and set their properties, you can validate the graph to identify data type mismatches between source and target nodes and highlight other possible issues that may manifest within the described graph topology.

To start a sequence of rule checks on the graph to test various aspects of the graph construction, click the

Graph Rule-Check Analytics icon (🔍) on the toolbar. The results are reported in the **Analytics Report** tab in the **Reports** tab. The following figure shows sample output for a graph.



Click a reported diagnostic to highlight the node that needs to be inspected again. In the case of data mismatches, both the source and target nodes for a given edge are highlighted. Review the **Node** tab in the **Properties** pane to address any changes that are needed.

Saving a Graph to a File

When a graph is in a consistent state and all the major issues are addressed, save it to a file:

- Save the graph to a GraphML* file by clicking the **Save** icon on the toolbar. GraphML format is an open-standard file format for representing graphs.

You can return to the graph later time to modify its topology and data types.

See the [Generating C++ Stubs](#) section for details on how to save the graph to C++ code form.

- Save the graph to a PNG image by clicking the **Generate Image** icon on the toolbar. The image is saved to the same directory where the corresponding GraphML file is saved.

When you load a `.graphml` file for the first time, the tool computes many pieces of information such as performance metrics and graph layout positions. This information computation can be expensive, and it is recommended that you save the graph after this first load. This information is saved in a `.metaxml` file and provides caching benefits that enable the tool to exhibit better performance on subsequent loads of the same `.graphml` file.

NOTE You may be asked if you wish to save the `.graphml` files after load even if no interaction or modification took place. This is due to the tool running the layout algorithm on the graph to display it in an intuitive manner.

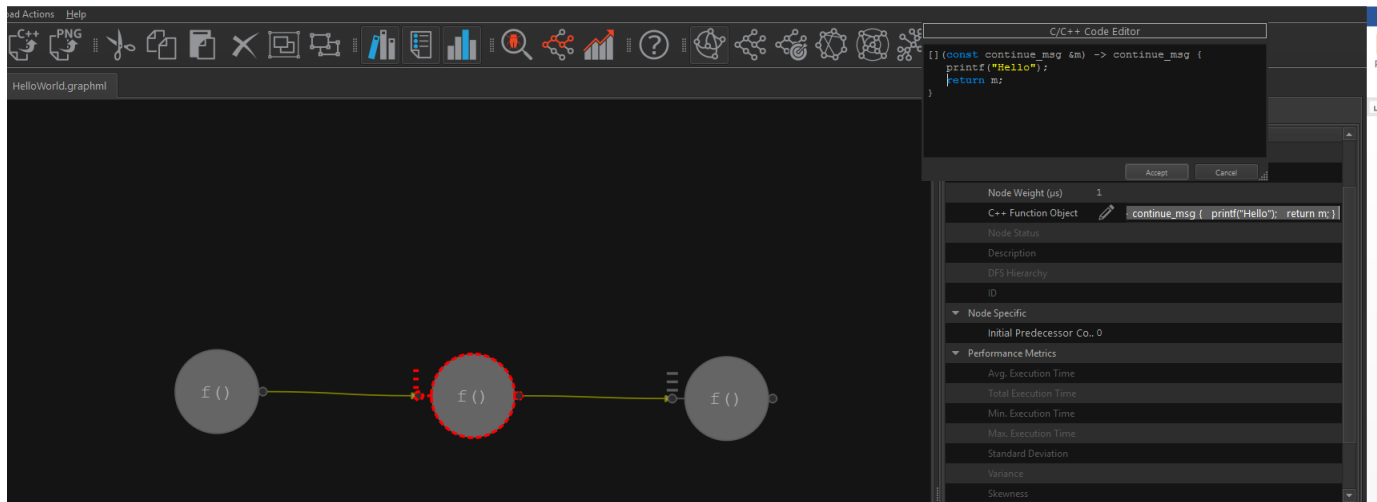
Generating C++ Stubs

To generate stubs for a working C++ application:

1. Create a graph in the canvas as described in the [Adding Nodes, Edges, and Ports](#) section.
2. Save the graph as described in the [Saving a Graph to a File](#) section.
3. Click the **Generate C++** icon on the toolbar to create C++ files.

Generate C++ Stubs for a Hello World Sample

For example, below is a three-node graph you can use to create a Hello World sample. This graph consists of a *source_node* followed by two *continue_node* objects. The first node is named `s0` and the next two nodes are named `c0` and `c1`. All nodes have *continue_msg* objects as their input and/or output types. The body of each node is defined by its **C++ Function Object** field, as shown below.



To generate C++ stubs from this Hello World sample graph:

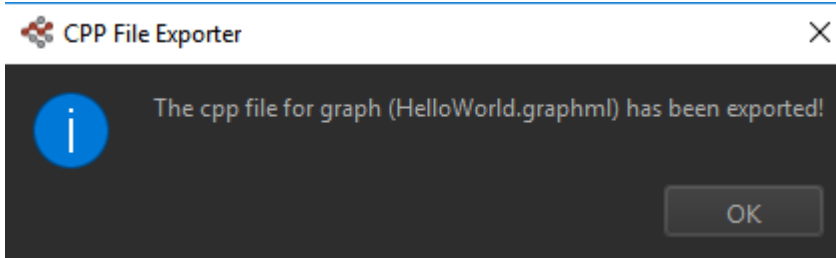
1. Create the sample graph by adding a **source_node** followed by two **continue_node** objects and connect them with edges. Modify the node names in the **Node Properties**: name the **source_node** as `s0` and the next two **continue_node** objects as `c0` and `c1`.
2. Set the following properties to the nodes:

Node Name	Input Port Type	Output Port Type	C++ Function Object
s0	None o u r c e — n o d e	continue_msg	<pre> [] (continue_msg &c) -> bool { static bool done = false; if (!done) { done = true; return true; } else { return false; } } </pre>
c0	continue_msg o n t i n u e — n o d e	continue_msg	<pre> [] (const continue_msg &m) -> continue_msg { printf("Hello"); return m; } </pre>
c1	continue_msg o n t i n u e — n o d e	continue_msg	<pre> [] (const continue_msg &m) -> continue_msg { printf(" World! \n"); return m; } </pre>

3. Click the **Save** icon on the toolbar to save this graph as HelloWorld.graphml.

4. Click the **Generate C++** icon on the toolbar to generate the C++ stubs.

The generation of the stub files should be reported as successful:



The result of C++ code generation is one file located in the same directory where the GraphML* file is last saved. The file name generated is `HelloWorld_stubs.cpp`. It should contain the following code:

```
//
// Automatically generated by Flow Graph Analyzer:
// C++ Code Generator Plugin version XYZ
//

#define TBB_PREVIEW_FLOW_GRAPH_NODES 1
#include "tbb/flow_graph.h"
#include "tbb/tick_count.h"
#include "tbb/atomic.h"
#include <cstdlib>

using namespace std;
using namespace tbb;
using namespace tbb::flow;

size_t key_from_message(char *k) {
    return reinterpret_cast<size_t>(k);
}

template<typename T>
size_t key_from_message(const T &k) {
    return static_cast<size_t>(k);
}

static void spin_for( double weight = 0.0 ) {
    if ( weight > 0.0 ) {
        tick_count t1, t0 = tick_count::now();
        const double weight_multiple = 1e-6;
        const double end_time = weight_multiple * weight;
        do {
            t1 = tick_count::now();
        } while ( (t1-t0).seconds() < end_time );
    }
}

int build_and_run_HelloWorld_g0() {
    graph HelloWorld_g0;

    source_node< continue_msg > s0( HelloWorld_g0,
    [](continue_msg &c) -> bool {
        static bool done = false;
        if (!done) {
            done = true;
            return true;
        } else {
            return false;
        }
    });
}
```

```

    }
    }, false);

    continue_node< continue_msg > c0( HelloWorld_g0, 0,
    [](const continue_msg &m) -> continue_msg {
        printf("Hello");
        return m;
    });

    continue_node< continue_msg > c1( HelloWorld_g0, 0,
    [](const continue_msg &m) -> continue_msg {
        printf(" World!\n");
        return m;
    });

    #if TBB_PREVIEW_FLOW_GRAPH_TRACE
    HelloWorld_g0.set_name("HelloWorld_g0");
    s0.set_name("s0");
    c0.set_name("c0");
    c1.set_name("c1");
    #endif

    make_edge( s0, c0 );
    make_edge( c0, c1 );

    s0.activate();
    HelloWorld_g0.wait_for_all();
    return 0;
}

int main(int argc, char *argv[]) {
    return build_and_run_HelloWorld_g0();
}

```

In the code above, note the `s0`, `c0`, and `c1` nodes reflect the properties described in the previous table.

If you have the paths to the Intel® oneAPI Threading Building Blocks (oneTBB) library set up in your environment, you can build this application from a command prompt:

- On a Windows* system, run the following command from a Microsoft Visual Studio* command prompt:

```
cl /EHsc HelloWorld_stubs.cpp tbb.lib
```

- On a Linux* system, run the following command:


```
g++ -std=c++11 HelloWorld_stubs.cpp -ltbb
```

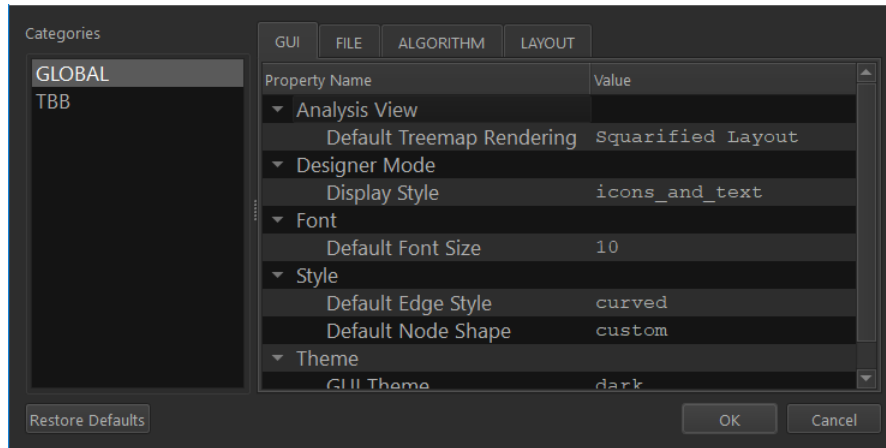
In addition, Flow Graph Analyzer allows you to control execution policies for nodes, such as setting lightweight for computational nodes and asynchronous nodes. If you set lightweight policies for any node, the current code generator generates stubs for oneTBB.

See more samples demonstrating this feature in the `samples/code_generation` directory.

Preferences

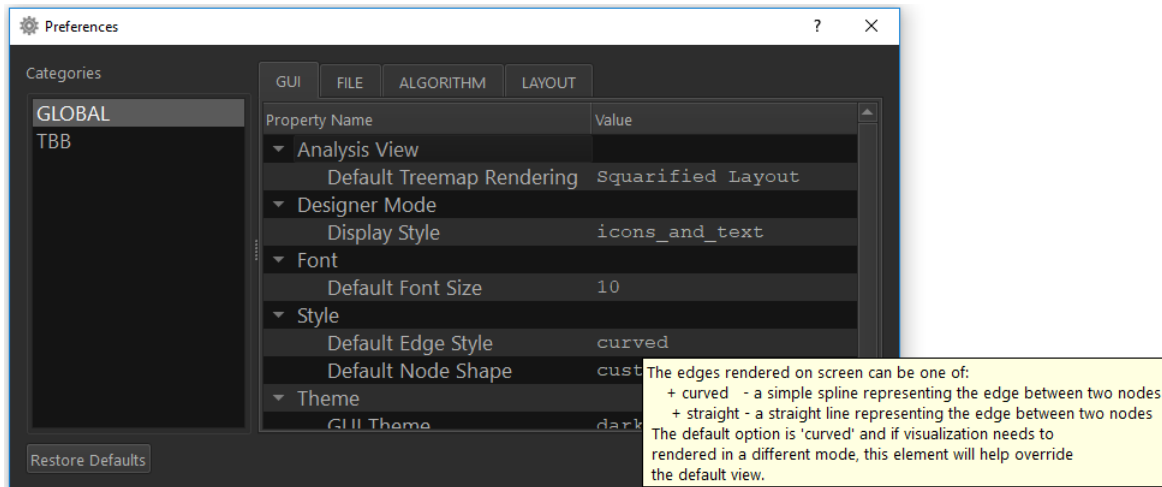
Use the **Preferences** dialog box to set your preferred global values for certain properties.

Go to the **Edit > Preferences** or click  on the toolbar. You should see the **Preferences** window:



To set a preference, click a property value and change it. The Flow Graph Analyzer applies your preferences to the entire session and restores the preferences after shutdown and restart.

To get more information about a preference property, hover your mouse over the property to enable a tooltip that contains more information on the property:

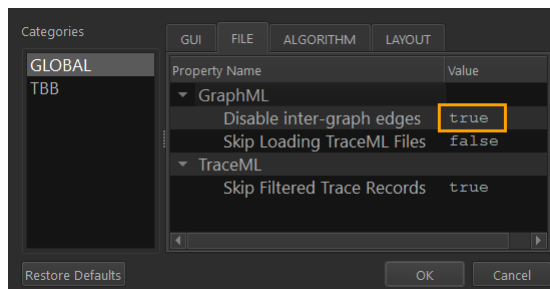


Some of the configurable preferences are:

Property	Description
Node Shape	This is a global preference in the Style group of the GUI tab. Possible values are box , circle , basic , uml , and custom . The default is the custom node shape.
GUI Theme	This is a global preference in the Theme group of the GUI tab. Possible themes are dark and light. The default is dark.
NOTE The Flow Graph Analyzer applies a theme change only after restart.	

Property**Description****Disable inter-graph edges**

This is a global preference in the **GraphML** group of the **File** tab. The Flow Graph Analyzer 2019 Update 4 or higher supports viewing GraphML* files with inter-graph edges. This is disabled by default (set to **true**).

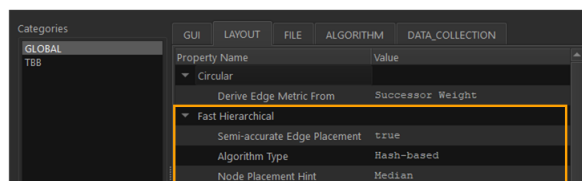
**NOTE**

Even though cross-graph edges are strongly discouraged in Intel® oneAPI Threading Building Blocks (oneTBB) documentation, some use-cases may require them.

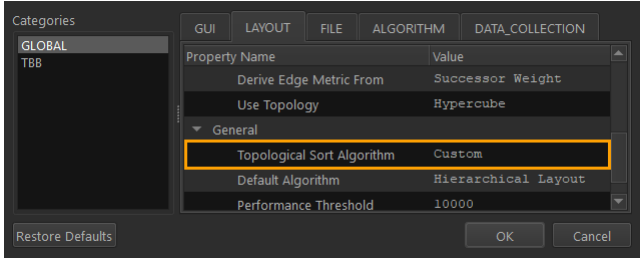
Fast Hierarchical (preferences group)

This is a global preferences group in the **File** tab. The new fast hierarchical layout algorithm has been introduced in Flow Graph Analyzer 2019 Update 5. It allows you to set the following preferences:

- Render graphs with **semi-accurate edge placement**.
- Render graphs with different **node placement** with respect to its children such as **Median**, **Average**, or **Minimum** distance.
- Choose an **algorithm type**. There are three different algorithms for this layout that can be applied, and each has its own performance characteristics:
 - **Simple** algorithm type sacrifices accuracy for speed.
 - **Hash-based** type honors node placement and hash-based **depth-first search (DFS)** helps with better visual quality.

**Topological Sort Algorithm**

This is a global preference in the **General** group of the **Layout** tab. Choose between two different DFS algorithms for a topological sort of a graph:

Property	Description
	<ul style="list-style-type: none"> Custom sort algorithm is optimized for topologies that are frequently encountered in oneTBB and OpenMP*. Boost library implementation of DFS. 

Scalability Analysis

When designing a parallel application, you need to know if the application performance continues to increase when you add more threads or tapers off. However, in a complex application, it is often not obvious what the overall parallelism of the application is.

Use the scalability analysis plugin to:

- Estimate the application parallelism provided by the topology of the graph.
- Estimate the inherent application parallelism provided within nodes that have unlimited concurrency.
- Identify contributing factors to overall parallelism.

The scalability analysis plugin allows you to run a graph with a varying number of threads and provides speedup information of the graph with respect to running the graph serially. You can specify any configuration of the graph to help design and analyze the graph.

Activating the Graph

To run the analysis, the graph must contain at least one *source_node* for activation. The *source_node* pushes the data items through the graph.

The plugin internally ensures that valid data types flow through the graph, so even if a rule check on the data types fails, the scalability analysis still runs. Therefore, selecting and connecting nodes on the canvas should be enough; there is no further need to edit the input and output data types to ensure they match.

NOTE

You may need to ensure input or output types for other plugins, like the code generator, to ensure accurate code generation.

Scalability Analysis Prerequisites

Before running the scalability analysis, set the following:

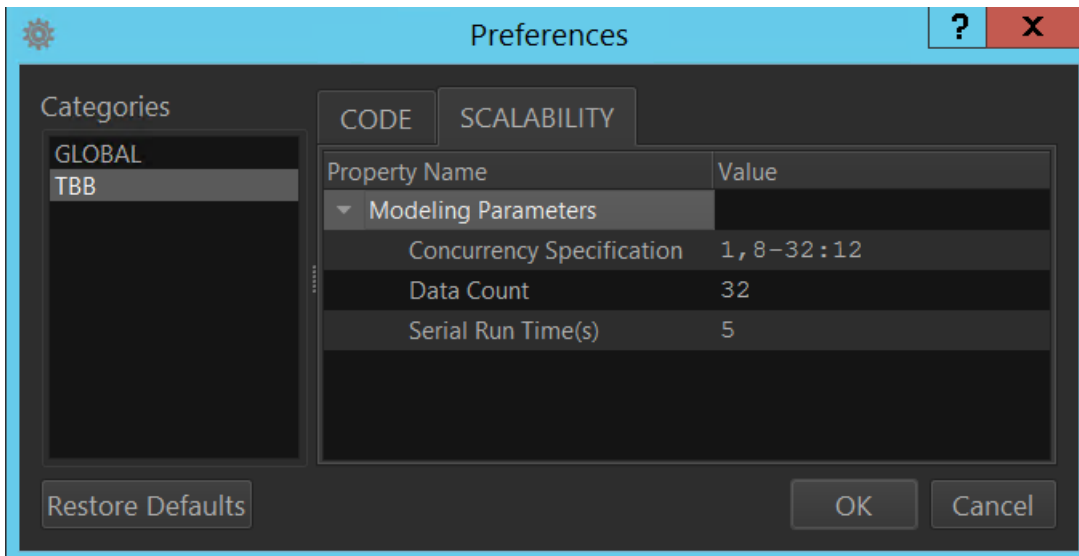
- A concurrency specification
- A data count value
- Node weights

Setting Concurrency Specification

The concurrency specification is a list of the number of threads on which the graph is run. To set the concurrency:

1. Open the **Preferences** window.

2. Go to the **Scalability > TBB** category.
3. Set the list in the **Concurrency Specification** field in the $1, a-b:n$ format, where:
 - 1 is the serial run. Because the speedup is estimated with respect to a serial run of a graph, by default, all graphs are first run serially before running on multiple threads. The addition of 1 is merely symbolic and informational, as its omission has no effect on the list.
 - a is the starting number of multiple threads.
 - b is the ending number of multiple threads.
 - n is the step size.



The default end value is the number of cores on the system and the default step size is a quarter of this value. The end value is always included in the list, even if it is not a direct multiple of the step size.

As an example, on a 32-core system, the default is $1, 8-32:8$. This results in the list 1, 8, 16, 24, 32, as shown below.

Debug Output	Statistics	Analytics Report
Graph Name	Graph	Threads
untitled...	Results(5)	
	Scalability projection	1
	Scalability projection	8
	Scalability projection	16
	Scalability projection	24
	Scalability projection	32

Other supported formats are a, b, c or only a if you want to see the speedup for only one set of threads.

You can also combine formats. The following shows some valid concurrency specifications:

- Range without a step size: 16-64 yields 1, 16, 32, 48, 64
- Range with a step size: 16-64:8 yields 1, 16, 24, 32, 40, 48, 56, 64
- Single value: 16 yields 1, 16
- List of values: 16, 32, 64 yields 1, 16, 32, 64
- Mix of range and list: 16-64, 40, 48, 56 yields 1, 16, 32, 40, 48, 56, 64

Setting Data Count

Data Count is the number of data items generated and pushed through a graph.

For graphs where potential parallelism is apparent from topology, pushing a single item through the graph is enough to explore the parallelism of the graph.

However, for cases where the parallelism is not readily apparent, as with a node with an unlimited concurrency, pushing a single item through the node is not enough to explore parallelism. Therefore, to ensure the graph is saturated, you need to set data count:

1. Open the **Preferences** windows.
2. Go to the **TBB** category, **Scalability** tab.
3. Change the default number of data items to the number of cores available on the system in the **Data Count** field, based on the topology of the graph and the type of nodes in the graph.

Setting Node Weight

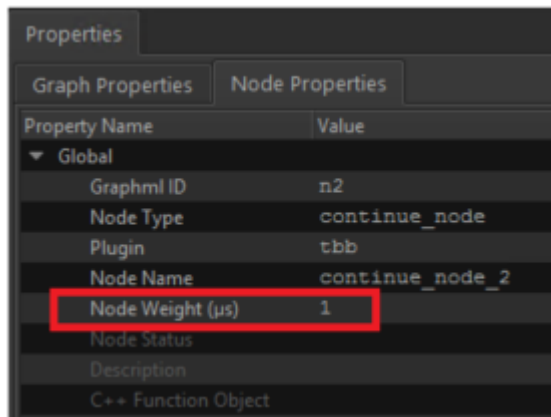
The weight determines the simulated amount of time the node spins or does active work per data item passed to the node. The default unit for the weight is microseconds. For example, per the default unit of microseconds, a weight value of 1000 makes a node spin for half a second.

NOTE

- If the specified weight is high relative to the unit, the computation might run for a longer time.
- If the graph has an associated trace, the unit of the weight is overwritten by the unit specified in the trace.

To edit a node weight:

1. Select a node in the canvas.
2. Click the **Node Weight** field value in the **Node** tab of the **Properties** pane and enter your value.



The weight value matters only for nodes with a body, such as *source_node*, *continue_node*, *function_node*, *multifunction_node*, *async_node*, and *tag_matching_join_node*. Other nodes that simply assist in the topology of the graph do not use the weight value. For example, specifying a weight value for a *broadcast_node* has no effect on the node.

Consider the following:

- To prevent long runtimes, the scalability analysis scales all runs with total serial times beyond a certain threshold. The current default threshold is 5 seconds. To modify this value:
 1. Go to the **Edit > Preferences**.
 2. Open the **TBB** category, **Scalability** tab.
 3. Modify a value in the **Serial Run Time(s)** field.

The total serial runtime also considers the number of data items passed through the graph. For example, a graph with a serial runtime of 10 seconds and 4 data items has a total serial runtime of 40 seconds.

- To ensure node weights are not scaled into regions that make the overhead dominant, the analysis uses original weights and does not implement scaling if no node has a weight of 100 microseconds after scaling.

- If a graph contains nodes with performance that could benefit from the use of the Intel® oneAPI Threading Building Blocks lightweight policy, the analysis activates the lightweight policy for the recommended nodes and lists the possible improvement in the results.

Running the Scalability Analysis

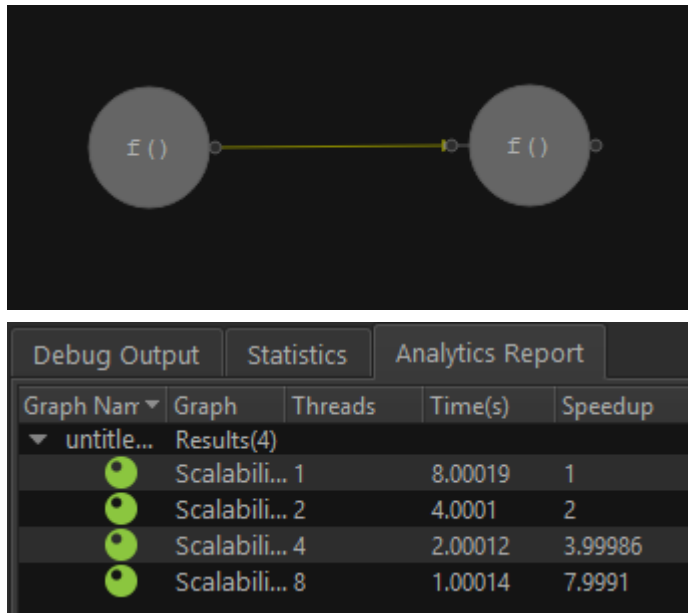
After you set the concurrency specification, data count, and node weights, you can run the scalability

analysis. To run the analysis, click the **Scalability** icon on the toolbar:



Exploring the Parallelism in a Concurrent Node

This example explores the parallelism inherent in a node with unlimited concurrency. The node used is a *function_node*. A *source_node* is connected to the *function_node*, as shown below, and eight items are pushed through to the *function_node* from the *source_node*. The *function_node* has a weight of 1s (weight = 1e6). To ensure there are only timing results from the *function_node*, the *source_node* has a comparatively negligible weight of 1e-6s (weight=1). The concurrency specification used is 1, 2, 4, 8.



The results are the following:

- For a serial execution with only 1 thread, the total time is 8s as the same thread evaluates the tasks one after the other.
- With 2 threads and 2 overlapping tasks, the total time is 4s.
- With 4 threads and 8 tasks, the total time is 2s.
- With 8 threads and 8 tasks, all tasks overlap, giving a total time of 1s.

Showing Non-Parallel Nature of a Serial Node





You can use the Scalability analysis plugin to identify nodes that have no parallelism or to tell when an unlimited node is running serially.

For this example, use a *source_node* and a *function_node* connected to it. Set the following:

1. Select the *function_node*.
2. In the **Node Concurrency** field, enter 1 or `serial`. This makes the node run serially.
3. In the **Node Weight** field, enter 1.
4. Select the *source_node*.
5. In the **Node Weight** field, enter 1e-6.

6. Go to the **Edit > Properties > TBB**.
7. Set the concurrency specification to 1, 2, 4, 8 and click **OK**.
8. Run the scalability analysis.

You will get the following results:

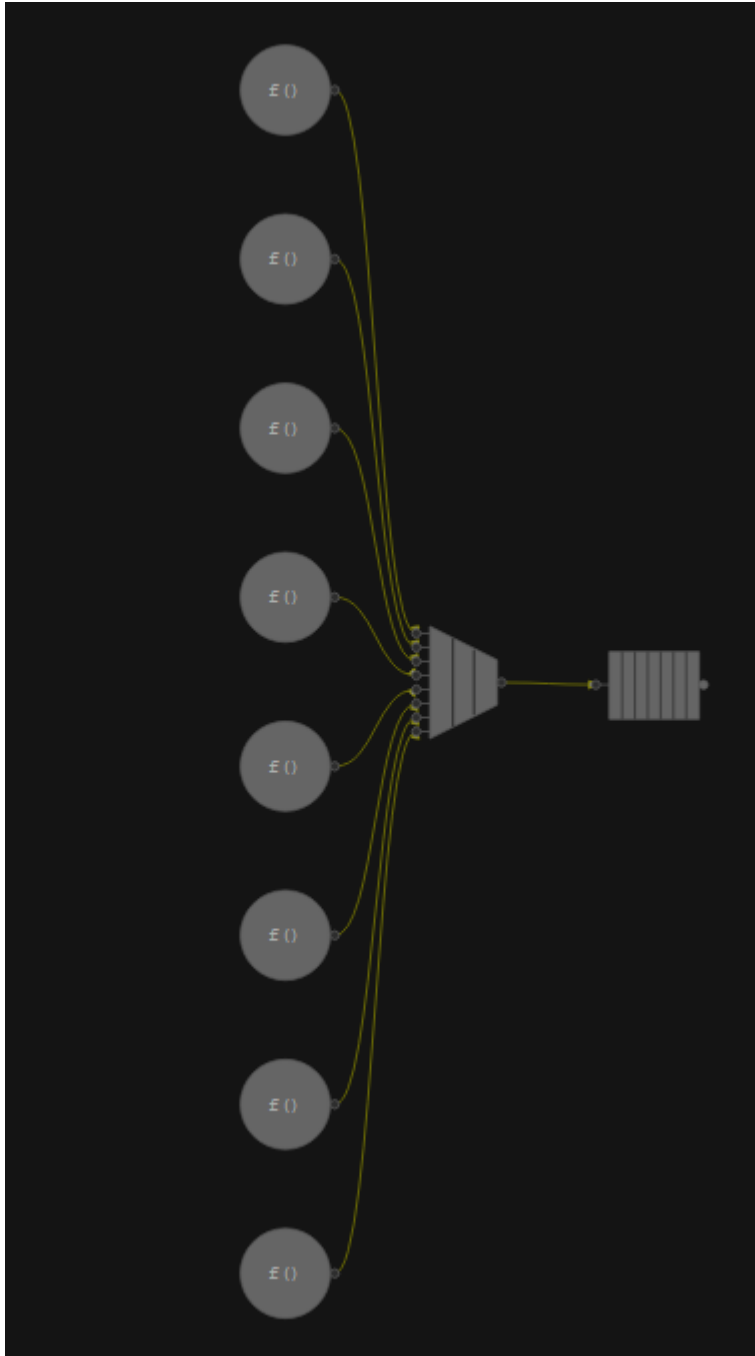
Debug Output	Statistics	Analytics Report			
Graph Name	Graph	Threads	Time(s)	Speedup	
▼ untitled_0	Results(4)				
	Scalability projection	1	8.00017	1	
	Scalability projection	2	8.00009	1.00001	
	Scalability projection	4	8.00007	1.00001	
	Scalability projection	8	8.00007	1.00001	

As expected, time is 8s, regardless of the number of threads used.





You can get the same results if you set the weight of the *source_node* to 1s and the weight to the *function_node* to a relatively negligible value. This is because a *source_node* executes its body serially.

Explore Parallelism Provided by the Topology of a Graph

This example explores the parallelism provided by the topology of a graph. To make the results as predictable as possible, use a graph that is explicitly parallel, as shown below:



Because the *source_node* is serial, there is no parallelism provided from within the node. This ensures all parallelism observed is provided by the topology of the graph. Eight *source_nodes* are connected to a *join_node* and then to a *queue_node*. In this graph, only the *source_nodes* do useful work. Because the parallelism is solely from the topology of the graph, one item per *source_node* is enough to through the graph. Each *source_node* has a weight of 1s(1e6). The results of scalability analysis of the graph are shown below.

Graph Name	Graph	Threads	Time(s)	Speedup
untitled_0	Results(4)			
	Scalability projection	1	8.00177	1
	Scalability projection	2	4.00031	2.00029
	Scalability projection	4	2.00035	4.00017
	Scalability projection	8	1.0005	7.99775







The speedup is directly proportional to the number of threads.

Understanding Analysis Color Codes

The results from the scalability analysis runs are color-coded:

- *Green* dots are results with the number of threads either equal to or less than the number of cores on the system.
- *Blue* dots are results with the number of threads greater than the number of cores.

For example, the following image shows the result for a concurrency specification of 1, 4, 8, 32, 48, 64 on a 32-core system.

Graph Name	Graph	Threads	Time(s)	Speedup
untitled_0	Results(6)			
	Scalability projection	1	8.00019	1
	Scalability projection	4	2.00016	3.99977
	Scalability projection	8	1.00011	7.99933
	Scalability projection	32	1.0001	7.99942
	Scalability projection	48	1.0001	7.99941
	Scalability projection	64	1.0001	7.99936

Results for runs with less than 32 threads are coded in green, while those with more than 32 threads are coded in blue.

Collecting Traces from Applications

This section explains how to collect traces from an application that uses the Intel® oneAPI Threading Building Blocks (oneTBB) flow graph interfaces.

Prerequisites

You need the following to collect traces from an application:

- An application that uses the oneTBB library flow graph interface
- The Flow Graph Collector library

Check the links in the [Additional Resources](#) section if you are missing any of these prerequisites.

Simple Sample Application

This section uses the sample code below as a running example. Assume this code is contained in a file `example.cpp`. You can also use your own application or sample instead of this simple example.

```
#include "tbb/flow_graph.h"
#include <iostream>
using namespace std;
using namespace tbb::flow;
int main() {
```

```
graph g;
continue_node< continue_msg> hello( g,
    []( const continue_msg &) {
        cout << "Hello";
    }
);
continue_node< continue_msg> world( g,
    []( const continue_msg &) {
        cout << " World\n";
    }
);
make_edge(hello, world);
hello.try_put(continue_msg());
g.wait_for_all();
return 0;
}
```

Building an Application for Trace Collection

To build an application enabled for trace collection:

1. Define the `TBB_USE_THREADING_TOOLS` macro and link against the `tbb` library. This macro activates the required instrumentations in the `flow_graph.h` header. The Intel® oneAPI Threading Building Blocks (oneTBB) library supports flow graph and algorithm profiling. All features other than `set_name` extensions are available as non-preview features.
2. Compile using oneTBB.

Refer to the OS-specific topics for instructions on how to build an application depending on your operating system.

Building an Application on Windows* OS

Building from a Microsoft Visual Studio* Command Prompt

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

After you open a Microsoft Visual Studio* command prompt and set up the proper paths for using the Intel® oneAPI Threading Building Blocks (oneTBB) library, use the following command line to build a Release executable for a running example:

```
cl /EHsc /DTBB_USE_THREADING_TOOLS example.cpp tbb.lib
```

This command defines the required macro and link the application against the appropriate `tbb` library, based on the oneTBB version you use.

Building from a Microsoft Visual Studio* IDE

To build a Release configuration of your application within a Microsoft Visual Studio* IDE, you must change your project to define the `TBB_PREVIEW_FLOW_GRAPH_TRACE/TBB_USE_THREADING_TOOLS` macro and link against the `tbb_preview.lib/tbb.lib`, as shown below for the Microsoft Visual Studio* 2015 IDE based on the oneTBB version you use.

1. Open the **Project Properties** dialog box, and select **Configuration Properties > C/C++ > Command Line**. In the **Additional Options** textbox, enter `/DTBB_USE_THREADING_TOOLS`.
2. Select **Configuration Properties > Linker > Input**.
3. In the **Additional Dependencies** field:

- For a Release build: Enter `tbb.lib`.
- For a Debug build: Enter `tbb_debug.lib`.

Building an Application on Linux* OS

Use the following command line to build an executable for the running example:

```
icpc -std=c++11 -DTBB_USE_THREADING_TOOLS example.cpp -ltbb
```

These command lines define the `TBB_USE_THREADING_TOOLS` macro and also link the application against the `tbb.lib` library. `-std=c++11` is present because the running example uses lambda expressions, which are a C++11 feature.

To map nodes to source code, use `-g`, `-DTBB_PREVIEW_FLOW_GRAPH_TRACE`, and `-DTBB_USE_THREADING_TOOLS` flags when building the application:

```
icpc -g -std=c++11 -DTBB_USE_THREADING_TOOLS -DTBB_PREVIEW_FLOW_GRAPH_TRACE example.cpp -ltbb_preview
```

Building an Application on macOS*

Use the following command line to build an executable for the running example:

```
clang++ -std=c++11 -DTBB_USE_THREADING_TOOLS example.cpp -ltbb
```

The command line define the `TBB_USE_THREADING_TOOLS` macro and also link the application against `tbb.lib` library of Intel® oneAPI Threading Building Blocks. `-std=c++11` is present because the running example uses lambda expressions, which are a C++11 feature.

Collecting Trace Files

While executing, your application collects trace files only if there is an appropriate collector library at the locations specified by the `INTEL_LIBITNOTIFY32` or `INTEL_LIBITNOTIFY64` environment variables. The `INTEL_LIBITNOTIFY32` path is searched by 32-bit executables and the `INTEL_LIBITNOTIFY64` path is searched by 64-bit executables.

To collect and convert trace files to use them with the Flow Graph Analyzer, chose one of the following options:

- Collect traces by starting your application from the Flow Graph Analyzer GUI.
- Set up your environment so a trace is collected when the application is started outside of the GUI.

NOTE

Both approaches require you to build the application following the steps from [Building an Application](#) section for your operating system.

Collect Traces In the Flow Graph Analyzer GUI

To run an existing Intel® oneAPI Threading Building Blocks (oneTBB) application and collect execution trace information for analysis, you can launch the *TBB trace collector* feature from the Flow Graph Analyzer GUI as follows, assuming the paths for the oneTBB libraries are set up (see Release Notes and Known Issues for limitations):

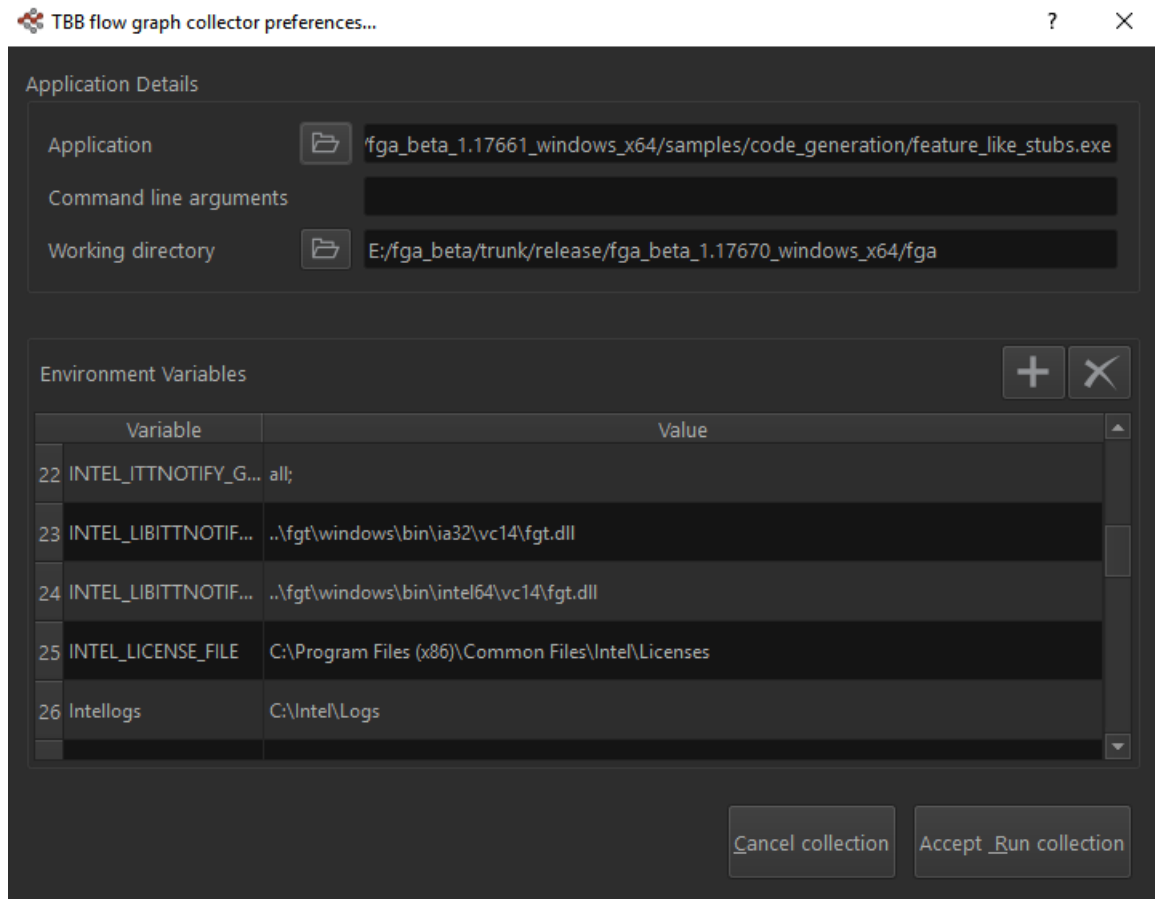
1. Run the trace collector using one of the following options:

- Go to the **Offload Actions > Run and collect traces** menu option.
-



Click the **Run and Collect Traces** icon on the toolbar.

The collector **Preferences** window opens, which lets you specify the application to run:



2. In the collector **Preferences** window:

- Specify an application to run in the **Application** field.
- Optional: Change the **Working Directory** value if required. The default is the Flow Graph Analyzer directory.
- View and set other environment variables, including `INTEL_LIBITTTNOTIFY32` and `INTEL_LIBITTTNOTIFY64`, using the **Environment Variables** pane of the dialog box.

The environment variables for running trace collection have default settings if you did not change their values in the environment from which the Flow Graph Analyzer is launched. The inherited values are used if the environment variables are set in the environment.

OS	Environment Variable	Default Value	Description
Windows*	INTEL_LIBITTTNOTIFY32	..\fgt\windows\bin\ia32\vc14.1\fgt.dll	32-bit collector library
	INTEL_LIBITTTNOTIFY64	..\fgt\windows\bin\intel64\vc14.1\fgt.dll	64-bit collector library
	INTEL_ITTNOTIFY_GROUPS	all;	Trace events from all groups

OS	Environment Variable	Default Value	Description
Linux*	INTEL_LIBITNOTIFY 32	../fgt/linux/lib/ ia32/ cc4.8_libc2.19_ker nel3.13.0/ libfgt.so	32-bit collector library
	INTEL_LIBITNOTIFY 64	../fgt/linux/lib/ intel64/ cc4.8_libc2.19_ker nel3.13.0/ libfgt.so	64-bit collector library
	INTEL_ITNOTIFY_GROUPS	all;	Trace events from all groups
macOS*	INTEL_LIBITNOTIFY 32	../fgt/macos/lib/ ia32/ osx10.12.6_kernel1 6.7.0/libfgt.dylib	32-bit collector library
	INTEL_LIBITNOTIFY 64	../fgt/macos/lib/ intel64/ osx10.12.6_kernel1 6.7.0/libfgt.dylib	64-bit collector library
	INTEL_ITNOTIFY_GROUPS	all;	Trace events from all groups

- Click the **Accept Run collection** button. If the application is executed correctly, the trace files are converted to a GraphML* format. The output file is stored in the working directory with a name based on the executable name and the time of the trace collection run.
- To examine the trace file, load the GraphML* file into the Flow Graph Analyzer GUI manually.

Collect Traces Outside the Flow Graph Analyzer GUI

To collect traces outside the Flow Graph Analyzer GUI:

- If you launch your application from a Windows* command prompt or a Linux* terminal, the simplest approach is using the `fgt_run` script to run your application.
- If you cannot launch your application from a prompt or terminal or you want to launch it from within an IDE, you must manually perform the steps performed by the `fgt_run` script.

In either case, you must update your `PATH` environment variable to add the paths to essential tools, as described below, and set the `FGT_ROOT` variable.

This section assumes the full path to your Flow Graph Analyzer installation is `<advisor-install-dir>\fga`. The version of your Visual Studio* compiler is `<vc_version>` with possible values of `vc12`, `vc14`, and `vc14.1`.

OS	Environment Variable	Value	Description
Windows*	FGT_ROOT	<advisor-install-dir>\fga\fgt	The path to the Flow Graph Collector installation

OS	Environment Variable	Value	Description
Linux*	PATH	%FGT_ROOT%\windows \bin;%FGT_ROOT% \windows\bin \ia32\<vc version>;%FGT_ROOT% \windows\bin \intel64\<vc version>;%PATH%	The path must include paths to fgtrun.bat and fgt2xml.exe.
	FGT_ROOT	<advisor-install-dir>/fga/fgt	The path to the Flow Graph Collector installation
	PATH	\${FGT_ROOT}/linux/ bin:\${FGT_ROOT}/ linux/bin/ia32/ cc4.8_libc2.19_kern el3.13.0:\${ {FGT_ROOT}/ linux/bin/intel64/ cc4.8_libc2.19_kern el3.13.0:\${PATH}	The path must include paths to fgtrun.sh, fgtrun.csh, and fgt2xml.
macOS*	FGT_ROOT	<advisor-install-dir>/fga/fgt	The path to the Flow Graph Collector installation
	PATH	\${FGT_ROOT}/macos/ bin:\${FGT_ROOT}/ macos/bin/ia32/ osx10.12.6_kernel16 .7.0:\${FGT_ROOT}/ macos/bin/intel64/ osx10.12.6_kernel16 .7.0:\${PATH}	The path must include paths to fgtrun.sh, fgtrun.csh, and fgt2xml.

Collecting Trace Files with fgtrun Script

Use the fgtrun script to collect the trace information from your application.

Before running the script, you must set the FGT_ROOT variable to <advisor-install-dir>\fga\fgt as described in the [Collecting Traces Outside the Flow Graph Analyzer GUI](#). The fgtrun script sets the paths necessary to execute your application and generate the GraphML* and TraceML* files that can be loaded into the Flow Graph Analyzer for visualization.

The following table lists the directories in which the scripts are located on a given system.

Operating System	Version	Location	Example Use
Windows*	fgtrun.bat	%FGT_ROOT%\windows \bin	fgtrun.bat <app- binary-name> [<binary-args>] [--

Operating System	Version	Location	Example Use
Linux*	fgtrun.sh	\${FGT_ROOT}/ linux/bin	ia32/ --intel64] [--vc12/ --vc14/ -- vc14.1] [--xml] fgtrun.sh <app- binary-name> [<binary-args>] [-- ia32/ --intel64] [--omp] [--xml]
macOS*	fgtrun.sh	\${FGT_ROOT}/ macos/bin	fgtrun.sh <app- binary-name> [<binary-args>] [-- ia32/ --intel64] [--omp] [--xml]

The `fgtrun` script tries to automatically detect the architecture and C/C++ runtime version (Windows* OS only) of the executable used to collect the traces and requires the presence of helper tools. If the helper tools are not available or fail to identify the required information, `fgtrun` scripts sets default values and runs the collection. Optionally you can override these default values by setting architecture and C/C++ runtime version information (Windows* OS only) using command line arguments when the script is invoked.

The `fgtrun` has the following options:

--omp	<p>Enable OpenMP* trace collection for applications linked with <code>-qopenmp</code>.</p> <p>The OpenMP* runtime environment must be set correctly before the script is launched.</p> <hr/> <p>NOTE This OpenMP* trace collection capability is currently not supported on the Windows* OS.</p>
--xml	<p>Collect traces in XML format.</p> <p>By default, the collector generates binary traces. If trace collection fails, you can switch to XML trace generation mode to debug the cause of the failure.</p>
--sym	<p>Get mapping between nodes and source code.</p> <hr/> <p>NOTE The symbol resolution feature is currently only supported on Linux* OS.</p>

Collecting Trace Files without fgtrun Script

You can choose to collect traces without the `fgtrun` script if you do not want to launch your application from a Visual Studio* command prompt or Linux* terminal. In this case, follow the steps below to collect and convert trace files manually.

NOTE

These steps do not outline steps required to capture symbol resolution information.

1. Set paths to the collector libraries.

Set up or update the environment variables shown below. For Windows* systems, specify also the proper version of the Microsoft Visual Studio* compiler (vc12, vc14, or vc14.1).

Operating System	Environment Variable	Default Value	Description
Windows*	INTEL_LIBITNOTIFY_32	..\fgt\windows\bin\ia32\<vc version>\fgt.dll	32-bit collector library
	INTEL_LIBITNOTIFY_64	..\fgt\windows\bin\intel64\<vc version>\fgt.dll	64-bit collector library
	INTEL_ITNOTIFY_GROUPS	all;	Trace events from all groups
Linux*	INTEL_LIBITNOTIFY_32	../fgt/linux/lib/ia32/cc4.8_libc2.19_kernel3.13.0/libfgt.so	32-bit collector library
	INTEL_LIBITNOTIFY_64	../fgt/linux/lib/intel64/cc4.8_libc2.19_kernel3.13.0/libfgt.so	64-bit collector library
	INTEL_ITNOTIFY_GROUPS	all;	Trace events from all groups
macOS*	INTEL_LIBITNOTIFY_32	../fgt/macos/lib/ia32/osx10.12.6_kernel16.7.0/libfgt.dylib	32-bit collector library
	INTEL_LIBITNOTIFY_64	../fgt/macos/lib/intel64/osx10.12.6_kernel16.7.0/libfgt.dylib	64-bit collector library
	INTEL_ITNOTIFY_GROUPS	all;	Trace events from all groups

If you want the Microsoft Visual Studio* IDE to use the environment variables set in a Microsoft Visual Studio command prompt, you can launch the Visual Studio* IDE from the command prompt using the following command:

```
devenv /useenv
```

2. Run the application.

If your paths are set up correctly, the application generates one or more files that start with `_fgt`. There is one file per thread that participates in executing the parallelism in the application. So, for example, if two threads participate in the execution of the flow graph, your run generates two files: `_fgt.0` and `_fgt.1`, with an autogenerated folder in the format `_fga_YYYYMMDD_HHMMSS` according to its creation (for example, `20191111_1111`).

3. Convert the trace files to GraphML* and TraceML* format.

Convert the `_fgt` binary files to the XML format understood by the Flow Graph Analyzer tool using the `fgt2xml.exe` converter in the directory containing the folder with the trace files:

```
fgt2xml.exe <desired_name>
```

This converter scans the current directory for all `_fgt` files within the most recent folder according to its name and generates two output files: `desired_name.graphml` and `desired_name.traceml`.

Nested Parallelism in Flow Graph Analyzer

The Flow Graph Analyzer supports visualization of applications that contain multiple levels of parallelism, such as nested Intel® oneAPI Threading Building Blocks (oneTBB) algorithms and OpenMP* parallel regions. This feature requires additional support from the parallel runtime libraries and can be used in combination with a oneTBB flow graph.

The sample code below is an example of nested parallelism that combines a oneTBB flow graph, a `oneTBBparallel_for` algorithm, and an OpenMP* parallel region.

```
#include "tbb/tbb.h"
#include "tbb/flow_graph.h"
#include <omp.h>
#include <iostream>

using namespace tbb;
using namespace tbb::flow;
int main() {
    graph g;
    const int size = 20;
    continue_node< continue_msg> hello( g,
        []( const continue_msg & ) {
            std::cout << "Hello\n";
            tbb::parallel_for(0, size, 1, [](int k) {
                std::cout << k << "\n"; });
        });
    continue_node< continue_msg> world( g,
        []( const continue_msg & ) {
            std::cout << " World\n";
            #pragma omp parallel for
            for (int i=0; i<20; i++) {
                std::cout << i << "\n"; }
        });
    make_edge(hello, world);
    hello.try_put(continue_msg());
    g.wait_for_all();
    return 0;
}
```

Tracing Nested Intel® oneAPI Threading Building Blocks (oneTBB) Algorithms

oneTBB enables general tracing of parallel algorithms, which is enabled by default and activated by the Flow Graph Analyzer trace collector.

As a result, Flow Graph Analyzer can display oneTBB library activity in nested and non-nested algorithms. Therefore, task context switches are captured and can be visualized in the Flow Graph Analyzer GUI. This work is similar to tasks in the timeline and is named according to its algorithm (for example, `parallel_for`).

NOTE

This information might not be available for user-defined task groups.

Tracing Nested OpenMP* Algorithms

For detailed information on Flow Graph Analyzer support for OpenMP* technology, see [Experimental Support for OpenMP* Applications](#).

Analyzer Workflow

NOTE

This section describes a recommended workflow to identify performance issues in the executed graph. This workflow may change as more analytics plugins are added. However, the fundamental principle should not change, as the goal is to maximize the throughput of the graph in a streaming case, and provide the best scaling performance with respect to the serial run.

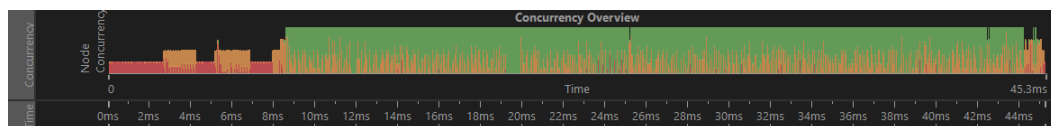
The Flow Graph Analyzer provides the following capabilities for analyzing flow graph performance:

- Display the graph for which the execution trace is captured. See the [Preferences](#) section for details on how to enable loading `.graphml` files that contain graphs with cross-graph edges.
- Display the trace information and highlight parallel performance issues.
- Map poorly scaling time regions to nodes executing at that time.
- Compute the critical path of the graph.
- View compute statistics for the computational nodes based on the execution traces.
- View prioritized diagnostics.

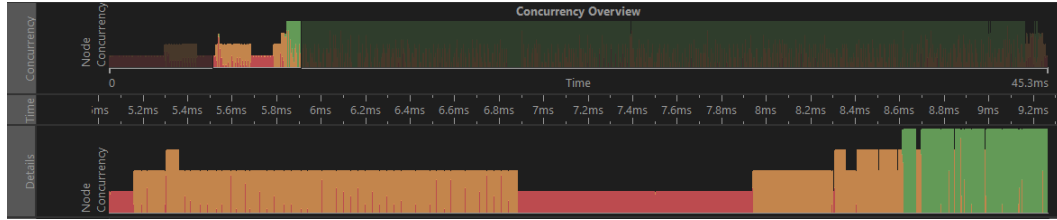
Follow the steps in this section to analyze performance.

Find Time Regions of Low Concurrency and Their Cause

1. Run the [trace collector](#).
2. In the **Execution Trace Views** tab, inspect the node concurrency histogram for regions in red, which indicate low concurrency.



3. Zoom in to a red region to inspect the data at a higher resolution and provide a better idea as to how concurrency varies over time.



4. Select a point in the chart where the concurrency is low to highlight the relevant node(s). Hover the mouse over the highlighted node to identify the node name.

Because the analysis tool does not have built-in symbol resolution, the determination of the C++ class of the body executed by a node must be explicitly encoded into the application. Explicit encoding affects the *Name* and/or *object_name* fields in the **Node Properties** tab. For example:

```
tbb::flow::graph g;
...
tbb::flow::source_node<int> s_node (g, source_node_body(),
false);
#ifdef TBB_PREVIEW_FLOW_GRAPH_TRACE
s_node.set_name("My Source Node");
#endif
```

This coding enables node annotation with the specified string during trace collection, and the annotation appears when you hover the mouse over the node.

NOTE

The `set_name` functions are only available when the `TBB_PREVIEW_FLOW_GRAPH_TRACE` macro is defined at compile-time.

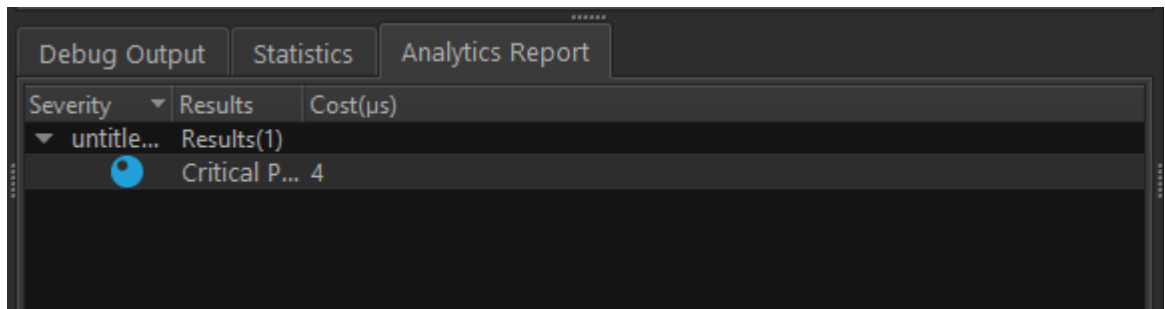
Finding a Critical Path

The *critical path* from a node N to a node M is the longest (in time) path from N to M. Its length is a lower-bound on the execution time of a computation that begins at N and ends at M.

The **Critical Path Analytic** in the Flow Graph Analyzer computes a critical path for each source node/sink node pair. A given graph has as many critical paths as the product of the number of source nodes and the number of sink nodes. Source nodes are the nodes in the graph without any predecessors, or nodes with an in-degree of zero. Sink nodes are the nodes in the graph without any successors, or nodes with an out-degree of zero.

Click the **Compute Critical Path** icon on the toolbar to calculate the critical paths in the graph. These critical paths are displayed in descending order by cost. Inspect the topmost critical path first because, as the longest critical path, it sets the lower bound on the execution time for the whole graph.

The screenshot below shows a sample critical path report.



Selecting a critical path in the **Analytics Report** window highlights all the nodes on the critical path.

Finding Tasks with Small Durations

Tasks executed by a flow graph are spawned as Intel® oneAPI Threading Building Blocks (oneTBB) tasks, so the task duration must be large enough to amortize the cost of a task spawn.

Finding Tasks with Small Durations using Statistics

1. Open the **Statistics** tab in the bottom pane.
2. Open the **Graph** tab to see execution time metrics. The metrics are computed as the mean and standard deviation for each node based on the execution traces.
3. Sort the resulting data by the **Avg Task Duration** column to identify the nodes with the smallest average durations.

Any node with an average duration of a few microseconds requires that you additionally inspect it, because any concurrency gained by its parallel execution may be overwhelmed by its scheduling costs.

Severity	For	Node Name	Instance Count	In-degree	Out-degree	Total Time (μs)	Avg. Task Duration (μs)	Std. Dev.	Average Concurrency
g0	Resu...								
	Node	src	251	0	1	1.04226e+09	4.15242e+06	1.2806e+06	4
	Node	buffers	unset	1	1	unset	unset	unset	unset
	Node	resource_join	unset	2	1	unset	unset	unset	unset
	Node	preprocess_fun..250	250	1	2	2.00888e+09	8.03554e+06	1.79901e+06	2.44
	Node	detect_A	250	1	1	1.55358e+09	6.2143e+06	745388	1.65
	Node	detect_B	250	1	1	1.57988e+09	6.31952e+06	635088	1.99
	Node	detection_join	unset	2	1	unset	unset	unset	unset
	Node	decide	250	1	1	4.46198e+07	178479	410715	1

The screenshot above shows a sample graph that executed 250 times. Notice the **Count** column, which is the number of times a node executes, is 250 for all functional nodes. Inspect the **Avg Task Duration** column for nodes that execute in a few microseconds.

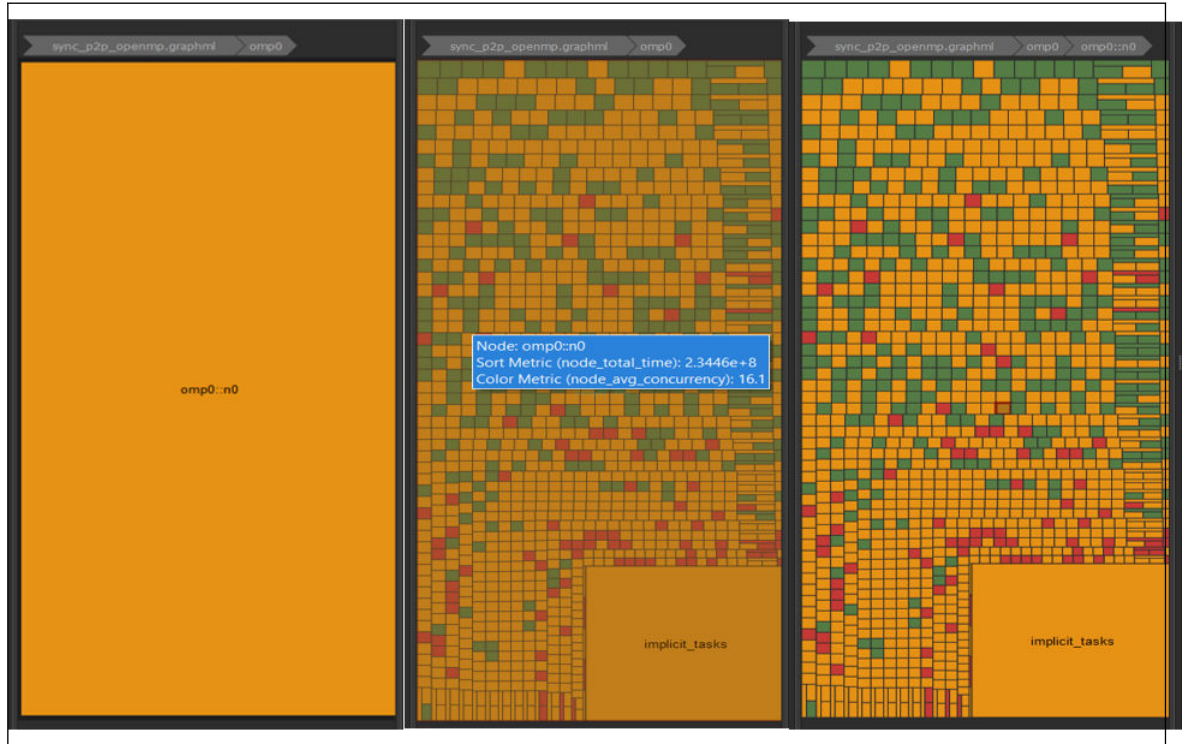
NOTE Times are provided in milliseconds.

Finding Tasks with Small Durations using Treemap

Another way to visualize task durations and the average concurrencies of each node is the Treemap view in the **Analysis Mode** tab. If multiple graphs are present in the application run, you see a high-level treemap showing the health of all the graphs in the run.

The Treemap view organizes the nodes in the graph by node durations. The larger the area of the square, the more time the node spent on the CPU. The node color is determined by the average observed concurrency of the graph while the node was running, and the colors use the same scale as in the active thread chart. A red node indicates poor concurrency when the node is executed on the system.

Hover a mouse over the Treemap to see the details about nodes in the graph. Double-click the graph in the Treemap view to keep the node treemap visible.



When a graph has nested subgraphs, the treemap presents this information by embedding the nested subgraph in the node which spawned the subgraph. This is visually represented by increasing the width of the border surrounding each node that contain subgraphs.

- Hover the mouse over a node that has embedded subgraphs to see the hidden subgraph.
- Double-click the node to zoom to the child-node level and see the contents of the embedded subgraph as a treemap.

Click any node in the Treemap view to highlight that node in the graph view on the canvas. If you select the default zoom factor or reset the zoom factor, clicking a node in the Treemap view zooms in and centers on the node in the graph view. The smaller the node size, the smaller the tasks executed by that node.

The Treemap view supports three different layouts for visualizing the treemap: squarified layout, alternating layout, and snake layout. All three layouts use:

- The node CPU time to determine the size of each node in the treemap
- The average concurrency observed in the graph while a node was active to determine the color

To switch between layouts:

1. Open the **Preferences** window.
2. Go to the **GUI > General**.
3. Change the value of the **Default Treemap Rendering** option in the **Analysis View** group.

Reduce Scheduler Overhead using Lightweight Policy

The Flow Graph API allows you to apply *lightweight policy* for computational nodes such as function node, multifunction node, continue node, and async node. Enabling the lightweight policy helps reduce scheduling overhead. It can limit parallel execution of tasks, so apply this policy on a per-node basis after careful evaluation.

The lightweight policy indicates that the body of a node contains a small amount of work and, if possible, should be executed without the overhead of scheduling a task. By default, the async node has the lightweight policy enabled because it has small computation weight. All other computational nodes do not have the lightweight policy enabled when they are dragged and dropped into the canvas.

Use the lightweight policy in the following cases:

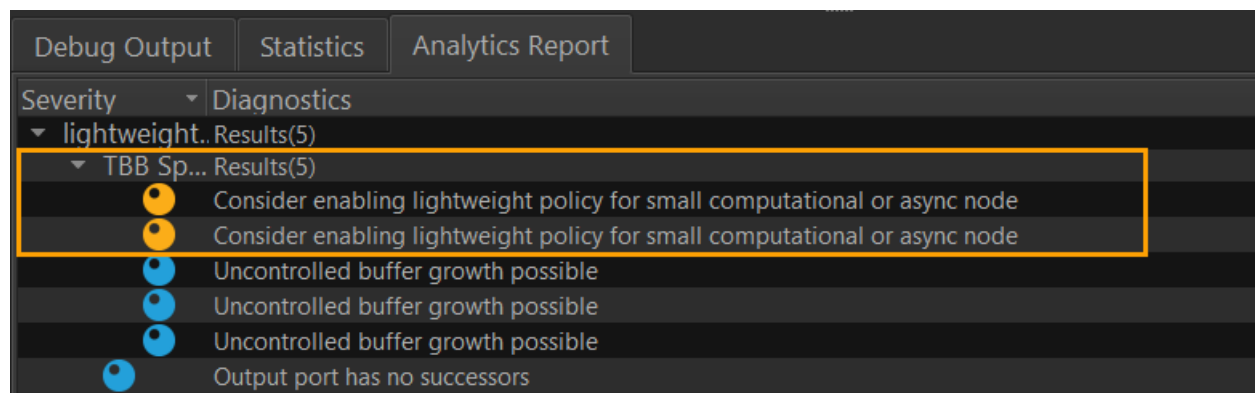
- Node weight is less than 1 microsecond when no trace information is available.
- Node average time is less than 1 microsecond if the graph is loaded into context with trace data.

When validating a graph, the graph rule check automatically identifies nodes that can use the lightweight policy. If the above conditions are not met but the lightweight policy is set, the graph rule check recommends removing the lightweight policy for the corresponding node.

To display recommendations for applying the lightweight policy:

1. Click the graph rule check icon on the toolbar to run the check.
2. Go to the **Analytics Report** tab to see the results.
3. Based on the results, set or disable the lightweight policy for certain nodes.
 - To set or disable the lightweight execution policy for a *single* node:
 1. Click the node to display the node properties on the right pane.
 2. Set the **Execution policy** property to `none` to disable lightweight policy or to `lightweight` to enable lightweight policy.
 - To set or disable the lightweight execution policy for *all* the nodes listed by graph rule check:
 1. Multi-select the report lines that say **Consider enabling lightweight policy for small computational or async node** or **Consider disabling lightweight policy for small computational or async node** to highlight all nodes in the canvas and display the common properties for all the selected nodes.
 2. Set the **Execution policy** property to `none` to disable lightweight policy or to `lightweight` to enable lightweight policy.

For example, to enable the lightweight policy for multiple nodes:



Another important attribute related to node execution policy is a *buffer policy*:

- If you set the buffer policy to `queueing` when lightweight policy is enabled, the Flow Graph Analyzer adds `queueing_lightweight` to the policy parameter of the node declaration during C++ code generation.
- If you set buffer policy to `rejecting`, the Flow Graph Analyzer adds `rejecting_lightweight` to the policy parameter of the node declaration during C++ code generation.

For example, by default, the async node is set to `queueing_lightweight`, and the Flow Graph Analyzer does not add any policy during code generation for async node. Setting buffer policy to `rejecting` for the async node adds `rejecting_lightweight` to the policy parameter of async node declaration during code generations.

Identifying Tasks that Operate on Common Input

Intel® oneAPI Threading Building Blocks (oneTBB) adds additional meta information to trace data, such as a frame number that a current task is working on. You can use this metadata to track different pipeline stages in execution, for example to identify an unbalanced pipeline. The following demonstrates how to use the user event tracing interface to enable the Flow Graph Analyzer *color-by-data* feature.

Highlighted in bold is code that enables the Flow Graph Analyzer to add a unique ID (frame ID) to group tasks.

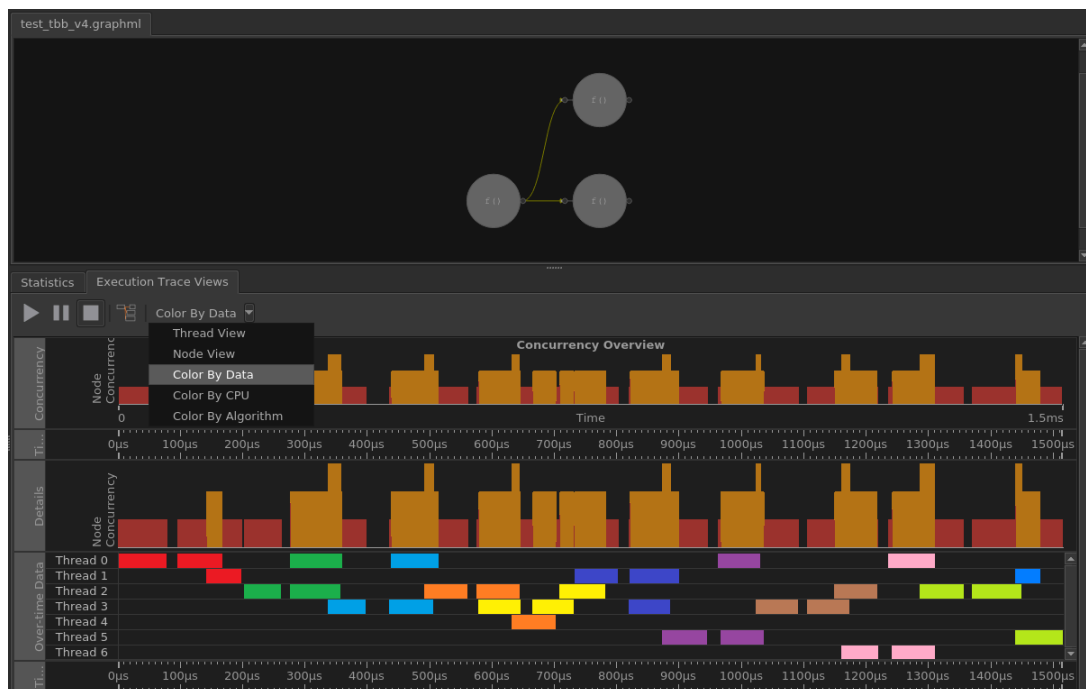
```
#include "tbb/flow_graph.h"
#include "tbb/tbb_profiling.h"
#include <string>
#include <vector>
int main() {
    tbb::flow::graph g;
    const int max_frames = 20;
    std::vector<tbb::profiling::event*> e;for(int i=0; i<nbr_of_frames;++i)e.push_back(new
tbb::profiling::event(std::to_string(i)));
    tbb::flow::source_node<int> source( g,
        [&]( int &v) -> bool {
            static int i = 0;
            if( i < max_frames ) {
                e[i]->emit();v = i++;
                return true;
            }
            return false;
        }, false);
    tbb::flow::function_node<int> foo( g, tbb::flow::unlimited,
        []( const int &input1) -> int {
            tbb::profiling::event::emit(std::to_string(input1));
            return input1;
        });
    tbb::flow::function_node<int> bar( g, tbb::flow::unlimited,TBB
        []( const int &input1) -> int {
            tbb::profiling::event::emit(std::to_string(input1));
            return input1;
        });
    make_edge(source, foo);
    make_edge(source, bar);
    source.activate();
    g.wait_for_all();
    return 0;
}
```

Compile the code with the `TBB_USE_THREADING_TOOLS` macro and link against the `tbb` library.

To enable the Flow Graph Analyzer *color-by-data* feature:

1. Assign IDs to events doing one of the following:
 - Option 1:
 1. Create an event object or a collection of events upfront, where the only argument is a string (*data ID*) that identifies the event.
 2. Call the `emit` function of the object to tag a task with a data ID.
 - Option 2: Call a static function inline (inside a task body).
2. In the **Execution Trace Views** tab in the bottom pane, choose **Color By Data** from the drop-down list.

Result: The Flow Graph Analyzer groups tasks that share the same data ID and displays them in a common color:



Support for SYCL

Flow Graph Analyzer is a feature of Intel® Advisor that allows you to explore, debug, and analyze graph computation problems. Since the SYCL* runtime constructs an asynchronous task graph from submitted work, Flow Graph Analyzer allows you to visualize and interact with the asynchronous task graph, and its execution traces. The tool introduces the following features:

- For a CPU device: Execution trace-based analytics.
- For CPU and GPU devices: Graph-related analytics.

NOTE

The data collection support for SYCL applications is currently supported only on Linux* OS.

The code sample below illustrates a simple example of a SYCL application that adds two vectors. The subsequent sections will use it as an example.

```
#include <CL/sycl.hpp>
#include <iostream>

#define VECTOR_SIZE 16384

using namespace cl::sycl;

void vec_add(queue &q, const float A[], const float B[], float C[],
             const int size) {
    // Create the buffers
    buffer<float, 1> bufA(A, range<1>(VECTOR_SIZE));
    buffer<float, 1> bufB(B, range<1>(VECTOR_SIZE));
    buffer<float, 1> bufC(C, range<1>(VECTOR_SIZE));

    q.submit([&](handler &cgh) {
        auto Acc = bufA.get_access<access::mode::read>(cgh);
```

```
    auto Bcc = bufB.get_access<access::mode::read>(cgh);
    auto Ccc = bufC.get_access<access::mode::write>(cgh);
    cgh.parallel_for<class saxpy_kernel>(range<1>(size), [=](id<1> idx) {
        Ccc[idx[0]] = Acc[idx[0]] + Bcc[idx[0]];
    });
});
}

int main(int argc, char **argv) {
    if (argc < 2) {
        std::cout << "Usage:- " << argv[0] << " [cpu, gpu]\n";
        return 1;
    }

    float A[VECTOR_SIZE], B[VECTOR_SIZE], C[VECTOR_SIZE];

    if (std::string("cpu") == argv[1]) {
        cpu_selector device;
        queue q(device);
        vec_add(q, A, B, C, VECTOR_SIZE);
    } else if (std::string("gpu") == argv[1]) {
        gpu_selector device;
        queue q(device);
        vec_add(q, A, B, C, VECTOR_SIZE);
    }

    return 0;
}
```

Collect SYCL Application Traces

NOTE SYCL trace collection is currently supported only on Linux* OS.

The command line collector for Flow Graph Analyzer enables you to capture trace data from SYCL applications. To collect SYCL traces, you will need trace-enabled SYCL run-times. Use the code from [Analyze Data Parallel C++ Application](#) as a sample application. To build it:

1. Copy the code snippet from [Analyze Data Parallel C++ Application](#) and save it as `va_const.cpp`.
2. Run the following command to build it:

```
icpx -fsycl -o vac ./va_const.cpp
```

To collect traces for the SYCL application using the built sample and create the XML files to view with Flow Graph Analyzer:

1. Set the environment variable `FGT_ROOT` to point to `<fga-install-dir>/fgt`:

```
export FGT_ROOT=<advisor-install-dir>/fga/fgt
```

2. Set the back end for the SYCL run-time to OpenCL™ by setting the following environment variable:

```
export SYCL_BE=PI_OPENCL
```

NOTE Current version of Flow Graph Collector does not support Level0.

3. Run the application using the Flow Graph Analyzer collector:

```
$FGT_ROOT/linux/bin/fgtrun.sh ./vac gpu
```

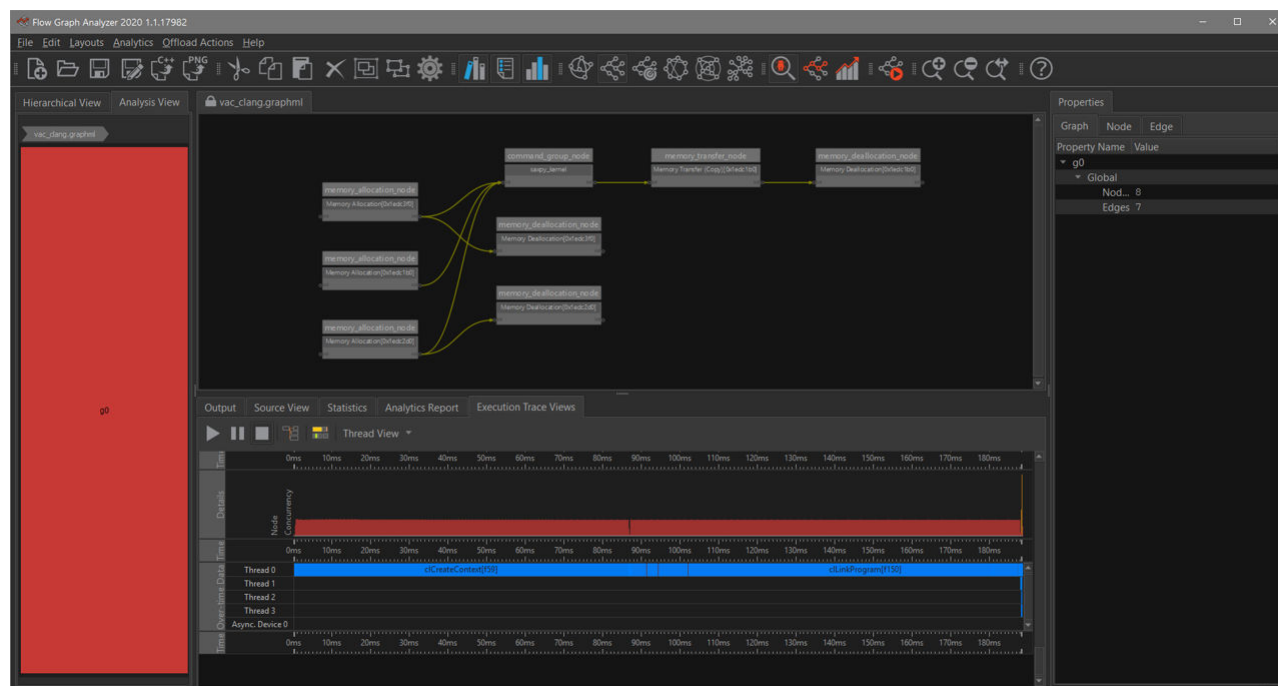
This command will generate two files: `vac.graphml`, which contains the semantic information of what was executed, as in the asynchronous task-graph, and `vac.traceml`, which contains the execution traces of the application. Open the files in the Flow Graph Analyzer on the current system or copy them to another system with Flow Graph Analyzer installed to investigate.

To launch the Flow Graph Analyzer graphical user interface, use the following command:

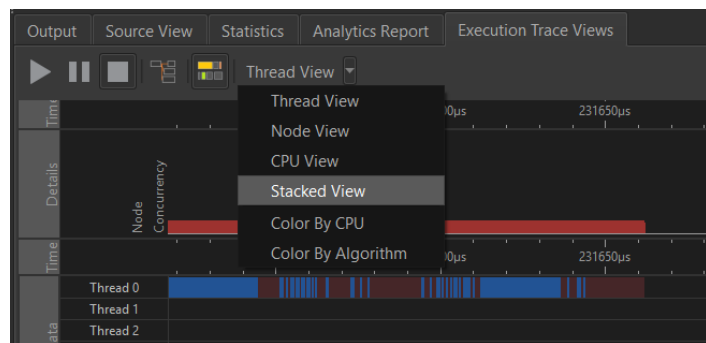
```
<fga-install-dir>/fga/run_fga.sh &
```

Examine a SYCL Application Graph

The visualization of SYCL applications is similar to data from other run-times, such as Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP*. The graph in SYCL represents the asynchronous task graph created from the end-user construct such as buffer accessors, command group handler, and data parallel constructs such as `parallel_for`.



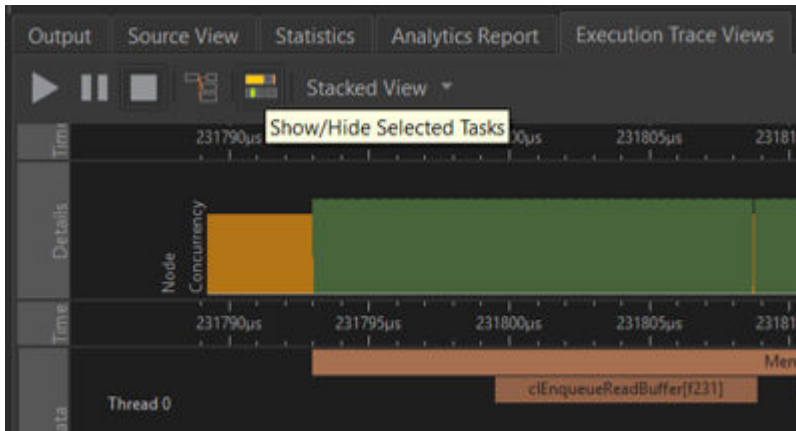
The data from the sample viewed in Flow Graph Analyzer is shown above. As with other runtimes, the graph view is correlated with the execution trace views. The workflows will provide similar information for SYCL. To better visualize the overlapping tasks in the execution trace view, select the **Stacked View** attribute from the pull-down menu as shown below:



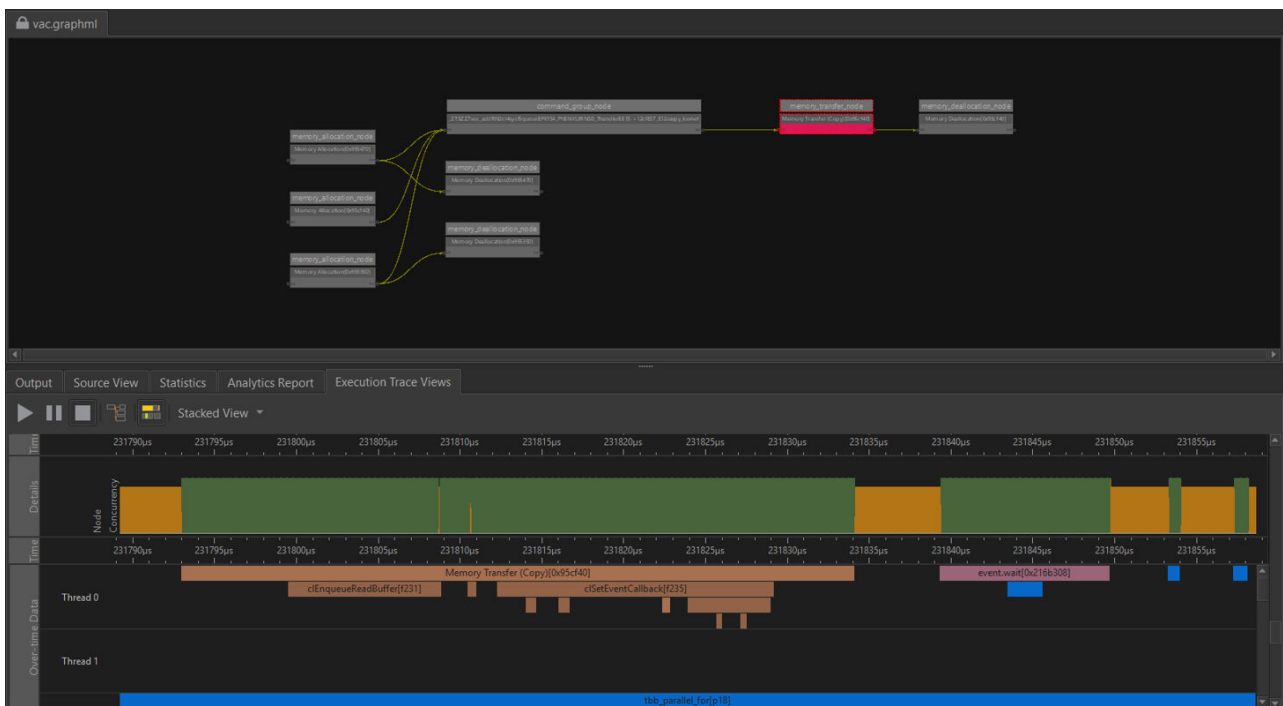
This will change the view to an icicle chart that displays everything in detail, and you can see the calls to the OpenCL™ stack.



Clicking on a task in the timeline views will highlight a node in the graph if that task belongs to the graph. If you want to highlight all the tasks that belong to a graph node, you should enable task highlighting button and select a node on the graph to see the associations.
















The screenshot below shows the tasks that belong to the `memory_transfer_node` are highlighted in a different color. Using the correlation features, one can debug the execution profile of the application to get a better understanding of the execution. Flow Graph Analyzer also includes features that target specific performance-related issues and the other sections go into detail for each one of these potential performance problems.



Hotspot View

Flow Graph Analyzer supports hot-spot views, but the data is limited to the objects in the graph. The data collection for Flow Graph Analyzer-based collectors is currently limited to time-based traces. To obtain the hotspot view, select the Statistics tab, and click the Graph pane for the data.

Output	Source View	Statistics	Analytics Report	Execution Trace Views					
Graph	Parallel Efficiency								
Severity	For	Node Name	Instance Count	In-degree	Out-d	Total Time (clks)	Avg. Task Duration (clks)	Std. Dev.	Average Concurrency
▼ g0	Resul...								
	Node	kernel1	7	1	1	2039185951	5.22868e+07	1.61754e+07	6.87
	Node	kernel2	7	1	1	8952324	241955	701279	3.87
	Node	kernel3	2	1	1	4428168	2.21408e+06	3.0952e+06	1.09
	Node	Memory Transfer (Unmap)[0x255ad...	10	1	1	333503	33350.3	80682.5	1.06
	Node	Memory Transfer (Map)[0x255ad10]	1	1	1	21391	21391	0	2.14
	Node	Memory Allocation[0x255ae30]	1	0	1	17907	17907	0	1
	Node	Memory Allocation[0x255ad10]	2	1	1	17224	8612	10081.9	1.09
	Node	Host Accessor Creation/Buffer Lock...	1	1	1	8458	8458	0	3
	Node	Memory Transfer (Copy)[0x255ae30]	1	1	1	5812	5812	0	1
	Node	Host Accessor Destruction/Buffer L...	1	1	1	994	994	0	0.999
	Node	Memory Deallocation[0x255ae30]	unset	1	0	0	unset	unset	unset
	Node	Memory Transfer (Copy)[0x255ad10]	unset	1	1	0	unset	unset	unset
	Node	Memory Deallocation[0x255ad10]	unset	1	1	0	unset	unset	unset

View Performance Inefficiencies of Data-parallel Constructs

The **Statistics Tab** also contains the efficiency information for each parallel construct if they are employed by the algorithm. This data will show up under the **Parallel Efficiency** tab in the **Statistics** group.

Output	Source View	Statistics	Analytics Report	Execution Trace Views					
Graph	Parallel Efficiency								
Severity	For	Efficiency(%)	Task Count	Duration	CPU Time(%)	Other Time(%)	Fork Imbalance(%)	Join Imbalance(%)	
▼ kernel2	[tid 4]	100	1	17537	100	0	0	0	
	Resul...	73.93	0	0	81.7883	18.2117	8.80649	17.2635	
	▼ p71	Resul...	83.2723	7	163761	85.1962	14.8038	2.05294	14.6748
	[tid 1]	95.8206	1	158334	99.6306	0.369366	3.82413	0.355241	
	[tid 2]	70.7898	1	116973	73.1886	26.8114	3.27766	25.9326	
	[tid 3]	96.6068	1	159633	99.8655	0.134503	3.26313	0.130114	
	[tid 4]	98.2256	1	162308	99.7762	0.223763	1.5541	0.220286	
	[tid 5]	99.1049	1	163761	100	0	0.895062	0	
▼ p41	[tid 6]	98.294	1	162421	99.8482	0.151843	1.55652	0.14948	
	[tid 7]	24.0644	1	39764	24.0644	75.9356	0	75.9356	
	Resul...	64.9884	7	157116	82.9371	17.0629	20.1198	14.8918	
	[tid 1]	43.9785	1	83560	66.863	33.137	34.226	21.7956	
	[tid 2]	82.6918	1	157116	99.7866	0.213398	17.1314	0.17684	
	[tid 3]	80.1971	1	152376	99.7702	0.229822	19.6182	0.184735	
	[tid 4]	38.2733	1	72720	38.2733	61.7267	0	61.7267	
	[tid 5]	65.4525	1	124361	100	0	34.5475	0	
▼ p29	[tid 6]	80.0992	1	152190	99.7686	0.23141	19.7151	0.185788	
	[tid 7]	64.2267	1	122032	76.0983	23.9017	15.6004	20.1729	
	Resul...	81.6849	7	140447	85.8631	14.1369	4.31775	13.9974	
	[tid 1]	72.2106	1	104865	74.9909	25.0091	3.70745	24.0819	
	[tid 2]	26.7957	1	38913	26.7957	73.2043	0	73.2043	
	[tid 3]	92.3186	1	134066	99.4917	0.508345	7.2097	0.471695	
	[tid 4]	93.9175	1	136388	99.9699	0.0300523	6.05422	0.0282328	
	[tid 5]	95.4524	1	138617	99.9502	0.0497527	4.50004	0.0475138	
▼ p23	[tid 6]	96.7126	1	140447	100	0	3.2874	0	
	[tid 7]	94.3865	1	137069	99.8434	0.15661	5.46546	0.14805	
	Resul...	70.9988	7	170831	72.3481	27.6519	1.65269	27.3485	
	[tid 0]	30.5581	1	53058	31.1025	68.8975	1.75027	67.6916	
	[tid 1]	67.1344	1	160654	60.6567	39.3433	3.52174	0.22463	
	[tid 2]	70.7898	1	116973	73.1886	26.8114	3.27766	25.9326	
	[tid 3]	96.6068	1	159633	99.8655	0.134503	3.26313	0.130114	
	[tid 4]	98.2256	1	162308	99.7762	0.223763	1.5541	0.220286	

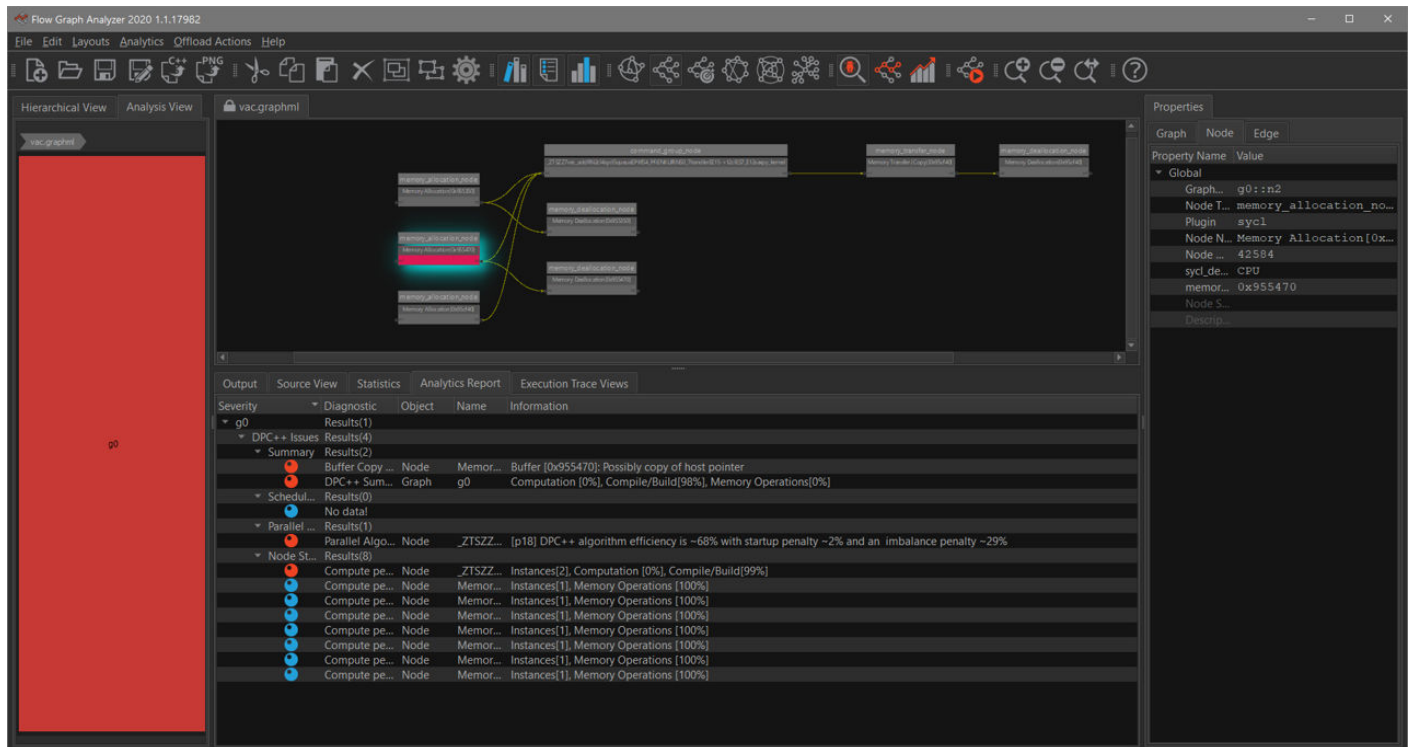
The data parallel construct efficiency for each instance of a kernel. The column provides information that is useful for understanding the execution, and makes inferences to improve performance.

- The parallel algorithms are nested under the kernel name when the kernel name can be demangled correctly.
- The Efficiency column indicates the efficiency of the algorithm, when associated with the algorithm name. For the participating worker threads, the efficiency column indicates the efficiency of the thread while participating in the execution. This data is typically derived from the total time spent on the parallel construct and the time the thread spent participating in other parallel constructs.
- Task Count column indicates the number of tasks executed by the participating thread.
- Duration indicates the time the participating thread spends executing tasks from the parallel construct.
- CPU time is the **Duration** column data expressed as a percentage of the wall clock time of the parallel construct.

- **Other Time** will be 0 if the thread fully participates in the execution of tasks from the parallel construct. However, in runtimes such as Intel® oneAPI Threading Building Blocks, the participating threads may steal tasks from other parallel constructs submitted to the device to provide better dynamic load balancing and throughput. In such cases, the **Other Time** column will indicate the percentage of the total wall clock time the participating thread spends executing tasks from other parallel constructs.
- **Fork Imbalance** indicates the penalty for waking up threads to participate in the execution of tasks from the parallel construct. For more information, see Startup Penalty.
- **Join Imbalance** indicates the degree of imbalanced execution of tasks from the parallel constructs by the participating worker threads. For more information, see [Data Parallel Efficiency](#).

Find Issues Using Static Rule-check Engine

The Rule-check engine in Flow Graph Analyzer includes SYCL specific analysis that can be invoked by clicking on the **Rule-check** button in the toolbar. Run it to check for issues.



NOTE In the sample code, the kernel name demangling has an issue. This is the current limitation of the demangling mechanism available in the runtime. For correct demangling of kernel names, use the open-source version of the Clang++ compiler.

The Rule-Check Engine currently identifies the following types of issues that may be present in an application:

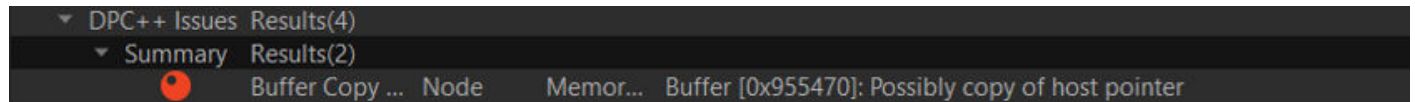
- Use of const reference to a host pointer to initialize a buffer
- Use of host pointer accessor in a loop
- Data parallel construct inefficiency

Issue: Const Reference to a Host Pointer Used to Initialize a Buffer

When porting applications from C++ to SYCL, these issues may be implicitly present. The convention for passing arguments to a function is defined in C++. If a function requires read-only parameters, they are sent in as const references and any parameter that has to be used for read-write will be without a const. This causes a secondary issue in SYCL algorithms if these const references are used by the algorithm to construct buffers. Since the type of access required to use this data is not known at the time of construction of the

buffer, the Intel® oneAPI DPC++/C++ Compiler is conservative and creates copies of const references while creating the buffers. If these are large arrays, the cost incurred is not trivial. This issue only affects the CPU device as everything is communicated through shared memory and the copying of data pointer to the host pointer is not necessary.

The `va_const.cpp` example demonstrates such an issue and indicates buffer copy in the application that needs to be looked at.



To eliminate this copy, you should change the code sample in the following way:

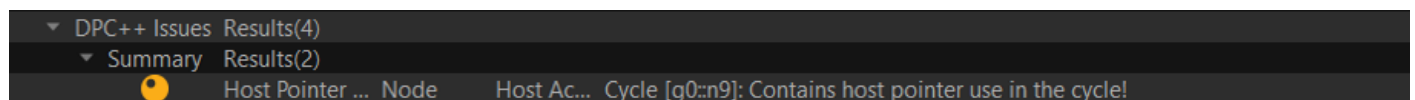
```
// Old code:
// The function prototype passed in the read-only buffers as const, which is the
// recommended practice in C++
//
// void vec_add(queue &q, const float A[], const float B[], ...) {
//
void vec_add(queue &q, float A[], float B[], float C[],
              const int size) {
    ...
}
```

NOTE The recommended practice in C++ is to pass in read-only parameters as const values. However, this causes the Intel® oneAPI DPC++/C++ Compiler to be conservative and create a copy. If you are porting C++ code to SYCL, the static rule-check should help you identify such issues in your application.

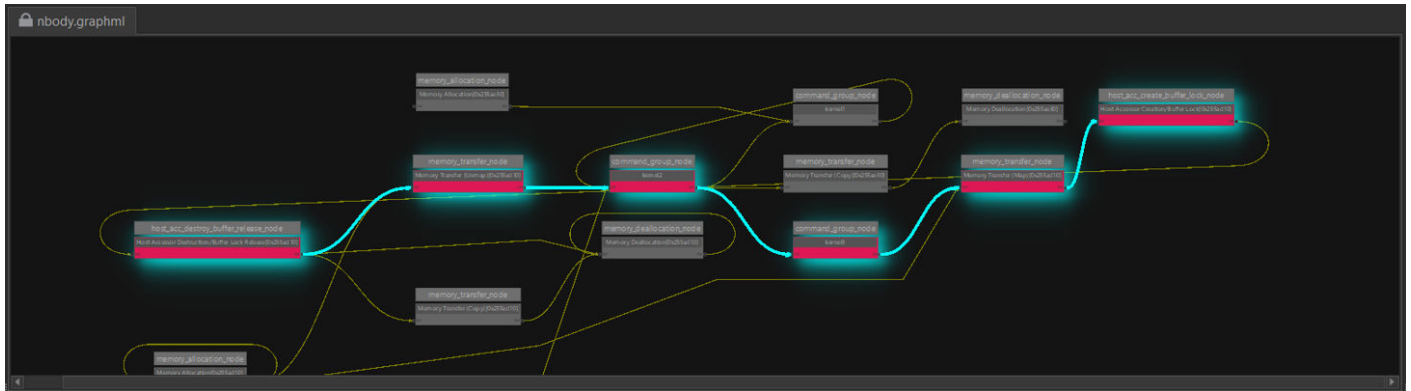
Issue: Host Pointer Accessor Used in a Loop

In many algorithms, it is likely that a lot of operations are performed on the device memory and some operations on the host memory. This is particularly true in simulation code where the host memory is updated using a host accessor. This causes many things to happen within the SYCL run-time where the locks the buffer the accessor points to and updates the copy of this buffer memory on the host device. This pattern of access could cause a lot of memory copies from the device to the host and back in order to keep the data coherent.

Flow Graph Analyzer reports such issues in the following way:



Click the issue to highlight the loop in the graph that consists of a host pointer accessor.



If the buffer pointed by the host pointer accessor is large, the costs incurred due to this access can be a significant portion of each loop.

Issue: Data Parallel Construct Inefficiency

The SYCL language allows to use data parallel constructs within each command group. This feature of rule-check analysis tries to capture the efficiency of a data parallel construct. The inefficiencies in the data parallel construct are broken down into two parts:

- Startup costs for kicking off the data parallel algorithm on the worker threads.
- Imbalance costs encountered during the execution of the algorithm.

The combination of these parts affects the overall efficiency of the data parallel algorithm. The data and screenshots shown in this section are from the Nbody sample that is available with Intel® oneAPI Toolkits.

Parallel Algorithms	Results(11)
Parallel Algo... Node	kernel1 [p51] DPC++ algorithm efficiency is ~96% with imbalance penalty ~3%
Parallel Algo... Node	kernel1 [p15] DPC++ algorithm efficiency is ~92% with startup penalty ~4% and an imbalance penalty ~2%
Parallel Algo... Node	kernel1 [p21] DPC++ algorithm efficiency is ~98%
Parallel Algo... Node	kernel1 [p57] DPC++ algorithm efficiency is ~98%
Parallel Algo... Node	kernel1 [p69] DPC++ algorithm efficiency is ~97% with imbalance penalty ~2%
Parallel Algo... Node	kernel2 [p17] DPC++ algorithm efficiency is ~68% with startup penalty ~15% and an imbalance penalty ~15%
Parallel Algo... Node	kernel2 [p23] DPC++ algorithm efficiency is ~70% with imbalance penalty ~27%
Parallel Algo... Node	kernel2 [p29] DPC++ algorithm efficiency is ~81% with startup penalty ~4% and an imbalance penalty ~13%
Parallel Algo... Node	kernel2 [p41] DPC++ algorithm efficiency is ~64% with startup penalty ~20% and an imbalance penalty ~14%
Parallel Algo... Node	kernel2 [p71] DPC++ algorithm efficiency is ~83% with startup penalty ~2% and an imbalance penalty ~14%
Parallel Algo... Node	kernel3 [p19] DPC++ algorithm efficiency is ~100%

Startup Penalty

The startup costs are primarily related to the worker threads participating in the parallel algorithm starting up slowly. If your application exhibits a lot of inefficiencies due to startup costs, the Linux* kernel maybe biased towards power. You can use the following command to ensure that it is set to performance:

```
echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

NOTE

You will need `sudo` privileges to run this command and this setting is reset after system reboot.

Imbalance Penalty

Imbalance penalties are usually due to static partitioning of a data parallel workload that have varying costs per iteration of the loop or when the granularity of the block size is too large when dynamic partitioning is used. Many times, addressing the startup penalties will improve the amount of imbalance in the algorithm, but if they are retained, the following options maybe tried to improve performance:

- If the algorithm uses `range`, the runtime is automatically picking the block size and may be causing the imbalance. You can override this by using `nd_range` and specifying a block size that would eliminate the imbalance.

- If `nd_range` is used, this issue may be caused by using a `block_size` that is larger than optimal. Reducing the block size may improve performance or using `range` and letting the runtime decide may also be an option.

Experimental Support for OpenMP* Applications

You can now trace, visualize, and analyze OpenMP* parallel regions, tasks, and task dependencies in your application with the Flow Graph Analyzer.

The Flow Graph Analyzer support for OpenMP technology is experimental and currently covers two basic scenarios:

- OpenMP parallel regions are nested inside a Intel® oneAPI Threading Building Blocks (oneTBB) flow graph. For this case, the Flow Graph Analyzer shows the execution of the parallel regions in the per-thread task execution timelines.

The sample code below, `omp_nested.cpp`, is an example of an OpenMP *construct* nested inside a oneTBB flow graph:

```
#include "tbb/tbb.h"
#include "tbb/flow_graph.h"
#include <omp.h>
#include <iostream>
using namespace tbb;
using namespace tbb::flow;

int main() {
    graph g;
    const int size = 20;
    continue_node< continue_msg> hello( g,
        []( const continue_msg &) {
            std::cout << "Hello\n";
            tbb::parallel_for(0, size, 1, [=](int k) {
                std::cout << k << "\n"; });
        });
    continue_node< continue_msg> world( g,
        []( const continue_msg &)
            std::cout << " World\n";
        #pragma omp parallel forfor (int i=0; i<20; i++) {std::cout << i << "\n";}
    );
    make_edge(hello, world);
    hello.try_put(continue_msg());
    g.wait_for_all();
    return 0;
}
```

- OpenMP tasks that use `depends` clauses. In this, the Flow Graph Analyzer shows task execution in the timelines and provides experimental support that lets you see the dependency relationships between OpenMP tasks as a graph in the graph canvas.

The sample code below, `omp_depend.cpp`, is a hello-world example of OpenMP task dependencies:

```
#include <omp.h>
#include <iostream>

int main() {
    #pragma omp parallel
    {
        std::string s = "";
        #prgma omp single
```

```

{
    #pragma omp task depend( out: i)
    {
        s = "hello";
        printf("%s", s);
    }
    #pragma omp task depend( out: s )
    {
        s = "world";
        printf("%s",s);
    }
}
}
return 0;
}

```

Collecting Traces for OpenMP* Applications

NOTE

OpenMP* trace collection is currently supported only on Linux* and macOS* operating systems.

To collect OpenMP traces for an application, you need an OMPT-enabled OpenMP* version 5.0 library, such as [LLVM-OpenMP](#).

To build a sample code (for example, the `omp_depend.cpp` described in the [Experimental Support for OpenMP* Applications](#)) on a Linux OS with the Intel® C++ Compiler Classic, use the following command:

```
icpc -std=c++11 -qopenmp omp_depend.cpp -o example
```

NOTE Please use `-g` compiler flag to enable symbol resolution information.

To collect traces for OpenMP applications and create XML files, follow these steps:

1. Set the `FGT_ROOT` variable and update your `PATH` environment variable as shown in the table below.

OS	Environment Variable	Value	Description
Linux*	FGT_ROOT	<advisor-install-dir>/fgt<version>/fgt	The path to the Flow Graph Collector installation
	PATH	\${FGT_ROOT}/linux/ bin:\${FGT_ROOT}/ linux/bin/ia32/ cc4.8_libc2.19_ker nel3.13.0:\$ {FGT_ROOT}/ linux/bin/intel64/ cc4.8_libc2.19_ker nel3.13.0:\${PATH}	The path must include paths to <code>fgtrun.sh</code> , <code>fgtrun.csh</code> , and <code>fgt2xml.exe</code> .

2. To enable tracing from OMPT, set the following variables:

OS	Environment Variable	Value	Description
Linux*	OMP_TOOL	Enabled	OMPT tool support enabler.
	OMP_TOOL_LIBRARIES	\${FGT_ROOT}/ linux/lib/intel64/ cc4.8_libc2.19_ker nel3.13.0/ libfgt.so	Path to OpenMP collector library.

3. Run the application.

If your paths are set up correctly, the application generates one or more files that start with `_fgt`. There is one file per thread that participates in executing the parallelism in the application. So, for example, if two threads participate in the execution of the flow graph, running the application generates two files, `_fgt.0` and `_fgt.1`, in an autogenerated folder in the format `_fga_YYYYMMDD_HHMMSS` according to its creation (for example, `_fga_20191111_1111`).

4. Convert the trace files to GraphML* and TraceML* format.

Convert the `_fgt` binary files to the XML format understood by the Flow Graph Analyzer using the `fgt2xml.exe` converter in the directory containing the folder with the trace files:

```
fgt2xml.exe <desired_name> --omp_experimental
```

or

```
fgt2xml.exe --omp_experimental <desired_name>
```

or

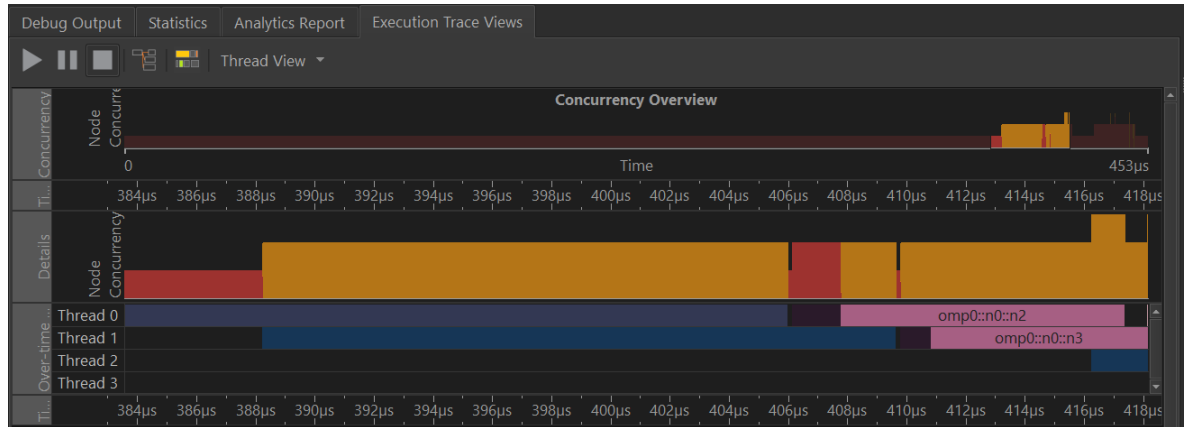
```
fgt2xml.exe --omp_experimental
```

This converter scans the current directory for all `_fgt` files within the most recent folder according to its name and generates two output files: `<desired_name>.graphml` and `<desired_name>.traceml`. If you do not provide a `<desired_name>`, the converter creates `unknown.graphml` and `unknown.traceml`.

The `omp_experimental` flag enables displaying a subgraph and tasks dependence graph. By default, display support is disabled and you see only information related to OpenMP constructs in the per-thread traces.

OpenMP* Constructs in the Per-Thread Task View

After you run the steps from the [Collecting Traces for OpenMP* Applications](#) section, you can see the execution of the OpenMP* tasks in the per-thread timelines. A display example for the `omp_depend.cpp` sample is shown below. The OpenMP region names are prefixed with `omp`.



OpenMP* Constructs in the Graph Canvas

To map OpenMP* parallel regions and task constructs to a graph, run the `fgt2xml` converter with the `--omp_experimental` flag. In such graph, nodes represent parallel regions and tasks, and edges represent task dependencies.

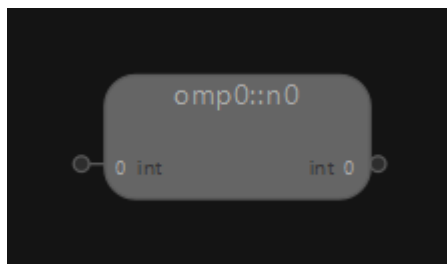
Parallel Regions

All OpenMP-related parallelism is contained within OpenMP parallel regions. In the Flow Graph Analyzer, a parallel region is mapped to a subgraph node in the graph canvas. Inside the subgraph node are at least two nodes:

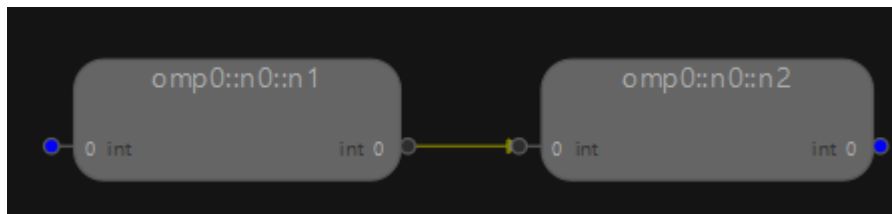
- A node that represents the start of the parallel region.
- A node that represents the implicit barrier at the end of the region.

For example, for an empty parallel region like the following, the Flow Graph Analyzer creates a subgraph node, such as `omp0::n0`, in the graph canvas.

```
#pragma omp parallel
{
}
```



When you double-click the subgraph node, you see the following, where `omp0::n0::n1` is the start of the parallel region and `omp0::n0::n2` is the implicit barrier at the end of the node.

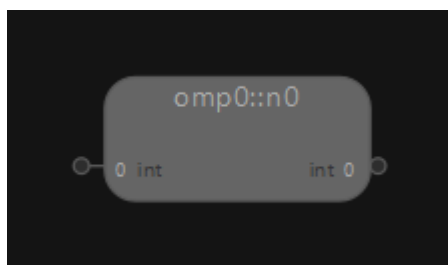


OpenMP* Tasks

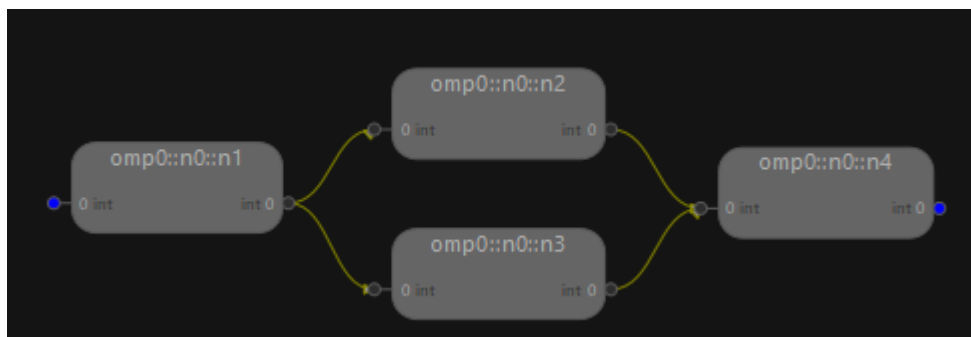
An OpenMP* task is a block of code contained in a parallel region that can be executed simultaneously with other tasks in the same region. In the Flow Graph Analyzer, an OpenMP task is mapped to a generic node. For example, in the code below, there are two tasks: one prints `hello` and the other prints `world`. The order in which these tasks execute is not specified, so they can execute in any order. However, the two tasks always start after the enclosing parallel region begins, and they complete before the enclosing parallel region ends.

```
#pragma omp parallel
{
  #pragma omp task
  { printf("hello "); }
  #pragma omp task
  { printf("world "); }
}
```

When you visualize this program in the Flow Graph Analyzer, it looks like this:



When you double-click this subgraph, you see the following, where `omp0::n0::n1` is the start of the parallel region, `omp0::n0::n4` is the implicit barrier at the end of the region, `omp0::n0::n2` is the "hello" task and `omp0::n0::n3` is the "world" task.



OpenMP* Task Dependencies

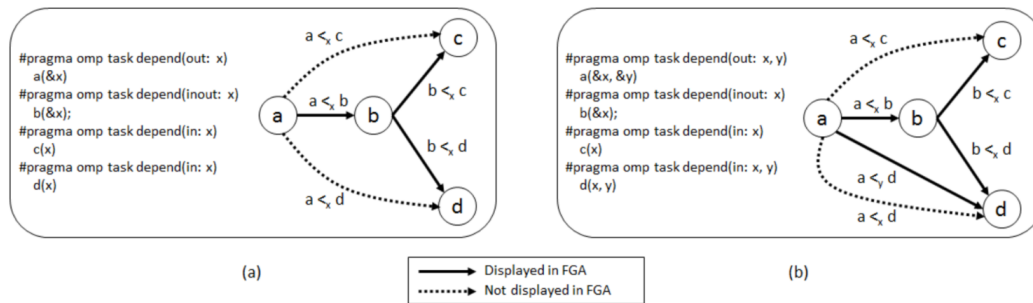
In the OpenMP* specification, a partial ordering of tasks can be expressed with `depend` clauses. The task dependence is fulfilled when the predecessor task completes. There are three dependency types supported by the OpenMP API: *in*, *out*, and *in-out*:

- *in* dependency type: The generated task is a dependent task of all previously generated sibling tasks that reference at least one of the list items in an *out* or *in-out* clause.
- *out* and *in-out* dependency types: The generated task is a dependent task of all previously generated sibling tasks that reference at least one of the list items in an *in*, *out*, or *in-out* clause.

In the Flow Graph Analyzer, task dependencies are represented by edges between the nodes that represent OpenMP tasks.

It is important to understand what dependencies are visualized in the Flow Graph Analyzer.

- The task dependency graph represents the partial order set by the `depend` clauses for the OpenMP tasks executed by the application. The nodes in the graphs are OpenMP tasks and the edges represent the partial order.
- To reduce the complexity of the graph, the Flow Graph Analyzer omits some **transitive** dependencies. A transitive dependence is a dependency between three tasks, such that if it holds between the first and the second tasks and between the second and the third tasks, it must hold between the first and the third tasks. In the figure below, the node **a** must execute before the node **b** in the partial order due to a dependency on the location **x** as $a <_x b$.



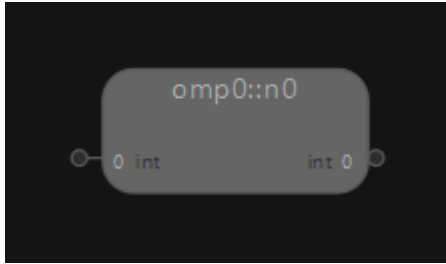
- Part (a) of the figure shows an example that only includes dependencies due to a single location **x**. Because $a <_x b$ and $b <_x d$, the Flow Graph Analyzer does not show the **transitive** edge $a <_x d$.
- Part (b) of the figure shows two locations **x** and **y** that determine the partial order. There are two potential dependency edges from **a** to **d**: $a <_x d$ and $a <_y d$. The Flow Graph Analyzer includes an edge from **a** to **d** because **a** is the direct source of **y** for **d**, but it excludes $a <_x d$.

NOTE If there are parallel edges between two nodes and at least one of them can be omitted due to transitivity, they all can be omitted without changing the partial order. The Flow Graph Analyzer includes edges like $a <_y d$ in the graph topology because including edges to satisfy all required data dependencies is the most natural representation.

For example:

```
#pragma omp parallel
{
    std::string s = "";
    #prgma omp single
    {
        #pragma omp task depend( out: s)
        {
            s = "hello";
            printf("%s", s);
        }
        #pragma omp task depend( out: s )
        {
            s = "world";
            printf("%s",s);
        }
    }
}
```

This application, when visualized with the Flow Graph Analyzer, has a single top-level subgraph node representing the OpenMP parallel region.



When you double-click this subgraph, you see the following:



The edge between `omp0::n0::n2` and `omp0::n0::n3` represents task dependency due to the variable `s`.

The main components of the Flow Graph Analyzer include the treemap view, the graph-topology canvas, the timeline and concurrency histogram view, and the critical-path report. OpenMP task traces map naturally to these views:

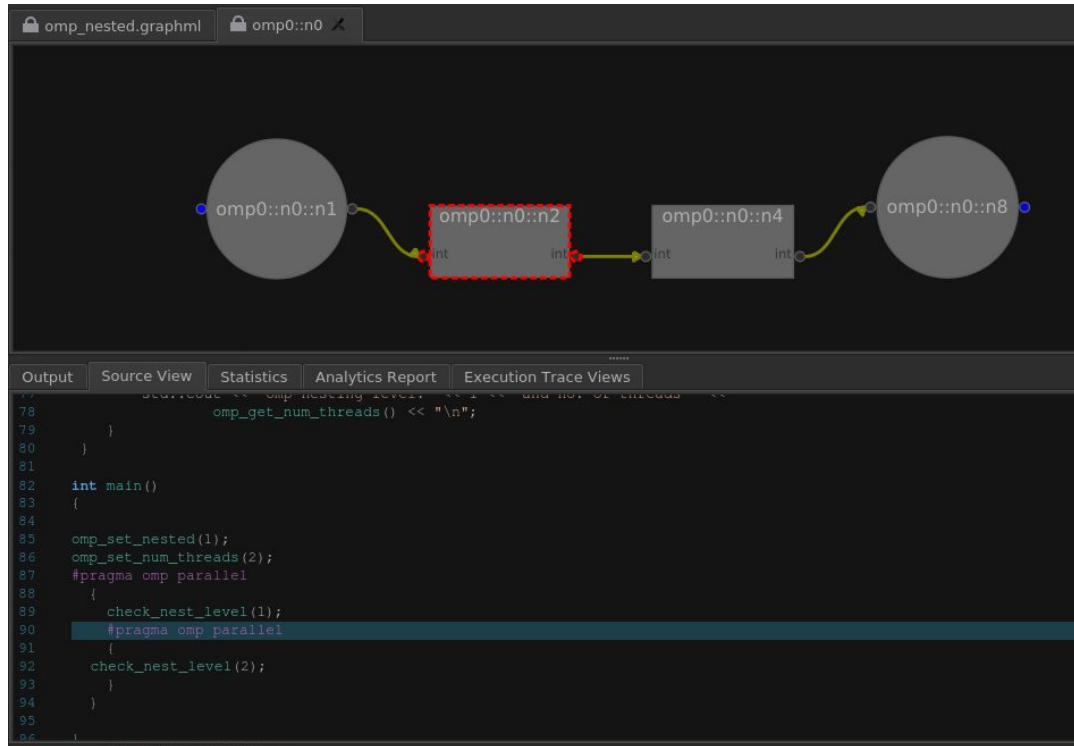
- The treemap view shows the time spent in each OpenMP parallel region, colored according to the average application concurrency during the time it was executing.
- The graph topology canvas shows the partial ordering of the tasks.
- The timeline and concurrency histogram view show the execution of each task on the OpenMP runtime threads and the application concurrency over time.
- The critical report shows the most time-consuming path from each source to each sink in the graph, sorted with the longest critical path at the top.

For more examples, see https://link.springer.com/chapter/10.1007/978-3-319-98521-3_12.

OpenMP* Nodes to Source Code Mapping:

In addition to the graphical view of OpenMP* task dependency graphs, the Flow Graph Analyzer also shows nodes mapping to corresponding source code. To get this information, you must build an OpenMP application with the `-g` flag.

For example, source code mapping with subgraph nodes in a parallel region looks as follows:



Sample Trace Files

The Flow Graph Analyzer includes five sample traces you can explore to get familiar with the features of the tool. These traces are in the `<advisor_installation>/fga/samples` directory. Whenever you launch the Flow Graph Analyzer, the **File** dialog box defaults to the `samples` directory.

The `samples` subdirectories contain samples you can load. The samples are described in the table below and explained in more detail in the sections that follow.

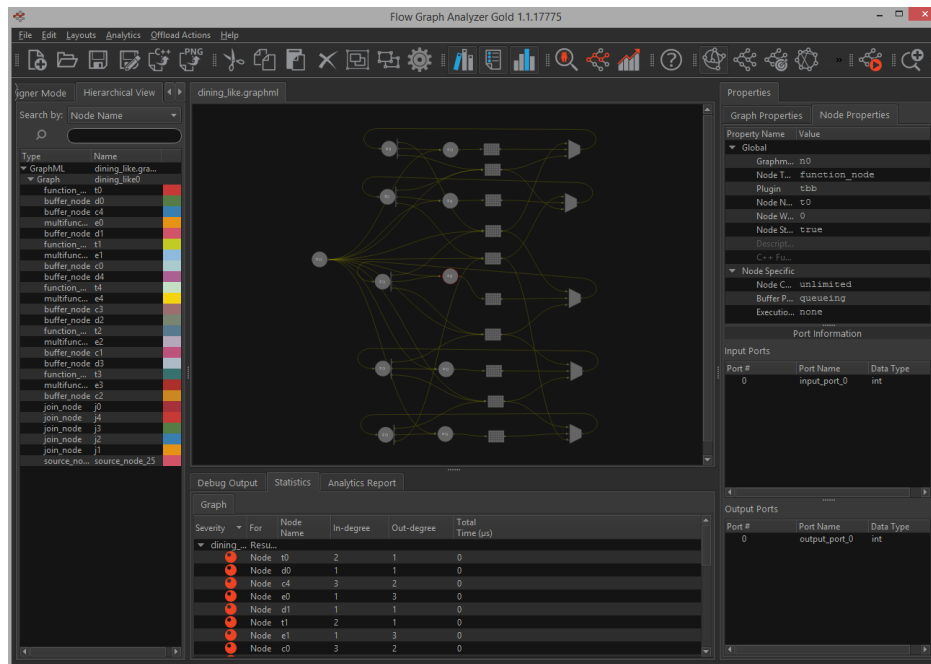
Location	XML files	Notes
samples/code_generation	dining_like.graphml	Generates a Dining Philosophers sample.
	feature_like.graphml	Generates a Features Detection sample.
samples/performance_analysis	feature_detection.graphml	Provides a runtime trace of a feature detection sample.
	feature_detection.traceml	
	forward_substitution.graphml	Provides a runtime trace of a forward substitution sample.
	forward_substitution.traceml	
	computer_vision.graphml	Provides a runtime trace of a computer vision sample.
	computer_vision.traceml	

NOTE The `performance_analysis` samples were captured by runtime tracing. Because runtime tracing cannot infer all types and cannot capture the user bodies of nodes, these samples do not contain complete descriptions of applications and cannot be used to regenerate the applications. You can generate a C++ code from these samples, but it will be incomplete and will need modification before compilation and execution. In contrast, the `code_generation` samples were completely described from within Flow Graph Analyzer. When you generate a C++ code from the `code_generation` samples, no modifications are necessary before compilation and execution.

code_generation Samples

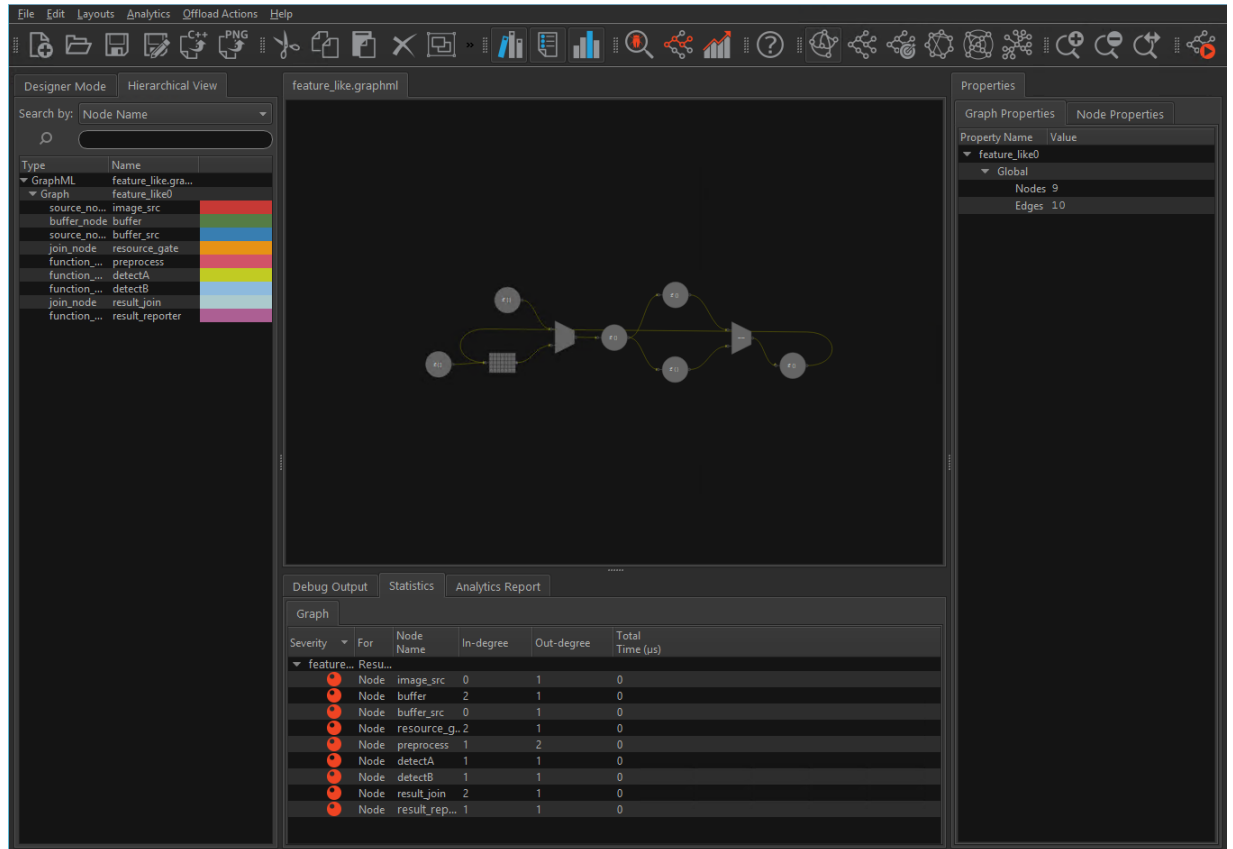
Dining Philosophers

The `dining_like.graphml` sample provides a complete description of a Intel® oneAPI Threading Building Blocks (oneTBB) flow graph application that implements a version of the dining philosophers problem. You can generate a complete oneTBB flow graph by loading this file in to the Flow Graph Analyzer and then following the instructions provided in the [Generating C++ Stubs](#) section.



Feature Detection

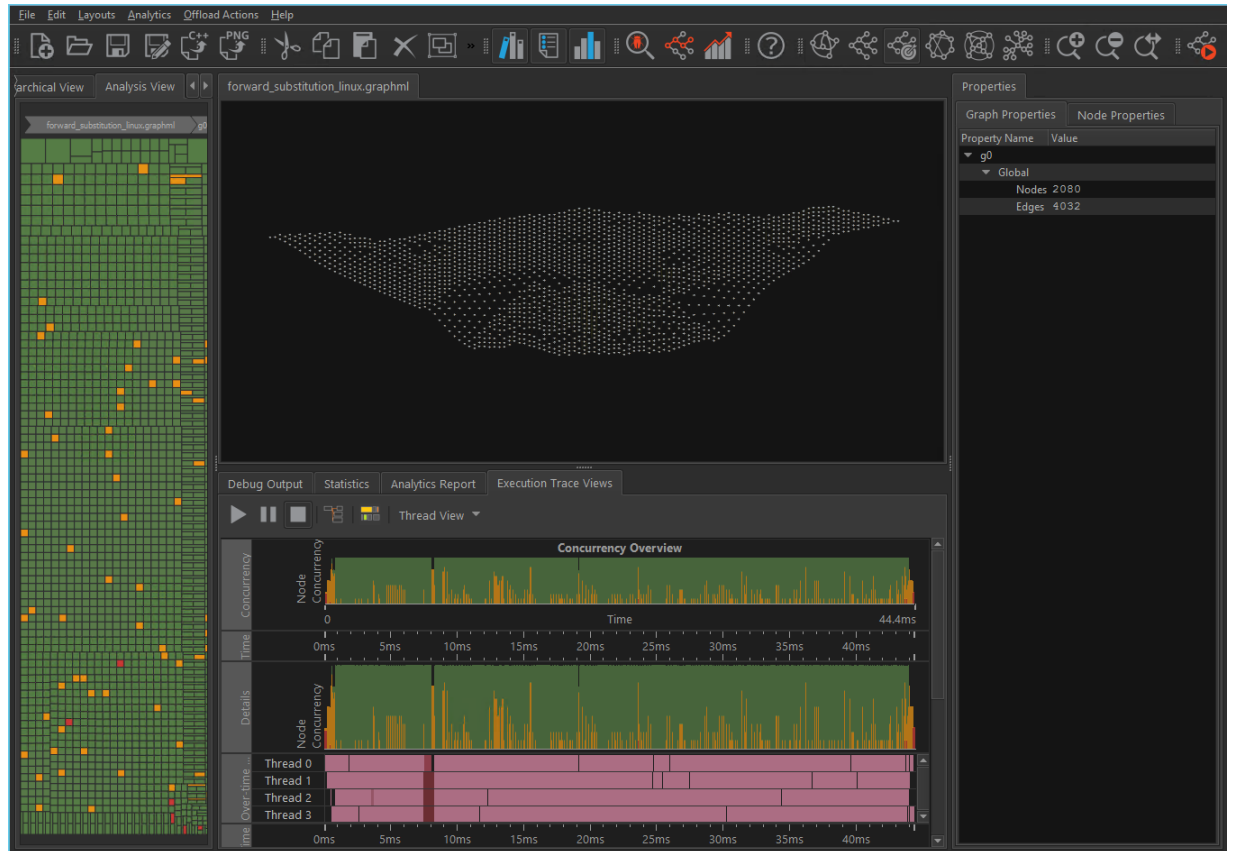
The `feature_like.graphml` sample provides a complete description of a feature detection application based on the example described in the blog posting at <https://software.intel.com/content/www/us/en/develop/blogs/a-feature-detection-example-using-the-intel-threading-building-blocks-flow-graph.html>. You can generate a complete oneTBB flow graph by loading this file into the Flow Graph Analyzer and then following the instructions provided in the [Generating C++ Stubs](#) section.



performance_analysis Samples

Forward Substitution with Trace

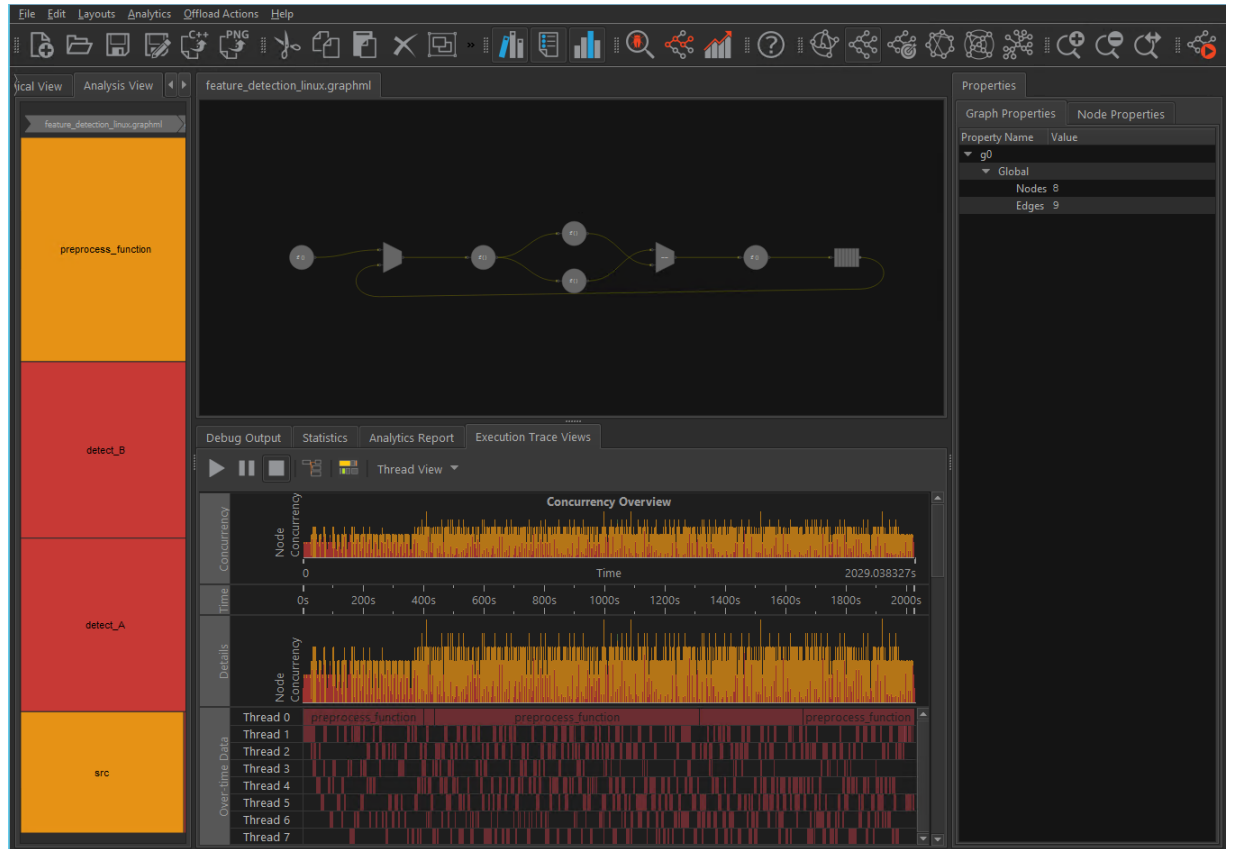
The `forward_substitution.graphml` sample shows the topology and behavior of a Intel® oneAPI Threading Building Blocks (oneTBB) flow graph application that provides an implementation of forward substitution on a lower-triangular matrix. The trace is for a single execution of the graph, using 4 threads for a 8192x8192 matrix with a block size of 128. The runtime trace of the application is contained in the matching `forward_substitution.traceml` file. This matching file is loaded automatically by the Flow Graph Analyzer.



Feature Detection with Trace

The `feature_detection.graphml` sample shows the topology and behavior of a oneTBB flow graph application.

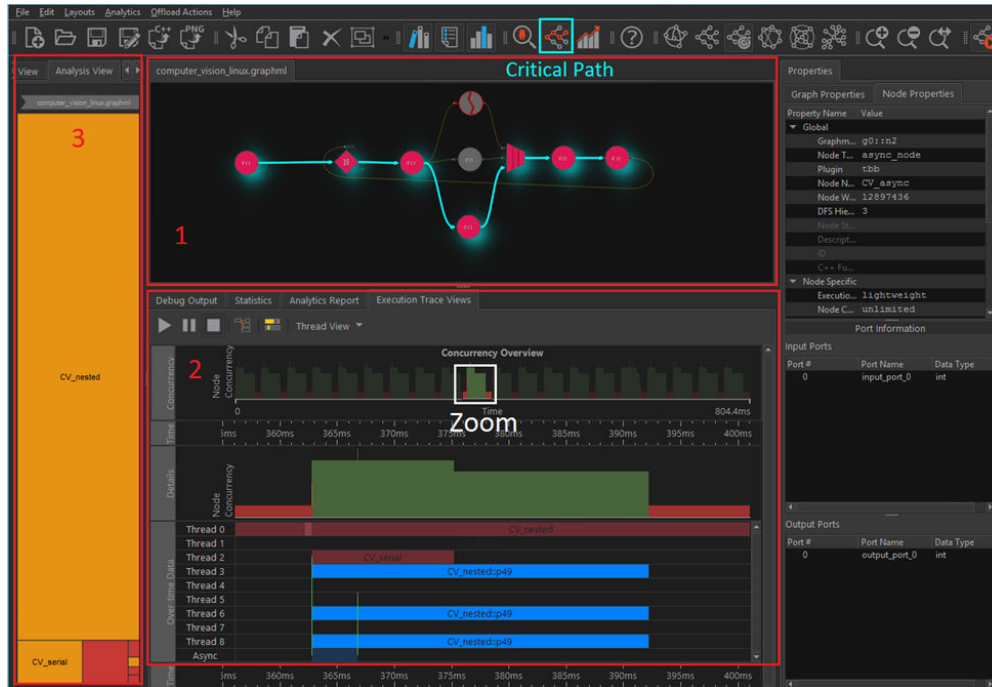
This trace was collected using 8 threads and 32 buffers provided to the buffer queue. The concurrency varies over time, but is limited to 8 threads at most.



Computer Vision with Trace

The `computer_vision.graphml` sample shows the topology and behavior of a oneTBB flow graph application that represents a classic example of data flow parallelism. It is composed of three different computer vision (CV) algorithms that process the same input data. The data is a video input stream, and you can observe a resulting regular pattern in the timeline chart (the trace contains around 20 frames).

Notice the following:



Red outlined area #1

You can use the critical path calculation functionality (turquoise box) to identify bottlenecks in the data flow. As a result of this feature, all nodes on the critical path are highlighted.

White box in the #2 area

Zoom in the timeline to analyze a single frame execution in detail. The frame execution flow is the following:

1. The source node spawns a task. This is the first stage of the image processing pipeline.
2. A limiter node is used to balance the pipeline. It forwards the frame only if the number of frames that are currently executed is below a user-specified threshold.
3. Three different algorithms are executed in parallel. Concurrency changes during the algorithm stage because less work is available. In the timeline, a high concurrency is colored in green.

Lower part of red outlined area #2

For a oneTBB flow graph, an external activity can be encapsulated in a predefined async node. This activity represents offloading work to an Accelerator (for example, FPGA, GPU). The beginning and end of this activity are displayed as green vertical lines in the timeline. You can find a single execution within a single frame for each CV algorithm (represented by the nodes CV serial, CV nested, CV async). CV nested represents a node with a nested oneTBB parallel for algorithm that consumes most of the CPU time on average.

Red outlined area #3

The Treemap shows the average node weight. `CV_nested` includes a `oneTBBparallel_for` algorithm and consumes most of the CPU time.

Additional Resources

- Flow Graph Analyzer is delivered with the Intel® Advisor. You can find out more information about Intel® Advisor and instructions on how to obtain it and its components at <https://software.intel.com/en-us/oneapi/advisor>.
- You can download or find out more about the Intel® oneAPI Threading Building Blocks library at <https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onetbb.html> or <https://github.com/oneapi-src/oneTBB>.

Minimize Analysis Overhead

Running your target application with the Intel® Advisor can take substantially longer than running your target application without the Intel® Advisor. Depending on an accuracy level and analyses you choose for a perspective, different overhead is added to your application execution time. For example:

Runtime Overhead / Analysis	Survey	Characterization	Dependencies	MAP
Target application runtime with Intel® Advisor compared to runtime without Intel® Advisor	1.1x longer	2 - 55x longer	5 - 100x longer	5 - 20x longer

The following tables summarize how to minimize overhead while collecting and finalizing Intel® Advisor analysis data.

Collection Controls

The following table is a summary. For more information, see [Collection Controls to Minimize Analysis Overhead](#).



Minimization Technique	Impacted Analyses	Summary
Pause collection/ Resume collection using API methods	<ul style="list-style-type: none"> Survey Characterization 	<p>Pause collection:</p> <ul style="list-style-type: none"> C++: <code>__itt_pause</code> Fortran: Use <code>ITTNOTIFY</code> statement to call <code>ITT_PAUSE()</code> subroutine <p>Resume collection:</p> <ul style="list-style-type: none"> C++: <code>__itt_resume</code> Fortran: Use <code>ITTNOTIFY</code> statement to call <code>ITT_RESUME()</code> subroutine
Pause collection/ Resume collection using annotations	<ul style="list-style-type: none"> Survey Dependencies <p>Some analysis types recognize the structural annotations typically used in</p>	<p>Pause collection:</p> <ul style="list-style-type: none"> C++: <code>ANNOTATE_DISABLE_COLLECTION_PUSH</code> Fortran: call <code>annotate_disable_collection_push()</code> C#: <code>Annotate.DisableCollectionPush();</code> <p>Resume collection:</p> <ul style="list-style-type: none"> C++: <code>ANNOTATE_DISABLE_COLLECTION_POP</code> Fortran: call <code>annotate_disable_collection_pop()</code> C#: <code>Annotate.DisableCollectionPop();</code>

Minimization Technique	Impacted Analyses	Summary
	the Threading perspective workflow.	<hr/> NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1. <hr/>
Start target application with collection paused	<ul style="list-style-type: none"> Survey Characterization 	<p>Start target application with collection paused:</p> <ul style="list-style-type: none"> GUI control: Workflow pane > Start paused control advisor CLI action option: <code>-start-paused</code> <hr/> NOTE You can use different techniques to resume collection. The most common is <code>__itt_resume</code> <hr/>
Start target application with collection paused/ Resume collection after N seconds	<ul style="list-style-type: none"> Survey Characterization 	<p>GUI control: Project Properties > Analysis Target > [Name] Analysis > Advanced > Automatically resume collection after (sec) checkbox</p> <p>advisor CLI action option: <code>-resume-after=<integer></code></p>
Stop collection after N seconds	All	<p>GUI control: Project Properties > Analysis Target > [Name] Analysis > Advanced > Automatically stop collection after (sec) checkbox and field</p> <p>advisor CLI action: <code>-stop-after=<integer></code></p>
Stop collection	All	<p>GUI control: Workflow pane > Stop current analysis control and Site Coverage widget</p> <p>advisor CLI: action option: <code>-command=stop</code></p>
Manually pause collection/ Manually resume collection	<ul style="list-style-type: none"> Survey Characterization 	<p>Pause collection:</p> <ul style="list-style-type: none"> GUI control: Workflow pane > Pause control advisor CLI action: <code>-command=pause</code> <p>Resume collection:</p> <ul style="list-style-type: none"> GUI control: Workflow pane > Resume control advisor CLI action: <code>-command=resume</code>
Attach to process/ Detach from process	<ul style="list-style-type: none"> Survey Characterization 	<p>Attach to process:</p> <ul style="list-style-type: none"> GUI control: Project Properties > Analysis Target > [Name] Analysis > Launch Application drop-down list > Attach to Process advisor CLI action options: <code>-target-pid=<unsigned integer></code> and <code>-target-process=<string></code> <p>Detach from process:</p>

Minimization Technique	Impacted Analyses	Summary
		<ul style="list-style-type: none"> GUI control: Workflow pane > Stop current analysis control advisor CLI action: <code>-command=detach</code>

Loop Markup

The following table is a summary. For more information, see [Loop Markup to Minimize Analysis Overhead](#).

Minimization Technique	Impacted Intel Advisor Analyses	Summary
Select loops by ID	<ul style="list-style-type: none"> Characterization Dependencies Memory Access Patterns 	GUI control: Survey Report  checkbox(es) advisor CLI action option: <code>-mark-up-list=<string></code>
Select loops by source file/line	<ul style="list-style-type: none"> Characterization Dependencies Memory Access Patterns 	GUI control: Survey Report  checkbox(es) advisor CLI action: <code>-mark-up-loops</code> with action option <code>-select=<string></code>
Select loops by criteria	<ul style="list-style-type: none"> Dependencies Memory Access Patterns 	advisor CLI: action <code>-mark-up-loops</code> or <code>-collect</code> with action option <code>-loops=<string></code>

Filtering

The following table is a summary. For more information, see [Filtering to Minimize Analysis Overhead](#).

Minimization Technique	Impacted Intel Advisor Analyses	Summary
Filter modules	<ul style="list-style-type: none"> Survey Characterization 	GUI control: Project Properties > Analysis Target > [Name] Analysis > Modules options and field advisor CLI: action option: <code>-module-filter-mode=include exclude</code> and <code>-module-filter=<string></code>

Execution Speed/Duration/Scope Properties

The following table is a summary. For more information, see [Execution Speed/Duration/Scope Properties to Minimize Analysis Overhead](#).

Minimization Technique	Impacted Intel Advisor Analyses	Summary
Change stackwalk mode from offline (after collection) to	Survey	GUI control: Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Stack unwinding mode > During collection advisor CLI action option: <code>-stackwalk-mode=online</code>

Minimization Technique	Impacted Intel Advisor Analyses	Summary
online (during collection)		
Disable stacks collection	<ul style="list-style-type: none"> Characterization 	<p>GUI controls:</p> <ul style="list-style-type: none"> Vectorization Workflow pane > Enable Roofline with Callstacks checkbox Project Properties > Analysis Target > Trip Counts and FLOP Analysis > Advanced > Collect stacks checkbox <p>advisor CLI action option: <code>-no-stacks</code> (or just ensure the CLI action option <code>-stacks</code> is omitted from the advisor command line)</p>
Disable stitch stacks	Survey	<p>GUI control: Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Stitch stacks checkbox</p> <p>advisor CLI action option: <code>-no-stack-stitching</code></p>
Increase sampling interval	Survey	<p>GUI control: Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Sampling interval field</p> <p>advisor CLI action option: <code>interval=<integer></code></p>
Limit collected analysis data	Survey	<p>GUI control: Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Collection data limit, MB field</p> <p>advisor CLI action option: <code>-data-limit=<integer></code></p>
Limit loop call count	<ul style="list-style-type: none"> Dependencies Memory Access Patterns 	<p>GUI control: Project Properties > Analysis Target > [Name] Analysis > Advanced > Loop Call Count Limit field</p> <p>advisor CLI action option: <code>-loop-call-count-limit=<integer></code></p>
Disable additional analysis	Survey	<p>GUI controls: Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced...</p> <ul style="list-style-type: none"> Analyze MKL loops and functions checkbox Analyze Python loops and functions checkbox Analyze loops that reside in non-executed code paths checkbox Enable register spill/fill analysis checkbox Enable static instruction mix analysis checkbox <p>advisor CLI action options:</p> <ul style="list-style-type: none"> <code>-no-mkl-user-mode</code> <code>-no-profile-python</code> <code>-no-support-multi-isa-binaries</code> <code>-no-spill-analysis</code> <code>-no-static-instruction-mix</code>

Miscellaneous Techniques

The following table is a summary. For more information, see [Miscellaneous Techniques to Minimize Analysis Overhead](#).

Minimization Technique	Impacted Intel Advisor Analyses	Summary
Disable cache simulation	<ul style="list-style-type: none"> Characterization Memory Access Patterns 	<p>GUI controls:</p> <ul style="list-style-type: none"> Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Enable cache simulation checkbox Project Properties > Analysis Target > Trip Counts and FLOP Analysis > Advanced > Enable cache simulation checkbox <p>advisor CLI action option: <code>-no-enable-cache-simulation</code></p>
Limit reported data	Memory Access Patterns	<p>GUI controls:</p> <ul style="list-style-type: none"> Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Report stack variables checkbox Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Report heap allocated variables checkbox <p>advisor CLI action options:</p> <ul style="list-style-type: none"> <code>-no-record-stack-frame</code> <code>-no-record-mem-allocations</code>
Minimize data set	<p>All, but especially:</p> <ul style="list-style-type: none"> Dependencies Memory Access Patterns 	Minimize number of instructions executed within a loop while thoroughly exercising target application control flow paths
Temporarily disable finalization until opening result in GUI	<ul style="list-style-type: none"> Survey Characterization 	<p>GUI control: Workflow pane > Cancel current analysis control during finalization</p> <p>advisor CLI action option: <code>-no-auto-finalize</code></p>

Collection Controls to Minimize Analysis Overhead

Issue

Running your target application with the Intel® Advisor can take substantially longer than running your target application without the Intel Advisor. Depending on an accuracy level and analyses you choose for a perspective, different overhead is added to your application execution time. For example:

Runtime Overhead / Analysis	Survey	Characterization	Dependencies	MAP
Target application runtime with Intel® Advisor compared to runtime without Intel® Advisor	1.1x longer	2 - 55x longer	5 - 100x longer	5 - 20x longer

Solutions

Use the following techniques to skip *uninteresting* parts of your target application, such as the initialization phase, and analyze only *interesting* parts.

Pause Collection/Resume Collection Using Annotations

Minimize collection overhead.

Applicable analyses: Survey, Dependencies.

Some analysis types recognize the structural annotations typically used in the Threading perspective workflow.

Use when:

- Modifying/recompiling your target application is not an issue.
- You do not want to analyze one or more uninteresting parts of your target application.
- The interesting parts of your target application involve large workloads (because Pause/Resume API call frequency is about 1 Hz, and the operations pause and resume data collection in all processes in the analysis run, with the corresponding collection state notification to the GUI).

To pause collection, add the following annotation to your code:

- C++: `ANNOTATE_DISABLE_COLLECTION_PUSH`
- Fortran: `call annotate_disable_collection_push()`
- C#: `Annotate.DisableCollectionPush();`

To resume collection, add the following annotation to your code:

- C++: `ANNOTATE_DISABLE_COLLECTION_POP`
- Fortran: `call annotate_disable_collection_pop()`
- C#: `Annotate.DisableCollectionPop();`

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

See [Pause Collection and Resume Collection Annotations](#) for more information.

Pause Collection/Resume Collection Using API Methods

Minimize collection overhead using the Intel® Instrumentation and Tracing Technology (ITT) API.

Tip For MPI applications, you can also use a standard MPI-specific function `MPI_Pcontrol(<N>)` to pause and resume data collection. This function requires no changes in the application building process, unlike the ITT API calls, which require linking of a static ITT API library. See [Control Collection with an MPI_Pcontrol Function](#).

Applicable analyses: Survey, Characterization with Trip Counts or FLOP collection enabled.

Use when:

- Modifying/recompiling your target application is not an issue.
- You do not want to analyze one or more uninteresting parts of your target application.
- The interesting parts of your target application involve large workloads (because Pause/Resume API call frequency is about 1 Hz, and the operations pause and resume data collection in all processes in the analysis run, with the corresponding collection state notification to the GUI).

Prerequisites:

- Add the following statements to every source file you want to instrument:
 - C/C++: `ittnotify.h`
 - Fortran: `USE ITTNOTIFY`

NOTE

- The `ittnotify` header file contains definitions of ITT API routines and important macros that provide the correct logic of API invocation from an application.
 - The ITT API is designed to incur almost zero overhead when tracing is disabled. If you need completely zero overhead, you can compile out all ITT API calls from an application by defining the `INTEL_NO_ITTNOTIFY_API` macro in your project at compile time, either on the compiler command line or in your source file prior to including the `ittnotify` header file.
-

- Configure your build system to reach ITT API header file and libraries, where `<install-dir>` is the Intel Advisor installation directory.
 - Add the appropriate entry to your INCLUDE path:
 - C++: `<install-dir>/sdk/include`
 - Fortran: `<install-dir>/sdk/include/lib32` or `<install-dir>/sdk/include/lib64`
 - Microsoft Visual Studio* IDE: **Project Properties > C/C++ | Fortran > General > Additional Include Directories**.
 - Add `<install-dir>/sdk/lib32` or `<install-dir>/sdk/lib64` to your LIBRARIES path.

Visual Studio IDE: **Project Properties > C/C++ | Fortran > General > Additional Include Libraries**.

NOTE The ITT API headers, static libraries, and Fortran modules previously located at `<install-dir>/include` and `<install-dir>/[lib32 | lib64]` folders were moved to the `<install-dir>/sdk/include` and `<install-dir>/sdk/[lib32 | lib64]` folder. Copies of these files are retained at their old locations for backward compatibility and these copies should not be used for new projects.

- Link your target application to the static library `libittnotify.a` (Linux* OS) or `libittnotify.lib` (Windows* OS) by passing `-littnotify` to your compiler. If tracing is enabled, this static library loads the ITT API implementation and forwards ITT API instrument data to the Intel Advisor. If tracing is disabled, the static library ignores ITT API calls, providing nearly zero instrumentation overhead.

Visual Studio IDE: **Project Properties > Linker > Input > Additional Dependencies**

- Insert `_itt_pause` (C/C++) or `CALL ITT_PAUSE` (Fortran) before uninteresting parts of your target application and the `_itt_resume` (C/C++) or `CALL ITT_RESUME` (Fortran) before interesting parts of your target application.

Example 1: The following snippet plus the standard run control collects analysis data twice - at the beginning and the middle of the snippet:

```
#include <ittnotify.h>
int main(int argc, char* argv[])
{
    // Do work here
    _itt_pause();
    // Do uninteresting work here
    _itt_resume();
    // Do work here
    _itt_pause();
    // Do uninteresting work here
    return 0;
}
```

Example 2: The following snippet plus the standard run control collects analysis data only once - in the middle of the snippet:

```
#include <ittnotify.h>
int main(int argc, char* argv[])
{
    _itt_pause();
    // Do uninteresting work here
    _itt_resume();
    // Do work here
    _itt_pause();
    // Do uninteresting work here
    return 0;
}
```

Example 3: The following snippet plus the standard run control collects analysis data only once - at the end of the snippet:

```
#include <ittnotify.h>
int main(int argc, char* argv[])
{
    _itt_pause();
    // Do uninteresting work here
    _itt_resume();
    // Do work here
    return 0;
}
```




Example 4: The following snippet plus the **Start Paused** control collects analysis data only once - at the end of the snippet:

```
#include <ittnotify.h>
int main(int argc, char* argv[])
{
    // Do uninteresting work here
    _itt_resume();
    // Do work here
    return 0;
}
```

After performing the prerequisites and recompiling, do one of the following:

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

- Click the standard run control or the  **Start Paused** control on the **Analysis Workflow** pane to run the desired analysis.
- Use the advisor CLI action `--collect` with or without the CLI action option `--start-paused` to run the desired analysis. For example:

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

To attach ITT APIs to a launched application, that is, to collect API data on an application that is already launched, point the target application to the `ittnotify_collector` library using an environment variable:

- Windows* OS:

```
set INTEL_LIBITTTNOTIFY32=<install_dir>\bin32\runtime\ittnotify_collector.dll
set INTEL_LIBITTTNOTIFY64=<install_dir>\bin64\runtime\ittnotify_collector.dll
```

- Linux* OS:

```
export INTEL_LIBITTTNOTIFY32=<install_dir>/lib32/runtime/libittnotify_collector.so
export INTEL_LIBITTTNOTIFY64=<install_dir>/lib64/runtime/libittnotify_collector.so
```

NOTE

Use the full path to the library without quotations marks.

After you complete configuration, start the instrumented application in the correct environment. Intel® Advisor collects API data even if the application is launched before the Intel® Advisor is launched.

You can find the *ITT API documentation* at <https://github.com/intel/ittapi>.

Start Target Application With Collection Paused

Minimize collection overhead.

Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled.

Use when you do not want to analyze the early phase(s) of your target application, such as the initialization phase, but you want analysis in *ready mode*.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following:

-



Click the associated control on the **Analysis Workflow** pane to run the desired analysis.

- Use the `advisor` CLI action option `--start-paused` when you run the desired analysis. For example:

```
advisor --collect=survey --start-paused --project-dir=./advi_results -- ./myApplication
```

NOTE

You can use different techniques to resume collection. The most common is `__itt_resume`.

Start Target Application With Collection Paused/Resume Collection After N Seconds

Minimize collection overhead.

Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled.

Use when...

- You do not want to modify/recompile your target application.
- You do not want to analyze the initialization phase of your target application.
- You have a good idea of the time interval of interest, but pinpointing the exact beginning of the interesting part of your target application is not important.
- The interesting part of your target application is more than a few loops.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following:

- Enable the **Project Properties > Analysis Target > [Name] Analysis > Advanced > Automatically resume collection after (sec)** checkbox and supply the desired value, where *[Name]* is Survey or Trip Counts and FLOP.

Click the standard run control on the **Analysis Workflow** pane to run the desired analysis. (Collection automatically starts in the paused state.)

- Use the `advisor` CLI action option `--resume-after=<integer>` when you run the desired analysis. For example:

```
advisor --collect=survey --resume-after=30 --project-dir=./advi_results -- ./myApplication
```

NOTE

Use a value representing seconds in the GUI field and milliseconds in the integer argument.

Stop Collection After N Seconds

Minimize collection overhead.

Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled, Dependencies, Memory Access Patterns.

This is the flip side of the *Start target application with collection paused* technique. Use when...

- You do not want to modify/recompile your target application.
- You do not want to analyze the end of your target application.
- You have a good idea of the time interval of interest, but pinpointing the exact end of the interesting part of your target application is not important.
- The interesting part of your target application is more than a few loops.

To implement, enable **Project Properties > Analysis Target > [Name] Analysis > Advanced > Automatically stop collection after (sec)** checkbox and supply the desired value, where *[Name]* is Survey or Trip Counts and FLOP.

Click the standard run control on the **Analysis Workflow** pane to run the desired analysis.

NOTE

Use a value representing seconds in both the GUI field and integer argument.

Stop Collection

Minimize collection overhead.

Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled, Dependencies, Memory Access Patterns.

Use when...

- You do not want to modify/recompile your target application.
- You do not want to analyze the end of your target application.
- You can detect the time interval of interest based on target application output.
- The interesting part of your target application is more than a few loops.

To implement, do one of the following:

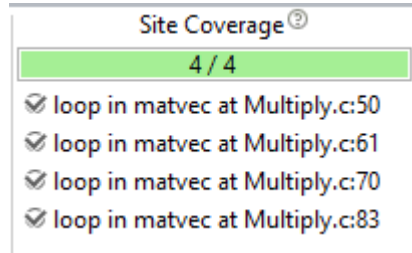
-



Click the associated control on the **Analysis Workflow** pane when running the desired analysis.

NOTE

If running a Dependencies or Memory Access Patterns analysis, use the **Site Coverage** bar to determine when all marked loops are analyzed at least once:



- Use the advisor CLI action `--command=stop` when you run the desired analysis. For example:

```
advisor --command=stop --result-dir=./myAdvisorResult
```

Manually Pause Collection/Manually Resume Collection



Minimize collection overhead.

Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled.

Use when...

- You can detect the time interval of interest based on target application output.
- Your need to pause or resume is unplanned and spontaneous.

To implement, do one of the following to pause analysis data collection (the target application continues running, but analysis data collection stops):

-  Click the associated  control on the **Analysis Workflow** pane when running the desired analysis.
- Use the advisor CLI action `--command=pause` when you run the desired analysis. For example:

```
advisor --command=pause --result-dir=./myAdvisorResult
```

Do one of the following to resume analysis data collection:

-  Click the associated  control on the **Analysis Workflow** pane.
- Use the advisor CLI action `--command=resume`. For example:

```
advisor --command=resume --result-dir=./myAdvisorResult
```

Attach to Process/Detach from Process

Minimize collection overhead.

Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled without call stacks.

This technique is similar to the *Start target application with collection paused* technique, except you can attach to an already running process. This is particularly beneficial if:

- The process is a service that runs forever.
- The launching infrastructure is relatively complicated, such as a sequence of scripts that must be modified to embed a *launch collection* command.



GUI:

1. Choose **Project Properties > Analysis Target > [Name] Analysis > Launch Application** drop-down list > **Attach to Process**, where *[Name]* is Survey or Trip Counts and FLOP.
2. Disable the **Inherit settings from Survey Hotspots Analysis Type** checkbox.
3. Choose the **Process name** or **PID** option and identify a process.
4. Supply other information as desired and close the **Project Properties** dialog box.
5. Click the standard run control on the **Analysis Workflow** pane to run the desired analysis.

CLI: Use the advisor CLI action option `--target-pid=<unsigned integer>` or `--target-process=<string>` to attach to a process when running the desired analysis. For example:

```
advisor --collect=survey --project-dir=./advi_results --result-dir=./myAdvisorResult --target-process=myProcess
```

Do one of the following to stop collecting analysis data on a process (the process continues running but analysis data collection stops):

-  Click the associated  control on the **Analysis Workflow** pane.
- Use the advisor CLI action `--command=detach`. For example:

```
advisor --command=detach --result-dir=./myAdvisorResult
```

NOTE

- Ensure call stacks are disabled (which is the default setting) if you run the Characterization with Trip Counts and FLOP collection enabled analysis:
 - Disable the **Project Properties > Analysis Target > Trip Counts and FLOP Analysis > Advanced > Collect stacks** checkbox.
 - Add the advisor CLI action option `--no-stacks` to the `--collect` command, or simply omit a `--stacks` action option on the `--collect` command.
- Using the advisor CLI action `--command=stop` kills the process (which also stops analysis data collection).

See Also

[Loop Markup to Minimize Analysis Overhead](#)

[Filtering to Minimize Analysis Overhead](#)

[Execution Speed/Duration/Scope Properties to Minimize Analysis Overhead](#)

[Miscellaneous Techniques to Minimize Analysis Overhead](#)

Loop Markup to Minimize Analysis Overhead

Issue

Running your target application with the Intel® Advisor can take substantially longer than running your target application without the Intel® Advisor. Depending on an accuracy level and analyses you choose for a perspective, different overhead is added to your application execution time. For example:

Runtime Overhead / Analysis	Survey	Characteri zation	Dependen cies	MAP
Target application runtime with Intel® Advisor compared to runtime without Intel® Advisor	1.1x longer	2 - 55x longer	5 - 100x longer	5 - 20x longer

Solutions

Use the following techniques to skip *uninteresting* loops and analyze only *interesting* loops.

Select Loops by ID

Goal: Minimize collection overhead.

Applicable analyses: Characterization with Trip Counts and FLOP collection enabled, Dependencies, Memory Access Patterns.

Use when...

- You want to perform a deeper analysis on only a few loops.
- CLI environment: You cannot identify source file/line numbers, such as when you are analyzing a target application for which you do not have access to source code.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Prerequisites:

1. Run a Survey analysis.
2. `advisor` CLI environment: Identify the loop IDs for the loops of interest.

```
advisor --report=survey --project-dir=./advi_results -- ./myApplication
```


In the report, the first column is the loop IDs.

Tip

Intel® Advisor reports tend to be very wide. Do one of the following to generate readable reports:

- Set your console width appropriately to avoid line wrapping.
 - Pipe your report using the appropriate truncation command if you care only about the first few report columns.
-

After performing the prerequisites, do one of the following:

- For Vectorization and CPU Roofline: Mark the loop(s) of interest by enabling the associated  checkbox on the **Survey Report**.

Then run a Characterization with Trip Counts and FLOP collection enabled, Dependencies, or Memory Access Patterns analysis.



- For Offload Modeling: Go to **Project Properties > Performance Modeling** and enter the CLI action option `--select=<string>` in the **Other parameters** field. For example, `--select=5,10,12`.
- Mark the loop(s) of interest using the CLI action option `--select=<string>` (recommended) or `--mark-up-list=<string>` when running a Characterization with Trip Counts and FLOP collection enabled, Dependencies, or Memory Access Patterns analysis. For example, with the `--select` option:

```
advisor --collect=tripcounts --flop --project-dir=./advi_results --select=5,10,12 -- ./myApplication
```

Then run a Characterization with Trip Counts and FLOP collections enabled, Dependencies, or Memory Access Patterns analysis.

NOTE

There are different ways to select loops in the CLI environment:

- The advisor CLI action options `--mark-up-list=<string>` and `--select=<string>` merely simulate enabling a GUI  checkbox when used within `-collect` action. They are active only for the duration of the `--collect` command.
- The same options used with advisor CLI action `--mark-up-loops` actually enable a GUI  checkbox. They are active beyond the duration of the `-mark-up-loops` command and applies to all downstream analyses, such as Characterization with Trip Counts and FLOP collection enabled, Dependencies, Memory Access Patterns.

Select Loops by Source File/Line Number

Minimize collection overhead.

Applicable analyses: Characterization with Trip Counts and FLOP collection enabled, Dependencies, Memory Access Patterns.

Use when...

- You want to perform a deeper analysis on only a few loops.
- CLI environment: You are analyzing a target application for which you have access to source code and can identify source file/line numbers.


Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Prerequisites:

1. Run a Survey analysis.
2. `advisor` CLI environment: If necessary, identify the source file and line number for the loops of interest.

```
advisor --report=survey --project-dir=./advi_results -- ./myApplication
```


After performing the prerequisites, do one of the following:

- For Vectorization and CPU Roofline: Mark the loop(s) of interest by enabling the associated  checkbox on the Survey report.

Then run a Characterization with Trip Counts and FLOP collection enabled, Dependencies, or Memory Access Patterns analysis.

- For Offload Modeling: Go to **Project Properties > Performance Modeling** and enter the CLI action option `--select=<string>` in the **Other parameters** field. For example, `--select=foo.cpp:34,bar.cpp:192`.
- Mark the loop(s) of interest using the CLI action option `--select=<string>` (recommended) or `--mark-up-list=<string>` for a Characterization with Trip Counts and FLOP collection enabled, Dependencies, or Memory Access Patterns analysis. For example, with the `-select` option:

```
advisor --collect=tripcounts --flop --project-dir=./advi_results --select=foo.cpp:34,bar.cpp:192 -- ./bin/myApplication
```

- Mark the loop(s) of interest by enabling the associated  checkbox on the **Survey Report**.

Then run a Characterization with Trip Counts and FLOP collection enabled, Dependencies, or Memory Access Patterns analysis.

- Mark the loop(s) of interest using the `advisor` CLI action `--mark-up-loops` and action option `--select=<string>`. For example:

```
advisor --mark-up-loops --select=foo.cpp:34,bar.cpp:192 --project-dir=./advi_results -- ./myApplication
```

Then run a Characterization with Trip Counts and FLOP collection enabled, Dependencies, or Memory Access Patterns analysis.

NOTE

- There is essentially no difference between selecting loops by ID and selecting loops by source file/line in the GUI environment. The difference is in the `advisor` CLI environment:
 - The `advisor` CLI action option `--mark-up-list=<string>` merely simulates enabling a GUI checkbox; therefore it persists only for the duration of the `--collect` command.
 - The `advisor` CLI action `--mark-up-loops` and action option `--select=<string>` actually enables a GUI checkbox; therefore it persists beyond the duration of the `--mark-up-loops` command and applies to downstream analyses, such as Characterization with Trip Counts and FLOP collection enabled, Dependencies, and Memory Access Patterns.
- If you use the `--mark-up-loops` CLI action to mark up loops, you can append and remove source file/line numbers for an analysis run after it using the `advisor` CLI action option `--append=<string>` and `--remove=<string>` respectively.

Select Loops by Criteria

Goal: Minimize collection overhead.

Applicable analyses: Dependencies, Memory Access Patterns.

Use when you want to perform a deeper analysis on loops chosen by criteria instead of by human input, such as when you are running the Intel® Advisor with a collection preset or using automated scripts.

To implement in the `advisor` CLI environment, run the commands similar to the following one by one from the command line or create a script similar to the following examples and run it to execute the commands automatically. Use the `--select` (recommended) or `--loops` option to select loops by criteria.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

For example, to analyze loop-carried dependencies in loops/functions that have the *Assumes dependency present* issue, use one of the following:

- Example 1:

```
advisor --collect=survey --project-dir=./advi_results -- ./bin/myApplication
```

```
advisor --collect=dependencies --project-dir=./advi_results -- ./myApplication
```

- Example 2:

```
advisor --collect=survey --project-dir=./advi_results -- ./bin/myApplication
```

```
advisor --collect=dependencies select="scalar,has-issue" --project-dir=./advi_results -- ./myApplication
```

Select Loops by Markup Algorithm

Goal: Minimize collection overhead.

Applicable analyses: Characterization with Trip Counts and FLOP collection enabled, Dependencies, Memory Access Patterns.

NOTE This is only applicable to the Offload Modeling perspective.

Use `--select=r:markup=<algorithm>` when you want to perform a deeper analysis on loops chosen by a pre-defined markup algorithm based on a programming model used and/or estimated offload profitability.

If you analyze an application that runs on a CPU, use the `gpu_generic` algorithm. This algorithm selects all potentially profitable loops/functions for additional analyses to collect more data and make sure they can be safely offloaded.

If you analyze code regions that are already offloaded and use a specific programming model, use one of the following algorithms:

- `omp` - Select OpenMP* loops.
- `icpx -fsycl` - Select SYCL loops.
- `ocl` - Select OpenCL™ loops.
- `daal` - Select Intel® oneAPI Data Analytics Library loops.
- `tbb` - Select Intel® oneAPI Threading Building Blocks loops.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

For example, to run the Offload Modeling and analyze potentially profitable code regions in details:

- **Example 1.** Use the `--select=r:markup=<algorithm>` option with the `--collect` action option to select loops only for the specific analysis.

```
advisor --collect=survey --project-dir=./advi_results --static-instruction-mix -- ./myApplication
```

```
advisor --collect=tripcounts --project-dir=./advi_results --flop --cache-simulation=single --
target-device=xehpg_512xve --stacks --data-transfer=light -- ./myApplication
```

```
advisor --collect=dependencies --filter-reductions --loop-call-count-limit=16 --select
markup=gpu_generic --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --project-dir=./advi_results
```

- **Example 2.** Use the `--select=r:markup=<algorithm>` option with the `--mark-up-loops` action option in a separate step to select loops for all analysis executed after this command.

```
advisor --collect=survey --project-dir=./advi_results --static-instruction-mix -- ./myApplication
```

```
advisor --collect=tripcounts --project-dir=./advi_results --flop --cache-simulation=single --
target-device=xehpg_512xve --stacks --data-transfer=light -- ./myApplication
```

```
advisor --mark-up-loops --project-dir=./advi_results --select markup=gpu_generic -- ./
myApplication
```

```
advisor --collect=dependencies --filter-reductions --loop-call-count-limit=16 --project-dir=./
advi_results -- ./myApplication
```

```
advisor --collect=projection --project-dir=./advi_results
```

NOTE Currently, there is no GUI equivalent of the markup strategies. The `gpu_generic` strategy is used by default.

See Also

[Collection Controls to Minimize Analysis Overhead](#)

[Filtering to Minimize Analysis Overhead](#)

[Execution Speed/Duration/Scope Properties to Minimize Analysis Overhead](#)

[Miscellaneous Techniques to Minimize Analysis Overhead](#)

Filtering to Minimize Analysis Overhead

Issue

Running your target application with the Intel® Advisor can take substantially longer than running your target application without the Intel Advisor. Depending on an accuracy level and analyses you choose for a perspective, different overhead is added to your application execution time. For example:

Runtime Overhead / Analysis	Survey	Characterization	Dependencies	MAP
Target application runtime with Intel® Advisor compared to runtime without Intel® Advisor	1.1x longer	2 - 55x longer	5 - 100x longer	5 - 20x longer

Solution

Use the following techniques to skip *uninteresting* modules and/or analyze only *interesting* modules.

Filter Modules

Minimize collection and finalization overhead.

Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled.

Use to...

- Exclude modules you cannot optimize, such as third-party code.
- Include a small number of modules of interest.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following before/while running the desired analysis:

- Set **Project Properties > Analysis Target > [Name] Analysis > Modules > Exclude the following module(s)** and identify the modules, where *[Name]* is Survey or Trip Counts and FLOP.
- Use the advisor CLI action options `--module-filter-mode=exclude` and `--module-filter=<string>`. For example:

```
advisor --collect=survey --project-dir=./advi_results --module-filter-mode=exclude --module-filter=foo1.so,foo2.so -- ./myApplication
```

- Set **Project Properties > Analysis Target > [Name] Analysis > Modules > Include only the following module(s)** and identify the modules.
- Use the advisor CLI action options `--module-filter-mode=include` and `--module-filter=<string>`. For example:

```
advisor --collect=survey --project-dir=./advi_results --module-filter-mode=include --module-filter=foo1.so,foo2.so -- ./myApplication
```

See Also

[Collection Controls to Minimize Analysis Overhead](#)

[Loop Markup to Minimize Analysis Overhead](#)

[Execution Speed/Duration/Scope Properties to Minimize Analysis Overhead](#)

Miscellaneous Techniques to Minimize Analysis Overhead

Execution Speed/Duration/Scope Properties to Minimize Analysis Overhead

Issue

Running your target application with the Intel® Advisor can take substantially longer than running your target application without the Intel Advisor. Depending on an accuracy level and analyses you choose for a perspective, different overhead is added to your application execution time. For example:

Runtime Overhead / Analysis	Survey	Characterization	Dependencies	MAP
Target application runtime with Intel® Advisor compared to runtime without Intel® Advisor	1.1x longer	2 - 55x longer	5 - 100x longer	5 - 20x longer

Solutions

Use the following techniques to minimize overhead while collecting Intel Advisor analysis data. The *Disabling additional analysis* technique also minimizes finalization overhead.

Change Stackwalk Mode from Offline (After collection) to Online (During Collection)

Minimize collection overhead.

Applicable analysis: Survey.

Set to offline/after collection when:

- Survey analysis runtime overhead exceeds 1.1x.
- A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following before/while running a Survey analysis:

- Set **Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Stack unwinding mode > During collection**.
- Use the advisor CLI action option `--stackwalk-mode=online`. For example:

```
advisor --collect=survey --project-dir=./advi_results --stackwalk-mode=online -- ./myApplication
```

Disable Stacks Collection

Minimize collection overhead.

Applicable analyses: Characterization with Trip Counts and FLOP collections enabled.

To implement, do one of the following before/while running the analysis:

- Disable the **Analysis Workflow** pane > **Characterization Collect call stacks** checkbox.
- Disable the **Project Properties > Analysis Target > Trip Counts and FLOP Analysis > Advanced > Collect stacks** checkbox.
- Ensure the CLI action option `--stacks` is omitted from the command line. Alternative: Use the CLI action option `--no-stacks`.

NOTE If you collected callstack-attributed data with `collect.py` or `advisor` (with `--collect=tripcounts`), but callstack attribution went wrong, disable using callstacks data for analysis with `analyze.py` to avoid using the wrong data. This is a possible fallback when data with stacks is broken. Notice that this reduces modeling accuracy.

Disable Stitch Stacks

Minimize collection overhead.

Applicable analysis: Survey.

The stitch stacks option restores a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload.

Disable when Survey analysis runtime overhead exceeds 1.1x.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following before/while running the analysis:

- Disable the **Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Stitch stacks** checkbox.
- Use the `advisor` CLI action option `--no-stack-stitching`. For example:

```
advisor --collect=survey --project-dir=./advi_results --no-stack-stitching -- ./myApplication
```

NOTE

Disabling stack stitching may decrease the overhead for applications using oneTBB.

Increase Sampling Interval

Minimize collection overhead.

Applicable analysis: Survey.

Increase the wait time between each analysis collection sample when your target application runtime is long.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following before/while running the analysis:

- Increase the value in the **Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Sampling interval** checkbox.
- Use the `advisor` CLI action option `--interval=<integer>` when running a Survey analysis. For example:

```
advisor --collect=survey --project-dir=./advi_results --interval=20 -- ./myApplication
```

Limit Collected Analysis Data

Minimize collection overhead.

Applicable analysis: Survey.

Decrease the amount of collected raw data when exceeding a size threshold could cause issues. For example: You have storage space limitations.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following before/while running the analysis:

- Decrease the value in the **Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Collection data limit, MB** field.
- Decrease the value in the `advisor` CLI action option `--data-limit=<integer>`. For example:

```
advisor --collect=survey --project-dir=./advi_results --data-limit=250 -- ./myApplication
```

Limit Loop Call Count

Minimize collection overhead.

Applicable analysis: Dependencies, Memory Access Patterns.

Decrease the maximum number of instances each marked loop is analyzed.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following before/while running the analysis:

- Supply a non-zero value in the **Project Properties > Analysis Target > [Name] Analysis > Advanced > Loop Call Count Limit** field.
- Supply a non-zero value in the `advisor` CLI action option `--loop-call-count-limit=<integer>`. For example:

```
advisor --collect=dependencies --project-dir=./advi_results --loop-call-count-limit=10 -- ./myApplication
```

Disable Additional Analysis

Minimize finalization overhead.

Applicable analysis: Survey.

Implement these techniques when the additional data is not important to you.

NOTE

The default setting for all the properties/options in the table below is disabled.

Path: Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced	CLI Action Options	Description
Disable the Analyze MKL loops and functions checkbox.	<code>--no-mkl-user-mode</code>	Do not show Intel® oneAPI Math Kernel Library loops and functions in Intel Advisor reports.
Disable the Analyze Python loops and functions checkbox.	<code>--no-profile-python</code>	Do not show Python* loops and functions in Intel Advisor reports.

Path: Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced	CLI Action Options	Description
Disable the Analyze loops that reside in non-executed code paths checkbox.	<code>--no-support-multi-isa-binaries</code>	Do not collect a variety of data for loops that reside in non-executed code paths, including: <ul style="list-style-type: none"> • Loop assembly code • Instruction set architecture (ISA) • Vector length <p>NOTE This capability is available only for binaries compiled using the <code>-ax</code> (Linux* OS)/<code>Qax</code> (Windows* OS) option with an Intel® compiler.</p>
Disable the Enable register spill/fill analysis checkbox.	<code>--no-spill-analysis</code>	Do not calculate the number of consecutive load/store operations in registers and related memory traffic.
Disable the Enable static instruction mix analysis checkbox.	<code>--no-static-instruction-mix</code>	Do not statically calculate the number of specific instructions present in the binary.

See Also

[Collection Controls to Minimize Analysis Overhead](#)

[Loop Markup to Minimize Analysis Overhead](#)

[Filtering to Minimize Analysis Overhead](#)

[Miscellaneous Techniques to Minimize Analysis Overhead](#)

Miscellaneous Techniques to Minimize Analysis Overhead

Issue

Running your target application with the Intel® Advisor can take substantially longer than running your target application without the Intel® Advisor. Depending on an accuracy level and analyses you choose for a perspective, different overhead is added to your application execution time. For example:

Runtime Overhead / Analysis	Survey	Characterization	Dependencies	MAP
Target application runtime with Intel® Advisor compared to runtime without Intel® Advisor	1.1x longer	2 - 55x longer	5 - 100x longer	5 - 20x longer

Solutions

The following techniques may help minimize overhead without limiting collection scope.

Disable Cache Simulation

Minimize collection overhead.

Applicable analyses:

- Memory Access Patterns (base simulation functionality)
- Characterization with Trip Counts and FLOP collection enabled (enhanced simulation functionality)

Implement these techniques when cache modeling information is not important to you:

NOTE

The default setting for all the properties/options in the table below is disabled.

From the **Analysis Workflow** pane, disable **Characterization > Enable CPU cache simulation** for the **Characterization** analysis.

From the **Project Properties**:

Path: Project Properties > Analysis Target...	CLI Action Options	Description
Disable the Memory Access Patterns Analysis > Advanced > Enable Memory-Level Roofline with cache simulation checkbox.	<code>--no-enable-cache-simulation</code>	Do not model cache misses, cache misses and cache line utilization, or cache misses and loop footprint.
Disable the Trip Counts and FLOP Analysis > Advanced > Enable CPU cache simulation checkbox.	<code>--no-enable-cache-simulation</code>	Do not: <ul style="list-style-type: none"> • Model multiple levels of cache for data, such as counts of loaded or stored bytes for each loop. • Create simulations for specific cache hierarchy configurations.

Limit Reported Data

Applicable analysis: Memory Access Patterns.

Implement these techniques when the additional data is not important to you.

NOTE

The default setting for all the properties/options in the table below is enabled.

Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced	CLI Action Options	Description
Disable the Report stack variables checkbox.	<code>--no-record-stack-frame</code>	Do not report stack variables for which memory access strides are detected.
Disable the Report heap allocated variables checkbox.	<code>--no-record-stack-frame</code>	Do not report heap-allocated variables for which memory access strides are detected.

Minimize Data Set

Minimize collection overhead.

Applicable analyses: All, but especially Dependencies, Memory Access Patterns.

When you run an analysis, the Intel® Advisor executes the target against the supplied data set. Data set size and workload have a direct impact on target application execution time and analysis speed

For example, it takes longer to process a 1000x1000 pixel image than a 100x100 pixel image. A possible reason: You may have loops with an iteration space of 1...1000 for the larger image, but only 1...100 for the smaller image. The exact same code paths may be executed in both cases. The difference is the number of times these code paths are repeated.

You can control analysis cost without sacrificing completeness by minimizing this kind of unnecessary repetition from target application execution.

Instead of choosing large, repetitive data sets, choose small, representative data sets that minimize the number of instructions executed within a loop while thoroughly exercising target application control flow paths.

Your objective: In as short a runtime period as possible, execute as many paths as you can afford, while minimizing the repetitive computation within each task to the bare minimum needed for good code coverage.

Data sets that run in about ten seconds or less are ideal. You can always create additional data sets to ensure all your code is checked.

Temporarily Disable Finalization

Minimize finalization overhead.

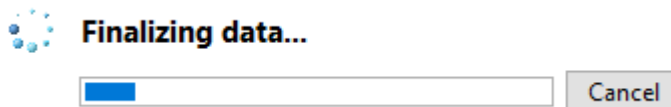
Applicable analyses: Survey, Characterization with Trip Counts and FLOP collection enabled.

Use when you plan to view collected analysis data on a different machine. Finalization automatically occurs when a result is opened in the GUI or a report is generated from the result.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

To implement, do one of the following while running an analysis:

- When the analysis **Finalizing data...** phase begins, click the associated **Cancel** button.



- Use the `advisor` CLI action option `--no-auto-finalize` when you run the desired analysis. For example:

```
advisor --collect=survey --project-dir=./advi_results --no-auto-finalize -- ./myApplication
```

When you open the result in GUI next time, the result is refinalized automatically.

To refinalize the result when running the analysis from CLI, use the `refinalize-survey` option with `--report` action. For example:

```
advisor --report=survey --search-dir src=./src bin=./bin --refinalize-survey --project-dir=./advi_results -- ./myApplication
```

See Also

[Collection Controls to Minimize Analysis Overhead](#)

[Loop Markup to Minimize Analysis Overhead](#)

[Filtering to Minimize Analysis Overhead](#)

[Execution Speed/Duration/Scope Properties to Minimize Analysis Overhead](#)

Analyze MPI Applications

With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

To start MPI jobs, use an MPI launcher such as `mpirun`, `mpiexec`, `srun`, `aprun`. You can use the Intel Advisor with the Intel® MPI Library or other MPI implementations only through the command line interface, but you can view the result using the standalone GUI, as well as the command line. The examples provided in this section use `mpirun` with the `advisor` command line interface (CLI) to spawn processes across the cluster and collect data about the application.

NOTE Use the MPI implementation that passes communication information using *environment variables*. The implementation needs to operate with the Intel Advisor process (`advisor`) being between the launcher process and the application process. Intel Advisor *does not* work on an MPI implementation that tries to pass communication information from its immediate parent process.

To analyze your MPI application:

1. Prerequisite: [Set up environment variables](#) to enable Intel Advisor CLI.
2. Optional: [Get pre-configured command lines](#).
3. Run Intel Advisor analyses with one of the following:
 - [Analyze a single rank of the MPI application with the Intel MPI Library](#)
 - [Analyze multiple or all ranks of the MPI application with Intel MPI Library](#)
 - [Analyze all ranks of the MPI application with non-Intel MPI library](#)
4. [View the results](#).

You can analyze your application as one of the following:

- If you have a directory shared between remote and local systems: Collect data remotely to the shared directory. In this case, you do not need to move a project between the systems.
- If you do not have a shared directory: Collect data to a directory on a remote system (for example, on a cluster), generate a snapshot (optionally), and copy the snapshot or a project to your local system to view the result. If you generate a snapshot, you do not need to configure the search paths for a project.

Get Preconfigured Command Lines

You can generate pre-configured command lines for collecting results for the Intel MPI Library launcher or a custom launcher using Intel Advisor graphical user interface (GUI). In this case, you do not need to type each command with all options and paths to a project directory and an application executable manually.

See [Generate Command Lines](#) for details.

Use Intel® MPI Library

With the Intel MPI Library, you can analyze a single MPI rank or several ranks of your MPI application with the Intel Advisor. This can help you to decrease analysis overhead.

Recommended MPI ranks to analyze are rank 1 and higher, because rank 0 might include time for configuration and not be a good representative for the general MPI application performance.

MPI Command Syntax

To collect performance data for an MPI application with Intel Advisor using the `mpirun` launcher of the Intel MPI Library, use the following command syntax:

```
mpirun -gtool "advisor --collect=<analysis-type> --search-dir src:r=<source-dir> [--no-auto-finalize] --project-dir=<project-dir>:<rank-set>" -n <N><application-name>
[<application-options>]
```

where:

- `-gtool` allows you to run Intel Advisor analyses for the specified MPI ranks only. This option is available for Intel MPI Library 5.0.2 or higher.
- `<analysis-type>` is an [Intel Advisor analysis](#) to run: `survey`, `tripcounts`, `map`, `dependencies`, `projection`.
- `<source-dir>` is the path to the directory where application sources are stored. Specify it if you disabled autofinalization.
- `<project-dir>` is the path/name of the project directory where the analysis results are saved. Specify the same project directory when running various Intel Advisor collections for the selected process.
- `<ranks-set>` is a set of MPI ranks to analyze. Each rank corresponds to an MPI *process* and is used to identify the result data. Separate ranks with a comma or use a dash " - " to set a range of ranks. Use with `-gtool` option only. Do not specify if you want to analyze all ranks.
- `<N>` is the number of MPI processes to launch.
- `--no-auto-finalize` disables result finalization on the target system to decrease overhead. The results are finalized when you import them or open in Intel Advisor GUI. *Do not* use this option if you do not use a shared directory and plan to copy results from cluster with snapshot. See [Temporarily Disable Finalization](#) for details.

Analyze a Single Rank of MPI Application with Intel MPI Library

Prerequisite: Set up environment variables to enable Intel Advisor CLI.

In the commands below:

- Data is collected remotely to a *shared* directory.
- The analyses are performed for an application running in *four* processes.
- Path to an application executable is `./mpi_sample`.

Note: In the commands below, make sure to replace the application path and name *before* executing a command. If your application requires additional command line options, add them after the executable name.

- Path to an Intel Advisor project directory is `./advi_results`.

This example shows how to run a Survey, Trip Counts, and Roofline analyses for the rank 1 of the MPI application with the `gtool` option of the Intel MPI Library.

1. Collect survey data for rank 1 into the shared `./advi_results` project directory on a target system.

```
mpirun -gtool "advisor --collect=survey --project-dir=./advi_results:1" -n 4 ./mpi_sample
```

2. Run the Trip Counts analysis with FLOP collection for rank 1 on the target system.

```
mpirun -gtool "advisor --collect=tripcounts --flop --project-dir=./advi_results:1" -n 4 ./mpi_sample
```

After you collect the Survey, Trip Counts, and FLOP data, you also get the Roofline report for your application.

3. If you did not collect data to a shared location and need to copy the data to the local system to view the results, do it now.
4. On the local system, [view the results](#) with your preferred method. You can view only one process data at a time.

Analyze Multiple Ranks of MPI Application with Intel MPI Library

Prerequisite: Set up environment variables to enable Intel Advisor CLI.

In the commands below:

- Data is collected remotely to a *shared* directory.
- The analyses are performed for an application running in *four* processes.

- Path to an application executable is `./mpi_sample`.

Note: In the commands below, make sure to replace the application path and name *before* executing a command. If your application requires additional command line options, add them after the executable name.

- Path to an Intel Advisor project directory is `./advi_results`.

Analyze a Set of Ranks

This example shows how to run a Survey, Trip Counts, and Roofline analyses for a *set* of ranks of the MPI application with the `gtool` option of the Intel MPI Library.

1. Collect survey data for ranks 1, 2, and 4 into the shared `./advi_results` project directory on a target system.

```
mpirun -gtool "advisor --collect=survey --project-dir=./advi_results:1-2,4" -n 4 ./mpi_sample
```

2. Run the Trip Counts analysis with FLOP collection for ranks 1, 2, and 4 on a target system.

```
mpirun -gtool "advisor --collect=tripcounts --flop --project-dir=./advi_results:1-2,4" -n 4./mpi_sample
```

After you collect the Survey, Trip Counts, and FLOP data, you also get the Roofline report for your application.

3. If you did not collect data to a shared location and need to copy the data to the local system to view the results, do it now.
4. On the local system, [view the results](#) with your preferred method. You can view only one process data at a time.

Analyze All Ranks

This example shows how to run a Survey, Trip Counts, and Roofline analyses for *all* ranks of the MPI application with the `gtool` option of the Intel MPI Library.

1. Collect survey data for all ranks into the shared `./advi_results` project directory on a target system.

```
mpirun -gtool "advisor --collect=survey --project-dir=./advi_results" -n 4 ./mpi_sample
```

2. Run the Trip Counts analysis with FLOP collection for all ranks on a target system.

```
mpirun -gtool "advisor --collect=tripcounts --flop --project-dir=./advi_results" -n 4./mpi_sample
```

After you collect the Survey, Trip Counts, and FLOP data, you also get the Roofline report for your application.

3. If you did not collect data to a shared location and need to copy the data to the local system to view the results, do it now.
4. On the local system, [view the results](#) with your preferred method. You can view only one process data at a time.

Use Non-Intel MPI Library

With non-Intel MPI library implementation, you can only analyze all ranks of your MPI application with Intel Advisor. This might increase analysis overhead.

MPI Command Syntax

To collect performance data for an MPI application with Intel Advisor using the `mpirun` launcher, use the following command syntax:

```
mpirun -n <N> "advisor --collect=<analysis-type> --search-dir src:r=<source-dir>--  
trace-mpi [--no-auto-finalize] --project-dir=<project-dir>" <application-name>  
[<application-options>]
```

where:

- `<N>` is the number of MPI processes to launch.
- `<analysis-type>` is an [Intel Advisor analysis](#) to run: `survey`, `tripcounts`, `map`, `dependencies`, `projection`.
- `<source-dir>` is the path to the directory where application sources are stored. Specify it if you disabled autofinalization.
- `<project-dir>` is the path/name of the project directory where the analysis results are saved. Specify the same project directory when running various Intel Advisor collections for the selected process.
- `--trace-mpi` enables analyzing non-Intel MPI library implementations. This option is required for non-Intel MPI implementation.
- `--no-auto-finalize` disables result finalization on the target system to decrease overhead. The results are finalized when you import them or open in Intel Advisor GUI. *Do not* use this option if you do not use a shared directory and plan to copy results from cluster with snapshot. See [Temporarily Disable Finalization](#) for details.

Analyze an MPI Application with Non-Intel MPI Library

Prerequisite: [Set up environment variables](#) to enable Intel Advisor CLI.

In the commands below:

- Data is collected remotely to a *shared* directory.
- The analyses are performed for an application running in *four* processes.
- Path to an application executable is `./mpi_sample`.

Note: In the commands below, make sure to replace the application path and name *before* executing a command. If your application requires additional command line options, add them after the executable name.

- Path to an Intel Advisor project directory is `./advi_results`.

This example shows how to run a Survey, Trip Counts, and Roofline analyses for *all 4* ranks of the MPI application.

1. Collect survey data for all ranks into the shared `./advi_results` project directory on a target system.

```
mpirun -n 4 "advisor --collect=survey --project-dir=./advi_results" ./mpi_sample
```

2. Run the Trip Counts analysis with FLOP collection on the target system.

```
mpirun -n 4 "advisor --collect=tripcounts --flop --project-dir=./advi_results" ./mpi_sample
```

After you collect the Survey, Trip Counts, and FLOP data, you also get the Roofline report for your application.

3. If you did not collect data to a shared location and need to copy the data to the local system to view the results, do it now.
4. On the local system, [view the results](#) with your preferred method. You can view only one process data at a time.

NOTE For all analysis types and MPI libraries: When using a shared partition on Windows* OS, specify the network paths to the project and executable location or use the MPI options `mapall` or `map` to specify these locations on the network drive.

For example:

```
mpiexec -gwdir \\<host1>\mpi -hosts 2 <host1> 1 <host2> 1 advisor --collect=survey
--project-dir=\\<host1>\mpi\advi_results -- \\<host1>\mpi\mpi_sample.exe

advisor --import-dir=\\<host1>\mpi\advi_results --project-dir=\\<host1>\mpi
\new_advi_results --search-dir src=\\<host1>\mpi --mpi-rank=1

advisor --report=survey --project-dir=\\<host1>\mpi\new_advi_results
```

or:

```
mpiexec -mapall -gwdir z:\ -hosts 2 <host1> 1 <host2> 1 advisor --collect=survey
--project-dir=z:\advi_results -- z:\mpi_sample.exe
```

or:

```
mpiexec -map z:\\<host1>\mpi -gwdir z:\ -hosts 2 <host1> 1 <host2> 1 advisor --
collect=survey --project-dir=z:\advi_results -- z:\mpi_sample.exe
```

View Results

Intel Advisor saves collection results into subdirectories for each rank analyzed under the project directory specified with `--project-dir`. The subdirectories are named as `rank.<n>`, where the numeric suffix `<n>` corresponds to an MPI rank analyzed. You can only view results for one rank at a time.

To view the performance results collected for a specific rank, you can do one of the following.

View Results in GUI

From the Intel Advisor GUI, open a result project file `*.advixeproj` that resides in the `<project-dir> / rank.<n>` directory.

You can also open the GUI from command line:

```
advisor-gui ./advi_results/rank.1
```

NOTE If you used `--no-auto-finalize` when collecting data, make sure to set paths to application binaries and sources *before* viewing the result so that Intel Advisor can finalize it properly.

View Results in Command Line

Run the Intel Advisor `--report` action to print the result summary in a terminal:

```
advisor --report=<analysis-type> --project-dir=<project-dir> --mpi-rank=<n>
```

where:

- `<analysis-type>` is the Intel Advisor analysis you want to print the results for.
- `<project-dir>` is the same project directory as you used for data collection.
- `<n>` is the number of MPI rank you want to view results for.

View Results in a File

You can save results for a specified rank to a TXT, CSV, or a XML file. For example, save the results to a `advisor_result.csv` file, run the following command:

```
advisor --report=<analysis-type> --format=csv --report-output=advisor_result.csv --
project-dir=<project-dir> --mpi-rank=<n>
```

where:

- <analysis-type> is the Intel Advisor analysis you want to print the results for.
- <project-dir> is the same project directory as you used for data collection.
- <n> is the number of MPI rank you want to view results for.
- --format specified the file format to save the results to. In the command above, it is CSV.

Additional MPI Resources

For more details on analyzing MPI applications, see the Intel MPI Library and online MPI documentation on the Intel® Developer Zone at <https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html>

For detailed syntax, refer to the [Intel® MPI Library Developer Reference for Linux* OS](#) or [Intel® MPI Library Developer Reference for Windows* OS](#).

Hybrid applications: Intel MPI Library and OpenMP* on the Intel Developer Zone at <https://www.intel.com/content/www/us/en/developer/articles/technical/hybrid-applications-mpi-openmp.html>

See Also

Model MPI Application Performance on GPU You can model your MPI application performance on a target graphics processing unit (GPU) device to determine whether you can get a performance speedup from offloading the application to the GPU.

[Control Collection with an MPI_Pcontrol Function](#)

Model MPI Application Performance on GPU

You can model your MPI application performance on a target graphics processing unit (GPU) device to determine whether you can get a performance speedup from offloading the application to the GPU.

Offload Modeling perspective of the Intel® Advisor includes the following stages:

1. Collecting the baseline performance data on a host device with the Survey, Characterization (Trip Counts, FLOP), and/or Dependencies analyses. You can collect data for one or more MPI ranks, where each rank corresponds to an MPI process.
2. Modeling application performance on a target device with the Performance Modeling analysis. You can model performance only for one rank at a time. You can run performance modeling several times for different ranks analyzed to examine the potential performance difference between them, but the topic does not cover this case.

Model Performance of MPI Application

Prerequisite: [Set up environment variables](#) to enable Intel Advisor CLI.

In the commands below:

- Data is collected remotely to a *shared* directory.
- The analyses are performed for an application running in *four* processes.
- Path to an application executable is `./mpi_sample`.

Note: In the commands below, make sure to replace the application path and name *before* executing a command. If your application requires additional command line options, add them after the executable name.

- Path to an Intel Advisor project directory is `./advi_results`.
- Performance is modeled for the default Intel® Arc™ graphics code-named Alchemist (`xehpg_512xve` configuration).

This example shows how to run Offload Modeling to model performance for the rank 1 of the MPI application. It uses the `gtool` option of the Intel MPI Library to collect performance data on a baseline CPU. For other collection options, see [Analyze MPI Applications](#).

1. *Optional, but recommended:* Generate preconfigured command lines for your application using the `--dry-run` option. For example, generate the command lines using Intel Advisor CLI:

```
advisor --collect=offload --dry-run --project-dir=./advi_results -- ./mpi_sample
```

After you run it, a list of analysis commands to run the Offload Modeling for the specified accuracy level is printed to the terminal/command prompt. For the command above, the commands are printed for the default *medium* accuracy:

```
advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./advi_results
-- ./mpi_sample
advisor --collect=tripcounts --flop --stacks --auto-finalize --enable-cache-simulation --data-
transfer=light --target-device=xehpg_512xve --project-dir=./advi_results -- ./mpi_sample
advisor --collect=projection --no-assume-dependencies --config=xehpg_512xve --project-dir=./
advi_results
```

You need to modify the printed commands for the MPI syntax to use an MPI launcher. See [Analyze MPI Applications](#) for syntax details.

2. Collect survey data for the rank 1 into the shared `./advi_results` project directory.

```
mpirun -gtool "advisor --collect=survey --auto-finalize --static-instruction-mix --project-dir=./
advi_results:1" -n 4 ./mpi_sample
```

3. Collect trip counts and FLOP data for the rank 1.

```
mpirun -gtool "advisor --collect=tripcounts --flop --stacks --auto-finalize --enable-cache-
simulation --data-transfer=light --target-device=xehpg_512xve --project-dir=./advi_results:1" -n
4 ./mpi_sample
```

4. If you did not collect data to a shared location and need to copy the data to the local system to view the results, do it now.
5. Model performance for the analyzed rank 1 of the MPI application that you ran the analyses for.

```
advisor --collect=projection --config=xehpg_512xve --mpi-rank=1 --project-dir=./advi_results
```

You can only model performance for one rank at a time. The results are generated for the rank specified in a corresponding `./advi_results/rank.1` directory.

6. If you did not collect data to a shared location and need to copy the data to the local system to view the results, do it now.
7. On a local system, [view the results](#) with your preferred method.

Configure Performance Modeling for MPI Application

By default, Offload Modeling is optimized to model performance for a single-rank MPI application. For multi-rank MPI applications, you can apply additional configuration and settings to adjust the performance model for a specific hardware or application. You can adjust the number of MPI ranks to run per GPU tile and/or exclude MPI time from the report.

In the commands below:

- Data is collected remotely to a *shared* directory.
- The analyses are performed for an application running in *four* processes.
- Path to an application executable is `./mpi_sample`.

Note: In the commands below, make sure to replace the application path and name *before* executing a command. If your application requires additional command line options, add them after the executable name.

- Path to an Intel Advisor project directory is `./advi_results`.
- Performance is modeled for Intel® Arc™ graphics code-named Alchemist (`xehpg_512xve` configuration).

Change the Number of MPI Processes per GPU Tile

Prerequisite: Set up [environment variables](#) to enable Intel Advisor CLI.

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

By default, Offload Modeling assumes that one MPI process, or rank, is mapped to one GPU tile. You can configure the performance model to adjust the number of MPI ranks to run per GPU tile to match your target device configuration.

To do this, you need to set the number of tiles per MPI process by scaling the `Tiles_per_process` target device parameter in a command line or a TOML configuration file. If you want to model performance for the Intel® Arc™ graphics code-named Alchemist, which is **XeHPG 256** or **XeHPG 512** configuration in Offload Modeling targets, use the `Stack_per_process` parameter. The parameter sets a fraction of a GPU tile that runs a single MPI process. For example, if you want to offload your MPI application with 8 processes to a target GPU device with 4 tiles, you need to adjust the performance model to run 2 MPI processes per tile, or to use 0.5 tile per process.

The number of tiles per process you set automatically adjusts:

- the number of execution units (EU)
- SLM, L1, L3 sizes and bandwidth
- memory bandwidth
- PCIe* bandwidth

The parameter accepts values from 0.01 to 12.0. Consider the following value examples:

Tiles_per_process/Stack_per_process Value	Number of MPI Ranks per Tile
1.0 (default)	1
12.0 (maximum)	1/12
0.25	4
0.125	8

To run the Offload Modeling with a custom tile-per-process parameter, you need to scale the parameter during the analysis. This change is one time and is applied only to the analysis you run it with. The commands below use the `Tiles_per_process` parameter for scaling. Replace it with `Stack_per_process` if needed.

1. Generate pre-configured command lines for your application with the `--set-parameter` option to change the number of tiles per process. Use the `--dry-run` option of the `collect.py` script to generate commands to adjust cache configuration to the scaled parameter.

For example, to generate commands for the `./advi_results` project and model performance with 0.25 tiles per process, which corresponds to four MPI ranks per tile:

```
advisor-python $APM/collect.py ./advi_results --set-parameter scale.Tiles_per_process=0.25 --dry-run -- ./mpi_sample
```

After you run it, a list of analysis commands to run the Offload Modeling with the specified accuracy level is printed to the terminal/command prompt similar to the following:

```
advisor --collect=survey --project-dir=./advi_results --static-instruction-mix -- ./mpi_sample
advisor --collect=tripcounts --project-dir=./advi_results --flop --ignore-checksums --data-
transfer=medium --stacks --profile-jit --cache-sources --enable-cache-simulation --cache-
config=8:64w:4k/1:192w:768k/1:4w:2m -- ./mpi_sample
python $APM/collect.py ./advi_results -m generic
advisor --collect=dependencies --project-dir=./advi_results --filter-reductions --loop-call-
count-limit=16 --ignore-checksums -- ./mpi_sample
```

2. Copy the generated commands to your preferred text editor and modify them for the MPI-specific syntax. You need to add the following:

- MPI launcher name and (optionally) `gtool` option for Intel® MPI Library
- Number of MPI processes to launch
- If you use `gtool`: MPI ranks to analyze

See [Analyze MPI Applications](#) for syntax details.

NOTE You can skip the mark-up and Dependencies analysis step (the last two commands) because they add high overhead. See [Check How Assumed Dependencies Affect Modeling](#) for details.

3. Run the modified commands for Survey, Trip Counts, and (optionally) Dependencies analyses one by one. For example, to run Survey and Trip Counts for the rank 1:

```
mpirun -gtool "advisor --collect=survey --static-instruction-mix -- ./mpi_sample --project-dir=./
advi_results:1" -n 4 ./mpi_sample
```

```
mpirun -gtool "advisor --collect=tripcounts --flop --ignore-checksums --data-transfer=medium --
stacks --profile-jit --cache-sources --enable-cache-simulation --cache-config=8:64w:4k/
1:192w:768k/1:4w:2m --project-dir=./advi_results:1" -n 4 ./mpi_sample
```

4. Run the performance modeling with the number of tiles per MPI processes specified using the `--set-parameters` option. For example, to model performance for the rank 1:

```
advisor --collect=projection --project-dir=./advi_results --set-parameter
scale.Tiles_per_process=0.25 --mpi-rank=1
```

NOTE Make sure to specify the same value for the `--set-parameter scale.Tiles_per_process` as for the dry-run step.

The result is generated for the rank specified in a corresponding `./advi_results/rank.1` directory. You can transfer them to the development system, if needed, and [view the results](#).

When you open the result in the Intel Advisor GUI or an interactive HTML report, you should see the tiles per process or stack per process parameter in the **Modeling Parameters** pane with the value you set. The parameter is in a read-only format. Notice that tiles per process or stack per process parameter shows the value per process, while other parameters in the pane show the value per device.

Ignore MPI Time

Prerequisite: [Set up environment variables](#) to enable Intel Advisor CLI.

For multi-rank MPI workloads, time spent in MPI runtime can differ from rank to rank, which may cause significant performance imbalance. Because of this, the whole application time and Offload Modeling results may be different from rank to rank. If MPI time is large and differs between ranks, and the MPI code does not include many computations, you can exclude time spent in MPI routines from the analysis so that it does not affect modeling results.

1. Collect Survey, Trip Counts, and (optionally) Dependencies data for your application. See [Analyze MPI Applications](#) for details.
2. Run the performance modeling with time in MPI calls ignored using the `--ignore=MPI` option.

```
advisor --collect=projection --project-dir=./advi_results --ignore=MPI --mpi-rank=1
```

The results are generated in a `./advi_results/rank.1` directory. You can transfer them to the development system and [view the results](#).

In the report generated, all *per-application* performance modeling metrics are calculated based on application self-time with time spent in MPI calls excluded from the analysis. This should improve modeling across ranks.

NOTE This option affects only metrics for *the whole program* in the **Summary** tab. Metrics for individual regions are not recalculated.

View Results

Intel Advisor saves collection results into subdirectories for each rank analyzed under the project directory specified with `--project-dir`. The modeling results are available only for the ranks that you ran the Performance Modeling for, for example, as specified with the `--mpi-rank` option.

To view the performance or dependency results collected for a specific rank, you can do one of the following.

View Results in GUI

From the Intel Advisor GUI, open a result project file `*.advixeproj` that resides in the `<project-dir> / rank.<n>` directory.

You can also open the GUI from command line:

```
advisor-gui ./advi_results/rank.1
```

NOTE If you used `--no-auto-finalize` when collecting data, make sure to set paths to application binaries and sources *before* viewing the result so that Intel Advisor can finalize it properly.

View Results in Command Line

After you run the Performance Modeling analysis, the summary result of the modeling is printed to a terminal/command prompt. Examine the data to learn the estimated speedup and top five offloaded regions.

View Results in an Interactive HTML Report

Open an interactive `advisor-report` HTML report generated in the respective rank directory at `<project-dir>/rank.<n>/e<NNN>/report` and a set of CSV reports in the respective rank directory at `<project-dir>/rank.<n>/p<NNN>/data.0`.

See Also

[Analyze MPI Applications](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

[Explore Offload Modeling Results](#)

Control Collection with an MPI_Pcontrol Function

By default, Intel® Advisor analyzes performance of a whole application. In some cases, you may want to focus on the most time-consuming section or disable collection for the initialization or finalization phases. This can decrease collection overhead.

Intel Advisor supports the MPI region control with the `MPI_Pcontrol()` function. This function allows you to enable and disable collection for specific application regions in the source code. The region control affects only MPI and OpenMP* metrics, while other metrics are collected for the entire application.

To use the function, add it to your application source code as follows:

- To pause data collection, add `MPI_Pcontrol(0)` *before* the code region that you want to disable the collection for.
- To resume data collection, add `MPI_Pcontrol(1)` where you want the collection to start again.
- To skip the initialization phase:
 1. Add the `MPI_Pcontrol(1)` function right after initialization.
 2. Build the application.
 3. Run Intel Advisor analyses with the `--start-paused` option. For example, to run the Survey analysis:

```
advisor --collect=survey --start-paused --project-dir=./advi_results -- ./mpi_sample
```

NOTE Intel Advisor accepts only 0 and 1 arguments for the `MPI_Pcontrol()` function. You can use other numbers to mark code regions, but Intel Advisor ignores them.

For code snippet examples, see [Region Control with MPI_Pcontrol](#).

See Also

[Analyze MPI Applications](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

[Model MPI Application Performance on GPU](#) You can model your MPI application performance on a target graphics processing unit (GPU) device to determine whether you can get a performance speedup from offloading the application to the GPU.

[Minimize Analysis Overhead](#)

Manage Results

Data collected by running Intel® Advisor tools is stored in a result. When you run one of its tools, the Intel® Advisor executes a target, identifies issues that may need handling, collects the results and shows it in the Results subdirectory in the **Solution Explorer** in Microsoft Visual Studio* or in the **Project Navigator** in the Intel® Advisor Standalone GUI.

View the data in the Result tab to help you choose the best places to add parallelism. There is one result for each project or each project in the Solution (on Windows* OS). If you run an Intel® Advisor tool on the same project, any previously collected results are overwritten.

Result Locations

The Intel® Advisor saves results in the `Results` directory in a solution directory on Windows OS (the default Microsoft Visual Studio* location) or a subdirectory of the specified Intel® Advisor project location in the Project Navigator on Linux* OS. You can specify a custom location for saving results.

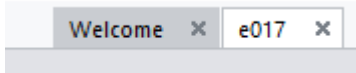
Results and the Solution Explorer

You can view results associated with a project in the **Solution Explorer** in Visual Studio or the **Project Navigator** in the Intel® Advisor GUI. However, sometimes other considerations outweigh accessibility. For example, in Visual Studio, do not display results in the **Solution Explorer** if you use a source code control system and you do not want to check in your **Solution Explorer** with embedded results.

Open a Result

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

There is one result for each project. To open a previously collected result for one project:

To Do This	Do This
After opening its Solution, open a result from the Microsoft Visual Studio* IDE or the Intel® Advisor GUI.	<p>After you open the Visual Studio solution or the Intel Advisor GUI project, do one of the following:</p> <ul style="list-style-type: none"> From the Microsoft Visual Studio* Solution Explorer, double-click the result in the Results program folder. From the Microsoft Visual Studio* menu: <ol style="list-style-type: none"> Choose Open > File. In the Open File dialog box, browse to and double-click the result. By default the <i>ennn.advixe</i> file in <i>project name</i> folder in the solution or project directory. From the Intel® Advisor GUI menu: <ol style="list-style-type: none"> Choose either File > Open > Result... or File > Recent Results. In the Project Navigator, browse to and double-click the result. By default the <i>ennn.advixe</i> file in <i>project name</i> folder in the solution or project directory. <hr/> <p>NOTE You can open multiple results from the same project only,</p>
Open a result from the Solution Explorer or the Intel® Advisor product GUI or its Project Navigator.	<p>After you open the Visual Studio solution, browse to and double-click the result. By default, the results are in the Advisor Result directory in the Solution Explorer.</p> <p>From the product GUI, click File > Open > Result... or File > Recent Results > name.</p> <p>When using the Project Navigator in the product GUI, navigate to the project and click its result name, such as <i>ennn</i>.</p> <hr/> <p>NOTE You can open multiple results from the same project only,</p>
View a specific result	<p>If you have opened multiple results for different projects and you would like to view a result that is not displayed, click its tab to view that result. The result appears showing the last report that you viewed.</p> <p>To view a specific result, click its tab name:</p>  <hr/> <p>NOTE You can open multiple results from the same project only,</p>

To rearrange the order of the displayed tabs, drag a tab to the desired location.

You can create a snapshot of the active result and save as a read-only result (see the help topic *Create a Read-only Result Snapshot*).

See Also

[Rename Existing Results](#)

[Create a Read-only Result Snapshot](#)

[Open a Result as a Read-only File](#)

Rename an Existing Result

NOTE

- If you rename results using the file system or Windows* Explorer software instead of the Intel® Advisor Standalone GUI or the Microsoft Visual Studio* IDE, you may create an error condition.
 - If you change the `.advixe` extension, you will not be able to open the result file in Intel® Advisor.
-

To rename an existing result

1. Right-click the result folder in the **Solution Explorer** or the result in the Intel® Advisor GUI to display a context menu, then choose **Rename**.
2. Type the new name.
3. Press the **Enter** key.

The result is renamed in the project and on your system.

See Also

[Delete a Result](#)

[Save Results to a Custom Location](#)

Delete a Result

To delete a result:

1. Right-click the result in the **Solution Explorer** in Visual Studio or the **Project Navigator** in the Intel® Advisor GUI to display a context menu.
2. When using Visual Studio, choose **Remove**. In the resulting dialog box, click the **Delete** button.
3. When using the Intel® Advisor GUI **Project Navigator**, click **Delete**.

The result is removed from the project and deleted from your system.

See Also

[Create a Read-only Result Snapshot](#)

[Save Results to a Custom Location](#)

Save Results to a Custom Location

The Intel® Advisor saves a result in a subdirectory of each project's directory. The project directory is in the default Visual Studio location or the directory specified when creating the Intel® Advisor Standalone GUI project. Instead of saving results within each project's directory, you can specify a custom, central location for saving all new results.

To save results to a custom location when using the Microsoft Visual Studio* or Intel® Advisor GUI:

1. From the Microsoft Visual Studio menu, choose **Tools > Options...**
2. From the Intel® Advisor GUI menu, choose **File > Options...**
3. In the **Options** dialog box, expand the **Intel Advisor** program folder and choose the **Result Location** page.

4. Select **Save all results in this directory:**.
5. Click **Browse** to select the custom location.
6. Click **OK**.

The Intel® Advisor saves future results to the custom location. The subdirectory name is the result name, such as e000.

See Also

[Open a Result as a Read-only File](#)

Work with Standalone HTML Reports

Export the interactive Intel® Advisor HTML reports that you can share or open on a remote machine using your web browser.

Offload Modeling HTML Reports

For the Offload Modeling perspective, you can export two types of HTML reports:

- An interactive HTML report that represents results in the same structure as in graphical user interface (GUI) and enables you to switch between Offload Modeling and Intel Advisor perspective results if you collect your data for an application running on GPU.
- A legacy HTML report that enables you to view an extended set of metrics for your offloaded and non-offloaded code regions

Export Offload Modeling HTML Reports

If you [run Offload Modeling perspective from command line interface \(CLI\)](#) Intel Advisor automatically saves both types of HTML reports. Once the execution is complete, you can find the reports stored in the following directories:

- `<project-dir>/e<NNN>/report/advisor-report.html` for interactive HTML report
- `<project-dir>/e<NNN>/report/report.html` for legacy HTML report

If you [run Offload Modeling perspective from GUI](#), Intel Advisor automatically saves only the legacy report into the `<project-dir>/e<NNN>/pp000/data.0` directory.

To get an interactive HTML report for results collected in the GUI, you can export the report using CLI. For example, to export an **interactive HTML report** as `offload_modeling_report.html` from the `advi_results` project, run the following command:

```
advisor --report=all --project-dir=./advi_results --report-output=./offload_modeling_report.html
```

where:

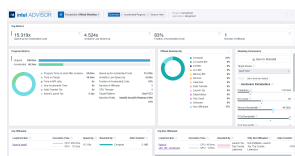
- `--project-dir` option specifies the path to your project directory where you collected the results.
- `--report-output` option specifies the directory to save the HTML report to and a file name. This option is required.

View Offload Modeling Interactive HTML Report

The structure of results in Offload Modeling interactive HTML report is similar to a GUI report with the following controls available:

- If you run both GPU-to-GPU Offload Modeling and GPU Roofline Insights perspectives or the GPU Roofline Insights perspective with the Performance Modeling analysis enabled, you can switch between the results using the **Perspective** drop-down menu in the top left corner of the report.
- Switch between report tabs to view estimated performance on a target accelerator. The **Summary** tab shows data for the whole application, the **Accelerated Regions** tab shows data per code region or kernel in detail.
- Use the sliders in the **Modeling Parameters** pane to change the hardware parameter values and use the updated target device configuration for remodeling. See the section below for details.

To explore the interactive HTML report, you can [download precollected Offload Modeling reports](#) and examine the results and structure.



For details about results interpretation, see [Explore Offload Modeling Results](#).

Remodel Performance for a Custom Target Device

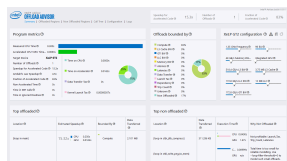
The interactive HTML report includes a Modeling Parameter pane, which you can use to examine the target device characteristics and modify the parameters as needed to model application performance for the future or a custom target:

1. Change the parameter values using the sliders.
2. Click **Save to Remodel** to save the custom device configuration.
3. Copy the command line generated under the hardware parameter list.
4. Paste the command to a terminal or command prompt and run it to model your application performance on a custom device.
5. View the updated interactive HTML report generated as `<project-dir>/e<NNN>/report/advisor-report.html`.

See [Model Application Performance on a Custom Target GPU Device](#) for a full workflow and pane description.

View Offload Modeling Legacy HTML Report

You can switch between the tabs to explore metrics for offloaded and non-offloaded code regions, estimate data transfer taxes, view or download configuration file for the selected target accelerator and examine execution logs.




GPU Roofline HTML Reports

Intel Advisor enables you to export two types of HTML reports:

- An interactive HTML report that represents results in the same structure as in GUI and enables you to switch between Offload Modeling and GPU Roofline Insights perspective results if you collect your data for an application running on GPU. This report contains grid data with GPU metrics, a GPU Roofline chart, GPU Details tab containing per-kernel compute and memory metrics, and a source view.
- An HTML Roofline chart that enables you to visualize your application performance on an interactive Roofline chart and view your platform information.

Export HTML GPU Roofline Chart using GUI

To export an interactive HTML GPU Roofline chart using GUI, do the following:

1. Run GPU Roofline Insights perspective.
2. Select FLOAT or INT data type using the filter pane at the top of the Roofline chart. You cannot change the data type after the report is generated.
3. Export the project results by clicking the  button and selecting **Export as HTML** option. To share your result as an image, consider selecting **Export as SVG** option and setting up the resolution.
4. Save the HTML report and open it in your browser.

Export GPU Roofline Report using CLI

Once the perspective executes, you can export both types of HTML reports using CLI.

To export an **interactive HTML report** as `gpu_roofline_report.html` from the `./advi_results` project, run the following command:

```
adviser --report=all --project-dir=./advi_results --report-output=./gpu_roofline_report.html
```

where:

- `--project-dir` option specifies the path to your project directory where you collected the results.
- `--report-output` option specifies the directory to save the HTML report to and a file name. This option is required.

To export an **HTML GPU Roofline chart** for floating-point operations data as `gpu_roofline.html`, make sure that the value of the `--project-dir` option specifies the path to your project directory and run the following command:

```
adviser --report=roofline --gpu --project-dir=./advi_results --report-output=./gpu_roofline.html --data-type=float
```

where:

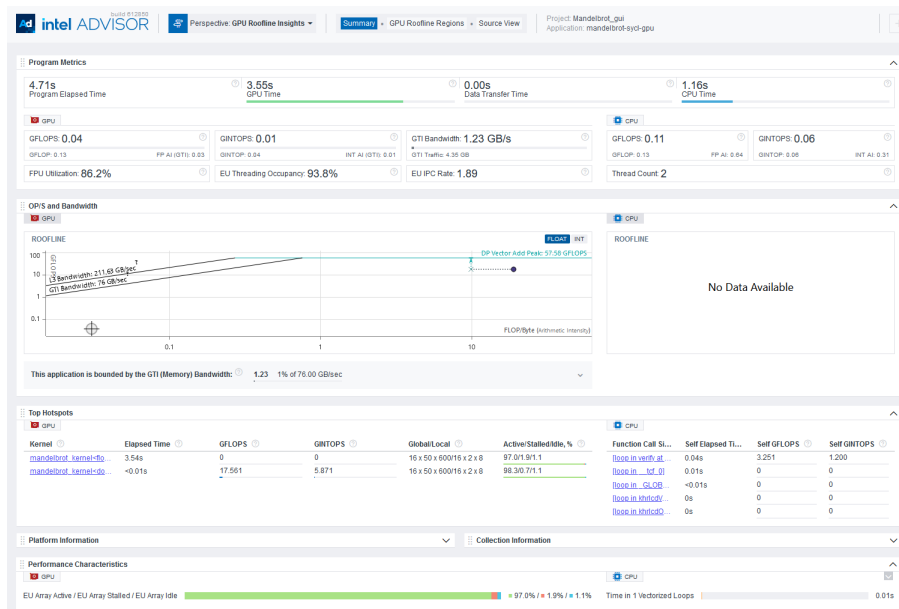
- `--project-dir` option specifies the path to your project directory where you collected the results.
- `--report-output` option specifies the directory to save the HTML report to and a file name. This option is required.
- `--gpu` option generates a Roofline chart for GPU kernels. This option is *required*.
- `--data-type=<type>` option specifies the data type to show in the Roofline chart. Available types are `float` (default) or `int`. You cannot change the data type after the report is generated.

Once report generation is complete, open it in your preferred web browser.

View Interactive HTML Report for GPU Roofline

You can switch between the **Summary** and **GPU Roofline Regions** tabs to examine how your application executes on a GPU, identify top hotspots, and define room for their optimization using a Roofline chart and GPU grid metrics.

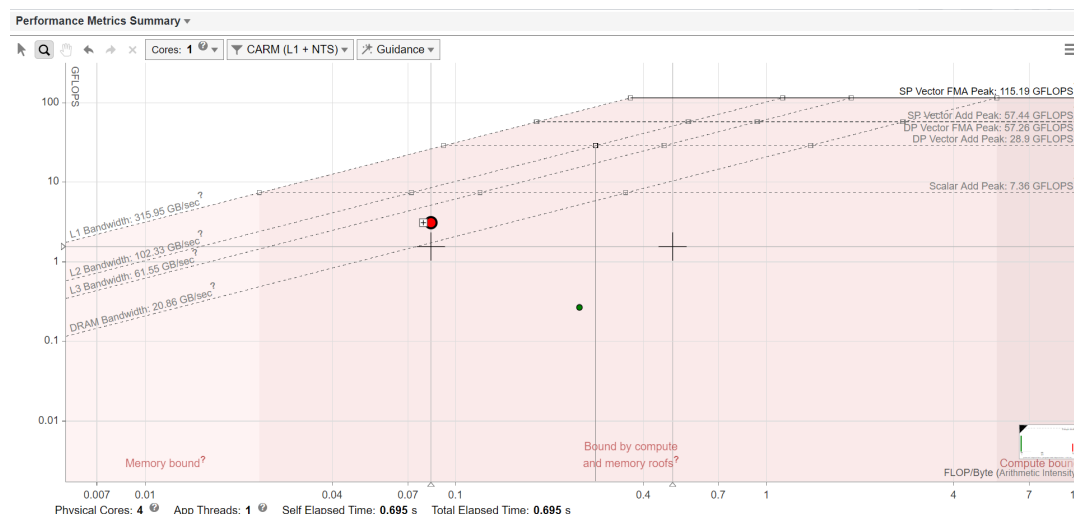
To explore the interactive HTML report, you can [download a precollected GPU Roofline report](#) and examine the results and structure.



For details about result interpretation, see [Explore GPU Roofline Results](#).

View HTML GPU Roofline Chart

Identify top hotspots and room for optimization of your application running on GPU using a Roofline chart and view the application execution and performance details in the **Performance Metrics Summary** drop-down section.



For details on results interpretation, see [Examine Bottlenecks on a GPU Roofline Chart](#).


HTML Roofline Chart for CPU Roofline

Export an interactive Roofline chart for CPU / Memory Roofline Insights perspective to share it or open on a remote machine using your web browser. This report enables you to visualize performance of your application running on CPU, identify factors limiting your application performance, and define headroom for optimization at different memory levels.

Export HTML CPU Roofline Chart Using GUI

To export an interactive HTML CPU Roofline chart using GUI, do the following:

1. Run CPU / Memory Roofline Insights perspective.
2. Select FLOAT or INT data type using the filter pane at the top of the Roofline chart. You cannot change the data type after the report is generated.
- 3.

Export the project results by clicking the  button and selecting **Export as HTML** option. To share your result as an image, consider selecting **Export as SVG** option and setting up the resolution.

4. Save the HTML report and open it in your browser.

Export HTML CPU Roofline Chart Using CLI

Intel Advisor enables you to export an HTML CPU Roofline chart using CLI. For example, to export an interactive CPU Roofline chart for floating-point operations data as `roofline.html`, make sure that the value of the `--project-dir` option specifies the path to your project directory and run the following command:

```
adviser --report=roofline --project-dir=./advi_results --report-output=./roofline.html
```

where:

- `--project-dir` option specifies the path to your project directory where you collected the results.
- `--report-output` option specifies the directory to save the HTML report to and a file name. This option is required.

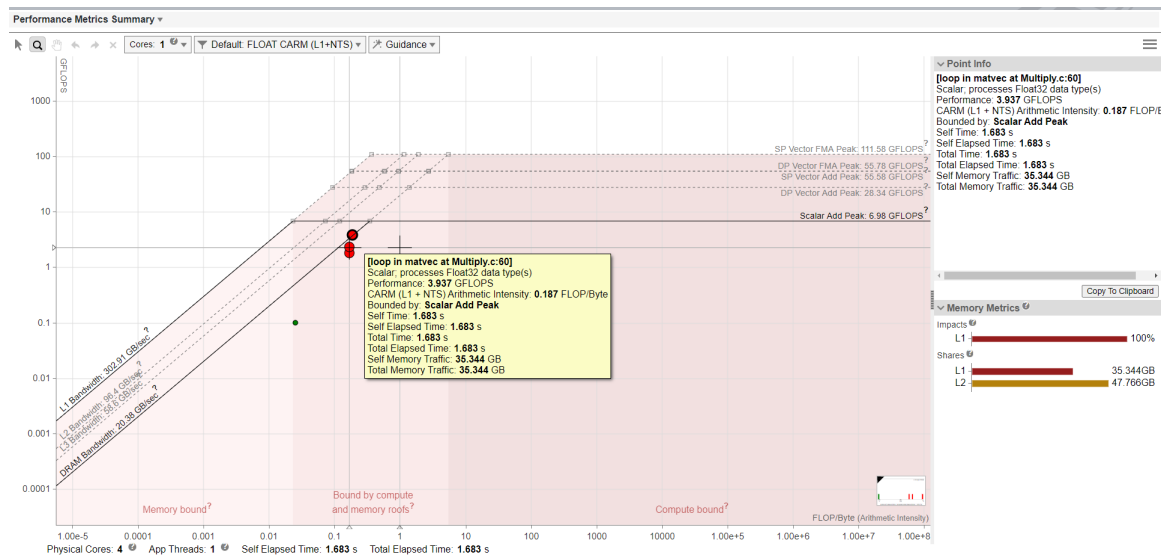
In addition to the options used in the example command, you can use the following additional options to extend the report with more data:

- Use the `--with-stack` option to enable call stack data in the HTML report. Use it if you collected CPU Roofline with call stack data using the `--stacks` option.
- Use the `--data-type=<type>` option to specify the data type to show in the Roofline chart. Available types are `float` (default), `int`, `mixed`. You *cannot* change the data type after the report is generated.
- Use the `--memory-level=<string>` option to show specific memory levels in the HTML report by default. Available memory levels are `L1` (default), `L2`, `L3`, and `DRAM`. You can combine several memory levels with an underscore (for example, `L1_L2`). Use this if you generated the Memory-level CPU Roofline with `--enable-cache-simulation`.

Once report generation is complete, open it in your preferred web browser.

View CPU HTML Roofline Chart

Identify hotspots for optimization and room for improvement of your application running on CPU and view the application execution and performance details in the **Performance Metrics Summary** drop-down section.



For more information on interpretation of Roofline charts, see [Examine Bottlenecks on CPU Roofline Chart](#).



See Also

- [Roofline Resources](#)
- [Offload Modeling Resources](#)

Create a Read-only Result Snapshot

Only the active result for a project can be modified by collecting new data. You can create a read-only result snapshot after you collect data from one or more Intel® Advisor tools. To create a snapshot of a result and save its data in a read-only result:

1. Open the active result (not a read-only result snapshot).
- 2.

Click the  /  button next the window caption. The **Create a Result Snapshot** dialog box appears.

3. Type the **Result name** of the Intel® Advisor read-only result. Provide a unique name, perhaps by adding an identifying suffix within the result name.
4. Click **OK**. With a large result, you may need to wait as the read-only result gets created.

You can visually compare saved read-only results against the current active result or other read-only results. The read-only result appears in the Intel® Advisor GUI **Project Navigator** and the Visual Studio Solution Explorer (for Windows* OS only). The words **(read-only)** appear after the name of a read-only result in the result tab. The icon of a read-only result differs from that of the active result.

Although you can only collect data to update the active project, you can update your sources regardless of what type of result (or project) you have open.

See Also

[Save Results to a Custom Location](#)

Create a Result Snapshot Dialog Box

Purpose

Intel® Advisor stores only the most recent analysis result. Use this dialog box to save a read-only result snapshot you can view any time.

Tip

- Visually comparing one or more snapshots to each other or to the most recent analysis result can be an effective way to judge performance improvement progress.
- To view a snapshot, choose **File > Open > Result...**
- Snapshots are identified by a different icon in the Visual Studio* Solution Explorer and the Intel® Advisor **Project Navigator**. The words **(read-only)** appear after the snapshot name in a result tab.

Access

1. Display an active result (not a snapshot).
- 2.

Click the  /  button.

Controls

Use This	To Do This
Result name field	Specify the name of the read-only result snapshot. Provide a unique name, perhaps by adding an identifying suffix within the result name.
Cache sources checkbox	Enable source code availability in the resulting snapshot.
Cache binaries checkbox	Enable binary availability in the resulting snapshot.
Pack into archive checkbox	Create a one-file archive with all snapshot data inside.
Result path text box	Specify the path to the resulting snapshot archive. Use the Browse... button to specify the address. Disabled by default. Enable by selecting the Pack into archive checkbox.

Open a Result as a Read-only File in Visual Studio

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

To open a previously collected result file as a read-only file when its corresponding Visual Studio* Solution is *not* open:

To Do This	Do This
Open a result from the Microsoft Visual Studio* IDE.	<p>Before opening the Solution, from the Microsoft Visual Studio* File menu:</p> <ol style="list-style-type: none"> 1. Choose Open > File. 2. In the Open File dialog box, browse to and double-click the Intel® Advisor result file (file suffix <code>.advixe</code>). By default, Intel® Advisor results are saved in the <code>ennn.advixe</code> file in <code>project name</code> directory in the solution or project directory. <p>NOTE This action launches the Microsoft Visual Studio* IDE if it is not already open.</p>
Open a result from the Windows* Explorer (or comparable) environment.	<p>From the Windows* Explorer (or comparable) environment:</p> <ol style="list-style-type: none"> 1. Open the Windows* Explorer (or comparable) environment. 2. Browse to and double-click the result. By default, Intel® Advisor results are saved in the <code>ennn.advixe</code> file in <code>project name</code> folder in the solution or project directory. <p>NOTE This action launches the Microsoft Visual Studio* IDE if it is not already open.</p>
Verify that the result is open as a read-only file.	A result opened as a read-only file contains (read-only) after its result name.
View a specific report for the result	<p>Each result contains several different reports. To view a specific report, click the button for the report you want to view, such as Survey Report, Suitability Report, or Dependencies Report.</p> <p>When opening a result as a read-only file, you are only viewing the previously collected data as a read-only file. The following limitations apply:</p> <ul style="list-style-type: none"> • The Annotation Report window does not contain any data. To view the Annotation Report, you need to open a result after opening its Solution. • The Suitability Report window shows data, but you cannot change the modeling parameters, such as the Target System or the Threading Model. • If you modify your sources or Intel® Advisor annotations, you need to open the solution to collect data and perform analysis (see Opening a Result) to update the result and view Annotation Report data.

See Also

Create a Read-only Result Snapshot

Open a Result (after opening a solution)

Command Line Interface

This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

- Use the [advisor Command Line Interface](#) to run any perspective: Vectorization and Code Insights, CPU / Memory Roofline Insights, Threading, Offload Modeling, GPU Roofline Insights.
- Use [Python* scripts](#) to run the Offload Modeling perspective.

Tip See [Intel Advisor cheat sheet](#) for quick reference on command line interface.

advisor Command Line Interface Reference

This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

The main advantage of using the Intel® Advisor command line interface, `advisor`, instead of the GUI is you can collect data as part of an automated or background task, and then view the result in a command line interface (CLI) report or in the GUI at your convenience.

Prerequisite: [Set Intel® Advisor environment variables](#) to start using the command line interface.


Tip

You can [generate command lines](#) from Intel® Advisor for selected configuration. In the **Analysis Workflow** pane:

-

To generate a set of commands for the whole perspective, click



- To generate a command for a specific analysis, expand the analysis you want to get a command for and click .

advisor Command Syntax

The `advisor` command syntax is:

```
advisor <--action> [--action-options] [--global-options] [--] target [target options]
```

where:

<code>advisor</code>	The name of the Intel® Advisor command line tool.
<code><--action></code>	The action to perform, such as <code>collect</code> or <code>report</code> . Each command has exactly one action. For example, you cannot use both the <code>collect</code> and <code>report</code> actions in the same command.
<code>--action-options</code>	Action options modify behavior specific to the action. You can have multiple action options per command. Using an action option that does not apply to the action results in a usage error.
<code>--global-options</code>	Global options modify behavior in the same manner for all actions. You can have multiple global options per action.
<code>target</code>	The target (application executable) to analyze.

[target-options] Options that apply to the target.

Action option/Global option rules:

- If opposing action options are used on the same command line the last specified action option applies.
- An action option that is redundant or has no meaning in the context of the specified action is ignored.
- Attempted use of an inappropriate action option that might lead to unexpected behavior returns a usage error

Syntax Alternatives

An action option or global option can be preceded by one or two dashes. This chapter uses one dash before the short form of an action option/global option, and two dashes before the long form of an action option/global option. For example: The following are equivalent:

```
advisor --help
```

```
advisor -help
```

An option-value pair can be separated by an equal sign (=) or by a space. This chapter uses an equal sign. For example: The following are equivalent:

```
advisor --report=survey
```

```
advisor --report survey
```

The target executable must be preceded by two dashes and a space. For example:

```
advisor --collect=survey -- ./myApplication
```

Some action options accept multiple arguments. Most of the time, you can pass these arguments in a comma-separated string (with no spaces), or by repeating the action option. For example: The following are equivalent.

```
advisor --collect=survey --project-dir=./advi_results --exclude-files=./src/  
foo,./src/bar -- ./myApplication
```

```
advisor --collect=survey --project-dir=./advi_results --exclude-files=./src/foo --  
exclude-files=./src/bar -- ./myApplication
```

Directories

Project Directory

By default, the project directory is your current working directory. Use the `project-dir` action option to write a result to a different directory. For example:

Survey the application for hotspots and write the result to the `./advi` project directory.

```
advisor --collect=survey --project-dir=./advi_results --search-dir all:=./src -- ./  
myApplication
```

Generate a Survey report from the Survey result and write it to the `./advi` project directory.

```
advisor --report=survey --project-dir=./advi_results --format=text --report-  
output=./out/survey.txt
```

Search Directory

Use the `search-dir` action option to specify the directories containing the source, symbol, and binary files that support analysis.

You can specify multiple search directories. For example:

```
advisor --collect=survey --project-dir=./advi_results --search-dir src:=./src1,./src2
-- ./myApplication
```

Tip

Always specify your search directories when using `collect` action.

User Data Directory

Use the `user-data-dir` action option to write result files to a directory other than `project-dir`, such as a remote directory or simply another directory when there is not enough space in `project-dir`.

For example: Collect Suitability data and write the result to a remote directory.

```
advisor --collect=suitability --project-dir=./advi_results --user-data-dir=./remote_dir
--search-dir src:=./src -- ./myApplication
```

advisor Command Action Reference

The advisor command currently supports the actions shown below.

Action	Description
<code>collect</code>	Run the specified type of analysis and collect data.
<code>command</code>	Control the Intel® Advisor while running analyses.
<code>create-project</code>	Create an empty project, if it does not already exist.
<code>help</code>	Explain command line actions with corresponding options.
<code>import-dir</code>	Import and finalize data collected on an MPI cluster.
<code>mark-up-loops</code>	After running a Survey analysis and identifying loops of interest, select loops (by file and line number or criteria) for deeper analysis.
<code>report</code>	Generate a report from data collected during a previous analysis.
<code>snapshot</code>	Create a read-only result snapshot you can view any time.
<code>version</code>	Display product version information.
<code>workflow</code>	Explain typical Intel® Advisor user scenarios, with corresponding command lines.

collect

Run the specified type of analysis and collect data.

GUI Equivalent**Analysis Workflow**

File > New > Start [Name] Analysis

Syntax

```
-c=<string> [--action-options] [--global-options] [--] <target> [<target options>]]
```

```
--collect=<string> [--action-options] [--global-options] [--] <target> [<target options>]]
```

Arguments

<string> is the type of analysis:

Argument	Description
survey	Survey the target (your executable application) and collect data about code that may benefit from (more) parallelism.
dependencies	Collect dependencies data to predict and eliminate data sharing problems.
map	Collect memory access patterns data.
offload	Run the Offload Modeling perspective analyses with a single command.
projection	Project performance on a target device.
roofline	Run the Survey analysis immediately followed by the Trip Counts & FLOP analysis to visualize actual performance against hardware-imposed performance ceilings.
suitability	Collect suitability data by executing annotated code to analyze the proposed threading parallelism opportunities and estimate where performance gains are most likely.
tripcounts	Collect the following data and add it to the Survey report: loop iteration, floating-point and integer operation, and memory traffic statistics, and more.

Default

No default argument

Modifiers

accuracy, app-working-dir, assume-dependencies, assume-hide-taxes, assume-ndim-dependency, assume-single-data-transfer, auto-finalize, batching, benchmarks-sync, cache-config, cache-simulation, cache-sources, cachesim, cachesim-associativity, cachesim-cacheline-size, cachesim-mode, cachesim-sampling-factor, cachesim-sets, check-profitability, config, count-logical-instructions, count-memory-instructions, count-memory-objects-accesses, count-mov-instructions, count-send-latency, cpu-scale-factor, custom-config, data-limit, data-reuse-analysis, data-transfer, data-transfer-histogram, data-transfer-page-size, delete-tripcounts, disable-fp64-math-optimization, dry-run, duration, enable-cache-simulation, enable-data-transfer-analysis, enforce-baseline-decomposition, enforce-fallback, enforce-offloads, estimate-max-speedup, evaluate-min-speedup, exclude-files, executable-of-interest, exp-dir, filter-by-scope, filter-reductions, flop, force-32bit-arithmetics, force-64bit-arithmetics, gpu, gpu-carm, gpu-sampling-interval, hide-data-transfer-tax, ignore, ignore-app-mismatch, ignore-checksums, instance-of-interest, integrated, interval, loop-call-count-limit, loop-filter-threshold, loops, mark-up, mark-up-list, mkl-user-mode, model-baseline-gpu, model-children, model-extended-math, model-system-calls, module-filter, module-filter-mode, mpi-rank, mrte-mode, ndim-depth-limit, option-file, overlap-taxes, profile-gpu, profile-intel-perf-lib, profile-jit, profile-python, profile-stripped-binariesproject-dir, quiet, record-mem-allocations, record-stack-frame, refinalize-survey, resume-after, return-app-exitcode, search-dir, search-n-dim, select, set-dependency, set-parallel, set-parameter, show-report, small-node-filter, spill-analysis, stack-access-granularity, stack-stitching, stack-unwind-limit, stacks, stackwalk-mode, start-paused, static-instruction-mix, strategy, support-multi-isa-binaries, target-device, target-gpu, target-pid, target-process, threads, trace-mode, trace-mpi, track-memory-objects, track-stack-accesses, track-stack-variables, trip-counts, verbose

Example

Survey the application to find candidates for code that may benefit from (more) parallelism.

```
advisor --collect=survey --search-dir src:r=./src --project-dir=./advi_results -- ./bin/
myApplication
```

Collect memory access patterns data on the specified loops.

```
advisor --collect=map --mark-up-list=5,10,12 --search-dir src:r=./src --project-dir=./
advi_results -- ./bin/myApplication
```

Collect survey data on four nodes of an MPI cluster into the shared ./advi project directory.

```
mpirun -n 4 "advisor --collect=survey --project-dir=./advi_results" -- <PATH>/mpi-sample/
1_mpi_sample_serial
```

Collect dependencies data for all innermost loops that account for over 2% of the total CPU time.

```
advisor --collect=dependencies --loops="loop-height=0,total-time>2 --project-dir=./advi_results"
-- ./bin/myApplication
```

Run the Offload Modeling perspective with low accuracy.

```
advisor --collect=offload --accuracy=low --config=xehpg_512xve --search-dir src:r=./src --
project-dir=./advi_results -- ./bin/myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

command

Control the Intel Advisor while running analyses.

GUI Equivalent

Analysis Workflow

Syntax

```
--C=<string> [--action-options] [--global-options] [--] <target> [<target options>]]
--command=<string> [--action-options] [--global-options] [--] <target> [<target
options>]]
```

Arguments

`<string>` is one of the following:

Argument	Description
cancel	Interrupt data collection without saving results.
detach	Similar to pause, except you can attach to an already running process.
pause	Pause data collection while the target application continues running.
resume	Resume paused data collection.
stop	Stop data collection.

Argument	Description
status	Print collection status.

Default

No default argument

Modifiers

[quiet](#), [result-dir](#), [verbose](#)

Usage

Usage can decrease collection overhead.

Example

Pause the analysis run that is currently collecting data into result directory `r000hs`.

```
advisor --command=pause -r=<PATH>/r000hs
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

create-project

Create an empty project, if it does not already exist.

GUI Equivalent

File > New > Project...

Syntax

```
--create-project [--action-options] [--global-options] [--] <target> [target options]
```

Modifiers

[project-dir](#), [quiet](#), [search-dir](#), [verbose](#)

Usage

Use the `--project-dir` action option to:

- Specify a project name.
- Create a project somewhere other than the current working directory.

Example

Create a new `advi` project in the current working directory.

```
advisor --create-project --project-dir=./advi_result -- ./bin/myApplication
```

See Also

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

help

Explain command line actions with corresponding options.

Syntax

```
-h
--help
-h <action>
--help <action>
```

Arguments

`<action>` is one of the following: [collect](#), [command](#), [create-project](#), [import-dir](#), [mark-up-loops](#), [report](#), [snapshot](#), [version](#), [workflow](#)

Description

Explain command line actions with corresponding options.

Examples

Display overall help.

```
advisor --help
```

Display help for the `collect` action.

```
advisor -h collect
```

See Also

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

import-dir

Import and finalize data collected on an MPI cluster.

Syntax

```
--import-dir=<PATH> [--action-options] [--global-options] [--] <target> [<target options>]
```

Arguments

`<PATH>` is the full path to a directory where previously collected data resides.

Default

No default argument

Modifiers

[mpi-rank](#), [project-dir](#), [quiet](#), [search-dir](#), [verbose](#)

Usage

For best results, specify the location of the source application files using the `search-dir` action option. Use the `mpi-rank` action option to specify the process data to import.

For MPI workloads:

1. Copy the data from the rank node to the home node.
2. Import the data.
3. View the data.

Example

Import data collected on rank 2 of the MPI cluster to the new project.

```
advisor --import-dir-./advi --mpi-rank-2 --search-dir src:r=./src --project-dir-./advi_results
```

See Also

[Analyze MPI Workloads](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

mark-up-loops

After running a Survey analysis and identifying loops of interest, select loops (by file and line number or criteria) for deeper analysis.

GUI Equivalent

Survey > 



Syntax

```
--mark-up-loops [--action-options] [--global-options] [--] target [target options]
```

Modifiers

[append](#), [clear](#), [loops](#), [mpi-rank](#), [project-dir](#), [quiet](#), [remove](#), [select](#), [verbose](#)

Usage

Do not confuse the `mark-up-loops` action with the `mark-up-list` action option. The `mark-up-loops` action coupled with the `select` action option enables a GUI  checkbox; therefore loop selection persists beyond the duration of the `mark-up-loops` action and applies to downstream analyses, such as Dependencies and Memory Access Patterns analyses. The `collect` action coupled with the `mark-up-list` action option simulates enabling a GUI  checkbox; therefore loop selection persists only for the duration of the `collect` action.

Example

Select loops for downstream analysis based on file and line number.

```
advisor --mark-up-loops --select=foo.cpp:34,bar.cpp:192 --project-dir=./advi_results -- ./bin/myApplication
```

Select loops for downstream analysis based on criteria.

```
advisor --mark-up-loops --loops="scalar,has-issue" --project-dir=./advi_results -- ./bin/
myApplication
```

See Also

[mark-up-list](#) After running a Survey analysis and identifying loops of interest, select loops (by file and line number or ID) for deeper analysis.

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#)

report

Generate a report from data collected during a previous analysis.

GUI Equivalent

File > Open > Results

File > Recent Results

Syntax

```
--report=<string> [--action-options] [--global-options] [--] target [target options]]
-R=<string> [--action-options] [--global-options] [--] target [target options]]
```

Arguments

<string> is the list of available reports:

Argument	Description
annotations	Report the annotations in the source code.
all	Generate a combined HTML report for Offload Modeling and GPU Roofline Insights perspectives.
custom	Generate a custom report.
dependencies	Report results of a Dependencies analysis.
joined	Combine results for several analyses into a single report.
map	Report results of a Memory Access Patterns analysis.
projection	Report results for Offload Modeling.
roofline	Report results of a Roofline analysis.
roofs	Report roof values.
suitability	Report results of a Suitability analysis.
summary	Report the analysis summary.
survey	Report results of a Survey analysis.

Argument	Description
threads	Report on threads.
top-down	Report results of a Survey analysis in top-down view.
tripcounts	Add trip counts data to a Survey report.

Default

No default argument

Modifiers

[bottom-up](#), [csv-delimiter](#), [data-type](#), [display-callstack](#), [dynamic](#), [enable-task-chunking](#), [filter](#), [gpu](#), [format](#), [limit](#), [memory-level](#), [memory-operation-type](#), [mix](#), [mpi-rank](#), [option-file](#), [project-dir](#), [quiet](#), [recalculate-time](#), [reduce-lock-contention](#), [reduce-lock-overhead](#), [reduce-site-overhead](#), [reduce-task-overhead](#), [refinalize-survey](#), [report-output](#), [report-template](#), [search-dir](#), [show-all-columns](#), [show-all-rows](#), [show-functions](#), [show-loops](#), [show-not-executed](#), [sort-asc](#), [sort-desc](#), [target-system](#), [threading-model](#), [top-down](#), [verbose](#), [with-stack](#)

Usage

Suitability reports are the most configurable.

Generate a report from data collected during a previous analysis.

Example

Generate a Suitability report.

```
advisor --report=suitability --search-dir src:r=./src --format=text --report-output=./out/suitability.txt --project-dir=./advi_results
```

Generate a Dependencies report for data collected on rank 3 of MPI cluster:

```
advisor --report=dependencies --mpi-rank=3 --search-dir src:r=./src --project-dir=./advi_results
```

See Also

[collect](#) Run the specified type of analysis and collect data.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command **syntax**: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

snapshot

Create a read-only result snapshot you can view any time.

GUI Equivalent

File > Create Data Snapshot

Syntax

```
--snapshot [--action-options] [--global-options] [--] target [target options]
```

Default

Save the current analysis result.

Modifiers

[cache-binaries](#), [cache-binaries-mode](#), [cache-sources](#), [mpi-rank](#), [pack](#), [project-dir](#), [quiet](#), [search-dir](#), [verbose](#)

Usage

Intel® Advisor stores only the most recent analysis result. Visually comparing one or more snapshots to each other or to the most recent analysis result can be an effective way to judge performance improvement progress.

Example

Create a new snapshot in the project directory. Name it `snapshotXXX` (default name).

```
advisor --snapshot --no-pack --project-dir=./advi_results
```

Create a new snapshot in the project directory. Name it `new_snapshot`.

```
advisor --snapshot --no-pack --project-dir=./advi_result -- new_snapshot
```

Create a new snapshot. Pack it into an archive. Put it in the current directory. Name it `snapshotXXX.advixeexpz` (default name).

```
advisor --snapshot --pack --project-dir=./advi_results
```

Create a new snapshot. Include sources and binaries. Pack it into an archive. Name it `/tmp/new_snapshot.advixeexpz`.

```
advisor --snapshot --pack --cache-sources --cache-binaries --project-dir=./advi_results -- /tmp/new_snapshot
```

See Also

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Option Reference](#)

version

Display product version information.

Syntax

`-v`

`--version`

Example

Write product version information to `stdout`.

```
advisor --version
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

workflow

Explain typical Intel® Advisor user scenarios, with corresponding command lines.

Syntax

--workflow

Usage

Explain typical Intel Advisor user scenarios, with corresponding command lines.

Add SIMD Parallelism:

1. Find hotspots.

```
advisor --collect=survey --search-dir src:r=./src --project-dir=./advi_results -- ./bin/
myApplication
advisor --report=survey --project-dir=./advi_results --search-dir src:r=./src --format=text --
report-output=./out/survey.txt
```

2. Determine the number of loop iterations.

```
advisor --collect=survey --search-dir src:r=./src --project-dir=./advi_results -- ./bin/
myApplication
advisor --report=survey --search-dir src:r=./src --format=text --report-output=./out/survey.txt
--project-dir=./advi_results
```

3. Check for possible dependencies.

```
advisor --collect=dependencies --search-dir src:r=./src --project-dir=./advi_results -- ./bin/
myApplication
advisor --report=dependencies --search-dir src:r=./src --format=text --report-output=./out/
dependencies.txt --project-dir=./advi_results
```

4. Check memory access patterns.

```
advisor --collect=map --search-dir src:r=./src --project-dir=./advi_results -- ./bin/
myApplication
advisor --report=map --search-dir src:r=./src --format=text --report-output=./out/map.txt --
project-dir=./advi_results
```

5. Update the application to enable automatic compiler vectorization, or explicitly mark the loops you need to vectorize. Rebuild the application and test.

Add Threading Parallelism

1. Find hotspots. This step is similar to the first step in the SIMD-parallel workflow (above).
2. Determine the number of loop iterations. This step is similar to the second step in the SIMD parallel workflow (above).
3. Add annotations to the application source code and rebuild the application.
4. Collect suitability data. Note: Annotations must be present in the source code for this collection to be successful.

```
advisor --collect=suitability --search-dir src:r=./src --project-dir=./advi_results -- ./bin/
myApplication
advisor --report=suitability --search-dir src:r=./src --format=text --report-output=./out/
suitability.txt --project-dir=./advi_results
```

5. Check for the possible dependencies.

```
advisor --collect=dependencies --search-dir src:r=./src --project-dir=./advi_results -- ./bin/
myApplication
advisor --report=dependencies --search-dir src:r=./src --format=text --report-output=./out/
dependencies.txt --project-dir=./advi_results
```

6. Display a list of currently used annotations.

```
advisor --report=annotations --search-dir src:r=./src --format=text --report-output=./out/
annotations.txt --project-dir=./advi_results
```

7. Update the application using the chosen parallel coding constructs. Rebuild the application and test.

Tip

Use an option file for efficiency. Enter one option on each line. No spaces are allowed in the option entry; use a new line. The option file must be in UTF-8 format.

```
advisor --report=annotations --option-file=./advi/option.txt
```

with an `option.txt` file that looks like this:

```
--project-dir
./advi_results
--search-dir
src:r=./src
--format=text
--report-output
./out/annotations.txt
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

advisor Command Option Reference

The `advisor` command currently supports the options shown below.

Option	Description
<code>accuracy</code>	Set an accuracy level for the Offload Modeling collection preset.
<code>append</code>	Add loops (by file and line number) to the loops selected for deeper analysis.
<code>app-working-dir</code>	Specify the directory where the target application runs during analysis, if it is different from the current working directory.
<code>assume-dependencies</code>	Assume that a loop has dependencies if the loop dependency type is unknown.
<code>assume-hide-taxes</code>	Estimate invocation taxes assuming the invocation tax is paid only for the first kernel launch.
<code>assume-ndim-dependency</code>	When searching for an optimal N-dimensional offload, assume there are dependencies between inner and outer loops.
<code>assume-single-data-transfer</code>	Assume data is only transferred once for each offload, and all instances share that data.
<code>auto-finalize</code>	Finalize Survey and Trip Counts & FLOP analysis data after collection is complete.

Option	Description
<code>batching</code>	Emulate the execution of more than one instance simultaneously for a top-level offload.
<code>benchmarks-sync</code>	Run benchmarks on only one concurrently executing Intel Advisor instance to avoid concurrency issues with regard to platform limits.
<code>bottom-up</code>	Generate a Survey report in bottom-up view.
<code>cache-binaries</code>	Enable binary visibility in a read-only snapshot you can view any time.
<code>cache-binaries-mode</code>	Select what binary files will be added to a read-only snapshot.
<code>cache-config</code>	Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.
<code>cache-simulation</code>	Simulate device cache behavior for your application.
<code>cache-sources</code>	Enable source code visibility in a read-only snapshot you can view any time (with the <code>--snapshot</code> action). Enable keeping source code cache within a project (with the <code>--collect</code> action).
<code>cachesim</code>	Enable cache simulation for Performance Modeling.
<code>cachesim-associativity</code>	Set the cache associativity for modeling CPU cache behavior during the Memory Access Patterns analysis.
<code>cachesim-cacheline-size</code>	Set the cache line size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.
<code>cachesim-mode</code>	Set the focus for modeling CPU cache behavior during Memory Access Patterns analysis.
<code>cachesim-sampling-factor</code>	Specify what percentage of total memory accesses should be processed during cache simulation.
<code>cachesim-sets</code>	Set the cache set size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.
<code>check-profitability</code>	Check the profitability of offload regions and add only profitable regions to a report.
<code>clear</code>	Clear all loops previously selected for deeper analysis.
<code>config</code>	Specify a device configuration to model your application performance for.
<code>count-logical-instructions</code>	Use the projection of x86 logical instructions to GPU logical instructions.
<code>count-memory-instructions</code>	Project x86 memory instructions to GPU SEND/SENDS instructions.

Option	Description
<code>count-memory-objects-accesses</code>	Count the number of accesses to memory objects created by code regions.
<code>count-mov-instructions</code>	Project x86 MOV instructions to GPU MOV instructions.
<code>count-send-latency</code>	Select how to model SEND instruction latency.
<code>cpu-scale-factor</code>	Specify a scale factor to approximate a host CPU that is faster than the baseline CPU by this factor.
<code>csv-delimiter</code>	Set the delimiter for a report in CSV format.
<code>custom-config</code>	Specify the absolute path or name for a custom TOML configuration file with additional modeling parameters.
<code>data-limit</code>	Limit the maximum amount (in MB) of raw data collected during Survey analysis.
<code>data-reuse-analysis</code>	Analyze potential data reuse between code regions.
<code>data-transfer</code>	Set the level of details for modeling data transfers during Characterization.
<code>data-transfer-histogram</code>	Estimate data transfers in details and latencies for each transferred object.
<code>data-transfer-page-size</code>	Specify memory page size to set the traffic measurement granularity for the data transfer simulator.
<code>data-type</code>	Show only floating-point data, only integer data, or data for the sum of both data types in a Roofline interactive HTML report.
<code>delete-tripcounts</code>	Remove previously collected trip counts data when re-running a Survey analysis with changed binaries.
<code>disable-fp64-math-optimization</code>	Do not account for optimized traffic for transcendentals on a GPU.
<code>display-callstack</code>	Show a callstack for each loop/function call in a report.
<code>dry-run</code>	List all steps included in Offload Modeling batch collection at a specified accuracy level without running them.
<code>duration</code>	Specify the maximum amount of time (in seconds) an analysis runs.
<code>dynamic</code>	Show (in a Survey report) how many instructions of a given type actually executed during Trip Counts & FLOP analysis.
<code>enable-batching</code>	Deprecated.
<code>enable-cache-simulation</code>	Model CPU cache behavior on your target application.

Option	Description
<code>enable-data-transfer-analysis</code>	Model data transfer between host memory and device memory.
<code>enable-grf-simulation</code>	Enable a simulator to model GRF.
<code>enable-slm</code>	Deprecated. SLM is modeled by default if available.
<code>enable-task-chunking</code>	Examine specified annotated sites for opportunities to perform task-chunking modeling in a Suitability report.
<code>enforce-baseline-decomposition</code>	Use the same local size and SIMD width as measured on a baseline device.
<code>enforce-fallback</code>	Emulate data distribution over stacks if stacks collection is disabled.
<code>enforce-offloads</code>	Offload all selected code regions even if offloading their child loops/functions is more profitable.
<code>estimate-max-speedup</code>	Estimate region speedup with relaxed constraints.
<code>evaluate-min-speedup</code>	Consider loops recommended for offloading only if they reach the minimum estimated speedup specified in a configuration file.
<code>exclude-files</code>	Exclude the specified files or directories from annotation scanning during analysis.
<code>executable-of-interest</code>	Specify an application for analysis that is not the starting application.
<code>exp-dir</code>	Specify a path to an unpacked result snapshot or an MPI rank result to generate a report or model performance.
<code>filter</code>	Filter data by the specified column name and value in a Survey and Trips Counts & FLOP report.
<code>filter-by-scope</code>	Enable filtering detected stack variables by scope (warning vs. error) in a Dependencies analysis.
<code>filter-reductions</code>	Mark all potential reductions by specific diagnostic during Dependencies analysis.
<code>flex-cachesim</code>	Enable flexible cache simulation to change cache configuration without re-running collection.
<code>flop</code>	Collect data about floating-point and integer operations, memory traffic, and mask utilization metrics for AVX-512 platforms during Trip Counts & FLOP analysis.
<code>force-32bit-arithmetics</code>	Consider all arithmetic operations as single-precision floating-point or int32 operations.
<code>force-64bit-arithmetics</code>	Consider all arithmetic operations as double-precision floating-point or int64 operations.
<code>format</code>	Set a report output format.

Option	Description
<code>gpu</code>	With Offload Modeling perspective, analyze OpenCL™ and oneAPI Level Zero programs running on Intel® Graphics. With GPU Roofline Insights perspective, create a Roofline interactive HTML report for data collected on GPUs.
<code>gpu-carm</code>	Collect memory traffic generated by OpenCL™ and Intel® Media SDK programs executed on Intel® Processor Graphics.
<code>gpu-kernels</code>	Deprecated. Use <code>--profile-gpu</code> or <code>--gpu</code> instead.
<code>gpu-sampling-intervals</code>	Specify time interval, in milliseconds, between GPU samples during Survey analysis.
<code>hide-data-transfer-tax</code>	Disable data transfer tax estimation.
<code>ignore</code>	Specify runtimes or libraries to ignore time spent in these regions when calculating per-program speedup.
<code>ignore-app-mismatch</code>	Ignore mismatched target or application parameter errors before starting analysis.
<code>ignore-checksums</code>	Ignore mismatched module checksums before starting analysis.
<code>instance-of-interest</code>	Analyze the Nth child process during Memory Access Patterns and Dependencies analysis.
<code>integrated</code>	Model traffic on all levels of the memory hierarchy for a Roofline report.
<code>interval</code>	Set the length of time (in milliseconds) to wait before collecting each sample during Survey analysis.
<code>limit</code>	Set the maximum number of top items to show in a report.
<code>loop-call-count-limit</code>	Set the maximum number of instances to analyze for all marked loops.
<code>loop-filter-threshold</code>	Specify total time, in milliseconds, to filter out loops that fall below this value.
<code>loops</code>	Select loops (by criteria instead of human input) for deeper analysis.
<code>mark-up</code>	Enable/disable user selection as a way to control loops/functions identified for deeper analysis.
<code>mark-up-list</code>	After running a Survey analysis and identifying loops of interest, select loops (by file and line number or ID) for deeper analysis.
<code>memory-level</code>	Model specific memory level(s) in a Roofline interactive HTML report, including L1, L2, L3, and DRAM.

Option	Description
<code>memory-operation-type</code>	Model only load memory operations, store memory operations, or both, in a Roofline interactive HTML report.
<code>mix</code>	Show dynamic or static instruction mix data in a Survey report.
<code>mkl-user-mode</code>	Collect Intel® oneAPI Math Kernel Library (oneMKL) loops and functions data during the Survey analysis.
<code>model-baseline-gpu</code>	Use the baseline GPU configuration as a target device for modeling.
<code>model-children</code>	Analyze child loops of the region head to find if some of the child loops provide more profitable offload.
<code>model-extended-math</code>	Model calls to math functions such as EXP, LOG, SIN, and COS as extended math instructions, if possible.
<code>model-system-calls</code>	Analyze code regions with system calls considering they are separated from offload code and executed on a host device.
<code>module-filter</code>	Specify application (or child application) module(s) to include in or exclude from analysis.
<code>module-filter-mode</code>	Limit, by inclusion or exclusion, application (or child application) module(s) for analysis.
<code>mpi-rank</code>	Specify MPI process data to import.
<code>mrte-mode</code>	Set the Microsoft* runtime environment mode for analysis.
<code>ndim-depth-limit</code>	When searching for an optimal N-dimensional offload, limit the maximum loop depth that can be converted to one offload.
<code>option-file</code>	Specify a text file containing command line arguments.
<code>overlap-taxes</code>	Enable asynchronous execution to overlap offload overhead with execution time.
<code>pack</code>	Pack a snapshot into an archive.
<code>profile-gpu</code>	Analyze OpenCL™ and oneAPI Level Zero programs running on Intel® Processor Graphics.
<code>profile-intel-perf-lib</code>	Show Intel® performance libraries loops and functions in Intel® Advisor reports.
<code>profile-jit</code>	Collect metrics about Just-In-Time (JIT) generated code regions during the Trip Counts and FLOP analysis.
<code>profile-python</code>	Collect Python* loop and function data during Survey analysis.

Option	Description
<code>profile-stripped-binaries</code>	Collect metrics for stripped binaries.
<code>project-dir</code>	Specify the top-level directory where a result is saved if you want to save the collection somewhere other than the current working directory.
<code>quiet</code>	Minimize status messages during command execution.
<code>recalculate-time</code>	Recalculate total time after filtering a report.
<code>record-mem-allocations</code>	Enable heap allocation tracking to identify heap-allocated variables for which access strides are detected during Memory Access Patterns analysis.
<code>record-stack-frame</code>	Capture stack frame pointers to identify stack variables for which access strides are detected during Memory Access Patterns analysis.
<code>reduce-lock-contention</code>	Examine specified annotated sites for opportunities to reduce lock contention or find deadlocks in a Suitability report.
<code>reduce-lock-overhead</code>	Examine specified annotated sites for opportunities to reduce lock overhead in a Suitability report.
<code>reduce-site-overhead</code>	Examine specified annotated sites for opportunities to reduce site overhead in a Suitability report.
<code>reduce-task-overhead</code>	Examine specified annotated sites for opportunities to reduce task overhead in a Suitability report.
<code>refinalize-survey</code>	Refinalize a survey result collected with a previous Intel® Advisor version or if you need to correct or update source and binary search paths.
<code>remove</code>	Remove loops (by file and line number) from the loops selected for deeper analysis.
<code>report-output</code>	Redirect report output from stdout to another location.
<code>report-template</code>	Specify the PATH/name of a custom report template file.
<code>result-dir</code>	Specify a directory to identify the running analysis.
<code>resume-after</code>	Resume collection after the specified number of milliseconds.
<code>return-app-exitcode</code>	Return the target exit code instead of the command line interface exit code.
<code>search-dir</code>	Specify the location(s) for finding target support files.
<code>search-n-dim</code>	Enable searching for an optimal N-dimensional offload.
<code>select</code>	Select loops (by file and line number, ID, or criteria) for deeper analysis.

Option	Description
<code>set-dependency</code>	Assume loops with specified IDs or source locations have a dependency.
<code>set-parallel</code>	Assume loops with specified IDs or source locations are parallel.
<code>set-parameter</code>	Specify a single-line parameter to modify in a target device configuration.
<code>show-all-columns</code>	Show data for all available columns in a Survey report.
<code>show-all-rows</code>	Show data for all available rows, including data for child loops, in a Survey report.
<code>show-functions</code>	Show only functions in a report.
<code>show-loops</code>	Show only loops in a report.
<code>show-not-executed</code>	Show not-executed child loops in a Survey report.
<code>show-report</code>	Generate a Survey report for data collected for GPU kernels.
<code>small-node-filter</code>	Specify the total time threshold, in milliseconds, to filter out nodes that fall below this value from PDF and DOT Offload Modeling reports.
<code>sort-asc</code>	Sort data in ascending order (by specified column name) in a report.
<code>sort-desc</code>	Sort data in descending order (by specified column name) in a report.
<code>spill-analysis</code>	Register flow analysis to calculate the number of consecutive load/store operations in registers and related memory traffic in bytes during Survey analysis.
<code>stack-access-granularity</code>	Specify stack access size to set stack memory access measurement granularity for the data transfer simulation.
<code>stack-stitching</code>	Restructure the call flow during Survey analysis to attach stacks to a point introducing a parallel workload.
<code>stack-unwind-limit</code>	Set stack size limit for analyzing stacks after collection.
<code>stacks</code>	Perform advanced collection of callstack data during Roofline and Trip Counts & FLOP analysis.
<code>stackwalk-mode</code>	Choose between online and offline modes to analyze stacks during Survey analysis.
<code>start-paused</code>	Start executing the target application for analysis purposes, but delay data collection.
<code>static-instruction-mix</code>	Statically calculate the number of specific instructions present in the binary during Survey analysis.

Option	Description
<code>strategy</code>	Specify processes and/or children for instrumentation during Survey analysis.
<code>support-multi-isa-binaries</code>	Collect a variety of data during Survey analysis for loops that reside in non-executed code paths.
<code>target-device</code>	Specify a device configuration to model cache for during Trip Counts collection.
<code>target-gpu</code>	Specify a target GPU to collect data for if you have multiple GPUs connected to your system.
<code>target-pid</code>	Attach Survey or Trip Counts & FLOP collection to a running process specified by the process ID.
<code>target-process</code>	Attach Survey or Trip Counts & FLOP collection to a running process specified by the process name.
<code>target-system</code>	Specify the hardware configuration to use for modeling purposes in a Suitability report.
<code>threading-model</code>	Specify the threading model to use for modeling purposes in a Suitability report.
<code>threads</code>	Specify the number of parallel threads to use for offload heads.
<code>top-down</code>	Generate a Survey report in top-down view.
<code>trace-mode</code>	Set how to trace loop iterations during Memory Access Patterns analysis.
<code>trace-mpi</code>	Configure collectors to trace MPI code and determine MPI rank IDs for non-Intel® MPI library implementations.
<code>track-memory-objects</code>	Attribute memory objects to the analyzed loops that accessed the objects.
<code>track-stack-accesses</code>	Track accesses to stack memory.
<code>track-stack-variables</code>	Enable parallel data sharing analysis for stack variables during Dependencies analysis.
<code>trip-counts</code>	Collect loop trip counts data during Trip Counts & FLOP analysis.
<code>use-collect-configs</code>	Deprecated.
<code>user-data-dir</code>	Deprecated.
<code>verbose</code>	Maximize status messages during command execution.
<code>with-stack</code>	Show call stack data in a Roofline interactive HTML report (if call stack data is collected).

accuracy

Set an accuracy level for the Offload Modeling collection preset.

GUI Equivalent

Analysis Workflow > Accuracy

Syntax

`--accuracy=<string>`

Arguments

Argument	Description
low	Run Offload Modeling with low accuracy: enable Survey, Characterization (Trip Counts and FLOP), and Performance Modeling analyses.
medium	Run Offload Modeling with medium accuracy: enable Survey, Characterization (Trip Counts and FLOP with cache simulation, callstacks, and light data transfer simulation), and Performance Modeling analyses.
high	Run Offload Modeling with high accuracy: enable Survey, Characterization (Trip Counts and FLOP with cache simulation, callstacks, and medium data transfer simulation), Dependencies, and Performance Modeling analyses.

Default

medium

Actions Modified

`collect=offload`

Usage

The higher accuracy value you choose, the higher modeling accuracy and runtime overhead.

Example

Run the Offload Modeling collection preset with high accuracy.

```
advisor --collect=offload --accuracy=high --project-dir=./advi_results
```

See Also

[Offload Modeling Accuracy Levels in Command Line](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

append

Add loops (by file and line number) to the loops selected for deeper analysis.

GUI Equivalent

Survey >

Syntax

`--append=<string>`

Arguments

<string> is a comma-separated list of files/line numbers in the following format: file1:line1.



Default

No default argument

Actions Modified

mark-up-loops

Usage

Do not confuse the `mark-up-loops` action with the `mark-up-list` action option. The `mark-up-loops` action coupled with the `select` action option enables a GUI  checkbox; therefore loop selection persists beyond the duration of the `mark-up-loops` action and applies to downstream analyses, such as Dependencies and Memory Access Patterns analyses. The `collect` action coupled with the `mark-up-list` action option simulates enabling a GUI  checkbox; therefore loop selection persists only for the duration of the `collect` action.

Example

1. Run a Survey analysis.
2. Select a loop for deeper analysis.
3. Add `bar.cpp:192` to the selection list.
4. Run a Dependencies analysis on both loops.

```
adviser --collect=survey --project-dir=./advi_results -- ./bin/myApplication
adviser --mark-up-loops --select=foo.cpp:34 --project-dir=./advi_results -- ./bin/myApplication
adviser --mark-up-loops --append=bar.cpp:192 --project-dir=./advi_results -- ./bin/myApplication
adviser --collect=dependencies --project-dir=./advi_results -- ./bin/myApplication
```

See Also

[adviser Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `adviser <--action> [--action-options] [--global-options] [--] target [target options]`.

app-working-dir

Specify the directory where the target application runs during analysis, if it is different from the current working directory.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Type] > Working directory

Syntax

`--app-working-dir=<PATH>`

Arguments

<PATH> is a string containing the PATH/name.

Default

Default is the current working directory.

Actions Modified

`collect`

Usage

If your data files are saved in a separate location from the target application, use the `app-working-dir` option to specify the target application working directory.

Example

Run a Survey analysis on `myApplication`. Use `work-dir` to launch `myApplication`.

```
advisor --collect=survey --app-working-dir=./work_dir --project-dir=./advi_results -- ./
myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

assume-dependencies

Assume that a loop has dependencies if the loop dependency type is unknown.

GUI Equivalent

Analysis Workflow > Offload Modeling > Performance Modeling > Assume Dependencies

Syntax

`--assume-dependencies`

`--no-assume-dependencies`

Default

On (assume-dependencies)

NOTE The GUI equivalent is disabled by default.

Actions Modified

`collect=projection`

`collect=offload`

Usage

Use the `no-assume-dependencies` option if you want to minimize the estimated time when your code is bounded by assumed dependencies without running the Dependencies analysis. In this case, if there is no information about a loop from a compiler or the loop is not explicitly marked as parallel, for example, with a programming model (OpenMP*, SYCL, Intel® oneAPI Threading Building Blocks), Intel® Advisor assumes it is parallel and does not have dependencies.

Example

1. Run a Survey analysis.

2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Project application performance assuming loops with an unknown dependency type do not have dependencies.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results  
-- ./myApplication
```

```
advisor --collect=projection --no-assume-dependencies --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

assume-hide-taxes

Estimate invocation taxes assuming the invocation tax is paid only for the first kernel launch.

GUI Equivalent

Analysis Properties > Offload Modeling > Performance Modeling > Single Kernel Launch Tax

Syntax

```
--assume-hide-taxes
```

```
--no-assume-hide-taxes
```

Default

Off (no-assume-hide-taxes)

Actions Modified

```
collect=projection
```

```
collect=offload
```

Usage

You can control how to model kernel invocation taxes for your application. When a high call count value is detected for a potentially profitable code region, Intel® Advisor assumes that the kernel invocation tax is paid as many times as the kernel is launched. This results in high invocation tax and cost of offloading, which means that this code region cannot benefit from offloading.

For simple applications where there is no need to wait for a kernel instance to finish, use the `assume-hide-taxes` to hide this cost every time except the very first one and minimize the invocation tax.

Enable the `assume-hide-taxes` option to analyze oneAPI applications (SYCL or OpenMP* target) [running on CPU](#).

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.

3. Model your application performance on a target device assuming the invocation tax is paid only when the kernel is launched for the first time.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --assume-hide-taxes --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

assume-ndim-dependency

When searching for an optimal N-dimensional offload, assume there are dependencies between inner and outer loops.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--assume-ndim-dependency
```

```
--no-assume-ndim-dependency
```

Default

On (assume-ndim-dependency)

Actions Modified

```
collect=projection
```

```
collect=offload
```

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device assuming there are no dependencies between inner and outer loops when searching for an optimal N-dimensional offload.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --no-assume-ndim-dependency --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

assume-single-data-transfer

Assume data is transferred only once for each offload, and all instances share that data.

Syntax

```
--assume-single-data-transfer
--no-assume-single-data-transfer
```

Default

Off (no-assume-single-data-transfer)

Actions Modified

`collect=projection`

`collect=offload`

Usage

When the option is disabled, use an *optimistic* approach to estimate data transfer taxes: assume data is only transferred *once* for each offload, and all instances share that data.

When the option is enabled, use a *pessimistic* approach to estimate data transfer taxes: assume *each* data object is transferred for every instance of an offload that uses it. This method assumes no data re-use between calls to the same kernel. SYCL, OpenMP* target, and OpenCL™ kernels running on a CPU are still counted only once because the call count for these kernels is usually inflated.

NOTE Make sure to enable the data transfer analysis during the Trip Counts collection with `--data-transfer=[light | medium | full]` or `--enable-data-transfer-analysis`.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage with the light data transfer.
3. Model your application performance assuming data is transferred only once.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --data-transfer=light --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --assume-single-data-transfer --project-dir=./advi_results
```

See Also

[data-transfer](#) Set the level of details for modeling data transfers during Characterization.

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

auto-finalize

Finalize Survey and Trip Counts & FLOP analysis data after collection is complete.

Syntax

```
--assume-single-data-transfer  
--no-assume-single-data-transfer
```

Default

On (auto-finalize)

Actions Modified

```
collect=survey  
collect=tripcounts  
collect=roofline  
collect=offload
```

Usage

Use the `no-auto-finalize` option in situations where you want to suppress finalization, such as when you want to view collected data on a different machine. If finalization is suppressed during data collection and analysis, it occurs automatically when you open the result in the GUI or generate a report from the result.

Disabling temporarily decreases overhead.

Example

Run a Survey analysis. Search recursively for source files in the `./src` search directory. Suppress finalization and write the unfinalized results to the `./advi` project directory instead of the default working directory.

```
advisor --collect=survey --no-auto-finalize --project-dir=./advi_results --search-dir src:=./src  
-- ./bin/myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

batching

Emulate the execution of more than one instance simultaneously for a top-level offload.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--batching  
--no-batching
```

Default

Off (no-batching)

Actions Modified

`collect=projection`

`collect=offload`

Usage

Use the `batching` option to enable job batching for a top-level offload and emulate the execution of more than one instance simultaneously. This option is equivalent to

`--threads=total_EU_count*threads_per_EU`.

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device and emulate the execution of several instances simultaneously for a top-level offload.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results  
-- ./myApplication
```

```
advisor --collect=projection --batching --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

benchmarks-sync

Run benchmarks on only one concurrently executing Intel Advisor instance to avoid concurrency issues with regard to platform limits.

Syntax

`--benchmarks-sync`

`--no-benchmarks-sync`

Default

On (benchmarks-sync)

Actions Modified

`collect`

Usage

Analyze a multi-process application running more than one process on the same host system.

Example

Run a Trip Counts & FLOP analysis for the MPI application `myApplication`. Disable benchmark synchronization to get platform limits corresponding to concurrent runs of multiple processes on the same host system.

```
mpirun -n 4 --gtool="advisor --collect=tripcounts --flop --no-benchmarks-sync --project-dir=./
advi_results:0-3" ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#)

bottom-up

Generate a Survey report in bottom-up view.

GUI Equivalent

Survey > Loop Information

Syntax

`--bottom-up`

`--no-bottom-up`

Default

On (bottom-up)

Actions Modified

`report=survey`

Example

1. Run a Survey analysis.
2. Generate a Survey report. Show a bottom-up list of target loops/functions.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
advisor --report=survey --bottom-up --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

cache-binaries

Enable binary visibility in a read-only snapshot you can view any time.

GUI Equivalent

File > Create Data Snapshot > Cache Binaries

Syntax

`--cache-binaries`

`--no-cache-binaries`

Default

Off (no-cache-binaries)

Actions Modified

snapshot

Example

1. Run a Survey analysis.
2. Create a snapshot. Include binaries.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
```

```
advisor --snapshot --cache-binaries --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command **syntax**: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

cache-binaries-mode

Select what binary files will be added to a read-only snapshot.

Syntax

`--cache-binaries-mode=<string>`

Arguments

`<string>` is one of the following:

Argument	Description
full	Add all binary files to a read-only snapshot.
light	Add only essential binary files to a read-only snapshot.
off	Do not add binary files to a read-only snapshot.

Default

light

Actions Modified

snapshot

Usage

Use `full` mode to add all binary files. This can increase the snapshot size.

If you only need minimal information in a snapshot, for example, only timings to compare them with other results, you can set the mode to **off** or **light** to reduce snapshot size.

Example

Create a snapshot `new_snapshot` and add all binary files.

```
advisor --snapshot --cache-binaries --cache-binaries-mode=full --project-dir=./advi_result --
new_snapshot
```

See Also

[advisor Command Line Interface](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Action Reference](#)

cache-config

Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.

GUI Equivalent

Project Properties > Analysis Target > Trip Counts and FLOP Analysis > Cache simulator configuration

Syntax

`--cache-config=<string>`

Arguments

`<string>` follows this template:

`[num_of_level1_caches]:[num_of_ways_level1_connected]:[level1_cache_size]:[level1_cacheline_size]/`
`[num_of_level2_caches]:[num_of_ways_level2_connected]:[level2_cache_size]:[level2_cacheline_size]/`
`[num_of_level3_caches]:[num_of_ways_level3_connected]:[level3_cache_size]:[level3_cacheline_size]`

For example: 4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l

Actions Modified

`collect=tripcounts --enable-cache-simulation`

`collect=roofline --enable-cache-simulation`

Usage

When no specific configuration is set, the Intel Advisor uses system cache hierarchy for modeling.

NOTE

Cache simulation modeling applies to the following:

- Memory Access Patterns analysis - This basic simulation functionality models accurate memory footprints, miss information, and cache line utilization for a downstream Memory Access Patterns report.
- CPU / Memory Roofline Insights perspective - This enhanced simulation functionality models multiple levels of cache for a downstream Memory-Level Roofline chart or Roofline interactive HTML report.

This option is applicable only to Trip Counts and FLOP and Roofline analyses.

Example

1. Run a Survey analysis.
2. Run a Trip Counts & FLOP analysis. Model cache behavior for the specified configuration.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication

advisor --collect=tripcounts --flop --enable-cache-simulation --cache-
config=4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l --project-dir=./advi_results -- ./
myApplication
```

Run Roofline analysis for all memory levels (Memory-Level Roofline) for the specified cache configuration.

```
advisor --collect=roofline --enable-cache-simulation --cache-config=4:8w:32k:64l/
4:4w:256k:64l/1:16w:6m:64l --project-dir=./advi_results -- ./myApplication
```

See Also

[cachesim-associativity](#) Set the cache associativity for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-cacheline_size](#) Set the cache line size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-mode](#) Set the focus for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-sets](#) Set the cache set size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[enable-cache-simulation](#) Model CPU cache behavior on your target application.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

cache-simulation

Simulate device cache behavior for your application.

GUI Equivalent

Analysis Workflow > Characterization > Cache Simulation Mode

Syntax

```
--cache-simulation=<string>
```

Arguments

<string> is one of the following:

Argument	Description
off	Disable cache simulation.
single	Simulate GPU memory hierarchy behavior only for the selected target device or CPU cache behavior if no target device is selected.
multi	Simulate cache behavior for all available target devices to remodel performance without re-running the collection.
grf	Simulate general register file (GRF) behavior.

Default

off

Actions Modified

`collect=tripcounts`

`collect=roofline`

`collect=offload`

Usage

Enabling cache simulation can increase analysis overhead.

- Use `off` to decrease overhead.
- Use `single` with Offload Modeling if you want to model performance for a single target device. You can use this mode for the CPU Roofline collection. This option is equivalent to `enable-cache-simulation`.
- Use `multi` with Offload Modeling if you want to model performance for several devices or with modified memory parameters without rerunning the Characterization (Trip Count and FLOP) analysis. This option is equivalent to the `flex-cachesim` option (deprecated).
- Use `grf` to ... This option is equivalent to the `enable-grf-simulation` option (deprecated).

Example

Run Characterization analysis with cache simulation enabled to model performance for all available devices.

```
advisor --collect=tripcounts --flop --cache-simulation=multi --target-device=xehpg_512xve --
project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Line Interface](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Action Reference](#)

cache-sources

Enable source code visibility in a read-only snapshot you can view any time (with `--snapshot action`).

Enable keeping source code cache within a project (with `--collect action`).

GUI Equivalent

To add source to a snapshot: **File > Create Data Snapshot > Cache Sources**

To keep cached source during data collection: **Project Properties > Analysis Target > Survey Hotspots Analysis > Source caching**

Syntax

`--cache-sources`

`--no-cache-sources`

Default

Off (no-cache-sources)

Actions Modified

`snapshot`

`collect`

Usage

When used with `collect` action, report is supplied with source code folded like a snapshot. Once set, this option triggers a flag in project configuration that prevents deleting cache with each analysis run unless you manually disable it.

Example

Create a read-only snapshot. Include performance data, sources, and binaries. Save the snapshot to the `tmp` directory. Name the snapshot `myAdvisorProjSnapshot.advixeexpz`.

```
advisor --snapshot --project-dir=./advi_results --pack --cache-sources --cache-binaries -- ./bin/myApplication
```

Run a Survey analysis and keep source cache.

```
advisor --collect=survey --cache-sources --project-dir=./advi_results -- ./bin/myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

cachessim

Enable cache simulation for Performance Modeling.

Syntax

`--cachessim`

`--no-cachessim`

Default

On (cachessim)

Actions Modified

`collect=projection`

Usage

Cache simulation data might be unavailable for some loops. Performance Modeling result may be incorrect. Use this option to enable the cache simulation data for the Performance Modeling.

NOTE Make sure to use the `--enable-cache-simulation` and `--cache-config` command line arguments when collection Trip Counts data to improve projection accuracy.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage with cache simulation for a specific cache configuration.

3. Model your application performance and explicitly enable cache simulation .

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication

advisor --collect=tripcounts --flop --enable-cache-simulation --cache-config=4:8w:32k:64l/
4:4w:256k:64l/1:16w:6m:64l --project-dir=./advi_results -- ./myApplication

advisor --collect=projection --cachesim --project-dir=./advi_results
```

See Also

[enable-cache-simulation](#) Model CPU cache behavior on your target application.

[cache-config](#) Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[cachesim-associativity](#)

Set the cache associativity for modeling CPU cache behavior during Memory Access Patterns analysis.

GUI Equivalent

Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Cache associativity

Syntax

`--cachesim-associativity=<integer>`

Arguments

`<integer>` is the number of cache locations where one memory entry can be placed: 1 | 2 | 4 | 8 | 16

Default

8

Actions Modified

`collect=map` --enable cache-simulation

Usage

1 stands for a direct mapped cache, where a memory entry can occupy only one cache line.

NOTE

Cache simulation modeling applies to the following:

- Memory Access Patterns analysis - This basic simulation functionality models accurate memory footprints, miss information, and cache line utilization for a downstream Memory Access Patterns report.
- CPU / Memory Roofline Insights perspective - This enhanced simulation functionality models multiple levels of cache for a downstream Memory-Level Roofline chart or Roofline interactive HTML report.

This option is applicable only to Memory Access Patterns analysis.

Example

Run a Memory Access Patterns analysis. Model four-way associative cache with default cache line and cache set size.

```
advisor --collect=map --enable-cache-simulation --cachesim-associativity=4 --cachesim-
mode=utilization --project-dir=./advi_results -- ./myApplication
```

See Also

[cache-config](#) Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.

[cachesim-cacheline_size](#) Set the cache line size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-mode](#) Set the focus for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-sets](#) Set the cache set size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[enable-cache-simulation](#) Model CPU cache behavior on your target application.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

cachesim-cacheline-size

Set the cache line size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

GUI Equivalent

Project Properties > Analysis Target > Memory Access Patterns > Advanced > Cache line size

Syntax

```
--cachesim-cacheline-size=<integer>
```

Arguments

<integer> is in bytes: 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536

Default

64

Actions Modified

`collect=map` `--enable-cache-simulation`

Usage

NOTE

Cache simulation modeling applies to the following:

- Memory Access Patterns analysis - This basic simulation functionality models accurate memory footprints, miss information, and cache line utilization for a downstream Memory Access Patterns report.
- CPU / Memory Roofline Insights perspective - This enhanced simulation functionality models multiple levels of cache for a downstream Memory-Level Roofline chart or Roofline interactive HTML report.

This option is applicable only to Memory Access Patterns analysis.

Example

Run a Memory Access Patterns analysis. Model four-way associative cache with 64-byte cache line size and default cache set size.

```
advisor --collect=map --enable-cache-simulation --cachesim-cacheline-size=64 --
cachesim-associativity=4 --cachesim-mode=utilization --project-dir=./advi_results -- ./
myApplication
```

See Also

[cache-config](#) Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.

[cachesim-associativity](#) Set the cache associativity for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-mode](#) Set the focus for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-sets](#) Set the cache set size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[enable-cache-simulation](#) Model CPU cache behavior on your target application.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[cachesim-mode](#)

Set the focus for modeling CPU cache behavior during Memory Access Patterns analysis.

GUI Equivalent

Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Cache simulation mode

Syntax

```
--cachesim-mode=<string>
```

Arguments

<string> is one of the following:

Argument	Description
cache-misses	Model cache misses only.
footprint	Model cache misses and memory footprint of a loop. Calculation: Cache line size x Number of unique cache lines accessed during simulation.
utilization	Model cache misses and cache lines utilization.

Default

utilization

Actions Modified

`collect=map --enable-cache-simulation`

Usage

For memory footprint simulation, the Intel Advisor tracks only a subset of accesses and cache lines, and scales it up to the total size of cache to calculate the final footprint value.

NOTE

Cache simulation modeling applies to the following:

- Memory Access Patterns analysis - This basic simulation functionality models accurate memory footprints, miss information, and cache line utilization for a downstream Memory Access Patterns report.
- CPU / Memory Roofline Insights perspective - This enhanced simulation functionality models multiple levels of cache for a downstream Memory-Level Roofline chart or Roofline interactive HTML report.

This option is applicable only to Memory Access Patterns analysis.

Tip

Usage can increase analysis overhead.

Example

Run a Memory Access Patterns analysis. Model cache misses for a default cache configuration.

```
advisor collect=map --enable-cache-simulation --cachesim-mode=cache-misses --project-dir=./  
advi_results -- ./myApplication
```

Run a Memory Access Patterns analysis. Model cache miss and memory footprint data for 1024-byte cache set size, default cache associativity and cache line size.

```
advisor collect=map --enable-cache-simulation --cachesim-sets=1024 --cachesim-mode=footprint --  
project-dir=./advi_results -- ./myApplication
```

See Also

[cache-config](#) Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.

[cachesim-associativity](#) Set the cache associativity for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-cacheline_size](#) Set the cache line size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-sets](#) Set the cache set size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[enable-cache-simulation](#) Model CPU cache behavior on your target application.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

cachesim-sampling-factor

Specify what percentage of total memory accesses should be processed during cache simulation.

Syntax

```
--cachesim-sampling-factor=<integer>
```

Arguments

`<integer>` is the percentage of total number of memory accesses to process: 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100

Default

10

Actions Modified

`collect=tripcounts` `--enable-cache-simulation`

`collect=roofline` `--enable-cache-simulation`

Usage

If loops in your application have mixed memory access patterns, Intel® Advisor might not be able to capture actual cache behavior when analyzing only the default 10% of memory accesses, which may decrease cache simulation accuracy. Increase the sampling factor to improve accuracy. This option also increases collection overhead.

NOTE Use with the `--enable-cache-simulation` option.

Example

Run the Trip Counts analysis with cache simulation processing 50% of memory accesses.

```
advisor --collect=tripcounts --enable-cache-simulation --cachesim-sampling-factor=50 --project-dir=./advi_results -- ./myApplication
```

See Also

[enable-cache-simulation](#) Model CPU cache behavior on your target application.

[advisor Command Line Interface](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Action Reference](#)

[Minimize Analysis Overhead](#)

cachesim-sets

Set the cache set size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

GUI Equivalent

Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Cache sets

Syntax

```
--cachesim-sets=<integer>
```

Arguments

<integer> is in bytes: 256 | 512 | 1024 | 2048 | 4096 | 8192

Default

4096

Actions Modified

`collect=map` --enable cache-simulation

Usage

NOTE

Cache simulation modeling applies to the following:

- Memory Access Patterns analysis - This basic simulation functionality models accurate memory footprints, miss information, and cache line utilization for a downstream Memory Access Patterns report.
- CPU / Memory Roofline Insights perspective - This enhanced simulation functionality models multiple levels of cache for a downstream Memory-Level Roofline chart or Roofline interactive HTML report.

This option is applicable only to Memory Access Patterns analysis.

Example

Run a Memory Access Patterns analysis. Model cache misses for 2048-byte cache set size, default cache associativity and cache line size.

```
advisor collect=map --enable-cache-simulation --cachesim-sets=2048 --cachesim-mode=cache-misses  
--project-dir=./advi_results -- ./myApplication
```

See Also

[cache-config](#) Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.

[cachesim-associativity](#) Set the cache associativity for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-cacheline_size](#) Set the cache line size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-mode](#) Set the focus for modeling CPU cache behavior during Memory Access Patterns analysis.

[enable-cache-simulation](#) Model CPU cache behavior on your target application.

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

check-profitability

Check the profitability of offload regions and add only profitable regions to a report.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--check-profitability
--no-check-profitability
```

Default

On (check-profitability)

Actions Modified

```
collect=projection
collect=offload
```

Usage

Use the `no-check-profitability` option to add all offloadable regions to a report even if they do not benefit from the increased speed after offloading.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device and do not check loop profitability.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results
-- ./myApplication
```

```
advisor --collect=projection --no-check-profitability --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

clear

Clear all loops previously selected for deeper analysis.

GUI Equivalent

Survey >  (deselect)

Syntax

```
--clear
```

--no-clear

Default

Off (no-clear)

Actions Modified

mark-up-loops

Example

Clear loops previously selected for analysis.

```
advisor --mark-up-loops --clear --project-dir=./advi_results -- ./bin/myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

config

Specify a device configuration to model cache configuration/your application performance for.

GUI Equivalent

Analysis Workflow > Offload Modeling > Target Platform Model

Project Properties > Analysis Target > Performance Modeling > Target Config

Syntax

--config=<string>

Arguments

<string> is one of the following device configurations:

Argument	Description
xehpg_256xve	Intel® Arc™ graphics with 256 vector engines
xehpg_512xve	Intel® Arc™ graphics with 512 vector engines
gen12_tgl	Intel® Iris® Xe graphics
gen12_dg1	Intel® Iris® Xe MAX graphics
gen11_icl	Intel® Iris® Plus graphics
gen9_gt2	Intel® HD Graphics 530
gen9_gt3	Intel® Iris® Graphics 550
gen9_gt4	Intel® Iris® Pro Graphics 580

Default

xehpg_512xve

Actions Modified

collect=projection

collect=tripcounts

collect=offload

Usage

Important

Make sure to specify the same configuration argument as for the [target-device](#) option during Trip Counts collection (collect=tripcounts).

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance for the gen12_dg1 configuration.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --target-device=gen12_dg1 --  
project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --config=gen12_dg1 --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

count-logical-instructions

Use the projection of x86 logical instructions to GPU logical instructions.

Syntax

--count-logical-instructions

--no-count-logical-instructions

Default

On (count-logical-instructions)

Actions Modified

collect=projection

collect=offload

Example

Model your application performance on a target device and do not project x86 logical instructions to GPU logical instructions.

```
advisor --collect=projection --no-count-logical-instructions --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

count-memory-instructions

Project x86 memory instructions to GPU SEND/SENDS instructions.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

`--count-memory-instructions`

`--no-count-memory-instructions`

Default

On (count-memory-instructions)

Actions Modified

`collect=projection`

`collect=offload`

Example

Model your application performance on a target device and disable projecting x86 instructions with memory.

```
advisor --collect=projection --no-count-memory-instructions --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

count-memory-objects-accesses

Count the number of accesses to memory objects created by code regions.

Syntax

`--count-memory-objects-accesses`

`--no-count-memory-objects-accesses`

Default

Off (no-count-memory-objects-accesses)

Actions Modified

`collect=projection --data-reuse-analysis`

Usage

Use as *one* of the following:

- Use the *full* data transfer with `collect=tripcounts`:

```
advisor --collect=tripcounts --flop --data-transfer=full --project-dir=<project-dir>
-- <target-application>
```

```
advisor --collect=projection --data-reuse-analysis --count-memory-objects-accesses
--project-dir=<project-dir>
```

- Enable the *basic data transfer analysis* and data-reuse-analysis with `collect=tripcounts`:

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis--data-reuse-
analysis --project-dir=<project-dir> -- <target-application>
```

```
advisor --collect=projection --data-reuse-analysis --count-memory-objects-accesses
--project-dir=<project-dir>
```

Example

With the full data transfer analysis:

1. Run the Survey analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage with the full data transfer analysis.
3. Analyze data reuse and count the number of accesses to memory objects when modeling your application performance.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --data-transfer=full --project-dir=./advi_results -- ./
myApplication
```

```
advisor --collect=projection --data-reuse-analysis --count-memory-objects-accesses --project-
dir=./advi_results
```

See Also

[data-reuse-analysis](#) Analyze potential data reuse between code regions.

[data-transfer](#) Set the level of details for modeling data transfers during Characterization.

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

count-mov-instructions

Project x86 MOV instructions to GPU MOV instructions.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

`--count-mov-instructions`

`--no-count-mov-instructions`

Default

Off (no-count-mov-instructions)

Actions Modified

`collect=projection`

`collect=offload`

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device and projecting x86 MOV instructions to GPU MOV.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results  
-- ./myApplication
```

```
advisor --collect=projection --count-mov-instructions --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

count-send-latency

Select how to model SEND instruction latency.

Syntax

`--count-send-latency=<string>`

Arguments

`<string>` is one of the following:

Argument	Description
all	Assume each SEND instruction has an uncovered latency.
first	Assume only the first SEND instruction in a thread has an uncovered latency.
off	Do not model SEND instruction latency.

Default

off

Actions Modified

`collect=projection`

`collect=offload`

Usage

Use `first` for CPU-to-GPU offload modeling.

Use `all` for GPU-to-GPU offload modeling.

Example

Model performance of your application on a target device assuming only the first send instruction has an uncovered latency.

```
advisor --collect=projection --count-send-latency=first --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

cpu-scale-factor

Specify a scale factor to approximate a host CPU that is faster than the baseline CPU by this factor.

Syntax

```
--cpu-scale-factor=<double>
```

Actions Modified

`collect`=projection

`collect`=offload

Usage

With this option, all original CPU times are divided by the factor specified.

Example

Model your application performance on a target device and approximate a host CPU that is 3 times faster than the original CPU.

```
advisor --collect=projection --cpu-scale-factor=3 --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

csv-delimiter

Set the delimiter for a report in CSV format.

Syntax

```
--csv-delimiter=<string>
```

Arguments

`<string>` is one of the following: comma | semicolon | tab

Default

comma

Actions Modified

`report`=[report type] --format=csv

Example

Generate a Dependencies report. Output in CSV format with tab delimiters.

```
advisor --report=dependencies --format=csv --csv-delimiter=tab --report-output=./out/advisor-Dependencies.csv --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

custom-config

Specify the absolute path or name for a custom TOML configuration file with additional modeling parameters.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Custom Device Configuration

Syntax

`--custom-config=<path>`

Arguments

`<path>` is the absolute path or a name of a custom TOML configuration file.

Actions Modified

`collect`=projection

`collect`=offload

Usage

If you specify the configuration file name, the configuration directory will be searched first, then the current directory. See [Advanced Modeling Configuration](#) for details about modeling parameters available.

You can specify this option more than once to provide multiple custom configuration files for the Performance Modeling. For example, you can provide a `scalers.toml` file generated from an Offload Modeling HTML report and a manually created configuration file with additional parameters.

When used with `--collect=offload`, the custom configuration file is checked for a memory structure description at the Trip Counts analysis step. This description is used to generate cache configuration for cache simulation. If you specify several configurations, Intel Advisor only uses the last configuration with memory structure parameters for Trip Counts and ignores other files. For example, if you specified a target device configuration with `config` and additional parameters with `custom-config`, Intel Advisor only uses the target device configuration for the Trip Counts analysis and ignores the custom configuration, but for the Performance Modeling, it uses both configuration files.

NOTE `target-device`, `config`, and `cache-config` options take precedence over memory structure parameters specified with `custom-config`.

Example

1. Create a TOML configuration file and specify additional parameters.
2. Run Survey Analysis.
3. Run Trip Counts and FLOP analyses of the Characterization stage.
4. Model your application performance on a target device with the custom `myConfig.toml` file.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results
-- ./myApplication
```

```
advisor --collect=projection --custom-config=./myConfig.toml --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

data-limit

Limit the maximum amount (in MB) of raw data collected during Survey analysis.

GUI Equivalent

Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Collection data limit

Syntax

```
--data-limit=<integer>
```

Arguments

`<integer>` is the maximum collection size, in MB.

Default

500

Actions Modified

`collect=survey`

Usage

This option is useful if you have storage space limitations. A smaller value can also decrease collection overhead.

Example

Run a Survey analysis. Stop data collection when a 250-MB limit is reached or upon normal completion.

```
advisor --collect=survey --data-limit=250 --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimizing Analysis Overhead](#)

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

data-reuse-analysis

Analyze potential data reuse between code regions.

GUI Equivalent

Analysis Workflow > Offload Modeling > Performance Modeling > Data Reuse Analysis

Syntax

```
--data-reuse-analysis
--no-data-reuse-analysis
```

Default

Off (no-data-reuse-analysis)

Actions Modified

```
collect=offload
collect=tripcounts --enable-data-transfer-analysis
collect=projection
```

Usage

With `collect=offload`, this option automatically applies the `data-reuse-analysis` option to all analyses it runs.

With `collect=tripcounts` and `collect=projection`, use as *one* of the following:

- Use the *full* data transfer with `collect=tripcounts` and specify `data-reuse-analysis` only for `collect=projection`:

```
advisor --collect=tripcounts --flop --data-transfer=full --project-dir=<project-dir>
-- <target-application>

advisor --collect=projection --data-reuse-analysis --project-dir=<project-dir>
```
- Enable the *basic data transfer analysis* with `collect=tripcounts` and specify `data-reuse-analysis` for both `collect=tripcounts` and `collect=projection`:

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis--data-reuse-
analysis --project-dir=<project-dir> -- <target-application>

advisor --collect=projection --data-reuse-analysis --project-dir=<project-dir>
```

Example

Run the Offload Modeling with data reuse analysis using a collection preset.

```
advisor --collect=offload --data-reuse-analysis --project-dir=./advi_results
```

With the full data transfer analysis:

1. Run the Survey analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage with the full data transfer analysis.

3. Analyze data reuse when modeling your application performance.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication  
advisor --collect=tripcounts --flop --data-transfer=full --project-dir=./advi_results -- ./myApplication  
advisor --collect=projection --data-reuse-analysis --project-dir=./advi_results
```

See Also

[data-transfer](#) Set the level of details for modeling data transfers during Characterization.

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

data-transfer

Set the level of details for modeling data transfers during Characterization.

GUI Equivalent

Analysis Workflow > Offload Modeling > Characterization > Data Transfer Simulation

Project Properties > Analysis Target > Survey Analysis Types > Trip Counts and FLOP analysis > Advanced > Data transfer simulation mode

Syntax

`--data-transfer=off | light | medium | full`

Arguments

Argument	Description
off	Disable data transfer simulation.
light	Model data transfer between host and device memory.
medium	Model data transfer between host and device memory, attribute memory objects to the analyzed loops that accessed the objects, and track accesses to stack memory.
full	Model data transfers, attribute memory objects, track accesses to stack memory, and identify where data can be potentially reused if transferred between host and target.

Default

off

Actions Modified

`collect=tripcounts`

`collect=offload`

Usage

Usage can increase analysis overhead.

NOTE

`data-transfer` option takes precedence over the `enable-data-transfer-analysis` option and its modifications. So if you specify both, `data-transfer` rewrites all `enable-data-transfer-analysis` modifications used.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage with full data transfer simulation.
3. Model your application performance on a target device.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --data-transfer=full --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

data-transfer-histogram

Estimate data transfers in details and latencies for each transferred object.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--data-transfer-histogram
```

```
--no-data-transfer-histogram
```

Default

Off (no-data-transfer-histogram)

Actions Modified

`collect=projection`

Usage

Use as *one* of the following:

- Use the *medium* or *full* data transfer with `collect=tripcounts` and specify only `data-transfer-histogram` for `collect=projection`. For example:

```
advisor --collect=tripcounts --flop --data-transfer=medium --project-dir=<project-dir> -- <target-application>
```

```
advisor --collect=projection --data-transfer-histogram --project-dir=<project-dir>
```

- Enable the *basic data transfer analysis* and `track-memory-objects` with `collect=tripcounts` and specify `track-memory-objects` and `data-transfer-histogram` for `collect=projection`:

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis--track-memory-objects --project-dir=<project-dir> -- <target-application>

advisor --collect=projection --track-memory-objects--data-transfer-histogram --project-dir=<project-dir>
```

Example

With the medium data transfer analysis:

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device and enable data transfer histogram.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication

advisor --collect=tripcounts --flop --data-transfer=medium --project-dir=./advi_results -- ./myApplication

advisor --collect=projection --data-transfer-histogram --project-dir=./advi_results
```

See Also

[data-transfer](#) Set the level of details for modeling data transfers during Characterization.

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[track-memory-objects](#) Attribute memory objects to the analyzed loops that accessed the objects.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

data-transfer-page-size

Specify memory page size to set the traffic measurement granularity for the data transfer simulator.

Syntax

```
--data-transfer-page-size=<integer>
```

Arguments

<integer> is a power-of-two value in range of 4 to 8192: 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192

Default

4096

Actions Modified

`collect=tripcounts --enable-data-transfer-analysis`

`collect=tripcounts --data-transfer=<mode>`

Example

Run a Trip Counts and FLOP analysis. Enable data transfer simulation with 512-bites memory page size.

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --data-transfer-page-size=512 --project-dir=./advi_results -- ./myApplication
```


See Also

[data-transfer](#) Set the level of details for modeling data transfers during Characterization.

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

data-type

Show only floating-point data, only integer data, or data for the sum of both data types in a Roofline interactive HTML report.

GUI Equivalent

Roofline > Default: FLOAT > Operations

Syntax

`--data-type=<string>`

Arguments

`<string>` is one of the following: float | int | mixed

Default

float

Actions Modified

`report=roofline`

Example

Generate a Roofline interactive HTML report. Include floating-point data; exclude integer data.

```
advisor --report=roofline --data-type=float --project-dir=./advi_results
```

See Also

[Command Line Interface Reference](#)

[advisor Command Action Reference](#)

delete-tripcounts

Remove previously collected trip counts data when re-running a Survey analysis with changed binaries.

GUI Equivalent

Warning message

Syntax

`--delete-tripcounts`

`--no-delete-tripcounts`

Default

On (delete-tripcounts)

Actions Modified

`collect=survey`

Usage

Enable to eliminate the risk of including out-of-date data in a new Survey analysis if you:

- Change binaries.
- Have previously collected trip counts/FLOP data.

In other cases, this option is ignored.

Example

Run a Survey analysis. Remove previously collected trip counts data during analysis.

```
adviser --collect=survey --search-dir src=./src bin=./bin --delete-tripcounts --project-dir=./
advi_results -- ./myApplication
```

See Also

[adviser Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `adviser <--action> [--action-options] [--global-options] [[--] target [target options]]`.

`disable-fp64-math-optimization`

*Do not account for optimized traffic for
transcendentals on a GPU.*

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--disable-fp64-math-optimization
--no-disable-fp64-math-optimization
```

Default

Off (no-disable-fp64-math-optimization)

Actions Modified

`collect=projection`

`collect=offload`

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device and disable accounting for optimized traffic for transcendentals .

```
adviser --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
adviser --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results
-- ./myApplication
```

```
adviser --collect=projection --disable-fp64-math-optimization --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

display-callstack

Show a callstack for each loop/function call in a report.

Syntax

`--display-callstack`

Actions Modified

`report`

Example

Generate a Suitability report. Include callstack data.

```
advisor --report=suitability --display-callstack --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

dry-run

List all steps included in Offload Modeling collection preset at a specified accuracy level without running them.

Syntax

`--dry-run`

`--no-dry-run`

Default

`no-dry-run`

Actions Modified

`collect=offload`

Example

List all steps for the high accuracy level of the Offload Modeling.

```
advisor --collect=offload --accuracy=high --config=xehpg_512xve --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

duration

Specify the maximum amount of time (in seconds) an analysis runs.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Type] > Advanced > Automatically stop collection after (sec)

Syntax

```
-d=<string>  
--duration=<string>
```

Arguments

<string> is maximum number of seconds or unlimited.

Default

unlimited

Actions Modified

collect

Usage

The target application also stops executing when the analysis stops running.

Example

Stop a Survey analysis after 60 seconds.

```
advisor --collect=survey --duration=60 --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

dynamic

Show (in a Survey report) how many instructions of a given type actually executed during Trip Counts & FLOP analysis.

GUI Equivalent**Code Analytics****Syntax**

```
--dynamic  
--no-dynamic
```

Default

On (dynamic)

Actions Modified

`report=survey`

Usage

Dynamic instruction mix is counted for the entire execution of the application; static instruction mix is counted per iteration. The `static-instruction-mix`, `dynamic`, and `mix` options work together in the following manner:

- Collect static instruction mix data: `--collect=survey --static-instruction-mix`
(In the GUI: Static instruction mix data is calculated on demand.)
- Collect dynamic instruction mix data (and static instruction mix data, from which dynamic mix data is calculated): `--collect=tripcounts --flop`
- Show static instruction mix data in a Survey report: `--report=survey --mix --no-dynamic`
- Show dynamic mix instruction data in a Survey report: `--report=survey --mix --dynamic`
- A Survey report cannot show both static and dynamic mix instruction data.
(In the GUI: **Code Analytics** can show both static and dynamic instruction mix data.)

Example

1. Run a Survey analysis.
2. Run a Trip Counts & FLOP analysis. Collect dynamic instruction mix data (and static instruction mix data, from which dynamic mix data is calculated).
3. Generate a Survey report. Show dynamic instruction mix data. (`dynamic` is on, by default).

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
advisor --collect=tripcounts --flop --project-dir=./advi_results -- ./myApplication
advisor --report=survey --mix --project-dir=./advi_results
```

1. Run a Survey analysis. Collect static instruction mix data.
2. Generate a Survey report. Show static instruction mix data.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
advisor --report=survey --mix --no-dynamic --project-dir=./advi_results
```

See Also

[mix](#) Show dynamic or static instruction mix data in a Survey report.

[static-instruction-mix](#) Statically calculate the number of specific instructions present in the binary during Survey analysis.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

enable-cache-simulation

Model CPU cache behavior on your target application.

GUI Equivalent

For basic modeling functionality: **Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Enable CPU cache simulation**

For enhanced modeling functionality:

Project Properties > Analysis Target > Trip Counts and FLOP Analysis > Enable CPU cache simulation or

Analysis Workflow > [CPU | GPU] Roofline > Characterization > Enable CPU cache simulation

Syntax

```
--enable-cache-simulation  
--no-enable-cache-simulation
```

Default

Off (no-enable-cache-simulation)

Actions Modified

```
collect=map  
collect=tripcounts  
collect=roofline  
collect=offload
```

Usage

Enabling can increase collection overhead.

NOTE

Cache simulation modeling applies to the following:

- Memory Access Patterns analysis - This basic simulation functionality models accurate memory footprints, miss information, and cache line utilization for a downstream Memory Access Patterns report.
 - CPU / Memory Roofline Insights perspective - This enhanced simulation functionality models multiple levels of cache for a downstream Memory-Level Roofline chart or Roofline interactive HTML report.
-

Example

Run a Memory Access Patterns analysis. Enable cache simulation with basic functionality and default cache parameters to collect cache modeling data for a downstream Memory Access Patterns report.

```
advisor --collect=map --enable-cache-simulation --project-dir=./advi_results -- ./myApplication
```

Run a Roofline analysis for all memory levels.

```
advisor --collect=roofline --enable-cache-simulation --project-dir=./advi_results -- ./myApplication
```

See Also

[cache-config](#) Set the cache hierarchy to collect modeling data for CPU cache behavior during Trip Counts & FLOP analysis.

[cachesim-associativity](#) Set the cache associativity for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-cacheline_size](#) Set the cache line size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-mode](#) Set the focus for modeling CPU cache behavior during Memory Access Patterns analysis.

[cachesim-sets](#) Set the cache set size (in bytes) for modeling CPU cache behavior during Memory Access Patterns analysis.

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

enable-data-transfer-analysis

Model data transfer between host memory and device memory.

Syntax

`--enable-data-transfer-analysis--no-enable-data-transfer-analysis`

Default

Off (no-enable-data-transfer-analysis)

Actions Modified

`collect=tripcounts`

Usage

NOTE

`data-transfer` option takes precedence over the `enable-data-transfer-analysis` option and its modifications. So if you specify both, `data-transfer` rewrites all `enable-data-transfer-analysis` modifications used.

`enable-data-transfer-analysis` with no modifications corresponds to `data-transfer=light`.

Example

Run a Trip Counts and FLOP analysis. Enable data transfer simulation.

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./
advi_results -- ./myApplication
```

See Also

[data-transfer-page-size](#) Specify memory page size to set the traffic measurement granularity for the data transfer simulator.

[track-memory-objects](#) Attribute memory objects to the analyzed loops that accessed the objects.

[track-stack-accesses](#) Track accesses to stack memory.

[data-reuse-analysis](#) Analyze potential data reuse between code regions.

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

enable-task-chunking

Examine specified annotated sites for opportunities to perform task-chunking modeling in a Suitability report.

GUI Equivalent

Suitability > Enable Task Chunking

Syntax

`--enable-task-chunking=<string>`

Arguments

`<string>` is a comma-separated list of annotated sites (no spaces).

Default

No default argument

Actions Modified

`report=suitability`

Example

Generate a Suitability report. Include task-chunking analysis for the annotated sites `myAnnotatedSiteJ` and `myAnnotatedSiteX`.

```
advisor --report=suitability --enable-task-chunking=myAnnotatedSiteJ,myAnnotatedSiteX --project-dir=./advi_results
```

See Also

[Enable Task Chunking](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

enforce-baseline-decomposition

Use the same local size and SIMD width as measured on a baseline device.

Syntax

`--enforce-baseline-decomposition`

`--no-enforce-baseline-decomposition`

Default

Off (no-enforce-baseline-decomposition)

Actions Modified

`collect=projection --profile-gpu`

Usage

This option is applicable only to the GPU-to-GPU performance modeling workflow.

When used with the Performance Modeling (`collect=projection`) as part of the GPU Roofline Insights perspective for an application executed on a GPU, your application performance is modeled for a baseline GPU device as a target. The estimated performance is compared with the actual application performance to add more recommendations for performance optimization.

This option is recommended to be used with the `model-baseline-gpu` option.

Example

Run the GPU-to-GPU Performance Modeling for the baseline GPU using the same local size and SIMD width as on the baseline device.

```
advisor --collect=projection --profile-gpu --model-baseline-gpu --enforce-baseline-decomposition  
--project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Line Interface](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Action Reference](#)

enforce-fallback

Emulate data distribution over stacks if stacks collection is disabled.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--enforce-fallback  
--no-enforce-fallback
```

Default

Off (no-enforce-fallback)

Actions Modified

```
collect=offload  
collect=projection
```

Usage

Use the `enforce-fallback` option to emulate data distribution over stacks after reducing collection overhead by removing `--stacks` option from Trip Counts collection (`--no-stacks` is default).

With `collect=offload`, this option automatically disables the stack collection and enables the fallback.

NOTE

This may reduce analysis accuracy.

Example

Run the Offload Modeling with enforced fallback using batch collection.

```
advisor --collect=offload --enforce-fallback --project-dir=./advi_results
```

Model your application performance emulating data distribution over stacks.

```
advisor --collect=projection --enforce-fallback --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

enforce-offloads

Offload all selected code regions even if offloading their child loops/functions is more profitable.

Syntax

`--enforce-offloads`
`--no-enforce-offloads`

Default

Off (no-enforce-offloads)

Actions Modified

`collect=projection`

Usage

Use this option if you want to check offload profitability for all selected loops or if your loop of interest is reported as not recommended for offloading to a target device. This option may be useful for remodeling when you run the Performance Projection several times with different modeling parameters or for different targets without rerunning the collection. This option skips the profitability check, disables analyzing child loops and functions to make sure the loops selected for offload are offloaded even if offloading child loops is more profitable.

You can use it with the `select` to specify code regions to offload by their source location, ID, selection criteria or using a pre-defined mark-up strategy.

Example

Run the Performance Modeling and offload all code regions.

```
advisor --collect=projection --enforce-offload --project-dir=./advi_results
```

See Also

advisor Command Line Interface This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

advisor Command Action Reference

estimate-max-speedup

Estimate region speedup with relaxed constraints.

Syntax

`--estimate-max-speedup`
`--no-estimate-max-speedup`

Default

On (estimate-max-speedup)

Actions Modified

`collect=projection`

Usage

Disabling can decrease collection overhead.

Example

Model your application performance on a target device and explicitly enable estimations with relaxed constraints.

```
advisor --collect=projection --estimate-max-speedup --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

evaluate-min-speedup

Consider loops recommended for offloading only if they reach the minimum estimated speedup specified in a configuration file.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--evaluate-min-speedup  
--no-evaluate-min-speedup
```

Default

Off (no-evaluate-min-speedup)

Actions Modified

`collect=projection`

Usage

You need to specify a minimum speedup for a region to be estimated as recommended for offloading in a custom TOML configuration file with a `min_required_speed_up` parameter. By default, the minimum speedup is set to 1.

Example

1. Create a custom configuration file with `min_required_speed_up` set to 2.
2. Run Survey Analysis.
3. Run Trip Counts and FLOP analyses of the Characterization stage.
4. Model your application performance on a target device with a minimum speedup set to 2.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results  
-- ./myApplication
```

```
advisor --collect=projection --custom-config=myConfig.toml --evaluate-min-speedup --project-  
dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

exclude-files

Exclude the specified files or directories from annotation scanning during analysis.

GUI Equivalent

Project Properties > Source Search > Exclude the following files

Syntax

`--exclude-files=<PATH>`

Arguments

`<PATH>` is a comma-separated list of file paths or directories to exclude during data collection (no spaces).

Default

No default argument

Actions Modified

`collect`

Example

Run a Suitability analysis. Exclude all files in the two specified source directories from annotation scanning during analysis.

```
advisor --collect=suitability --exclude-files=./src1,./src2 --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

executable-of-interest

Specify an application for analysis that is not the starting application.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Type] > Child Application

Syntax

`--executable-of-interest=<PATH>`

Arguments

`<PATH>` is a string specifying the PATH/name of the executable to be analyzed.

Actions Modified

collect

Usage

Specify an executable child process to analyze, instead of the script or application that spawns the child process.

Example

Run a Survey analysis launched from myScript. Specify myApplication as the executable of interest.

```
advisor --collect=survey --project-dir=./advi_results --executable-of-interest=./  
myApplication.exe -- ./myScript
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

exp-dir

Specify a path to an unpacked result snapshot or an MPI rank result to generate a report or model performance.

Syntax

`--exp-dir=<path>`

Arguments

`<path>` is the path or a name of a directory to an unpacked snapshot or an MPI rank result.

Default

Current working directory

Actions Modified

collect=projection

report

Usage

If you specify the `--exp-dir` option, you do not need to specify a project directory.

With `--collect=projection`, if you have a result snapshot, you can run Performance Modeling without an original project or application source/executable.

NOTE If you have a packed snapshot, unpack it first using the [import-dir](#) action.

Example

Model performance for a mySnapshot snapshot.

```
advisor --collect=projection --exp-dir=./mySnapshot
```

Generate a Survey report from the result in the `advi_result` directory for the MPI rank 2. Output the report in text format as `mpi_survey.txt`.

```
advisor --report=survey --format=text --report-output=./out/mpi_survey.txt --exp-dir=./advi_results/rank.2
```

See Also

[advisor Command Line Interface](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Action Reference](#)

[Create a Read-only Result Snapshot](#)

filter

Filter data by the specified column name and value in a Survey and Trips Counts & FLOP report.

GUI Equivalent

Filters

Syntax

```
--filter=<string>
```

Arguments

`<string>` is in the following format: "[column name]"="[value]".

Default

No default argument

Actions Modified

`report=survey`

`report=tripscounts`

Example

Generate a Survey report. Show the top five self-time hotspots that were not vectorized because of a *not inner loop* msg id.

```
advisor --report=survey --limit=5 --filter="Vectorization Message(s)"="loop was not vectorized: not inner loop" --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

filter-by-scope

Enable filtering detected stack variables by scope (warning vs. error) in a Dependencies analysis.

GUI Equivalent

Project Properties > Analysis Target > Dependencies Analysis > Advanced > Filter stack variables by scope

Syntax

--filter-by-scope
--no-filter-by-scope

Default

Off (no-filter-by-scope)

Actions Modified

collect=dependencies

Usage

Variables initiated inside the specified loop(s) are considered potential dependencies (warning). Variables initiated outside the specified loop(s) are considered dependencies (error).

Disabling can decrease collection overhead.

Example

Run a Dependencies analysis. Analyze innermost scalar loops. Filter detected stack variables by scope.

```
advisor --collect=dependencies --loops="scalar,loop-height=0" --filter-by-scope --project-dir=./  
advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

filter-reductions

Mark all potential reductions by specific diagnostic during Dependencies analysis.

GUI Equivalent

Project Properties > Analysis Target > Dependencies Analysis > Advanced > Filter reduction variables

Syntax

--filter-reductions
--no-filter-reductions

Default

Off (no-filter-reductions)

Actions Modified

collect=dependencies
collect=offload

Usage

Enabling can increase collection overhead.

Example

Run a Dependencies analysis. Analyze innermost scalar loops. Mark all potential reductions by specific diagnostic.

```
advisor --collect=dependencies --loops=scalar,loop-height=0" --filter-reductions --project-dir=./
advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

flop

Collect data about floating-point and integer operations, memory traffic, and mask utilization metrics for AVX-512 platforms during Trip Counts & FLOP analysis.

GUI Equivalent

Analysis Workflow > Characterization > Collect FLOP data

Project Properties > Trip Counts and FLOP Analysis > Advanced > Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage

Syntax

```
--flop
--no-flop
```

Default

Off (no-flop)

On (flop) for `--collect=offload`

Actions Modified

`collect=tripcounts`

`collect=offload`

Usage

Enabling can increase analysis overhead.

Example

Run the Trip Counts & FLOP analysis. Collect both Trip Counts and FLOP data. (Default for the `trip-counts` option is on, so it is not explicitly stated in the command line.)

```
advisor -collect=tripcounts --flop --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

force-32bit-arithmetics

Consider all arithmetic operations as single-precision floating-point or int32 operations.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--force-32bit-arithmetics  
--no-force-32bit-arithmetics
```

Default

Off (no-force-32bit-arithmetics)

Actions Modified

```
collect=projection  
collect=offload
```

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Force 32-bit arithmetics when modeling your application performance on a target device.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --force-32bit-arithmetics --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

force-64bit-arithmetics

Consider all arithmetic operations as double-precision floating-point or int64 operations.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--force-64bit-arithmetics  
--no-force-64bit-arithmetics
```

Default

Off (no-force-64bit-arithmetics)

Actions Modified

`collect=projection`

`collect=offload`

Example

Force 64-bit arithmetics when modeling your application performance on a target device.

```
advisor --collect=projection --force-64bit-arithmetics --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

format

Set a report output format.

Syntax

`--format=<string>`

Arguments

`<string>` is one of the following:

Argument	Description
csv	Tabular (.csv) file format
text	.txt file format
xml	.xml file format

Default

text

Actions Modified

`report`

Usage

By default, the `advisor` writes a report to standard output in text format; however, it provides a number of options for generating a report:

- Use the `report-output` option to write a report to a file.
- Use the `csv-delimiter` option to set a delimiter other than a comma.

Example

Generate a Dependencies report. Output in XML format. Save it as `advisor-Dependencies.xml`.

```
advisor --report=dependencies --format=xml --report-output=./out/advisor-Dependencies.xml --project-dir=./advi_results
```

Generate a Dependencies report. Output in CSV format with tab delimiters. Save it as `advisor-Dependencies.csv`.

```
advisor --report=dependencies --format=csv --csv-delimiter=tab --report-output=./out/advisor-Dependencies.csv --project-dir=./advi_results
```

See Also

[csv-delimiter](#) Set the delimiter for a report in CSV format.

[report-output](#) Redirect report output from stdout to another location.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

gpu

With Offload Modeling perspective, analyze OpenCL™ and oneAPI Level Zero programs running on Intel® Graphics. With GPU Roofline perspective, create a Roofline interactive HTML report for data collected on GPUs.

GUI Equivalent

For Offload Modeling perspective: **Analysis Workflow > Baseline Device > GPU**

For Offload Modeling perspective: **Project Properties > Analysis Target > Performance Modeling > GPU**

Syntax

`--gpu`

`--no-gpu`

Default

Off (no-gpu)

Actions Modified

`collect=offload`

`report=roofline`

Usage

For Offload Modeling perspective:

Use this option if you want to run GPU-to-GPU modeling using the collection presets. This option automatically applies the `--profile-gpu` option to all analyses it runs.

For GPU Roofline perspective:

Prerequisites: Collect Roofline data with `--profile-gpu` option enabled.

The default Roofline interactive HTML report is created for data collected on CPUs. To create a report for data collected on GPUs, use this option.

Example

Run the GPU-to-GPU Offload Modeling with collection preset.

```
advisor --collect=offload --gpu --project-dir=./advi_results
```

Generate a Roofline interactive HTML report for data collected on GPUs.

```
advisor --report=roofline --report-output=./out/roofline.html --gpu --project-dir=./advi_results
```

See Also

[advisor Command Action Reference](#)

[Run GPU-to-GPU Performance Modeling from Command Line](#) With Intel® Advisor, you can model performance of SYCL, OpenCL™, or OpenMP* target application running on a graphics processing unit (GPU) for a different GPU device without its CPU version. For this, run the GPU-to-GPU modeling workflow of the Offload Modeling perspective.

gpu-carm

Collect memory traffic generated by OpenCL™ and Intel® Media SDK programs executed on Intel® Processor Graphics.

Syntax

```
--gpu-carm--no-gpu-carm
```

Default

On (gpu-carm)

Actions Modified

```
collect=tripcounts --profile-gpu
```

```
collect=roofline --profile-gpu
```

Usage

Important

GPU profiling is applicable only to Intel® Processor Graphics.

This option may affect the performance of your application on the CPU side.

Example

1. Run a Survey analysis with GPU profiling enabled.

```
advisor --collect=survey --profile-gpu --project-dir=./advi_results -- ./myApplication
```

2. Run a Trip Count and FLOP analysis, enable GPU profiling and explicitly enable CARM traffic metrics collection.

```
advisor --collect=tripcounts --flop --profile-gpu --gpu-carm --project-dir=./advi_results -- ./myApplication
```

See Also

[Command Line Interface Reference](#)

[advisor Command Action Reference](#)

gpu-sampling-interval

Specify time interval, in milliseconds, between GPU samples during Survey analysis.

GUI Equivalent

Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > GPU sampling interval

Syntax

```
--gpu-sampling-interval=<double>
```

Arguments

<double> is the number from 0.1 to 10 milliseconds between GPU samples.

Default

1

Actions Modified

`collect=survey --profile-gpu`

`collect=roofline --profile-gpu`

Usage

Important

GPU profiling is applicable only to Intel® Processor Graphics.

This option may affect the performance of your application on the CPU side. Increasing the wait time between samples can decrease collection overhead.

Example

Run a Survey analysis. Enable GPU profiling and increase the GPU sampling interval to 3 ms.

```
advisor --collect=survey --profile-gpu --gpu-sampling-interval=3 --project-dir=./
advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

hide-data-transfer-tax

Disable data transfer tax estimation.

Syntax

```
--hide-data-transfer-tax
```

```
--no-hide-data-transfer-tax
```

Default

no-hide-data-transfer-tax (Off)

Actions Modified

`collect=projection`

`collect=offload`

Usage

When you use this option, data transfer tax is ignored when potential speedup is estimated.

Example

Run the Performance Modeling analysis and disable data transfer tax estimation.

```
advisor --collect=projeciton --hide-data-transfer-tax --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command **syntax**: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

ignore

Specify runtimes or libraries to ignore time spent in these regions when calculating per-program speedup.

Syntax

`--ignore=<string>`

Arguments

`<string>` is a comma-separated list of runtimes and libraries to ignore:

Argument	Description
OMP	Ignore time spent in OpenMP* code regions.
MPI	Ignore time spent in MPI code regions.
TBB	Ignore time spent in Intel® oneAPI Threading Building Blocks code regions.
MKL	Ignore time spent in Intel® Math Kernel Library code regions.
DAAL	Ignore time spent in Intel® Data Analytics Acceleration Library code regions.
OMPTARGET	Ignore time spent in code regions with OpenMP <code>target</code> construct.
OCL	Ignore time spent in OpenCL™ code regions.
L0	Ignore time spent in Intel® oneAPI Level Zero code regions.
SVML	Ignore time spent in Short Vector Math Library (SVML) code regions.

NOTE This option is *not* case sensitive.

Actions Modified

`collect=projection`

`collect=offload`

Usage

This option does not affect individual offloads, only the per-program metrics.

Example

Model your application performance on a target device and ignore time spent in OpenMP and Intel oneAPI Threading Building Blocks regions.

```
advisor --collect=projection --ignore=OMP,TBB --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

ignore-app-mismatch

Ignore mismatched target or application parameter errors before starting analysis.

GUI Equivalent

Survey > Your Project Properties changed since the last analysis run error

(Click **Continue** or **Cancel**)

Syntax

`--ignore-app-mismatch`

`--no-ignore-app-mismatch`

Default

Off (no-ignore-app-mismatch)

Actions Modified

`collect`

Usage

If you continue collection, trip counts data might become unreliable. To synchronize trip counts and survey data:

1. Run a Survey analysis for the updated parameters using `delete-tripcounts`.
2. Run a Trip Counts analysis.

Example

Run a Survey analysis. Ignore mismatched target or application parameter errors before starting analysis.

```
advisor --collect=survey --ignore-app-mismatch --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

ignore-checksums

Ignore mismatched module checksums before starting analysis.

GUI Equivalent

Survey > Target binaries changed since the last analysis run error

(Click **Continue** or **Cancel**)

Syntax

```
--ignore-checksums
--no-ignore-checksums
```

Default

Off (no-ignore-checksums)

Actions Modified

collect

Usage

If you continue collection, trip counts data might become unreliable. To synchronize trip counts and survey data:

1. Run a Survey analysis for the updated binaries using `delete-tripcounts`.
2. Run a Trip Counts analysis.

Example

Run a Survey analysis. Ignore mismatched module checksums before starting analysis.

```
advisor --collect=survey --ignore-checksums --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

instance-of-interest

Analyze the Nth child process during Memory Access Patterns and Dependencies analysis.

GUI Equivalent

Project Properties > [Analysis Type] > Advanced > Instance of interest

Syntax

```
--instance-of-interest=<integer>
```

Arguments

Argument	Description
0	Analyze all processes.
1	Analyze first process of the specified name in the application process tree.
<integer>	Analyze subsequent process of the specified name in the application process tree.

Default

0

Actions Modified

`collect=map`

`collect=dependencies`

Example

Run a Memory Access Patterns analysis. Analyze the first process of the specified name in the application process tree.

```
advisor --collect=map --instance-of-interest=1 --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

integrated

Model traffic on all levels of the memory hierarchy for a Roofline report.

GUI Equivalent

Analysis Workflow > CPU Roofline > Characterization > Enable CPU cache simulation

Syntax

`--integrated`

`--no-integrated`

Default

Off (no-integrated)

Actions Modified

`collect=roofline`

Example

Run a Roofline analysis. Model traffic on all levels of the memory hierarchy.

```
advisor --collect=roofline --integrated --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

interval

Set the length of time (in milliseconds) to wait before collecting each sample during Survey analysis.

GUI Equivalent

Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Sampling interval

Syntax

```
--interval=<integer>
```

Arguments

<integer> is the number of milliseconds between sampling (sampling interval).

Default

10

Actions Modified

`collect=survey`

Usage

Increasing the wait time between each analysis collection sample can decrease collection overhead.

Example

Run a Survey analysis. Increase the sampling interval to 20 milliseconds.

```
advisor --collect=survey --interval=20 --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

limit

Set the maximum number of top items to show in a report.

GUI Equivalent

Survey > Customize View > Top

Syntax

```
--limit=<integer>
```

Arguments

<integer> is the maximum number of top items.

Default

Off (unlimited lines in output report)

Actions Modified

`report`

Example

Generate a Survey report. Show the top five self-time hotspots that were not vectorized because of a *not inner loop* msg id.

```
advisor --report=survey --limit=5 --filter="Vectorization Message(s)"="loop was not vectorized:
not inner loop" --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

loop-call-count-limit

Set the maximum number of instances to analyze for all marked loops.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Type] > Advanced > Loop Call Count Limit

Syntax

`--loop-call-count-limit=<integer>`

Arguments

Argument	Description
0	Analyze all instances of all marked loops.
<integer>	Analyze up to <i>n</i> number of instances for all marked loops.

Default

0 (analyze all instances of all marked loops)

Actions Modified

`collect=dependencies`

`collect=map`

`collect=offload`

Usage

Assumes similar runtime properties, such as the same memory access patterns, over different call instances. If this is not true, using this option may produce skewed results.

A smaller, non-zero value can minimize collection overhead.

Example

Run a Memory Access Patterns analysis. Limit analysis to the first ten invocations of marked loops.

```
advisor --collect=map --loop-call-count-limit=10 --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimizing Analysis Overhead](#)

advisor Command Option Reference

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

loop-filter-threshold

Specify total time, in milliseconds, to filter out loops that fall below this value.

Syntax

`--loop-filter-threshold=<integer>`

Arguments

`<integer>` is a total execution time threshold, in milliseconds.

Default

20

Actions Modified

`collect=projection`

`collect=offload`

Usage

Use to ignore loop nests with total time less than the threshold.

Example

Ignore loops with total time less than 50 milliseconds when modeling your application performance on a target device.

```
advisor --collect=projection --loop-filter-threshold=50 --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

loops

Select loops (by criteria instead of human input) for deeper analysis.

Syntax

`--loops=<string>`

Arguments

`<string>` is a double-quote-enclosed, comma-separated list of criteria (no spaces):

Arguments for Dependencies Analysis	Description
scalar	Include only scalar serial loops.

Arguments for Dependencies Analysis	Description
total-time> <i>n</i>	Include only loops above <i>n</i> % of total CPU time.
has-source	Exclude only loops without source location.
has-issue	Include only loops with <i>Vector Dependence Prevents Vectorization</i> issue.
loop-height= <i>n</i>	<p>Include only loops by specific hierarchical position. 0 = Innermost loops.</p> <p>For example: Use the following criteria to include specific loops in the following scenario:</p> <ul style="list-style-type: none"> • loop-height=0 selects only innermost loop 3 • loop-height=1 selects only loop 2 and loop 3 • loop-height=2 selects all three loops <pre>main -> loop 1 -> loop 2 -> loop 3</pre>
top= <i>n</i>	Include only loops with the largest self-time.

Arguments for Memory Access Patterns Analysis	Description
total-time> <i>n</i>	Include only loops above <i>n</i> % of total CPU time.
has-source	Exclude only loops without source location.
has-issue	Include only loops with <i>Possible Inefficient Memory Access Pattern</i> issue.
loop-height= <i>n</i>	<p>Include only loops by specific hierarchical position. 0 = Innermost loops.</p> <p>For example: Use the following criteria to include specific loops in the following scenario:</p> <ul style="list-style-type: none"> • loop-height=0 selects only innermost loop 3 • loop-height=1 selects only loop 2 and loop 3 • loop-height=2 selects all three loops <pre>main -> loop 1 -> loop 2 -> loop 3</pre>
top= <i>n</i>	Include only loops with the largest self-time.

Default

For Memory Access Patterns analysis: "loop-height=0,total-time>0.1"

For Dependencies analysis: "scalar,loop-height=0,total-time>0.1"

Actions Modified

`collect=map`

`collect=dependencies`

`mark-up-loops`

Usage

All criteria must be met to select loops for analysis.

This option is particularly useful when you are using automated scripts.

You can accomplish the same objective using the `mark-up-loops` action followed by a `collect` action for a Dependencies or Memory Access Patterns analysis.

Usage can decrease collection overhead.

Example

Run a Dependencies analysis. Analyze only innermost loops with sources.

```
advisor --collect=dependencies --loops="loop-height=0,has-source" --search-dir src:=./src --
project-dir=./advi_results -- ./myApplication
```

See Also

[mark-up-loops](#) After running a Survey analysis and identifying loops of interest, select loops (by file and line number or criteria) for deeper analysis.

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

mark-up

Enable/disable user selection as a way to control loops/functions identified for deeper analysis.

Syntax

`--mark-up`

`--no-mark-up`

Default

On (mark-up)

Actions Modified

`collect=dependencies`

`collect=map`

Usage

Intel Advisor offers two ways to identify loops/functions for deeper analysis:

- **Source annotations** - Via code that pinpoints the start and end of loops and iterations
This is the only identification method for the Suitability analysis.

- **User selection** - In the CLI via `mark-up-loops` and `mark-up-list`, and in the GUI via **Survey** > 🔥
This is the primary identification method for Vectorization Advisor analyses.

User selection via `mark-up-loops` and **Survey** > 🔥 persists for downstream analyses. User selection via `mark-up-list` persists only for the duration of a `collect` action.

Use `mark-up` to use both ways to identify loops/functions for deeper analysis. Use `no-mark-up` to use only source annotations.

NOTE

There is no *clear selection* option in the Intel Advisor CLI. `no-mark-up` is the closest equivalent if user-selected loops/functions persist but you want to use only source annotations to identify loops/functions for deeper analysis.

Example

Run a Memory Access Patterns analysis. Analyze only loops/functions identified by source annotations.

```
advisor --collect=map --no-mark-up --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

mark-up-list

After running a Survey analysis and identifying loops of interest, select loops (by file and line number or ID) for deeper analysis.

GUI Equivalent

Survey > 🔥

Syntax

```
--mark-up-list=<string>
```

Arguments

`<string>` is a comma-separated list (no spaces) of loop IDs, file/line numbers in the format `file1:line1`, or both.

Default

The existing selection in **Survey** > 🔥, which is persistent.

If there is no GUI selection:

- For Trip Counts & FLOP and Roofline analyses: all loops
- `loops` default
 - For Memory Access Patterns analysis: `"loop-height=0,total-time>0.1"`
 - For Dependencies analysis: `"scalar,loop-height=0,total-time>0.1"`

Actions Modified



`collect=tripcounts`

`collect=map`

`collect=dependencies`

`collect=roofline`

Usage

Do not confuse the `mark-up-loops` action with the `mark-up-list` action option. The `mark-up-loops` action coupled with the `select` action option enables a GUI  checkbox; therefore loop selection persists beyond the duration of the `mark-up-loops` action and applies to downstream analyses, such as Dependencies and Memory Access Patterns analyses. The `collect` action coupled with the `mark-up-list` action option simulates enabling a GUI  checkbox; therefore loop selection persists only for the duration of the `collect` action.

Example

1. Run a Survey analysis.
2. Run a Trip Counts & FLOP analysis on Survey analysis loops 1 and 3; and source location `my_source.cpp:132`.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
advisor --collect=tripcounts --mark-up-list=1,my_source.cpp:132,3 --search-dir src:=./src --
project-dir=./advi_results -- ./myApplication
```

See Also

[loops](#) Select loops (by criteria instead of human input) for deeper analysis.

[mark-up-loops](#) After running a Survey analysis and identifying loops of interest, select loops (by file and line number or criteria) for deeper analysis.

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#)

memory-level

Model specific memory level(s) in a Roofline interactive HTML report, including L1, L2, L3, and DRAM.

GUI Equivalent

Roofline > Default: FLOAT CARM (L1+NTS) > Memory Level

Syntax

`--memory-level=<string>`

Arguments

`<string>` is an underscore-separated list of memory levels (no spaces).

Default

L1 (subject to change)

Actions Modified

`report=roofline`

Usage

Enable pinpointing memory bandwidth bottlenecks by specific memory layer.

Example

Generate a Roofline interactive HTML report. Show data for L2 and L3 memory.

```
advisor --report=roofline --memory-level=L2_L3 --project-dir=./advi_results
```

See Also

[integrated](#) Model traffic on all levels of the memory hierarchy for a Roofline report.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

memory-operation-type

Model only load memory operations, store memory operations, or both, in a Roofline interactive HTML report.

GUI Equivalent

Roofline > Default: FLOAT > Memory Operation Type

Syntax

`--memory-operation-type=<string>`

Arguments

`<string>` is one of the following: load | store | all

Default

all

Actions Modified

`report=roofline`

Example

Generate a Roofline interactive HTML report. Show only load memory operations.

```
advisor --report=roofline --memory-operation-type=load --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

mix

Show dynamic or static instruction mix data in a Survey report.

GUI Equivalent

Code Analytics

Syntax

```
--mix
--no-mix
```

Default

Off (no-mix)

Actions Modified

`report=survey`

Usage

Dynamic instruction mix is counted for the entire execution of the application; static instruction mix is counted per iteration. The `static-instruction-mix`, `dynamic`, and `mix` options work together in the following manner:

- Collect static instruction mix data: `--collect=survey --static-instruction-mix`
(In the GUI: Static instruction mix data is calculated on demand.)
- Collect dynamic instruction mix data (and static instruction mix data, from which dynamic mix data is calculated): `--collect=tripcounts --flop`
- Show static instruction mix data in a Survey report: `--report=survey --mix --no-dynamic`
- Show dynamic mix instruction data in a Survey report: `--report=survey --mix --dynamic`
- A Survey report cannot show both static and dynamic mix instruction data.

(In the GUI: **Code Analytics** can show both static and dynamic instruction mix data.)

Example

1. Run a Survey analysis.
2. Run a Trip Counts & FLOP analysis. Collect dynamic instruction mix data (and static instruction mix data, from which dynamic mix data is calculated).
3. Generate a Survey report. Show dynamic instruction mix data. (`dynamic` is on, by default).

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication
advisor --collect=tripcounts --flop --project-dir=./advi_results -- ./myApplication
advisor --report=survey --mix --project-dir=./advi_results
```

1. Run a Survey analysis. Collect static instruction mix data.
2. Generate a Survey report. Show static instruction mix data.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
advisor --report=survey --mix --no-dynamic --project-dir=./advi_results
```

See Also

[dynamic](#) Show (in a Survey report) how many instructions of a given type actually executed during Trip Counts & FLOP analysis.

[static-instruction-mix](#) Statically calculate the number of specific instructions present in the binary during Survey analysis.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

mkl-user-mode

Collect Intel® oneAPI Math Kernel Library (oneMKL) loops and functions data during the Survey analysis

GUI Equivalent

Project Properties > Analysis Target > Survey Analysis > Advanced > Analyze MKL loops and functions

Syntax

```
--mkl-user-mode  
--no-mkl-user-mode
```

Default

On (mkl-user-mode)

Actions Modified

`collect=survey`

Usage

Disabling can decrease finalization overhead.

Example

Run a Survey analysis. Disable collecting oneMKL loops and functions data.

```
advisor --collect=survey --no-mkl-user-mode --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

model-baseline-gpu

Use the baseline GPU configuration as a target device for modeling.

Syntax

```
--model-baseline-gpu  
--no-model-baseline-gpu
```

Default

Off (no-model-baseline-gpu)

Actions Modified

`collect=projection --profile-gpu`

Usage

This option is applicable only to the GPU-to-GPU performance modeling workflow.

Use this option when you run the Performance Modeling (`collect=projection`) as part of the GPU Roofline Insights perspective for an application executed on a GPU. With this analysis executed, your application performance is modeled for a baseline GPU device as a target. The estimated performance is compared with the actual application performance to add more recommendations for performance optimization.

This option automatically enables the `enforce-baseline-decomposition` option. You can use only `model-baseline-gpu` to simplify a command.

Example

Run the GPU-to-GPU Performance Modeling for the baseline GPU.

```
advisor --collect=projection --profile-gpu --model-baseline-gpu --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

model-children

Analyze child loops of the region head to find if some of the child loops provide more profitable offload.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--model-children
--no-model-children
```

Default

On (model-children)

Actions Modified

```
collect=projection
collect=offload
```

Usage

Use the `model-children` option to:

- Minimize overhead
- Estimate offload speedup of a not-offloaded region head that has a *Less or equally profitable than children offloads* message

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.

3. Model only offloading of loop heads when modeling your application performance on a target device.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results
-- ./myApplication
advisor --collect=projection --no-model-children --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

model-extended-math

Model calls to math functions such as EXP, LOG, SIN, and COS as extended math instructions, if possible.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--model-extended-math
--no-model-extended-math
```

Default

On (model-extended-math)

Actions Modified

```
collect=projection
collect=offload
```

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Do not model calls to math functions as extended math instructions when modeling your application performance on a target device.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results
-- ./myApplication
advisor --collect=projection --no-model-extended-math --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

model-system-calls

Analyze code regions with system calls considering they are separated from offload code and executed on a host device.

Syntax

```
--model-system-calls  
--no-model-system-calls
```

Default

On (model-system-calls)

Actions Modified

```
collect=projection  
collect=offload
```

Usage

NOTE

The presence of system calls inside a region may reduce model accuracy.

Example

Model your application performance on a target device and disable analyzing system calls.

```
advisor --collect=projection --model-system-calls --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

module-filter

Specify application (or child application) module(s) to include in or exclude from analysis.

GUI Equivalent

Modules

Syntax

```
--module-filter=<string>
```

Arguments

<string> is a comma-separated list of module names (no spaces).

Default

None - so when coupled with `module-filter-mode default (exclude)`, the Intel Advisor analyzes all modules.

Actions Modified

`collect`=[analysis type] `--module-filter-mode`

Usage

Usage can decrease collection and finalization overhead.

Example

Run a Survey analysis. Exclude modules `foo1.so` and `foo2.so`.

```
advisor --collect=survey --module-filter-mode=exclude --module-filter=foo1.so,foo2.so --project-dir=./advi_results -- ./myApplication
```

See Also

[module-filter-mode](#) Limit, by inclusion or exclusion, application (or child application) module(s) for analysis.

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

module-filter-mode

Limit, by inclusion or exclusion, application (or child application) module(s) for analysis.

GUI Equivalent

Modules

Syntax

`--module-filter-mode=<string>`

Arguments

Argument	Description
include	Include the modules specified in <code>module-filter</code> .
exclude	Exclude the modules specified in <code>module-filter</code> .

Default

Exclude - so when coupled with `module-filter` default (empty), the Intel Advisor analyzes all modules.

Actions Modified

`collect`=[analysis type] `--module-filter`

Usage

Usage can decrease collection and finalization overhead.

Example

Run a Survey analysis. Exclude modules `foo1.so` and `foo2.so`.

```
advisor --collect=survey --module-filter-mode=exclude --module-filter=foo1.so,foo2.so --project-dir=./advi_results -- ./myApplication
```

See Also

[module-filter](#) Specify application (or child application) module(s) to include in or exclude from analysis.

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

mpi-rank

Specify MPI process data to import.

Syntax

`--mpi-rank=<integer>`

Arguments

`<integer>` is the rank of the process with data to import.

Default

If an MPI rank is not specified, and there is more than one result directory in the project because the result partition is shared, a rank is chosen at random. Recommendation: Specify a rank.

Actions Modified

`import-dir`, `mark-up-loops`, `report`

Usage

When you collect analysis data on a cluster, the data is stored in unique subdirectories under the project directory, named `rank.#`. Use this option to specify the process with data to import for viewing. You can import data from only one process at a time.

Example

Import MPI analysis data from the `rank.3` cluster. Read source files from the specified search directory. Write the result to the `advi_results` project directory.

```
advisor --import-dir=./advi --mpi-rank=3 --project-dir=./advi_results -- ./myApplication
```

See Also

[Analyze MPI Workloads](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

mrte-mode

Set the Microsoft* runtime environment mode for analysis.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Types] > Managed code profiling mode

Syntax

```
--mrte-mode=<string>
```

Arguments

<string> is the mode type:

Argument	Description
auto	Automatically detect the type of target executable and switch to that mode.
native	Collect data for native code and do not attribute data to managed code.
mixed	Collect data for both native and managed code, and attribute data to managed code as appropriate. Consider using this option when analyzing a native executable that makes calls to the managed code.
managed	Collect data for both native and managed code, resolve samples attributed to native code, attribute data to managed source only. The call stack in the analysis result displays data for managed code only.

Default

auto

Actions Modified

collect

Usage

Applies to Windows* OS only.

Example

Run a Suitability analysis. Set a native runtime environment mode.

```
advisor --collect=suitability --mrte-mode=native --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

ndim-depth-limit

When searching for an optimal N-dimensional offload, limit the maximum loop depth that can be converted to one offload.

Syntax

```
--ndim-depth-limit=<integer>
```

Arguments

`<integer>` is a number in the range 1 \leq `<integer>` \leq 6.

Default

3

Actions Modified

`collect=projection` `--search-n-dim`

`collect=offload`

NOTE You can skip the `--search-n-dim` option because it is enabled by default.

Example

Model your application performance on a target device and limit the maximum depth of an offload to 5.

```
advisor --collect=projection --ndim-depth-limit=5 --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

option-file

Specify a text file containing command line arguments.

Syntax

`--option-file=<PATH>`

Arguments

`<PATH>` is the PATH/name of a text file containing command line arguments.

Actions Modified

`collect`, `report`

Usage

Put commonly used options in a UTF-8 text file to shorten the command line and create a reusable invocation syntax. Enter one option on each line. No spaces are allowed in the option entry; use a new line instead.

Arguments specified in an option file are processed before any arguments specified on the command line; therefore, options specified on the command line can override options in an option file.

Example

Create a reusable option file you can use whenever you want to generate a report that specifies the project directory, directory to search for source files, and PATH/name of the output text file.

```
--project-dir=./advi_results
--search-dir
```

```
src=./src
--format=text
--report-output=./out/annotations.txt
```

Generate a Suitability report. Use an option file named `my_suitability_analysis.txt`.

```
advisor --report=suitability --option-file=./my_suitability_analysis.txt
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

overlap-taxes

Enable asynchronous execution to overlap offload overhead with execution time.

Syntax

```
--overlap-taxes
--no-overlap-taxes
```

Default

Off (no-overlap-taxes)

Actions Modified

```
collect=projection
collect=offload
```

Example

Model your application performance on a target device and enable asynchronous execution.

```
advisor --collect=projection --overlap-taxes --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

pack

Pack a snapshot into an archive.

GUI Equivalent

File > Create Data Snapshot > Pack into archive

Syntax

```
--pack
--no-pack
```

Default

On (pack)

Actions Modified

snapshot

Usage

By default, the archive is saved to the current directory with the default `snapshotXXX.adviceexpz` name.

Example

Create a new snapshot in the project directory. Do not pack it into an archive.

```
advisor --snapshot --no-pack --project-dir=./advi_results
```

Create a new snapshot. Pack it into an archive. Save it in the current directory with the default name.

```
advisor --snapshot --pack --project-dir=./advi_results
```

Create a new snapshot. Pack it into an archive named `new_snapshot.adviceexpz`.

```
advisor --snapshot --pack --project-dir=./advi_results -- ./new_snapshot
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

profile-gpu

Analyze OpenCL™ and oneAPI Level Zero programs running on Intel® Graphics.

GUI Equivalent

Analysis Workflow > Baseline Device > GPU

Project Properties > Analysis Target > Performance Modeling > GPU

Syntax

`--profile-gpu`

`--no-profile-gpu`

Default

Off (`no-profile-gpu`)

Actions Modified

`collect=survey`

`collect=tripcounts`

`collect=roofline`

`collect=projection`

Usage

Prerequisite: [Set up system environment](#) to enable GPU kernel profiling.

Use this option to analyze a GPU-enabled application that uses SYCL, OpenMP* target, or OpenCL™ programming model.

- For the GPU Roofline Insights, use this option to analyze code regions running on a CPU and code regions running on a GPU. This option may affect the performance of your application on the CPU side.
- For the Offload Modeling, use this option to analyze *only* code regions running on a GPU. This is a *preview* feature.

NOTE Make sure to use this option with the Survey, Trip Counts, and Performance Modeling analyses.

NOTE GPU profiling is applicable only to Intel® Graphics.

Example

Run the Roofline analysis and enable GPU profiling to analyze OpenCL™ and oneAPI Level Zero programs running on Intel® Graphics.

```
advisor --collect=roofline --profile-gpu --project-dir=./advi_results -- ./myApplication
```

1. Run the Survey analysis with the GPU kernel profiling enabled.
2. Run the Trip Counts and FLOP analysis with the GPU kernel profiling enabled.
3. Run the Performance Modeling with the GPU kernel profiling enabled.

```
advisor --collect=survey --static-instruction-mix --profile-gpu --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --profile-gpu --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --profile-gpu --project-dir=./advi_results -- ./myApplication
```

See Also

[Run GPU-to-GPU Performance Modeling from Command Line](#) With Intel® Advisor, you can model performance of SYCL, OpenCL™, or OpenMP* target application running on a graphics processing unit (GPU) for a different GPU device without its CPU version. For this, run the GPU-to-GPU modeling workflow of the Offload Modeling perspective.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

profile-intel-perf-libs

Show Intel® performance libraries loops and functions in Intel® Advisor reports.

Syntax

```
--profile-intel-perf-libs
```

```
--no-profile-intel-perf-libs
```

Default

On (profile-intel-perf-libs)

Actions Modified

collect

Example

Run a Survey analysis and disable showing Intel® performance libraries in the report.

```
advisor --collect=survey --no-profile-intel-perf-libs --project-dir=./advi_results  
-- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

profile-jit

Collect metrics about Just-In-Time (JIT) generated code regions during the Trip Counts and FLOP analysis.

GUI Equivalent

Project Properties > Analysis Target > Trip Counts and FLOP Analysis > Advanced > Capture metrics for dynamic loops and functions

Syntax

```
--profile-jit  
--no-profile-jit
```

Default

On (profile-jit)

Actions Modified

`collect=tripcounts`

Usage

Enabling can increase collection overhead.

Example

1. Run a Survey analysis.
2. Run a Trip Counts and FLOP analysis. Explicitly enable collecting metrics for JIT generated code.

```
advisor --collect=survey --project-dir=./advi_results -- ./myApplication  
advisor --collect=tripcounts --flop --profile-jit --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

profile-python

Collect Python loop and function data during Survey analysis.

GUI Equivalent

Project Properties > Analysis Target > Survey Analysis > Advanced > Analyze Python loops and functions

Syntax

```
--profile-python  
--no-profile-python
```

Default

Off (no-profile-python)

Actions Modified

`collect=survey`

Usage

Enabling can increase collection overhead.

Example

Run a Survey analysis. Explicitly disable collecting Python loop and function data.

```
advisor --collect=survey --no-profile-python --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimizing Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

profile-stripped-binaries

Collect metrics for stripped binaries.

GUI Equivalent

Project Properties > Analysis Target > Survey Analysis Types > Trip Counts and FLOP analysis > Advances > Capture metrics for stripped binaries

Syntax

```
--profile-stripped-binaries  
--no-profile-stripped-binaries
```

Default

Off (no-profile-stripped-binaries)

Actions Modified

`collect=tripcounts`

Usage

Tip Enabling can increase overhead.

Example

Collect Trip Counts for stripped binaries.

```
advisor --collect=tripcounts --profile-stripped-binaries --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

project-dir

Specify the top-level directory where a result is saved if you want to save the collection somewhere other than the current working directory.

Syntax

`--project-dir=<PATH>`

Arguments

`<PATH>` is the PATH/name of a directory.

Default

Current working directory

Actions Modified

`collect`, `create-project`, `import-dir`, `mark-up-loops`, `report`, `snapshot`

Usage

Recommendation: Specify the project directory when you:

- Generate a report from a collection.
- Import MPI collections.

Examples

Run a Survey analysis on `myApplication`. Search the `src` directory for all source, binary, and symbol files. Write the result to `advi_results`.

```
advisor --collect=survey --search-dir all=./src --project-dir=./advi_results -- ./myApplication
```

Generate a Survey report from the result in the `advi` directory. Output the report in text format as `survey.txt`.

```
advisor --report=survey --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

See Also

[Command Line Interface Reference](#)

advisor Command Action Reference

quiet

Minimize status messages during command execution.

Syntax

-q
--quiet

Default

Off

Actions Modified

collect, command, import-dir, mark-up-loops, report, snapshot

Example

Generate a Suitability report. Output to `stderr`. Show warnings, errors, and fatal errors.

```
advisor --report=suitability --quiet --project-dir=./advi_results
```

See Also

[verbose](#) Maximize status messages during command execution.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

recalculate-time

Recalculate total time after filtering a report.

GUI Equivalent

Filters

Syntax

--recalculate-time
--no-recalculate-time

Default

On (recalculate-time)

Actions Modified

report=survey --filter
report=tripcounts --filter

Example

Generate a Survey report. Show data only for scalar loops. Do not recalculate total time.

```
advisor --report=survey --filter="Type"="Scalar" --no-recalculate-time --project-dir=./advi_results
```

Generate a Survey report. Show data only for loops/functions from `my_module1`. Explicitly recalculate total time.

```
advisor --report=survey --filter="Module"="my_module1" --recalculate-time --project-dir=./
advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

record-mem-allocations

Enable heap allocation tracking to identify heap-allocated variables for which access strides are detected during Memory Access Patterns analysis.

GUI Equivalent

Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Report heap allocated variables

Syntax

```
--record-mem-allocations
--no-record-mem-allocations
```

Default

On (record-mem-allocations)

Actions Modified

`collect=map`

Usage

Disabling can decrease collection overhead.

Example

Run a Memory Access Patterns analysis. Disable heap allocation tracking.

```
advisor --collect=map --no-record-mem-allocations --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

record-stack-frame

Capture stack frame pointers to identify stack variables for which access strides are detected during Memory Access Patterns analysis.

GUI Equivalent

Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Report stack variables

Syntax

```
--record-stack-frame  
--no-record-stack-frame
```

Default

On (record-stack-frame)

Actions Modified

`collect=map`

Usage

Disabling can decrease collection overhead.

Example

Run a Memory Access Patterns analysis. Disable capturing stack frame pointers.

```
advisor --collect=map --no-record-stack-frame --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

reduce-lock-contention

Examine specified annotated sites for opportunities to reduce lock contention or find deadlocks in a Suitability report.

GUI Equivalent

Suitability > Lock Contention

Syntax

```
--reduce-lock-contention=<string>
```

Arguments

`<string>` is a comma-separated list of annotated sites (no spaces).

Default

No default argument

Actions Modified

`report=suitability`

Usage

Lock contention is the time one thread spends waiting for a lock to be released while another thread holds that lock (as opposed to lock overhead, which is the time spent creating, destroying, acquiring, and releasing locks). You can reduce lock contention by using different locks for unrelated data when you convert to a parallel framework.

Usage of this option simulates parallel execution with the assumption that lock contention is zero for a specified site.

Example

Generate a Suitability report. Examine the annotated sites `myAnnotatedSiteJ` and `myAnnotatedSiteX` for opportunities to reduce lock contention. Write the report to `stdout`.

```
advisor --report=suitability --reduce-lock-contention=myAnnotatedSiteJ,myAnnotatedSiteX --  
project-dir=./advi_results
```

See Also

[Reducing Lock Contention](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

reduce-lock-overhead

Examine specified annotated sites for opportunities to reduce lock overhead in a Suitability report.

GUI Equivalent

Suitability > Lock Overhead

Syntax

`--reduce-lock-overhead=<string>`

Arguments

`<string>` is a comma-separated list of annotated sites (no spaces).

Default

No default argument

Actions Modified

`report=suitability`

Usage

Lock overhead is the time spent creating, destroying, acquiring, and releasing locks (as opposed to lock contention, which is the time spent waiting for a lock held by another task). Think of lock overhead as the cost of lock operations, assuming the lock is always available.

Usage of this option simulates parallel execution with the assumption that lock overhead is zero for a specified site.

Example

Generate a Suitability report. Examine the annotated sites `myAnnotatedSiteJ` and `myAnnotatedSiteX` for opportunities to reduce lock overhead. Write the report to `stdout`.

```
advisor --report=suitability --reduce-lock-overhead=myAnnotatedSiteJ,myAnnotatedSiteX --project-dir=./advi_results
```

See Also

[Reduce Lock Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

reduce-site-overhead

Examine specified annotated sites for opportunities to reduce site overhead in a Suitability report.

GUI Equivalent

Suitability > Site Overhead

Syntax

```
--reduce-site-overhead=<string>
```

Arguments

`<string>` is a comma-separated list of sites (no spaces).

Default

No default argument

Actions Modified

`report=suitability`

Usage

Site overhead is the time spent starting up (and shutting down) parallel execution. It includes creating threads, scheduling those threads onto cores, and waiting for the threads to begin executing.

Usage of this option simulates parallel execution with the assumption that site overhead is zero for a specified site.

Example

Generate a Suitability report. Examine the annotated sites `myAnnotatedSiteJ` and `myAnnotatedSiteX` for opportunities to reduce site overhead. Write the report to `stdout`.

```
advisor --report=suitability --reduce-site-overhead=myAnnotatedSiteJ,myAnnotatedSiteX --project-dir=./advi_results
```

See Also

[Reduce Site Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#)

reduce-task-overhead

Examine specified annotated sites for opportunities to reduce task overhead in a Suitability report.

GUI Equivalent

Suitability > Task Overhead

Syntax

```
--reduce-task-overhead=<string>
```

Arguments

<string> is a comma-separated list of sites (no spaces).

Default

No default argument

Actions Modified

`report=suitability`

Usage

Task overhead is the time spent creating a task, assigning it to a thread, and stopping or pausing the thread when the task is complete.

Usage of this option simulates parallel execution with the assumption that task overhead is zero for a specified site.

Example

Generate a Suitability report. Examine the annotated sites `myAnnotatedSiteJ` and `myAnnotatedSiteX` for opportunities to reduce task overhead. Write the report to `stdout`.

```
advisor --report=suitability --reduce-task-overhead=myAnnotatedSiteJ,myAnnotatedSiteX --project-dir=./advi_results
```

See Also

[Reducing Task Overhead](#)

[Command Line Interface Reference](#)

[advisor Command Action Reference](#)

refinalize-survey

Refinalize a survey result collected with a previous Intel® Advisor version or if you need to correct or update source and binary search paths.

Syntax

```
--refinalize-survey
```

```
--no-refinalize-survey
```

Default

Off (no-refinalize-survey)

Actions Modified

`report=survey`

Usage

Typical usage scenarios:

- Source files were moved after compilation.
- You open remotely collected results on a viewing system before setting the search paths appropriately.

Example

Refinalize the result of the Survey analysis. Search recursively for source files in the `./src` search directory and for binary files in the `./bin` search directory. Write the refinalized results to the `./advi_results` project directory instead of the default working directory.

```
advisor --report=survey --search-dir src=./src bin=./bin --refinalize-survey --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

remove

Remove loops (by file and line number) from the loops selected for deeper analysis.

GUI Equivalent

Survey > 

Syntax

`--remove=<string>`

Arguments

`<string>` is a comma-separated list of files/line numbers in the following format: `file1:line1`

Default



No default string

Actions Modified

`mark-up-loops`

Usage

Removing loops that are not of interest can decrease collection overhead.

Do not confuse the `mark-up-loops` action with the `mark-up-list` action option. The `mark-up-loops` action coupled with the `select` action option enables a GUI  checkbox; therefore loop selection persists beyond the duration of the `mark-up-loops` action and applies to downstream analyses, such as Dependencies and Memory Access Patterns analyses. The `collect` action coupled with the `mark-up-list` action option simulates enabling a GUI  checkbox; therefore loop selection persists only for the duration of the `collect` action.

Example

1. Select two loops for deeper analysis.

2. Remove one loop from the selection list.

```
advisor --mark-up-loops --select=foo.cpp:34,bar.cpp:192 --project-dir=./advi_results -- ./myApplication
advisor --mark-up-loops --remove=bar.cpp:192 --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

report-output

Redirect report output from stdout to another location.

Syntax

`--report-output=<PATH>`

Arguments

`<PATH>` is the directory PATH/filename.

Default

stdout

Actions Modified

report

Example

Generate a Suitability report. Output the report in text format. Save it as `suitability.txt` in the `out` output directory.

```
advisor --report=suitability --format=text --report-output=./out/suitability.txt --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

report-template

Specify the PATH/name of a custom report template file.

Syntax

`--report-template=<PATH>`

Arguments

`<PATH>` is the directory PATH/name.

Default

No default argument

Actions Modified

report

Example

Generate a custom report. Use the `./template/suitability.template` file.

```
advisor --report=custom --report-template=./template/suitability.template --project-dir=./
advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

result-dir

Specify a directory to identify the running analysis.

Syntax

```
--r=<PATH>
--result-dir=<PATH>
```

Arguments

`<PATH>` is the directory PATH.

Default

No default argument

Actions Modified

command

Example

Pause the analysis running in the `r000hs` directory.

```
advisor --command=pause -r=./r000hs
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

resume-after

Resume collection after the specified number of milliseconds.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Type] > Advanced > Automatically resume collection after (sec)

Syntax

```
--resume-after=<integer>
```

--resume-after=<double>

Arguments

You can specify the option value as one of the following:

<integer> is the number of milliseconds to wait before resuming data collection.

<double> is the number of seconds or fraction of seconds to wait before resuming data collection. For example, 1.56 is 1 sec 560 ms.

Default

Off

Actions Modified

collect

Usage

Skip *uninteresting* parts of your target application, such as the initialization phase, and analyze only *interesting* parts.

Collection automatically starts in the paused state.

Usage can decrease collection overhead.

NOTE The value in the corresponding GUI property is only in *seconds*.

Example

Run a Survey analysis. Launch the application with collection paused. Start collection after 30 milliseconds.

```
advisor --collect=survey --resume-after=30 --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

return-app-exitcode

Return the target exit code instead of the command line interface exit code.

Syntax

--return-app-exitcode

--no-return-app-exitcode

Default

Off (no-return-app-exitcode)

Actions Modified

collect

Example

```
advisor --collect=survey --return-app-exitcode --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

search-dir

Specify the location(s) for finding target support files.

GUI Equivalent

Project Properties > Binary/Symbol Search

Project Properties > Source Search

Syntax

```
--search-dir <keyword>=<PATH>
```

Arguments

Combine keywords with arguments in the following syntax: `<all | bin | src | sym[:<p | r>]>=<PATH>`

`<PATH>` is the PATH/name of the search directory, and can include environment paths and absolute paths.

Keyword	Description
all	Search all types of directories.
bin	Search binary directories.
src	Search source directories. This is used for most <code>collect</code> actions.
sym	Search symbol directories.
:r	Perform a recursive search of all subdirectories.
:p	Specify the highest priority search (directories to search prior to others, including environment paths and absolute paths).
:rp	Combine :r and :p.

Actions Modified

`collect`, `create-project`, `import-dir`, `report`

Usage

Use `--search-dir src:=<PATH>` when performing `collect` actions.

To exclude files from analysis, use the `exclude-files` option.

Example

Run a Suitability analysis on `myApplication`. Search for source files in the specified search directory. Write the result to the specified project directory.

```
advisor --collect=suitability --search-dir src:=./src1 --project-dir=./advi_results -- ./myApplication
```

The following two commands are equivalent. Each runs a Suitability analysis on `myApplication` and searches for source files in the two specified search directories.

```
advisor --collect=suitability --search-dir src:=./src1 --search-dir src:=./src2 --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=suitability --search-dir src:=./src1,./src2 --project-dir=./advi_results -- ./myApplication
```

See Also

[exclude-files](#) Exclude the specified files or directories from annotation scanning during analysis.

[Binary/Symbol Search Tab](#)

[Source Search Tab](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

search-n-dim

Enable searching for an optimal N-dimensional offload.

Syntax

`--search-n-dim`

`--no-search-n-dim`

Default

On (`search-n-dim`)

Actions Modified

`collect=projection`

`collect=offload`

Usage

`search-n-dim` enables combining up to three nested *parallel* loops into an N-dimensional offload. This can reduce estimated execution time on GPU and Time by Compute. Use `no-search-n-dim` to estimate offloading for each loop separately.

NOTE Only loops with no dependencies can be combined into an N-dimensional offload.

Do *not* use this option with `--threads` or `--enable-batching`.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Do not search for an optimal N-dimensional offload when modeling your application performance on a target device.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --no-search-n-dim --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

select

Select loops (by file and line number, ID, or criteria) for deeper analysis.

GUI Equivalent

To select individual loops/functions: **Vectorization and Code Insights > Survey & Roofline > 🔍**

To select loops/functions with markup strategy for Offload Modeling: **Offload Modeling > Analysis Workflow > Dependencies > Markup type**

Syntax

```
--select=<string>
```

Arguments

<string> is a comma-separated list of file name and line number, loop ID, and/or criteria in the `[(r|recursive):]<id>|<file>:<line>|<criteria>[,<id>|<file>:<line>|<criteria>,..]` format. Add `r:` (or `recursive:`) prefix to select all loops in call trees starting from heads selected by criteria, source location or ID. <criteria> can be one of the following:

Criteria	Description
scalar	Include scalar serial loops.
total-time>N	Include loops above N% of total CPU time.
has-source	Exclude loops without source location.
has-issue	Include loops with <i>Vector Dependence Prevent Vectorization</i> or <i>Possible Inefficient Memory Access Pattern</i> issue.
loop-height=N	Include loops at a specific hierarchical position. 0 = Innermost loops.
markup=name	Select loops using a pre-defined mark-up algorithm. Supported algorithms are: <ul style="list-style-type: none"> • <code>gpu_generic</code> - Select loops executed on a GPU. • <code>omp</code> - Select OpenMP* loops. • <code>icpx -fsycl</code> - Select SYCL loops. • <code>ocl</code> - Select OpenCL™ loops. • <code>daal</code> - Select Intel® oneAPI Data Analytics Library loops. • <code>tbb</code> - Select Intel® oneAPI Threading Building Blocks loops.

Default

No default argument

Actions Modified

`collect`



`mark-up-loops`

NOTE With the `--collect=projection` action, the `select` option accepts only loop/function IDs and source locations (in the `<file-name>:<line>` format).

Usage

Use `+` to combine criteria with AND logic. For example, use `--select=scalar+has-source` to select all scalar loops that have source location.

NOTE Selecting loops of interest can decrease collection overhead.

Do not confuse the `mark-up-loops` action with the `mark-up-list` action option. The `mark-up-loops` action coupled with the `select` action option enables a GUI  checkbox; therefore loop selection persists beyond the duration of the `mark-up-loops` action and applies to downstream analyses, such as Dependencies and Memory Access Patterns analyses. The `collect` action coupled with the `mark-up-list` action option simulates enabling a GUI  checkbox; therefore loop selection persists only for the duration of the `collect` action.

Example

1. Run a Survey analysis to identify loops of interest.
2. Select those loops for deeper analysis.

```
adviser --collect=survey --project-dir=./advi_results -- ./myApplication
adviser --mark-up-loops --select=foo.cpp:34,bar.cpp:192 --project-dir=./advi_results -- ./
myApplication
```

See Also

[Minimize Analysis Overhead](#)

[adviser Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `adviser <--action> [--action-options] [--global-options] [--] target [target options]`.

set-dependency

Assume loops with specified IDs or source locations have a dependency.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

`--set-dependency=<string>`

Arguments

`<string>` is a comma-separated list of loop IDs or source locations.

Actions Modified

`collect=projection`

`collect=offload`

Usage

If the list is empty, assume all loops have a dependency.

NOTE

`--set-dependency` option takes precedence over `--set-parallel`, so if the loop is listed in both, it is considered as having a dependency.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device assuming loops at source locations `my_source.cpp:132` and `my_source.cpp:155` have dependencies.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results  
-- ./myApplication
```

```
advisor --collect=projection --set-dependency=my_source.cpp:132,my_source.cpp:155 --project-  
dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

set-parallel

Assume loops with specified IDs or source locations are parallel.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

`--set-parallel=<string>`

Arguments

`<string>` is a comma-separated list of loop IDs or source locations.

Actions Modified

`collect=projection`

`collect=offload`

Usage

If the list is empty, assume all loops are parallel.

NOTE

`--set-dependency` option takes precedence over `--set-parallel`, so if the loop is listed in both, it is considered as having a dependency.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device assuming loops at source locations `my_source.cpp:132` and `my_source.cpp:155` do not have dependencies.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results  
-- ./myApplication
```

```
advisor --collect=projection --set-parallel=my_source.cpp:132,my_source.cpp:155 --project-dir=./  
advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

set-parameter

Specify a single-line parameter to modify in a target device configuration.

Syntax

```
--set-parameter=<string>
```

Arguments

`<string>` is a parameter to modify and its new value in the following format:

"`<group>.<parameter>=<new-value>`". You can specify a comma-separated list of parameters to modify.

Actions Modified

`collect=projection`

`collect=offload`

Usage

You can specify a comma-separated list of parameters or specify this option more than once to change several parameters.

Example

Run the Performance Modeling and change the minimum speedup from 1 (default) to 2:

```
advisor --collect=projection --set-parameter="min_required_speed_up=2" --project-dir=./  
advi_results
```

See Also

[Advanced Modeling Configuration](#)

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

show-all-columns

Show data for all available columns in a Survey report.

GUI Equivalent

Survey > Customize View > Settings > Configure Columns

Syntax

```
--show-all-columns  
--no-show-all-columns
```

Default

Off (no-show-all-columns)

Actions Modified

```
report=survey  
report=top-down
```

Example

Generate a Survey report. Output in CSV format. Show data for all available columns.

```
advisor --report=survey --show-all-columns --format=csv --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

show-all-rows

Show data for all available rows, including data for child loops, in a Survey report.

GUI Equivalent

Survey > Function Calls Sites and Loops > +

Syntax

```
--show-all-rows  
--no-show-all-rows
```

Default

On (show-all-rows)

Actions Modified

```
report=survey
```

Example

Generate a default Survey report. Output in CSV format. Show data for present child loops.

```
advisor --report=survey --format=csv --report-output=./out/survey.csv --project-dir=./advi_results
```

Generate a default Survey report. Output in CSV format. Do not show data for present child loops.

```
advisor --report=survey --format=csv --no-show-all-rows --report-output=./out/survey.csv --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

show-functions

Show only functions in a report.

GUI Equivalent

Filters

Syntax

--show-functions

--no-show-functions

Default

Off (no-show-functions)

Actions Modified

`report`

Usage

The `show-loops` option, which shows only loops in generated reports, is switched on by default. The `show-functions` option, which shows only functions in generated reports, is switched off by default.

Example

Generate a Survey report showing **both loops and functions**.

```
advisor --report=survey --show-functions --project-dir=./advi --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

Generate a Survey report showing **functions only**.

```
advisor --report=survey --no-show-loops --show-functions --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

Generate a default Survey report showing **loops only**.

```
advisor --report=survey --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

Generate an **empty** Survey report - you disabled showing information about loops, and showing information about functions is off by default.

```
advisor --report=survey --no-show-loops --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

See Also

[show-loops](#) Show only loops in a report.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

show-loops

Show only loops in a report.

GUI Equivalent

Filters

Syntax

`--show-loops`

`--no-show-loops`

Default

On (show-loops)

Actions Modified

`report`

Usage

The `show-loops` option, which shows only loops in generated reports, is switched on by default. The `show-functions` option, which shows only functions in generated reports, is switched off by default.

Example

Generate a Survey report showing **both loops and functions**.

```
advisor --report=survey --show-functions --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

Generate a Survey report showing **functions only**.

```
advisor --report=survey --no-show-loops --show-functions --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

Generate a default Survey report showing **loops only**.

```
advisor --report=survey --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

Generate an **empty** Survey report - you disabled showing information about loops, and showing information about functions is off by default.

```
advisor --report=survey --no-show-loops --format=text --report-output=./out/survey.txt --project-dir=./advi_results
```

See Also

[show-functions](#) Show only functions in a report.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

show-not-executed

Show not-executed child loops in a Survey report.

GUI Equivalent

Survey > Show all loops (with zero and non-zero time)

Syntax

`--show-not-executed`

`--no-show-not-executed`

Default

Off (no-show-not-executed)

Actions Modified

`report=survey`

`report=top-down`

Example

Generate a top-down Survey report. Output in text format. Show data for not-executed child loops.

```
advisor --report=top-down --show-not-executed --format=text --report-output=./out/topdown.txt --
project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

show-report

Generate a Survey report for data collected for GPU kernels.

Syntax

`--show-report`

`--no-show-report`

Default

On (show-report)

Actions Modified

`collect=survey --profile-gpu`

Example

Run the Survey analysis for the GPU kernels and explicitly enable adding the collected data to a report.

```
advisor --collect=survey --profile-gpu --show-report --project-dir=./advi_results
```

See Also

[profile-gpu](#) Analyze OpenCL™ and oneAPI Level Zero programs running on Intel® Graphics.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

small-node-filter

Specify the total time threshold, in milliseconds, to filter out nodes that fall below this value from PDF and DOT reports.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--small-node-filter=<integer>
```

Arguments

<integer> is total time threshold, in milliseconds.

Default

0

Actions Modified

`collect=projection`

`collect=offload`

Usage

This option affects only PDF and DOT reports, which is a graphical representation of an application call tree.

Important

The PDF and DOT reports are available only on Linux* OS if the DOT* tool is installed.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device and filter out loops with total time less than 5 milliseconds from the PDF report.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results  
-- ./myApplication
```

```
advisor --collect=projection --small-node-filter=5 --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

sort-asc

Sort data in ascending order (by specified column name) in a report.

GUI Equivalent

Sort **Survey** by column

Syntax

`--sort-asc=<string>`

Arguments

`<string>` is column name.

Actions Modified

`report`

Usage

Data in string format is sorted lexicographically.

Example

Generate a Survey report. Sort data in ascending order by **Total Time**.

```
advisor --report=survey --sort-asc="Total Time" --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

sort-desc

Sort data in descending order (by specified column name) in a report.

GUI Equivalent

Sort **Survey** by column

Syntax

`--sort-desc=<string>`

Arguments

`<string>` is column name.

Actions Modified

`report`

Usage

Data in string format is sorted lexicographically.

Example

Generate a Survey report. Sort data in descending order by **Total Time**.

```
advisor --report=survey --sort-desc="Total Time" --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

spill-analysis

Register flow analysis to calculate the number of consecutive load/store operations in registers and related memory traffic in bytes during Survey analysis.

GUI Equivalent

Project Properties > Analysis Target > Survey Analysis > Advanced > Enable register spill/fill analysis

Syntax

```
--spill-analysis  
--no-spill-analysis
```

Default

Off (no-spill-analysis)

Actions Modified

```
collect=survey  
collect=offload
```

Usage

Enabling can increase finalization overhead.

Example

Run a Survey analysis. Enable spill/fill analysis.

```
advisor --collect=survey --spill-analysis --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

stack-access-granularity

Specify stack access size to set stack memory access measurement granularity for the data transfer simulation.

Syntax

```
--stack-access-granularity=<integer>
```

Arguments

<integer> is a power-of-two value from 16 to 1048576.

Default

1048576

Actions Modified

```
collect=tripcounts --data-transfer=[medium | full]
```

```
collect=tripcounts --enable-data-transfer-analysis --track-stack-accesses
```

Usage

Decrease the stack access granularity if your application has multiple small objects on a stack to improve analysis accuracy. Decreasing the granularity increases collection overhead.

Example

Run the Trip Counts analysis with medium data transfer and decrease the stack access size to 32:

```
advisor --collect=tripcounts --data-transfer=medium --stack-access-granularity=16 --project-dir=./advi_results -- ./myApplication
```

See Also

[data-transfer](#) Set the level of details for modeling data transfers during Characterization.

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[track-stack-accesses](#) Track accesses to stack memory.

[advisor Command Line Interface](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Action Reference](#)

[Minimize Analysis Overhead](#)

stack-stitching

Restructure the call flow during Survey analysis to attach stacks to a point introducing a parallel workload.

GUI Equivalent

Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Stitch stacks

Syntax

```
--stack-stitching
```

```
--no-stack-stitching
```


Default

On (stack-stitching)

Actions Modified

`collect=survey`

Usage

The option restores a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload.

Disable when Survey analysis runtime overhead exceeds 1.1x

Disabling can decrease collection overhead and significantly decrease finalization overhead depending on workload.

Example

Run a Survey analysis. Disable stack stitching.

```
advisor --collect=survey --no-stack-stitching --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

stack-unwind-limit

Set stack size limit when analyzing stacks after collection.

Syntax

`--stack-unwind-limit=<integer>`

Arguments

`<integer>` is the maximum stack size to analyze.

Default

8388608

Actions Modified

`collect=survey --stackwalk-mode=offline`

Usage

Use to set the stack size limit when analyzing stacks after collection, which is the offline callstack unwinding mode. The offline mode is default, so you can skip the `--stackwalk-mode=offline` option.

Example

Run the Survey analysis and set the stack size limit to 10000.

```
advisor --collect=survey --stack-unwind-limit=10000 --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Line Interface](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[advisor Command Action Reference](#)

stacks

Perform advanced collection of callstack data during Roofline and Trip Counts & FLOP analysis.

GUI Equivalent

Project Properties > Analysis Target > Trip Counts & FLOP Analysis > Advanced > Collect stacks
Analysis Workflow > [CPU | GPU] Roofline > Characterization > Collect stacks

Syntax

```
--stacks
--no-stacks
```

Default

Off (no-stacks)

Actions Modified

```
collect=roofline
collect=tripcounts
collect=offload
```

Usage

Enabling can increase collection overhead.

Example

Run a Roofline analysis. Collect advanced callstack data.

```
advisor --collect=roofline --stacks --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimizing Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

stackwalk-mode

Choose between online and offline modes to analyze stacks during Survey analysis.

GUI Equivalent

Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Stack unwinding mode

Syntax

```
--stackwalk-mode=<string>
```

Arguments

Argument	Description
online	Analyze stacks during collection.
offline	Analyze stacks after collection.

Default

offline

Actions Modified

collect=survey

Usage

Set to `offline` in the following cases:

- Survey analysis overhead exceeds 1.1x
- A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions

Otherwise, set to `online`. This mode improves stack accuracy but increases overhead.

Example

Run a Survey analysis. Analyze stacks during collection.

```
advisor --collect=survey --stackwalk-mode=online --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

start-paused

Start executing the target application for analysis purposes, but delay data collection.

GUI Equivalent

Analysis Workflow >



To resume data collection: **Analysis Workflow** >

Syntax

```
--start-paused
```

Default

Off

Actions Modified

`collect`

Usage

Skip *uninteresting* parts of your target application, such as the initialization phase, and analyze only *interesting* parts.

You can use different techniques to resume collection, such as `__itt_resume`.

Usage can decrease analysis overhead.

Example

Launch the `sample` application with Suitability data collection paused.

```
advisor --collect=suitability --start-paused --project-dir=./advi_results -- ./myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

[static-instruction-mix](#)

Statically calculate the number of specific instructions present in the binary during Survey analysis.

GUI Equivalent

Project Properties > Analysis Target > Survey Analysis > Advanced > Enable static instruction mix analysis

Syntax

`--static-instruction-mix`

`--no-static-instruction-mix`

Default

Off (`no-static-instruction-mix`)

Actions Modified

`collect=survey`

`collect=offload`

Usage

Dynamic instruction mix is counted for the entire execution of the application; static instruction mix is counted per iteration. The `static-instruction-mix`, `dynamic`, and `mix` options work together in the following manner:

- Collect static instruction mix data: `--collect=survey --static-instruction-mix`
(In the GUI: Static instruction mix data is calculated on demand.)

- Collect dynamic instruction mix data (and static instruction mix data, from which dynamic mix data is calculated): `--collect=tripcounts --flop`
- Show static instruction mix data in a Survey report: `--report=survey --mix --no-dynamic`
- Show dynamic mix instruction data in a Survey report: `--report=survey --mix --dynamic`
- A Survey report cannot show both static and dynamic mix instruction data.

(In the GUI: **Code Analytics** can show both static and dynamic instruction mix data.)

Enabling `static-instruction-mix`:

- Is necessary in scenarios involving the Python* API.
- Can increase finalization overhead.

Example

1. Run a Survey analysis. Collect static instruction mix data.
2. Generate a Survey report. Show static instruction mix data.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
advisor --report=survey --mix --no-dynamic --project-dir=./advi_results
```

See Also

dynamic Show (in a Survey report) how many instructions of a given type actually executed during Trip Counts & FLOP analysis.

mix Show dynamic or static instruction mix data in a Survey report.

Minimize Analysis Overhead

advisor Command Option Reference

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

strategy

Specify processes and/or children for instrumentation during Survey analysis.

Syntax

`--strategy=<string>`

Arguments

`<string>` is a comma-separated list (no spaces) in the format `[process1 | child1:]profiling mode`.

Available profiling modes include the following:

Argument	Description
trace:trace	Instrument process and all children.
trace:notrace	Instrument process but not children.
notrace:trace	Instrument children but not process.
notrace:notrace	Do not instrument process or children.

Default

Instrument all processes and children (strategy=trace:trace)

Actions Modified

`collect=survey`

Example

Process_A starts several processes:

```
Root >
  Process_A >
    Child_of_A_1
    Child_of_A_2
  Process_B >
    Child_of_B
```

Run a Survey analysis. Instrument Child_of_A_2 and all children of Process_B.

```
advisor --collect=survey --
strategy=Root:notrace:notrace,Child_of_A_2:trace:notrace,Process_B:notrace:trace --project-dir=./
advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

support-multi-isa-binaries

Collect a variety of data during Survey analysis for loops that reside in non-executed code paths.

GUI Equivalent

Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Analyze loops that reside in non-executed code paths

Syntax

```
--support-multi-isa-binaries
--no-support-multi-isa-binaries
```

Default

Off (no-support-multi-isa-binaries)

Actions Modified

`collect=survey` for binaries compiled using the `ax` (Linux* OS)/`Qax` (Windows* OS) option with an Intel compiler

Usage

Disabling can decrease finalization overhead.

Example

Run a Survey analysis. Explicitly disable collecting data for loops that reside in non-executed code paths.

```
advisor --collect=survey --no-support-multi-user-binaries --project-dir=./advi_results -- ./
myApplication
```

See Also

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

target-device

Specify a device configuration to model cache for during Trip Counts collection.

GUI Equivalent

Analysis Workflow > Offload Modeling > Target Platform Model

Syntax

`--target-device=<string>`

Arguments

`<string>` is one of the following device configurations:

Argument	Description
pvc_xt_448 xve	Intel® Data Center GPU Max 448
pvc_xt_512 xve	Intel® Data Center GPU Max 512
xehpg_256x ve	Intel® Arc™ graphics with 256 vector engines
xehpg_512x ve	Intel® Arc™ graphics with 512 vector engines
gen12_tgl	Intel® Iris® Xe graphics
gen12_dg1	Intel® Iris® Xe MAX graphics
gen11_icl	Intel® Iris® Plus graphics
gen9_gt2	Intel® HD Graphics 530
gen9_gt3	Intel® Iris® Graphics 550
gen9_gt4	Intel® Iris® Pro Graphics 580

Default

No default

Actions Modified

`collect=tripcounts --enable-cache-simulation`

`collect=offload`

Usage

Important

Make sure to specify the same configuration argument as for the `config` option during Performance Modeling (`collect=projection`).

Do not confuse the `target gpu` option with the `target-device` option. The `target-device` option modifies only the `--collect=tripcounts` action and is used to simulate memory cache of a specific GPU platform for Offload Modeling. The `--target-gpu` modifies the `--collect=roofline`, `--collect=survey`, and `--collect=tripcounts` and is used to select a target GPU to collect data and plot a GPU Roofline for if you have several GPUs.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance for the `gen9_gt2` configuration.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-cache-simulation --target-device=gen9_gt2 --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --config=gen9_gt2 --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

target-gpu

Specify a target GPU to collect data for if you have multiple GPUs connected to your system.

Syntax

`--target-gpu=<string>`

Arguments

`<string>` is a bus/device/function of a GPU adapter with *decimal* numbers in the following format:

`<domain>:<bus>:<device-number>.<function-number>`

Default

The most recent GPU device family that Intel® Advisor detects. If you have several such GPUs, the first device on a PCI bus is the default.

Actions Modified

`collect=survey --profile-gpu`

`collect=tripcounts --profile-gpu`

`collect=roofline --profile-gpu`

Usage

Make sure to specify the same device configuration for both Survey and Trip Counts & FLOP analyses..

To see the list of option arguments for your system, run `advisor --help target-gpu` and see the option description.

For a list of GPUs with their bus/device/function adapters:

- On Windows* OS, see **Task Manager**.
- On Linux* OS, run `lspci -D`.

Important

`--target-gpu` option accepts *only* decimal numbers in a GPU address, and the option help lists the available arguments in the acceptable format. If you use a different way to get the GPU adapter list, for example, the `lspci -D` command, and get the GPU address with numbers in a hexadecimal format, you should convert it to the decimal format before passing to the `--target-gpu` option.

For example, if you have the address `0:4d:0.0`, you should convert it to the decimal format first and pass it to the Intel Advisor as `0:77:0.0`.

Do not confuse the `target gpu` option with the `target-device` option. The `target-device` option modifies only the `--collect=tripcounts` action and is used to simulate memory cache of a specific GPU platform for Offload Modeling. The `--target-gpu` modifies the `--collect=roofline`, `--collect=survey`, and `--collect=tripcounts` and is used to select a target GPU to collect data and plot a GPU Roofline for if you have several GPUs.

Example

1. Run Survey analysis for the GPU adapter `0:0:2.0`.
2. Run Trip Counts and FLOP analyses of the Characterization stage for the same device.

```
advisor --collect=survey --profile-gpu --target-gpu=0:0:2.0 --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --profile-gpu --target-gpu=0:0:2.0 --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

target-pid

Attach Survey or Trip Counts & FLOP collection to a running process specified by the process ID.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Type] > Attach to Process > PID

Syntax

```
--target-pid=<integer>
```

Arguments

`<integer>` is a process ID.

Default

No default argument

Actions Modified

`collect=survey` - with call stacks disabled (default)

`collect=tripcounts` - with call stacks disabled (default)

Usage

The usage scenario is similar to starting a target application with collection paused, except you can attach to an already running process.

Usage can decrease collection overhead.

Use the `command` action with the arguments shown below to:

- `detach` - the process continues running but analysis data collection stops.
- `stop` - kill the process, which also stops analysis data collection.

Example

Attach Survey collection to the running process with the process ID 5.

```
advisor --collect=survey --target-pid=5 --project-dir=./advi_results -- ./myApplication
```

See Also

[command](#) Control the Intel Advisor while running analyses.

[stacks](#) Perform advanced collection of callstack data during Roofline and Trip Counts & FLOP analysis.

[start-paused](#) Start executing the target application for analysis purposes, but delay data collection.

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

target-process

Attach Survey or Trip Counts & FLOP collection to a running process specified by the process name.

GUI Equivalent

Project Properties > Analysis Target > [Analysis Type] > Attach to Process > Process name

Syntax

`--target-process=<string>`

Arguments

`<string>` is a process name.

Default

No default argument

Actions Modified

`collect=survey` - with call stacks disabled (default)

`collect=tripcounts` - with call stacks disabled (default)

Usage

The usage scenario is similar to starting a target application with collection paused, except you can attach to an already running process.

Usage can decrease collection overhead.

Use the `command` action with the arguments shown below to:

- `detach` - the process continues running but analysis data collection stops.
- `stop` - kill the process, which also stops analysis data collection.

Example

Attach Survey collection to the running process `MyProcess`.

```
advisor --collect=survey --target-process=MyProcess --project-dir=./advi_results -- ./
myApplication
```

See Also

[command](#) Control the Intel Advisor while running analyses.

[stacks](#) Perform advanced collection of callstack data during Roofline and Trip Counts & FLOP analysis.

[start-paused](#) Start executing the target application for analysis purposes, but delay data collection.

[Minimize Analysis Overhead](#)

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

target-system

Specify the hardware configuration to use for modeling purposes in a Suitability report.

GUI Equivalent

Suitability > Target System

Syntax

`--target-system=<string>`

Arguments

`<string>` is one of the following: `cpu` | `xeon-phi` | `offload-to-xeon-phi`

Default

`cpu`

Actions Modified

`report=suitability`

Example

Generate a Suitability report. Use Intel® Xeon Phi™ as the target configuration

```
advisor --report=suitability --target-system=xeon-phi --project-dir=./advi_results
```

See Also[advisor Command Option Reference](#)[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.**threading-model***Specify the threading model to use for modeling purposes in a Suitability report.*

GUI Equivalent**Suitability > Threading Model****Syntax**`--threading-model=<string>`**Arguments**

<string> is one of the following: tbb | openmp | tpl (Windows* OS only) | other

Default

tbb

Actions Modified`report=suitability`**Usage**

Use only one threading model. Mixing threading models is not supported.

Example

Generate a Suitability report for the OpenMP* parallel framework.

```
advisor --report=suitability --threading-model=openmp --project-dir=./advi_results
```

See Also[advisor Command Option Reference](#)[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.**threads***Specify the number of parallel threads to use for offload heads.*

GUI Equivalent**Project Properties > Analysis Target > Performance Modeling > Other parameters****Syntax**`--threads=<integer>`**Arguments**

<integer> is a number of parallel threads. Specify 0 to let the model decide based on a selected target device configuration.

Default

0

Actions Modified

`collect=projection`

`collect=offload`

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses of the Characterization stage.
3. Model your application performance on a target device using 3 parallel threads for each offload head.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --project-dir=./advi_results
-- ./myApplication
```

```
advisor --collect=projection --threads=3 --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

top-down

Generate a Survey report in top-down view.

GUI Equivalent

Survey > Top Down

Syntax

`--top-down`

Default

Off

Actions Modified

`report=survey`

Usage

Equivalent to `report=top-down`

Example

Generate a Survey report in top-down view.

```
advisor --report=survey --top-down --project-dir=./advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

trace-mode

Set how to trace loop iterations during Memory Access Patterns analysis.

Syntax

```
--trace-mode=<string>
```

Arguments

<string> is one of the following:

Argument	Description
full	Trace all loop iterations.
linear	Trace loop iterations using linear step.
fibonacci	Trace loop iterations in Fibonacci sequence.

Default

fibonacci

Actions Modified

collect=map

Usage

Specifying a less extensive tracing method can decrease collection overhead.

Example

Run a Memory Access Patterns analysis on the specified loops. Trace all loop iterations.

```
advisor --collect=map --mark-up-list=3,4,5 --trace-mode=full --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

trace-mpi

Configure collectors to trace MPI code and determine MPI rank IDs for non-Intel® MPI library implementations.

Syntax

```
--trace-mpi  
--no-trace-mpi
```

Default

Off (no-trace-mpi)

Actions Modified

collect

Example

Run a Survey analysis. Use a non-Intel MPI library on a Windows* OS.

```
mpiexec -n 4 "advisor --collect=survey --trace-mpi --no-auto-finalize --project-dir=./
advi_results" ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

track-memory-objects

Attribute memory objects to the analyzed loops that accessed the objects.

GUI Equivalent

Project Properties > Analysis Target > Performance Modeling > Other parameters

Syntax

```
--track-memory-objects
--no-track-memory-objects
```

Default

Off (no-track-memory-objects)

Actions Modified

```
collect=tripcounts --enable-data-transfer-analysis
collect=projection
```

Usage

Use as *one* of the following:

- Use the *medium* or *full* data transfer with `collect=tripcounts` and specify `track-memory-objects` only for `collect=projection`. For example:

```
advisor --collect=tripcounts --flop --data-transfer=full --project-dir=<project-dir>
-- <target-application>

advisor --collect=projection --track-memory-objects --project-dir=<project-dir>
```
- Enable the *basic data transfer analysis* with `collect=tripcounts` and specify `track-memory-objects` for both `collect=tripcounts` and `collect=projection`:

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis--track-memory-
objects --project-dir=<project-dir> -- <target-application>

advisor --collect=projection --track-memory-objects --project-dir=<project-dir>
```

NOTE Enabling can increase overhead.

Example

1. Run Survey Analysis.
2. Run Trip Counts and FLOP analyses and track memory objects.
3. Model your application performance on a target device and track memory objects.

```
advisor --collect=survey --static-instruction-mix --project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=tripcounts --flop --data-transfer=medium --target-device=xehpg_512xve --  
project-dir=./advi_results -- ./myApplication
```

```
advisor --collect=projection --track-memory-objects --project-dir=./advi_results
```

See Also

[data-transfer](#) Set the level of details for modeling data transfers during Characterization.

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

track-stack-accesses

Track accesses to stack memory.

Syntax

```
--track-stack-accesses
```

```
--no-track-stack-accesses
```

Default

Off (no-track-stack-accesses)

Actions Modified

```
collect=tripcounts --enable-data-transfer-analysis
```

Usage

By default, the Intel® Advisor filters out all accesses to stack memory. When you enable this option, all accesses to stack memory are included in data transfer calculations.

This corresponds with `data-transfer=medium` or `data-transfer=full`. **Do not use the `data-transfer=<mode>` with the `track-stack-accesses` option because `data-transfer=<mode>` overrides the `enable-data-transfer-analysis` and `track-stack-accesses`.**

Tip Enabling can increase overhead.

Example

Run a Trip Counts and FLOP analysis. Enable data transfer simulation and analyze accesses to stack memory.

```
advisor --collect=tripcounts --flop --enable-data-transfer-analysis --track-stack-  
accesses --project-dir=./advi_results -- ./myApplication
```

See Also

[enable-data-transfer-analysis](#) Model data transfer between host memory and device memory.

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

track-stack-variables

Enable parallel data sharing analysis for stack variables during Dependencies analysis.

GUI Equivalent

Project Properties > Analysis Target > Dependencies Analysis > Analyze stack variables

Syntax

```
--track-stack-variables
--no-track-stack-variables
```

Default

On (track-stack-variables)

Actions Modified

`collect=dependencies`

Usage

Disabling can decrease collection overhead.

Example

Run a Dependencies analysis. Disable parallel data sharing analysis for stack variables.

```
advisor --collect=dependencies --no-track-stack-variables --search-dir src=./src --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

Command Line Interface Reference This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

trip-counts

Collect loop trip counts data during Trip Counts & FLOP analysis.

GUI Equivalent

Analysis Workflow > Characterizatoion > Collect trip counts

Project Properties> Analysis Target> Trip Counts and FLOP Analysis> Collect information about loop trip counts

Syntax

```
--trip-counts
--no-trip-counts
```

Default

On (trip-counts)

Actions Modified

`collect=tripcounts`

Usage

Use the option, which allows you to dynamically identify the number of times loops are invoked and executed, to:

- Detect loops with too-small trip counts and trip counts that are not a multiple of vector length.
- Analyze parallelism granularity more deeply.

Disabling can decrease analysis overhead.

Example

Run a Trip Counts & FLOP analysis. Collect trip counts, FLOP, and call stack data.

```
advisor --collect=tripcounts --flop --stacks --project-dir=./advi_results -- ./myApplication
```

Run a Trip Counts & FLOP analysis. Collect only FLOP data.

```
advisor --collect=tripcounts --no-trip-counts --search-dir src:=./src --project-dir=./advi_results -- ./myApplication
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

verbose

Maximize status messages during command execution.

Syntax

`-v`

`--verbose`

Default

Off

Actions Modified

`collect`, `command`, `import-dir`, `mark-up-loops`, `report`, `snapshot`,

Example

Generate a Suitability report. Output to `stderr`. Show additional status output.

```
advisor --report=suitability --verbose --project-dir=./advi_results
```

See Also

[quiet](#) Minimize status messages during command execution.

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

with-stack

Show call stack data in a Roofline interactive HTML report (if call stack data is collected).

GUI Equivalent

Roofline> Default: FLOAT > With Callstacks

Syntax

```
--with-stack
--no-with-stack
```

Default

Off (no-with-stack)

Actions Modified

`report=roofline`

Example

Generate a Roofline interactive HTML report. Show call stack data.

```
advisor --report=roofline --report-output=./out/roofline.html --with-stacks --project-dir=./
advi_results
```

See Also

[advisor Command Option Reference](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

Offload Modeling Command Line Reference

This reference section describes the command line options available for each of the Python scripts that you can use to run the Offload Modeling perspective.*

To use the Offload Modeling, run one or two of the following scripts, depending on a method you chose:

- Use [run_oa.py](#) script to collect performance data and model performance on a target device using a single command with a set of default recommended settings.
- Use [collect.py](#) to collect baseline performance data for your application on a host device.
- Use [analyze.py](#) to model your application performance on a target device.

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Command Syntax

The syntax for Offload Modeling commands is as follows:

```
advisor-python <APM>/<script-name>.py <project-dir> [--options] [-- <target> [target-options]]
```

where:

`advisor-python`

A call to the Intel® Advisor Python* command line tool.

	<p><code>advisor-python</code> is recommended to run the scripts. The main advantage of using this command line tool is that it does not require you to install a specific Python version on your system because it calls to an internal Python version of the Intel Advisor.</p>
<code><APM></code>	<p>The environment variable that points to the directory with the scripts. Replace it with:</p> <ul style="list-style-type: none"> • <code>\$APM</code> on Linux* OS • <code>%APM%</code> on Windows* OS
<code><script-name></code>	<p>A script name to run: <code>run_oa.py</code>, <code>collect.py</code>, or <code>analyze.py</code>.</p>
<code><project-dir></code>	<p>The path to a project directory to save collection results.</p>
<code><--options></code>	<p>Options to modify behavior specific to the script. You can specify several options per script. Using an option not supported by the script causes a usage error.</p>
<code><target></code>	<p>A target application to analyze.</p>
	<hr/> <p>Important You do not need to specify a target executable and target options when running the <code>analyze.py</code> script.</p> <hr/>
<code>[target-options]</code>	<p>Options to modify target application behavior.</p>

Syntax Rules and Alternatives

- An option can be preceded by one or two dashes. This section uses two dashes before long version of options and one dash before short version of options. For example, the following commands are equivalent:

```
advisor-python $APM/run_oa.py -h
```

```
advisor-python $APM/run_oa.py --help
```

- The path to a project directory must always follow after a script name. For example:

```
advisor-python $APM/analyze.py ./advi_results
```

- If an option accepts values, they can be separated by a space or by an equal sign (=). This document uses space for all such options. For example, the following are equivalent:

```
advisor-python $APM/analyze.py ./advi_results --out-dir ./report
```

```
advisor-python $APM/analyze.py ./advi_results --out-dir=./report
```

- The target executable must be preceded by two dashes and a space. For example:

```
advisor-python $APM/analyze.py ./advi_results --out-dir=./report
```

```
advisor-python $APM/collect.py ./advi_results -- myApplication
```

- If you have Python 3.6 or 3.7 installed and it is the default Python version on your system, you can run Offload Modeling with your system Python instead of the `advisor-python` tool:

```
python $APM/run_oa.py ./advi_results -- ./myApplication
```

```
python3.6 $APM/run_oa.py ./advi_results -- ./myApplication
```

```
python3.7 $APM/run_oa.py ./advi_results -- ./myApplication
```

run_oa.py Options

Collect basic data, do markup, and collect refinement data. Then proceed to run analysis on profiling data. This script combines the separate scripts `collect.py` and `analyze.py`.

Usage

```
advisor-python <APM>/run_oa.py <project-dir> [--options] -- <target> [target-options]
```

NOTE Replace `<APM>` with `$APM` on Linux* OS or `%APM%` on Windows* OS.

Options

The following table describes options that you can use with the `run_oa.py` script. The target application to analyze and application options, if any, must be preceded by two dashes and a space and placed at the end of a command.

Option	Description
<code><project-dir></code>	Required. Specify the path to the Intel® Advisor project directory.
<code>-h</code>	Show all script options.
<code>--help</code>	
<code>-v <verbose></code>	Specify output verbosity level:
<code>--verbose <verbose></code>	<ul style="list-style-type: none"> • 1 - Show only error messages. This is the least verbose level. • 2 - Show warning and error messages. • 3 (default) - Show information, warning, and error messages. • 4 - Show debug, information, warning, and error messages. This is the most verbose level.
<hr/> <p>NOTE This option affects the console output, but does not affect logs and report results.</p> <hr/>	
<code>--assume-dependencies (default) --no-assume-dependencies</code>	Assume that a loop has a dependency if the loop type is not known. When disabled, assume that a loop does not have dependencies if the loop dependency type is unknown.
<code>--assume-hide-taxes [<loop-id> <file-name>:<line-number>]</code>	Use an optimistic approach to estimate invocation taxes: hide all invocation taxes except the first one.

Option	Description
<code>--assume-never-hide-taxes</code> (default)	You can provide a comma-separated list of loop IDs and source locations to hide taxes for. If you do not provide a list, taxes are hidden for all loops.
<code>--assume-parallel</code> <code>--no-assume-parallel</code> (default)	Use a pessimistic approach to estimate invocation taxes: do not hide invocation taxes.
<code>--check-profitability</code> (default) <code>--no-check-profitability</code>	Assume that a loop is parallel if the loop type is not known.
<code>-c {basic, refinement, full}</code>	Check the profitability of offloading regions. Only regions that can benefit from the increased speed are added to a report.
<code>--collect {basic, refinement, full}</code>	When disabled, add all evaluated regions to a report, regardless of the profitability of offloading specific regions.
	Specify the type of data to collect for the application:
	<ul style="list-style-type: none"> <code>basic</code> - Collect basic performance data (Survey, Trip Counts, FLOP), analyze data transfer between host and device memory, attribute memory objects to loops, and track accesses to stack memory. <code>refinement</code> - Collect refined data (Dependencies) for marked loops only. Do not analyze data transfers. <code>full</code> (default) - Collect both basic data for application and refined data for marked loops, analyze data transfer between host and device memory and potential data reuse, attribute memory objects to loops, and track accesses to stack memory.
	<p>NOTE For <code>--collect full</code>, make sure to use <code>--data-reuse-analysis</code> and <code>--track-memory-objects</code>.</p> <p>For <code>--collect basic</code>, make sure to use the <code>--track-memory-objects</code>.</p>
<code>--config <config></code>	<p>Specify a configuration file by absolute path or name. If you choose the latter, the model configuration directory is searched for the file first, then the current directory.</p> <p>The following device configurations are available: <code>xehpg_512xve</code> (default), <code>xehpg_256xve</code>, <code>gen11_icl</code>, <code>gen12_tgl</code>, <code>gen12_dgl</code>, <code>gen9_gt4</code>, <code>gen9_gt3</code>, <code>gen9_gt2</code>.</p>

Option	Description
	<hr/> <p>NOTE You can specify several configurations by using the option more than once.</p> <hr/>
<code>--cpu-scale-factor <integer></code>	<p>Assume a host CPU that is faster than the original CPU by the specified value.</p> <p>All original CPU times are divided by the scale factor.</p>
<code>--data-reuse-analysis (default) --no-data-reuse-analysis</code>	<p>Estimate data reuse between offloaded regions. Disabling can decrease analysis overhead.</p> <hr/> <p>Important Use with <code>--collect full</code>.</p> <hr/>
<code>--data-transfer (default) --no-data-transfer</code>	<p>Analyze data transfer.</p> <hr/> <p>NOTE Disabling can decrease analysis overhead.</p> <hr/>
<code>--dry-run</code>	<p>Show the Intel® Advisor CLI commands for <code>advisor</code> appropriate for the specified configuration. No actual collection is performed.</p>
<code>--enable-batching --disable-batching (default)</code>	<p>Enable job batching for top-level offloads. Emulate the execution of more than one instance simultaneously.</p>
<code>--enable-edram</code>	<p>Enable eDRAM modeling in the memory hierarchy model.</p>
<code>--enable-slm</code>	<p>Enable SLM modeling in the memory hierarchy model. Use both with <code>collect.py</code> and <code>analyze.py</code>.</p>
<code>--exclude-from-report <items-to-exclude></code>	<p>Specify items to exclude from a report. Available items: <code>memory_objects</code>, <code>sources</code>, <code>call_graph</code>, <code>dependencies</code>, <code>strides</code>.</p> <p>By default, you can exclude the following items from the report:</p> <ul style="list-style-type: none"> For the CPU-to-GPU modeling: <ul style="list-style-type: none"> memory objects sources call graph dependencies For the GPU-to-GPU modeling: <ul style="list-style-type: none"> memory objects sources

Option	Description
	<p>Use this option if your report is heavy weight, for example, due to containing a lot of memory objects or sources, which slows down opening in a browser.</p> <hr/> <p>NOTE This option affects only data shown in the HTML report and does not affect data collection.</p> <hr/>
<code>--executable-of-interest <executable-name></code>	<p>Specify an executable process name to profile if it is not the same as the application to run. Use this option if you run your application via script or other binary.</p> <hr/> <p>NOTE Specify the <i>name</i> only, not the full path.</p> <hr/>
<code>--flex-cachesim <cache-configuration></code>	<p>Use flexible cache simulation to model cache data for several target devices. The flexible cache simulation allows you to change a device for an analysis without recollecting data. By default, when no configuration is set, cache data is simulated for all supported target platforms.</p> <p>You can also specify a list of cache configurations separated with a forward slash in the format <code><size_of_level1>:<size_of_level2>:<size_of_level3></code>. For each memory level size, specify a unit of measure as <i>b</i> - bytes, <i>k</i>- kilobytes, or <i>m</i> - megabytes.</p> <p>For example, <code>8k:512k:8m/24k:1m:8m/32k:1536k:8m</code>.</p>
<code>--gpu (recommended) --profile-gpu --analyze-gpu-kernels-only</code>	<p>Model performance only for code regions running on a GPU. Use <i>one</i> of the three options.</p> <hr/> <p>NOTE This is a <i>preview</i> feature. <code>--analyze-gpu-kernels-only</code> is deprecated and will be removed in future releases.</p> <hr/>
<code>--ignore <list></code>	<p>Specify a comma-separated list of runtimes or libraries to ignores time spent in regions from these runtimes and libraries when calculating per-program speedup.</p> <hr/> <p>NOTE This does not affect estimated speedup of individual offloads.</p> <hr/>

Option	Description
<code>--include-to-report <items-to-include></code>	<p>Specify items to include to a report. Available items: <code>memory_objects</code>, <code>sources</code>, <code>call_graph</code>, <code>dependencies</code>, <code>strides</code>.</p> <p>By default, you can add the following items from the report:</p> <ul style="list-style-type: none"> For the CPU-to-GPU modeling: <code>strides</code> For the GPU-to-GPU modeling: <ul style="list-style-type: none"> <code>call_graph</code> <code>dependencies</code> <code>strides</code> <p>Use this option if you want to add more data to the report or see that some data, for example, <code>sources</code> or <code>memory_objects</code>, are missing from the report, though you collected this data.</p> <hr/> <p>NOTE This option affects only data shown in the HTML report and does not affect data collection.</p>
<code>-m [{all, generic, regions, omp, icpx -fsycl, daal, tbb}]</code> <code>--markup [{all, generic, regions, omp, icpx -fsycl, daal, tbb}]</code>	<p>Mark up loops after survey or other data collection. Use this option to limit the scope of further collections by selecting loops according to a provided parameter:</p> <ul style="list-style-type: none"> <code>all</code> - Get lists of loop IDs to pass as the option for further collections. <code>generic</code> (default) - Mark up all regions and select the most profitable ones. <code>regions</code> - Select already existing parallel regions. <code>omp</code> - Select outermost loops in OpenMP* regions. <code>icpx -fsycl</code> - Select outermost loops in SYCL regions. <code>daal</code> - Select outermost loops in Intel® oneAPI Data Analytics Library regions. <code>tbb</code> - Select outermost loops in Intel® oneAPI Threading Building Blocks (oneTBB) regions. <p><code>omp</code>, <code>icpx -fsycl</code>, or <code>generic</code> selects loops in the project so that the corresponding collection can be run without loop selection options.</p> <p>You can specify several parameters in a comma-separated list. Loops are selected if they fit any of specified parameters.</p>
<code>--model-system-calls (default) --no-model-system-calls</code>	<p>Analyze regions with system calls inside. The actual presence of system calls inside a region may reduce model accuracy.</p>

Option	Description
<code>--mpi-rank <mpi-rank></code>	Specify a MPI rank to analyze if multiple ranks are analyzed.
<code>--no-cache-sources</code>	Disable keeping source code cache within a project.
<code>--no-cachesim</code>	Disable cache simulation during collection. The model assumes 100% hit rate for cache.
	NOTE Usage decreases analysis overhead.
<code>--no-profile-jit</code>	Disable JIT function analysis.
<code>--no-stacks</code>	Run data collection without collecting data distribution over stacks. You can use this option to reduce overhead at the potential expense of accuracy.
<code>-o <output-dir></code>	Specify the directory to put all generated files into. By default, results are saved in <code><advisor-project>/e<NNN>/pp<MMM>/data.0</code> . If you specify an existing directory or absolute path, results are saved in specified directory. The new directory is created if it does not exist.
<code>--out-dir <output-dir></code>	If you only specify the directory <code><name></code> , results are stored in <code><advisor-project>/e<NNN>/pp<MMM>/<name></code> .
	NOTE If you use this options, you might not be able to open the analysis results in the Intel Advisor GUI.
<code>-p <output-name-prefix></code>	Specify a string to add to the beginning output result filenames.
<code>--out-name-prefix <output-name-prefix></code>	
	NOTE If you use this options, you might not be able to open the analysis results in the Intel Advisor GUI.
<code>--set-parameter <CLI-config></code>	Specify a single-line configuration parameter to modify in a format " <code><group>.<parameter>=<new-value></code> ". For example: "min_required_speed_up=0", "scale.Tiles_per_process=0.5". You can use this option more than once to modify several parameters.
<code>--track-heap-objects (default) --no-track-heap-objects</code>	Deprecated. Use <code>--track-memory-objects</code> .

Option	Description
<code>--track-memory-objects</code> (default) <code>--no-track-memory-objects</code>	Attribute heap-allocated objects to the analyzed loops that accessed the objects. Disable to decrease analysis overhead. Important Currently, this option affects only the analysis step.
<code>--track-stack-accesses</code> (default) <code>--no-track-stack-accesses</code>	Track accesses to stack memory. Important Currently, this option does not affect the collection.

Examples

- Collect full data on `myApplication`, run analysis with default configuration, and save the project to the `./advi` directory. The generated output is saved to the default `advi/perfmodels/mNNNN` directory.

```
advisor-python $APM/run_oa.py ./advi_results -- ./myApplication
```

- Collect full data on `myApplication`, run analysis with default configuration, save the project to the `./advi` directory, and save the generated output to the `advi/perf_models/report` directory.

```
advisor-python $APM/run_oa.py ./advi_results --out-dir report -- ./myApplication
```

- Collect refinement data for SYCL code regions on `myApplication`, run analysis with a custom configuration file `config.toml`, and save the project to the `./advi` directory. The generated output is saved to the default `advi/perf_models/mNNNN` directory.

```
advisor-python $APM/run_oa.py ./advi_results --collect refinement --markup icpx -fsycl --config ./config.toml -- ./myApplication
```

collect.py Options

Depending on options specified, collect basic data, do markup, and collect refinement data. By default, execute all steps. For any step besides markup, you must specify an application argument.

Usage

```
advisor-python <APM>/collect.py <project-dir> [--options] -- <target> [target-options]
```

NOTE Replace `<APM>` with `$APM` on Linux* OS or `%APM%` on Windows* OS.

Options

The following table describes options that you can use with the `collect.py` script. The target application to analyze and application options, if any, must be preceded by two dashes and a space.

Option	Description
<code><project-dir></code>	Required. Specify the path to the Intel® Advisor project directory.

Option	Description
-h	Show all script options.
--help	
-v <verbose>	Specify output verbosity level:
--verbose <verbose>	<ul style="list-style-type: none"> 1 - Show only error messages. This is the least verbose level. 2 - Show warning and error messages. 3 (default) - Show information, warning, and error messages. 4 - Show debug, information, warning, and error messages. This is the most verbose level. <hr/> <p>NOTE This option affects the console output, but does not affect logs and report results.</p> <hr/>
-c {basic, refinement, full}	Specify the type of data to collect for an application:
--collect {basic, refinement, full}	<ul style="list-style-type: none"> <code>basic</code> - Collect basic performance data (Survey, Trip Counts, FLOP), analyze data transfer between host and device memory, attribute memory objects to loops, and track accesses to stack memory. <code>refinement</code> - Collect refined data (Dependencies) for marked loops only. Do not analyze data transfers. <code>full</code> (default) - Collect both basic data for application and refined data for marked loops, analyze data transfer between host and device memory and potential data reuse, attribute memory objects to loops, and track accesses to stack memory. <hr/> <p>NOTE For <code>--collect full</code>, make sure to use <code>--data-reuse-analysis</code> and <code>--track-memory-objects</code> for the Performance modeling with <code>analyze.py</code> or <code>advisor --collect=projection</code>. For <code>--collect basic</code>, make sure to use the <code>--track-memory-objects</code> for the Performance modeling with <code>analyze.py</code> or <code>advisor --collect=projection</code>.</p> <hr/>
--config <config>	Specify a configuration file by absolute path or name. If you choose the latter, the model configuration directory is searched for the file first, then the current directory.

Option	Description
<code>--data-reuse-analysis --no-data-reuse-analysis (default)</code>	<p>You can specify several configurations by using the option more than once.</p> <p>Estimate data reuse between offloaded regions. Disabling can decrease analysis overhead.</p> <hr/> <p>Important <code>--collect basic</code> and <code>--collect full</code> overwrite this option. To add the data reuse analysis results to the Offload Modeling report, make sure to use the <code>--data-reuse-analysis</code> option for the Performance modeling with <code>analyze.py</code> or <code>advisor --collect=projection</code>.</p> <hr/>
<code>--data-transfer (default) --no-data-transfer</code>	<p>Analyze data transfer.</p> <hr/> <p>NOTE Disabling can decrease analysis overhead.</p> <hr/>
<code>--dry-run</code>	<p>Show the Intel® Advisor CLI commands for <code>advisor</code> appropriate for the specified configuration. No actual collection is performed.</p>
<code>--enable-edram</code>	<p>Enable eDRAM modeling in the memory hierarchy model.</p> <hr/> <p>Important Make sure to use this option with both <code>collect.py</code> and <code>analyze.py</code>.</p> <hr/>
<code>--enable-slm</code>	<p>Enable SLM modeling in the memory hierarchy model.</p> <hr/> <p>Important Make sure to use this option with both <code>collect.py</code> and <code>analyze.py</code>.</p> <hr/>
<code>--executable-of-interest <executable-name></code>	<p>Specify the executable process name to profile if it is not the same as the application to run. Use this option if you run your application via script or other binary.</p> <hr/> <p>NOTE Specify the <i>name</i> only, not the full path.</p> <hr/>
<code>--flex-cachesim <cache-configuration></code>	<p>Use flexible cache simulation to model cache data for several target devices. The flexible cache simulation allows you to change a device for an</p>

Option	Description
<code>--gpu (recommended) --profile-gpu --analyze-gpu-kernels-only</code>	<p>analysis without recollecting data. By default, when no configuration is set, cache data is simulated for all supported target platforms.</p> <p>You can also specify a list of cache configurations separated with a forward slash in the format <code><size_of_level1>:<size_of_level2>:<size_of_level3></code>. For each memory level size, specify a unit of measure as <code>b</code> - bytes, <code>k</code> - kilobytes, or <code>m</code> - megabytes.</p> <p>For example, <code>8k:512k:8m/24k:1m:8m/32k:1536k:8m</code>.</p> <p>Model performance only for code regions running on a GPU. Use <i>one</i> of the three options.</p> <hr/> <p>Important Make sure to specify this option for both <code>collect.py</code> and <code>analyze.py</code>.</p> <hr/> <p>NOTE This is a <i>preview</i> feature. <code>--analyze-gpu-kernels-only</code> is deprecated and will be removed in future releases.</p> <hr/>
<code>--no-profile-jit (default)</code> <code>-m [{all, generic, regions, omp, icpx -fsycl, daal, tbb}]</code> <code>--markup [{all, generic, regions, omp, icpx -fsycl, daal, tbb}]</code>	<p>Disable JIT function analysis.</p> <p>Mark up loops after survey or other data collection. Use this option to limit the scope of further collections by selecting loops according to a provided parameter:</p> <ul style="list-style-type: none"> <code>all</code> - Get lists of loop IDs to pass as the option for further collections. <code>generic</code> (default) - Mark up all regions and select the most profitable ones. <code>regions</code> - Select already existing parallel regions. <code>omp</code> - Select outermost loops in OpenMP* regions. <code>icpx -fsycl</code> - Select outermost loops in SYCL regions. <code>daal</code> - Select outermost loops in Intel® oneAPI Data Analytics Library regions. <code>tbb</code> - Select outermost loops in Intel® oneAPI Threading Building Blocks (oneTBB) regions. <p><code>omp, icpx -fsycl, or generic</code> selects loops in the project so that the corresponding collection can be run without loop selection options.</p>

Option	Description
<code>--model-system-calls (default) --no-model-system-calls</code>	You can specify several parameters in a comma-separated list. Loops are selected if they fit any of specified parameters.
<code>--mpi-rank <mpi-rank></code>	Analyze regions with system calls inside. The actual presence of system calls inside a region may reduce model accuracy.
<code>--no-cache-sources</code>	Specify a MPI rank to analyze if multiple ranks are analyzed.
<code>--no-cachesim</code>	Disable keeping source code cache within a project.
	Disable cache simulation during collection. The model assumes 100% hit rate for cache.
	NOTE Usage decreases analysis overhead.
<code>--no-stacks</code>	Run data collection without collecting data distribution over stacks. You can use this option to reduce overhead at the potential expense of accuracy.
<code>-o <output-dir></code>	Specify the directory to put all generated files into. By default, results are saved in <code><advisor-project>/e<NNN>/pp<MMM>/data.0</code> . If you specify an existing directory or absolute path, results are saved in specified directory. The new directory is created if it does not exist.
<code>--out-dir <output-dir></code>	If you only specify the directory <code><name></code> , results are stored in <code><advisor-project>/e<NNN>/pp<MMM>/<name></code> .
	NOTE If you use this options, you might not be able to open the analysis results in the Intel Advisor GUI.
<code>-p <output-name-prefix></code>	Specify a string to add to the beginning output result filenames.
<code>--out-name-prefix <output-name-prefix></code>	
	NOTE If you use this options, you might not be able to open the analysis results in the Intel Advisor GUI.
<code>--set-parameter <CLI-config></code>	Specify a single-line configuration parameter to modify in a format " <code><group>.<parameter>=<new-value></code> ". For example: <code>"min_required_speed_up=0",</code> <code>"scale.Tiles_per_process=0.5"</code> . You can use this option more than once to modify several parameters.

Option	Description
	<hr/> Important Make sure to use this option for both <code>collect.py</code> and <code>analyze.py</code> with the same value. <hr/>
<code>--track-heap-objects --no-track-heap-objects</code>	Deprecated. Use <code>--track-memory-objects</code> .
<code>--track-memory-objects (default) --no-track-memory-objects</code>	<p>Attribute heap-allocated objects to the analyzed loops that accessed the objects.</p> <hr/> Important This option is always enabled with <code>--collect basic</code> and <code>--collect full</code> . To add the data reuse analysis results to the Offload Modeling report, make sure to use also the <code>--track-memory-objects</code> option for the Performance modeling with <code>analyze.py</code> or <code>advisor --collect=projection</code> . <hr/>
<code>--track-stack-accesses (default) --no-track-stack-accesses</code>	<p>Track accesses to stack memory.</p> <hr/> Important This option is always enabled with <code>--collect basic</code> and <code>--collect full</code> . To add the data reuse analysis results to the Offload Modeling report, make sure to use also the <code>--track-memory-objects</code> option for the Performance modeling with <code>analyze.py</code> or <code>advisor --collect=projection</code> . <hr/>

Examples

- Collect full data on `myApplication` with default configuration and save the project to the `./advi` directory.

```
advisor-python $APM/collect.py ./advi_results -- ./myApplication
```

- Collect refinement data for OpenMP* and SYCL loops on `myApplication` with a custom configuration file `config.toml` and save the project to the `./advi` directory.

```
advisor-python $APM/collect.py ./advi_results --collect refinement --markup [omp,icpx -fsycl] --config ./config.toml -- ./myApplication
```

- Get commands appropriate for a custom configuration specified in the `config.toml` file to collect data separately with `advisor`. The commands are ready to copy and paste.

```
advisor-python $APM/collect.py ./advi_results --dry-run --config ./config.toml
```

analyze.py Options

This script allows you to run an analysis on profiling data and generate report results.

Usage

```
advisor-python <APM>/analyze.py <project-dir> [--options]
```

NOTE Replace <APM> with \$APM on Linux* OS or %APM% on Windows* OS.

Options

The following table describes options that you can use with the `analyze.py` script.

Option	Description
<project-dir>	Required. Specify the path to the Intel® Advisor project directory.
-h	Show all script options.
--help	
--version	Display Intel® Advisor version information.
-v <verbose>	Specify output verbosity level:
--verbose <verbose>	<ul style="list-style-type: none"> 1 - Show only error messages. This is the least verbose level. 2 - Show warning and error messages. 3 (default) - Show information, warning, and error messages. 4 - Show debug, information, warning, and error messages. This is the most verbose level.
<p>NOTE This option affects the console output, but does not affect logs and report results.</p>	
--assume-dependencies (default) --no-assume-dependencies	Assume that a loop has a dependency if the loop type is not known. When disabled, assume that a loop does not have dependencies if the loop dependency type is unknown.
--assume-hide-taxes [<loop-id> <file-name>:<line-number>]	Use an optimistic approach to estimate invocation taxes: hide all invocation taxes except the first one. You can provide a comma-separated list of loop IDs and source locations to hide taxes for. If you do not provide a list, taxes are hidden for all loops.
--assume-never-hide-taxes (default)	Use a pessimistic approach to estimate invocation taxes: do not hide invocation taxes.
--assume-ndim-dependency (default) --no-assume-ndim-dependency	When searching for an optimal N-dimensional offload, assume there are dependencies between inner and outer loops.
--assume-parallel --no-assume-parallel (default)	Assume that a loop is parallel if the loop type is not known.

Option	Description
<code>--assume-single-data-transfer (default) --no-assume-single-data-transfer</code>	<p>Assumed data is transferred once for each offload, and all instances share the data. When disabled, assume each data object is transferred for every instance of an offload that uses it.</p> <p>This method assumes no data re-use between calls to the same kernel.</p> <hr/> <p>Important This option requires you to enable the following options during the Trip Counts collection:</p> <ul style="list-style-type: none"> • With <code>collect.py</code>, use <code>--collect basic</code> or <code>--collect full</code>. • With <code>advisor --collect=tripcounts</code>, use <code>data-transfer=<mode></code>. <hr/>
<code>--atomic-access-pattern <pattern></code>	<p>Select an atomic access pattern. Possible options: <code>sequential</code>, <code>partial_sums_16</code>, <code>same</code>. By default, it is set to <code>partial_sums_16</code>.</p>
<code>--assume-atomic-optimization-ratio <ratio></code>	<p>Model atomic accesses as a number of parallel sums. Specify one of the following values: 8, 16, 32, 64, 128 to model a specific number of parallel sums. Specify 0 value to search for an optimal number of parallel sums.</p> <p>Default value: '16'.</p>
<code>--check-profitability (default) --no-check-profitability</code>	<p>Check the profitability of offloading regions. Only regions that can benefit from the increased speed are added to a report.</p> <p>When disabled, add all evaluated regions to a report, regardless of the profitability of offloading specific regions.</p>
<code>--config <config></code>	<p>Specify a configuration file by absolute path or name. If you choose the latter, the model configuration directory is searched for the file first, then the current directory.</p> <p>The following device configurations are available: <code>xehpg_512xve</code> (default), <code>xehpg_256xve</code>, <code>gen11_icl</code>, <code>gen12_tgl</code>, <code>gen12_dgl</code>, <code>gen9_gt4</code>, <code>gen9_gt3</code>, <code>gen9_gt2</code>.</p> <hr/> <p>NOTE You can specify several configurations by using the option more than once.</p> <hr/>
<code>--count-logical-instructions (default) --no-count-logical-instructions</code>	<p>Use the projection of x86 logical instructions to GPU logical instructions.</p>

Option	Description
<code>--count-memory-instructions (default) --no-count-memory-instructions</code>	Use the projection of x86 instructions with memory to GPU SEND/SENDS instructions.
<code>--count-mov-instructions --no-count-mov-instructions (default)</code>	Use the projection of x86 MOV instructions to GPU MOV instructions.
<code>--count-send-latency {all, first, off}</code>	<p>Select how to model SEND instruction latency.</p> <ul style="list-style-type: none"> <code>all</code> - Assume each SEND instruction has an uncovered latency. This is a <i>default</i> value for GPU-to-GPU modeling with <code>--gpu</code>, <code>--profile-gpu</code>, or <code>--analyze-gpu-kernels-only</code>. <code>first</code> - Assume only the first SEND instruction in a thread has an uncovered latency. This is a <i>default</i> value for CPU-to-GPU modeling. <code>off</code> - Do not model SEND instruction latency.
<code>--cpu-scale-factor <integer></code>	<p>Assume a host CPU that is faster than the original CPU by the specified value.</p> <p>All original CPU times are divided by the scale factor.</p>
<code>--data-reuse-analysis --no-data-reuse-analysis (default)</code>	<p>Estimate data reuse between offloaded regions. Disabling can decrease analyze overhead.</p> <hr/> <p>Important This option requires you to enable the following options during the Trip Counts collection:</p> <ul style="list-style-type: none"> With <code>collect.py</code>, use <code>--collect full</code>. With <code>advisor --collect=tripcounts</code>, use <code>data-transfer=full</code>. <hr/>
<code>--data-transfer-histogram (default) --no-data-transfer-histogram</code>	<p>Estimate fine-grained data transfer and latencies for each object transferred and add a memory object histogram to a report.</p> <hr/> <p>Important This option requires you to enable <code>track-memory-objects</code> or <code>data-transfer=medium</code> or higher (for <code>advisor CLI</code> only) during the Trip Counts collection.</p> <hr/>
<code>--disable-fp64-math-optimization</code>	Disable accounting for optimized traffic for transcendentals on the GPU.
<code>--enable-batching --disable-batching (default)</code>	Enable job batching for top-level offloads. Emulate the execution of more than one instance simultaneously.
<code>--enable-edram</code>	Enable eDRAM modeling in the memory hierarchy model.

Option	Description
	<hr/> NOTE Make sure to use this option with both <code>collect.py</code> and <code>analyze.py</code> . <hr/>
<code>--enable-slm</code>	<p>Enable SLM modeling in the memory hierarchy model.</p> <hr/> NOTE Make sure to use this option with both <code>collect.py</code> and <code>analyze.py</code> . <hr/>
<code>--enforce-baseline-decomposition --no-enforce-baseline-decomposition (default)</code>	<p>Use <i>the same</i> local size and SIMD width as measured on the baseline. When disabled, search for an optimal local size and SIMD width to optimize kernel execution time.</p> <p>Enable the option for the GPU-to-GPU performance modeling.</p>
<code>-e, --enforce-offloads --no-enforce-offloads (default)</code>	<p>Skip the profitability check, disable analyzing child loops and functions, and ensure that the rows marked for offload are offloaded even if offloading child rows is more profitable.</p>
<code>--estimate-max-speedup (default) --no-estimate-max-speedup</code>	<p>Estimate region speedup with relaxed constraints.</p> <hr/> NOTE Disabling can decrease performance model overhead. <hr/>
<code>--evaluate-min-speedup</code>	<p>Enable offload fraction estimation that reaches minimum speedup defined in a configuration file. Disabled by default.</p>
<code>--exclude-from-report <items-to-exclude></code>	<p>Specify items to exclude from a report. Available items: <code>memory_objects</code>, <code>sources</code>, <code>call_graph</code>, <code>dependencies</code>, <code>strides</code>.</p> <p>By default, you can exclude the following items from the report:</p> <ul style="list-style-type: none"> For the CPU-to-GPU modeling: <ul style="list-style-type: none"> memory objects sources call graph dependencies For the GPU-to-GPU modeling: <ul style="list-style-type: none"> memory objects sources <p>Use this option if your report is heavy weight, for example, due to containing a lot of memory objects or sources, which slows down opening in a browser.</p>

Option	Description
	<hr/> <p>NOTE This option affects only data shown in the HTML report and does not affect data collection.</p> <hr/>
<code>--force-32bit-arithmetics</code>	Force all arithmetic operations to be considered single-precision FPs or int32.
<code>--force-64bit-arithmetics</code>	Force all arithmetic operations to be considered double-precision FPs or int64.
<code>--gpu (recommended) --profile-gpu --analyze-gpu-kernels-only</code>	<p>Model performance only for code regions running on a GPU. Use <i>one</i> of the three options.</p> <hr/> <p>Important Make sure to specify this option for both <code>collect.py</code> and <code>analyze.py</code>.</p> <hr/> <p>NOTE This is a <i>preview</i> feature. <code>--analyze-gpu-kernels-only</code> is deprecated and will be removed in future releases.</p> <hr/>
<code>--hide-data-transfer-tax --no-hide-data-transfer-tax (default)</code>	<p>Disable data transfer tax estimation.</p> <p>By default, the data transfer tax estimation is enabled.</p>
<code>--ignore <list></code>	<p>Specify a comma-separated list of runtimes or libraries to ignores time spent in regions from these runtimes and libraries when calculating per-program speedup.</p> <hr/> <p>NOTE This does not affect estimated speedup of individual offloads.</p> <hr/>
<code>--include-to-report <items-to-include></code>	<p>Specify items to include to a report. Available items: <code>memory_objects</code>, <code>sources</code>, <code>call_graph</code>, <code>dependencies</code>, <code>strides</code>.</p> <p>By default, you can add the following items from the report:</p> <ul style="list-style-type: none"> • For the CPU-to-GPU modeling: <code>strides</code> • For the GPU-to-GPU modeling: <ul style="list-style-type: none"> • <code>call graph</code> • <code>dependencies</code> • <code>strides</code>

Option	Description
	<p>Use this option if you want to add more data to the report or see that some data, for example, sources or memory objects, are missing from the report, though you collected this data.</p> <hr/> <p>NOTE This option affects only data shown in the HTML report and does not affect data collection.</p> <hr/>
<code>--loop-filter-threshold <threshold></code>	Specify the loop filter threshold in seconds. The default is 0.02. Loop nests with total time less than the threshold are ignored.
<code>-m <markup></code>	Select <code>markup_analyze</code> , affecting which regions to mark up for data collection and analysis.
<code>--markup <markup></code>	
<code>--model-children (default) --no-model-children</code>	Analyze child loops of the region head to find if some of the loops provide more profitable offload.
<code>--model-extended-math (default) --no-model-extended-math</code>	Model calls to math functions such as <code>EXP</code> , <code>LOG</code> , <code>SIN</code> , and <code>COS</code> as extended math instructions, if possible.
<code>--model-system-calls (default) --no-model-system-calls</code>	Analyze regions with system calls inside. The actual presence of system calls inside a region may reduce model accuracy.
<code>--mpi-rank <mpi-rank></code>	Model performance for the specified MPI rank if multiple ranks were analyzed.
<code>--ndim-depth-limit <N></code>	When searching for an optimal N-dimensional offload, limit the maximum loop depth that can be converted to one offload. The limit must be in the range $1 \leq N \leq 6$. The default value is 3.
<code>--no-cachesim</code>	Disable cache simulation during collection. The model assumes 100% hit rate for cache.
	<hr/> <p>NOTE Usage decreases analysis overhead.</p> <hr/>
<code>--no-stacks</code>	Run data analysis without using callstacks data. You can use this option to avoid bad callstacks attributed data at the expense of accuracy.
<code>--non-accel-time-breakdown</code>	Provide a detailed breakdown of non-offloaded parts of offloaded regions.
<code>-o <output-dir></code>	Specify the directory to put all generated files into. By default, results are saved in <code><advisor-project>/e<NNN>/pp<MMM>/data.0</code> . If you
<code>--out-dir <output-dir></code>	

Option	Description
	<p>specify an existing directory or absolute path, results are saved in specified directory. The new directory is created if it does not exist.</p> <p>If you only specify the directory <code><name></code>, results are stored in <code><advisor-project>/e<NNN>/pp<MMM>/<name></code>.</p> <hr/> <p>NOTE If you use this options, you might not be able to open the analysis results in the Intel Advisor GUI.</p> <hr/>
<p><code>-p <output-name-prefix></code></p> <p><code>--out-name-prefix <output-name-prefix></code></p>	<p>Specify a string to add to the beginning output result filenames.</p> <hr/> <p>NOTE If you use this options, you might not be able to open the analysis results in the Intel Advisor GUI.</p> <hr/>
<p><code>--overlap-taxes --no-overlap-taxes</code> (default)</p>	<p>Enable asynchronous execution to overlap offload overhead with execution time.</p> <p>When disabled, assume no overlap of execution time and offload overhead.</p>
<p><code>--refine-repeated-transfer --no-refine-repeated-transfer</code> (default)</p>	<p>Reduce over-estimation of data transfer when <code>--no-assume-single-data-transfer</code> is used. This option counts how many times each data object is modified and limits the number of data transfers based on that result. For example, constant data may be used in each call to a loop, but needs to be transferred to a device only once.</p> <hr/> <p>Important This option requires you to enable the following options during the Trip Counts collection:</p> <ul style="list-style-type: none"> • With <code>collect.py</code>, use <code>--collect full</code>. • With <code>advisor --collect=tripcounts</code>, use <code>data-transfer=full</code>. <hr/>
<p><code>--search-n-dim</code> (default) <code>--no-search-n-dim</code></p>	<p>Enable search for optimal N-dimensional offload.</p>
<p><code>-l [<file-name>:<line-number>]</code></p> <p><code>--select-loops [<file-name>:<line-number>]</code></p>	<p>Limit the analysis to specified loop nests determined by passing a topmost loop. The parameter must be a comma-separated list of source locations in the following format: <code><file-name>:<line-number></code>.</p>
<p><code>--set-dependency [<IDs/source-locations>]</code></p>	<p>Assume loops have dependencies if they have IDs or source locations from the specified comma-separated list. If the list is empty, assume all loops have dependencies.</p>

Option	Description
	<hr/> <p>NOTE <code>--set-dependency</code> option takes precedence over <code>--set-parallel</code>, so if a loop is listed in both, it is considered as having a dependency.</p> <hr/>
<code>--set-parallel [<IDs/source-locations>]</code>	<p>Assume loops are parallel if they have IDs or source locations from a specified comma-separated list. If the list is empty, assume all loops are parallel.</p> <hr/> <p>NOTE <code>--set-dependency</code> option takes precedence over <code>--set-parallel</code>, so if a loop is listed in both, it is considered as having a dependency.</p> <hr/>
<code>--set-parameter <CLI-config></code>	<p>Specify a single-line configuration parameter to modify in a format "<code><group>.<parameter>=<new-value></code>". For example: <code>"min_required_speed_up=0",</code> <code>"scale.Tiles_per_process=0.5"</code>. You can use this option more than once to modify several parameters.</p> <hr/> <p>Important Make sure to use this option for both <code>collect.py</code> and <code>analyze.py</code> with the same value.</p> <hr/>
<code>--small-node-filter <threshold></code>	<p>Specify the total time threshold, in seconds, to filter out nodes in the <code>program_tree.dot</code> and <code>program_tree.pdf</code> that fall below this value. The default is 0.0.</p>
<code>--threads <number-of-threads></code>	<p>Specify the number of parallel threads to use for offload heads.</p>
<code>--track-heap-objects --no-track-heap-objects</code>	<p>Deprecated. Use <code>--track-memory-objects</code>.</p>
<code>--track-memory-objects (default) --no-track-memory-objects</code>	<p>Attribute heap-allocated objects to the analyzed loops that accessed the objects. Disabling can decrease collection overhead.</p> <hr/> <p>Important This option requires you to enable the following options during the Trip Counts collection:</p> <ul style="list-style-type: none"> • With <code>collect.py</code>, use <code>--collect basic</code> or <code>--collect full</code>. • With <code>advisor --collect=tripcounts</code>, use <code>data-transfer=medium</code> or <code>data-transfer=full</code>. <hr/>

Option	Description
<code>--use-collect-configs --no-use-collect-configs (default)</code>	Use configuration files from collection phase in addition to default and custom configuration files.

Examples

- Run analysis with default configuration on the project in the `./advi` directory. The generated output is saved to the default `advi/perf_models/mNNTNN` directory.

```
adviser-python $APM/analyze.py ./advi_results
```

- Run analysis using the Intel® Iris® Xe MAX graphics (`gen12_dg1` configuration) configuration for the specific loops of the `./advi` project. Add both analyzed loops to the report regardless of their offloading profitability. The generated output is saved to the default `advi/perf_models/mNNTNN` directory.

```
adviser-python $APM/analyze.py ./advi_results --config gen12_dg1 --select-loops
[foo.cpp:34,bar.cpp:192] --no-check-profitability
```

- Run analysis for a custom configuration on the `./advi` project. Mark up regions for analysis and assume a code region is parallel if its type is unknown. Save the generated output to the `advi/perf_models/report` directory.

```
adviser-python $APM/analyze.py ./advi_results --config ./myConfig.toml --markup --assume-parallel --out-dir report
```

Generate Pre-configured Command Lines

You can use Intel® Advisor to generate command lines for perspective or analysis configuration and copy the lines to a clipboard to run from a terminal/command prompt. If you use pre-configured command lines, you do not need to provide analysis configuration, project directory, target application, and application options manually for each analysis you run.

To generate the command lines, you can do one of the following:

- Generate the command lines from Intel Advisor graphical user interface (GUI).
- For the Offload Modeling perspective only: Generate the command lines from Intel Advisor command line interface (CLI).

Generate Command Lines from GUI

Prerequisite: [Set up a project.](#)

- Select a perspective from a **Perspective Selector** or a drop-down menu in the **Analysis Workflow** pane.

The **Analysis Workflow** pane opens for a selected perspective.

- In the **Analysis Workflow** pane, do one of the following:

- To use a pre-defined set of analyses and properties, select an accuracy level.
- Select analyses and options manually from the **Analysis Workflow** tab. To set additional options, go to **File > Project Properties** and set the properties for desired analyses in the respective left-side sections. See [Dialog Box: Project Properties - Analysis Target](#) for option details.

- From the **Analysis Workflow** pane, generate the command lines:

-

For all selected analysis: Click the  **Get Command Line** button.

-

For a specific analysis: Expand the analysis controls and click the  **Command Line** button.

The **Copy Command Line to Clipboard** dialog box opens, which provides commands to launch the perspective or the analysis with selected configuration. Options with default values are hidden by default.

4. Click the **Copy** button to copy the generated command lines to the clipboard.
5. Paste the copied command lines to a terminal/command prompt one by one and run them.

Generate Command Lines from GUI for an MPI Application

Prerequisite: [Set up a project.](#)

1. Go to **File > Project Properties** and select the analysis you want to generate the command line for from the left-side pane. For example, go to **Survey Hotspots Analysis** to generate command line for the Survey analysis.
2. Set properties to configure the analysis, if required. See [Dialog Box: Project Properties - Analysis Target](#) for option details
3. Select the **Use MPI Launcher** checkbox.
4. Specify MPI run parameters:
 - If you select the **Intel MPI** as the MPI launcher: specify the number of ranks to profile, select **Selective** and specify ranks to profile, if required.
 - If you select **Other** as the MPI launcher: specify an MPI launcher executable and the number of ranks to profile. Make sure the **All** is selected under **Profile ranks**.
 - Click **OK** to save the parameters.

Important If you need to generate commands for more than one analysis or for the whole perspective, specify the MPI run parameters for *each* analysis you want to run. Otherwise, Intel Advisor uses default parameters for all analyses you do not configure properly.

5. Open the **Analysis Workflow** pane and select a perspective to run.
6. Select an accuracy level to preconfigure the perspective or configure it manually using the checkboxes. Make sure to select only analyses that you have specified the MPI run parameters for.

Important Do not change the accuracy level if you specified additional parameters in the **Project Properties**. Preconfigured accuracy levels reset your configuration.

7. Generate the command lines:

-

For all selected analysis: Click the  **Get Command Line** button.

-

For a specific analysis: Expand the analysis controls and click the  **Command Line** button.

The **Copy Command Line to Clipboard** dialog box opens, which provides commands to launch the perspective or the analysis with selected configuration. Options with default values are hidden by default.

8. In the **Copy Command Line to Clipboard** dialog box, select the **Generate command line for MPI** checkbox to apply the MPI syntax you configured.
9. Click the **Copy** button to copy the generated command lines to the clipboard.
10. Paste the copied command lines to a terminal/command prompt one by one and run them.

Generate Command Lines for Offload Modeling from CLI

You can generate pre-configured command lines to analyze your application with Offload Modeling. Use this feature if you want to:

- Analyze an MPI application

- Customize preset Offload Modeling commands

Note: In the commands below, make sure to replace the `myApplication` with your application executable path and name *before* executing a command. If your application requires additional command line options, add them *after* the executable name.

Prerequisite: [Set up environment variables](#) to enable Intel Advisor CLI.

1. Generate pre-configured command lines using the `--dry-run` option of the `--collect=projection` analysis or `collect.py` script. Specify accuracy level and paths to your project directory and application executable. For example, to generate commands for the default medium accuracy with `--collect=projection`:

```
advisor --collect=offload --dry-run --project-dir=./advi_results -- ./myApplication
```

If you want to scale parameters, for example, to change the number of GPU tiles per MPI process, generate the commands using the `collect.py` script as follows:

```
advisor-python $APM/collect.py ./advi_results --set-parameter <parameter-to-modify> --dry-run -- ./myApplication
```

NOTE Families of Intel® X^e graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® X^e Graphics](#).

After you run the command, a list of analysis commands to run the Offload Modeling with the specified accuracy level is printed to the terminal/command prompt.

2. **If you analyze an MPI application:** Copy the generated commands to your preferred text editor and modify each command to use an MPI launcher: add `mpirun` or `mpiexec` and a number of MPI processes to launch to each command. For details about the syntax, see [Analyze MPI Applications](#).
3. Run the generated commands one by one from a command prompt or a terminal. You can add more options or modify the commands to better fit your goal. See [Command Line Option Reference](#) for available options.
4. If you generated commands with the script: Run Performance Modeling step with the scaled parameter:

```
advisor --collect=projection --project-dir=./advi_results --set-parameter <parameter-to-modify>
```

For details about MPI application analysis with Offload Modeling, see [Model MPI Application Performance on GPU](#).

See Also

[Command Line Interface](#) This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

[Analyze MPI Applications](#) With Intel® Advisor, you can analyze parallel tasks running on a cluster to examine performance of your MPI application.

Troubleshooting

You may encounter unexpected situations while using Intel® Advisor. This reference section describes symptoms and possible correction strategies for several such situations.

Issue	Details
Error Message: Application Sets Its Own Handler for Signal	When you run Survey or Suitability analysis, the following error message appears: <i>Application sets its own handler for signal <conflicting_signal> that is used for internal needs of the tool. Collection cannot continue.</i>
Error Message: Cannot Collect GPU Hardware Metrics for the Selected GPU Adapter	When you run GPU Roofline, the following error message appears: <i>Cannot collect GPU hardware metrics for the selected GPU adapter.</i>
Error Message: Memory Model Cache Hierarchy Incompatible	<i>Error: Memory model cache hierarchy incompatible</i> message appears when you run performance modeling step of the Offload Modeling perspective with <code>analyze.py</code> on the results collected with a previous release of the Intel® Advisor.
Error Message: No Annotations Found	<i>No annotations detected in your project sources</i> message appears when you open the Summary or Annotation Report after running the Suitability analysis of the Threading perspective.
Error Message: No Data Is Collected	When you open a report, the following error message appears: <i>Error 0x40000024 (No data) – No data is collected.</i>
Error Message: Stack Size Is Too Small	When you run a Survey or a Suitability analysis, the following error message appears: <i>Stack size provided to sigaltstack is too small. Please increase the stack size to 64K minimum.</i>
Error Message: Undefined Linker References to <code>dlopen</code> or <code>dlsym</code>	When linking your application program on Linux* OS, you see linker (ld) messages such as <i>undefined reference to 'dlopen'</i> or <i>undefined reference to 'dlsym'</i> .
Problem: Broken Call Tree	After running the Offload Modeling perspective, in the Accelerated Regions tab, you see unexpected issues with code regions displayed or measured data reported, such as incorrect number of trip counts or code region type.
Problem: Code Region is not Marked up	After running the Offload Modeling perspective, you see that a code region of interest is not analyzed and has <i>Outside of Marked Region</i> message in the Details pane of the Accelerated Regions tab.
Problem: Debug Information Not Available	After you run any Intel Advisor analysis, the displayed report contains information about your target's <i>debug (symbol) information</i> that is unexpected or does not make sense.
Problem: No Data	A <i>No Data</i> message appears when you open a report.
Problem: Source Not Available	After you run the Intel Advisor analysis, the displayed report may contain information about your target's source code that is unexpected or does not make sense.

Issue	Details
Problem: Stack in the Top-Down Tree Window Is Incorrect	After you run any Intel Advisor analysis, the Top-Down view in the displayed shows application stack that is unexpected or does not make sense.
Problem: Survey Analysis does not Display Report	After you run the Survey analysis in any Intel Advisor perspective, a message appears instead of the report saying that your target runs too quickly or that the target does not contain debug symbol information.
Problem: Unexpected C/C++ Compilation Errors After Adding Annotations	After adding Intel Advisor annotations, you see unexpected compiler messages when building your C/C++ target executable.
Problem: Unexpected Unmatched Annotations in the Dependencies Report	After running Intel Advisor Dependencies analysis, you see unmatched problems reported caused by unmatched annotations execution that you did not expect.
Warning: Analysis of Debug Build	A message appears when you start the Survey or Suitability analysis and the current application is built in a Debug mode.
Warning: Analysis of Release Build	A message appears when you start the Dependencies analysis and the current application is built in a Release mode.

Error Message: Application Sets Its Own Handler for Signal

Symptoms

When you run a Survey or a Suitability analysis, the following error message appears: *Application sets its own handler for signal <conflicting_signal> that is used for internal needs of the tool. Collection cannot continue.*

NOTE This message is for Linux* OS only.

Cause

Intel® Advisor cannot profile applications that set up a signal handler for a signal used by the tool.

Possible Solution

NOTE This solution is applicable only to the Survey and Suitability analyses.

Do one of the following:

- When collecting data with `advisor` command line interface (CLI), pass the `--run-pass-thru=--profiling-signal=<not_used_signal>` option, where `<not_used_signal>` is the signal that should not be used by your application. You need to select the signal from `SIGRTMIN..SIGRTMAX`. For example, for the matrix multiply (`mmult`) application:

```
advisor --collect=survey --run-pass-thru=--profiling-signal=35 -- ./mmult
```

- Before collecting data from the Intel Advisor GUI or CLI, set the environment variable `ADVIXE_RUNTOOL_OPTIONS=-- profiling-signal=<not_used_signal>`.

Command Line Interface This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

Error Message: Cannot Collect GPU Hardware Metrics for the Selected GPU Adapter

Symptoms

When you run GPU Roofline, the following error message appears: *Cannot collect GPU hardware metrics for the selected GPU adapter.*

Cause

To collect GPU hardware metrics and GPU utilization data on Linux* OS or Windows* OS with a driver versions higher than 27.20.100.8280, Intel® Advisor uses the Intel® Metric Discovery API library that is delivered with the product. This error message displays if Intel Advisor cannot access the selected GPU adapter.

Possible Solution

- If you run Intel Advisor from the command line: Make sure that you have correctly set the target GPU with the `--target-gpu=<address>`. The `<address>` should be in the following format: `<domain>:<bus>:<device>.<function>`. The list of GPU adapters available on your system is available in the option description in the Intel Advisor CLI help.

For example, to run the Survey analysis for the GPU adapter 0:0:2:0:

```
advisor --collect=survey --project-dir=./advi_results --profile-gpu --target-gpu=0:0:2:0 -- ./myApplication
```

- For Windows systems, update the driver for the selected GPU adapters.
- For Linux systems, install the Intel Metric Discovery API library 1.6.0 or higher to support the selection of video adapters. To collect metrics from the video card of your choice, disable other adapters in the BIOS first.

Command Line Interface This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

GPU Roofline Perspective Measure and visualize the actual performance of GPU kernels using benchmarks and hardware metric profiling against hardware-imposed performance ceilings, as well as determine the main limiting factor, by running the GPU Roofline Insights perspective.

Error Message: Memory Model Cache Hierarchy Incompatible

Symptoms

You see the *Error: Memory model cache hierarchy incompatible* error message when you run performance modeling with `analyze.py` on the results collected with a previous release of the Intel® Advisor.

Cause

The cache configuration file from a previous release is incompatible with higher versions of the Intel® Advisor.

Possible Solution

Delete the `perf_models` directory from the results and re-run `analyze.py`.

Run Offload Modeling Perspective Find high-impact opportunities to offload/run your code and identify potential performance bottlenecks on a target graphics processing unit (GPU) by running the Offload Modeling perspective.

Error Message: No Annotations Found

Symptoms

- A message appears when you click the **Annotation Report** button and you have not yet added Intel® Advisor annotations to your program, or you have not yet run the Suitability and Dependencies tools after adding the annotations.
- When using the Intel® Advisor GUI while viewing the **Summary** window, you see a message **No annotations detected in your project sources** when your sources do not contain annotations.

Cause

The Suitability and Dependencies tools use the annotations you added to your program to analyze your running program and populate the **Annotation Report** window with data. However, before you run these tools to collect data about your running program, you need to add annotations and perform related actions.

When using the Intel® Advisor GUI, make sure that the appropriate project properties have been specified so the Intel® Advisor tools can find the correct source location(s).

Also, if your sources contain huge source files that contain annotations, be aware that only the first 8 MB of each file will be parsed for annotations (for performance reasons). This could possibly cause mismatched or no annotations found messages.

Possible Solution

- Do the following:
 1. Use the Survey analysis to find where your program spends its time. Choose at least one possible parallel code region (site) and identify code that might execute independently as a task.
 2. Use the code editor to add at least one pair of parallel site annotations that contain task annotation(s) into your program. You can copy annotation code using the bottom of the **Survey Report** or **Survey Source** windows.
 3. Make sure that these annotations are executed by the selected project or the selected startup project (Windows* OS).
 4. Make sure that you reference the annotations definitions file in the source modules where you added the annotations.
 5. Reference the annotations definition directory and provide other build settings.
 6. If you are using the Intel® Advisor GUI, check the **Project Properties** dialog to make sure that source locations are specified in the **Source Search** tab.
 7. If your sources include huge source files that contain annotations (more than 8 MB per file), consider breaking each huge source file into several source files.
- **Windows OS only:** If you selected the wrong startup project, select and build the correct startup project, run either the Suitability or Dependencies tool, and click the **Annotation Report** button.

Tip

For the most current information on optimal C/C++ and Fortran build settings, see [Build Your Target Application](#)[Build Your Target Application](#).

See Also

[No Data](#)

[Source Search Tab](#)

Copying Annotations and Build Settings Using the Annotation Assistant Pane
Annotating Code for Deeper Analysis
Intel Advisor Annotations Definition File
Annotations

Error Message: No Data Is Collected

Symptoms

When you open a report, the following error message appears: *Error 0x40000024 (No data) – No data is collected.*

Another possible message: *Error 0x4000002a (Database interface error) -- Cannot run data transformation 'Add Fake Loop Data'.*

Cause

The possible causes are:

- The data collection period is too short (less than 500 ms on CPU or GPU), and the Intel® Advisor cannot capture performance data since it uses time sampling to collect CPU or GPU time.
- The application crashed during an Intel Advisor analysis.
- The application could not find required shared libraries, for example, an OpenMP* runtime, if compiler environment was not sourced.

Possible Solution

Do one of the following:

- Verify that you can run your application without the Intel Advisor.
 1. Open a new console window. You can keep the console window where you launch the Intel Advisor.
 2. Run the application from a new console window with the same environment set up.
 3. If you see an error message reporting problem with loading shared libraries, set up the application environment. For example, do the following:
 - Set the `LD_LIBRARY_PATH` variable.
 - Source an Intel® Compiler environment.
 4. Once the application runs successfully, start the Intel Advisor console window with the same environment set up.
- If the data collection duration is too short, do one of the following:
 - Increase the workload for your application.
 - [Decrease sampling interval](#) to 1ms.

Command Line Interface This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

Error Message: Stack Size Is Too Small

Symptoms

When you run a Survey or a Suitability analysis, the following error message appears: *Stack size provided to sigaltstack is too small. Please increase the stack size to 64K minimum.*

NOTE This message is for Linux* OS only.

Cause

When setting up a `SIGPROF` signal handler, the Intel® Advisor configures the signals to use the alternative stack size with the `sigaltstack()` API. This verifies that its signal handler does not depend on the profiled application stack size. If the application uses an alternative signal stack, the Intel Advisor requires it to be not less than 64 KB.

However, if the application uses the `SIGSTKSZ` constant for the alternative stack size (which is 8192 bytes), the data collection may terminate with the error message.

Possible Solution

NOTE This solution is applicable only to the Survey and Suitability analyses.

Configure the Intel Advisor so that it does not set up the alternative stack and uses the stack provided by the application. Do one of the following:

- When collecting data with advisor command line interface (CLI), pass the `--run-pass-thru=--no-altstack` option to the tool. For example, for the `mmult` application:

```
advisor --collect=survey --run-pass-thru=--no-altstack -- ./mmult
```

- Before collecting data from the Intel Advisor GUI or CLI, set the environment variable `ADVISE_RUNTOOL_OPTIONS=--no-altstack`.

Command Line Interface This reference section describes the Intel® Advisor command line interface (CLI) used to run the analysis.

Error Message: Undefined Linker References to `dlopen` or `dlsym`

Symptoms

When linking your application program on Linux* OS, you see linker (`ld`) messages such as:

- undefined reference to ``dlopen'`
- undefined reference to ``dlsym'`

Cause

Intel® Advisor uses dynamic loading. After you add the `#include` (C/C++) line to include the Intel® Advisor annotation definition file, you must specify the linker option `-ldl` to enable dynamic loading.

NOTE In most cases, you do not need source annotations when using Intel® Advisor, except for the Suitability analysis of the [Threading](#) perspective. When analyzing your application with other perspectives, such as [Vectorization and Code Insights](#) or [Offload Modeling](#), you can analyze all parts of your code automatically or use Intel Advisor [mark-up capabilities](#), which do not require you to recompile your application.

Possible Solution

- Do the following:
 1. Add the linker option `-ldl` to your command line, script, or make file.
 2. Review the options listed in the [Build Your Target Application](#) to ensure that you specified all required compiler and linker options (use the link below under See Also). If omitted, add missing options to your command line, script, or make file.

3. Rebuild your program.

See Also

[Unexpected C/C++ Compilation Errors After Adding Annotations](#)

[Intel Advisor Annotations Definition File](#)

Problem: Broken Call Tree

Symptoms

After executing the Offload Modeling perspective, in the **Accelerated Regions** tab, you see one of the following:

- A code region is duplicated.
- A code region is located at a wrong place.
- A code region has incorrect number of trip counts reported in any column of the Trip Counts column group.
- A code region with your code has a *System Modulediagnostics* message and *Cannot be modeled: System Modulareason* for not offloading.

Any of these symptoms mean that the Intel® Advisor detected the application call tree incorrectly during Survey.

Details

A broken call tree often happens if you use a program model with SYCL or Intel® oneAPI Threading Building Blocks. These program models run code in many threads using a complicated scheduler, and the Intel Advisor sometimes cannot correctly detect their call stacks. As a result, some code instances might have no metrics or incorrect metrics in a report and a call tree is broken.

Cause

This can happen due to the following reasons:

- Call stacks were detected incorrectly.
- A heavy optimization was used.
- Debug information has issues.

Possible Solution

NOTE

This is not an issue if all hotspots and code you are interested in are outside of the broken part of the call tree. You can ignore it in this case.

To fix a broken call tree, do the following:

- Make sure you compiled binary with `-g` option.

NOTE

You can recompile it with the `-debug inline-debug-info` option to get enhanced debug information.

- Recompile the binary with a lower optimization level: use `-O2`.
- **If you collect performance metrics with `advisor` CLI:** When running the Survey analysis, try the following:

- Remove `--stackwalk-mode=online` option if you used it when running the Survey analysis.
- Add `--no-stack-stitching` option.
- Offload only specific code regions if their estimated execution time on a target device is greater than or equal to the original execution time. Rerun the performance modeling with `--select-loops` to specify loops of interest and `--enforce-offloads` to make sure all of them are offloaded. For example:

```
advisor-python <APM>/analyze.py <project-dir> --select-loops=[<file-name1>:<line-number1>,<file-name1>:<line-number2>,<file-name2>:<line-number3>] --enforce-offloads
```

NOTE Replace `<APM>` with `$APM` on Linux* OS or `%APM%` on Windows* OS.

For details, see [Enforce Offloading for Specific Loops](#)

- If you model a multithreaded code that runs with a complicated scheduler, you might see a code region with suspiciously low trip counts and multiple instances of the same region loop present in the scheduler. This means that the Offload Modeling could not correctly detect the call stacks. Use the `--enable-batching` option to artificially increase the number of trip counts by using total number of executions instead of average number trip counts.

See Also

[advisor Command Option Reference](#)

[analyze.py Options](#)

[Command Line Interface Reference](#) This reference section describes the CLI actions and options used in the command syntax: `advisor <--action> [--action-options] [--global-options] [--] target [target options]`.

Problem: Code Region is not Marked Up

Symptoms

A code region of interest is not analyzed and has *Outside of Marked Region* why-not-offloaded message in the Details pane of the **Accelerated Regions** tab after you execute the Offload Modeling perspective.

Details

To limit the scope of collections, the Intel® Advisor selects loops that match certain criteria and marks them up for analysis. By default, the Intel Advisor performs a smart region selection using the *generic* markup.

If a code region does not satisfy the markup criteria, you should see the *Outside of Marked Region* why-not-offloaded message or the *System Module* diagnostics message in the Details pane of the **Accelerated Regions** tab.

Cause

Your code region does not satisfy one or more markup rules for a specified markup mode. If you use the default generic mark-up strategy, make sure your loop of interest satisfies the following rules:

- It is not a system module or a system function.
- It has instruction mixes.
- It is executed.
- Its execution time is not less than 0.02 seconds, which is a sampling interval of the Intel Advisor. For more information about execution time limitations, see [Total Time is Too Small for Reliable Modeling](#).

Possible Solution

If a code region does not satisfy the generic markup rules, but you want to analyze it, do one of the following:

- You can change the markup strategy by using a `--markup=<markup_mode>` option of `analyze.py` or `--select markup=<markup-mode>` for `--collect=performance`. The following parameters select only loops inside regions that are already parallel:
 - `generic` or `gpu_generic` (default) - Select loops executed on a GPU.
 - `omp` - Select loops only in OpenMP parallel regions.
 - `icpx -fsycl` - Select loops only in SYCL parallel regions.
 - `daal` - Select loops only in Intel® oneAPI Data Analytics Library parallel regions.
 - `tbb` - Select loops only in Intel® oneAPI Threading Building Blocks parallel regions.

NOTE

`omp`, `icpx -fsycl`, and `generic/gpu_generic` select loops in the project so you can run another collection or performance modeling without markup or loop selection options.

- If your loops of interest are not marked up because they have no static instruction mixes or not executed, you can limit the analysis to these specific loops by using the `--select-loops` option with the `analyze.py` script. With this option, only the loops specified are analyzed. For example:

```
advisor-python <APM>/analyze.py <project-dir> --select-loops=[<file-name1>:<line-number1>,<file-name1>:<line-number2>,<file-name2>:<line-number3>]
```

NOTE Replace `<APM>` with `$APM` on Linux* OS or `%APM%` on Windows* OS.

With `--collect=performance`, use `--select` option to select specific loops to analyze by source location, ID, or other criteria.

See Also





[analyze.py Options](#)

Problem: Debug Information Not Available**Symptoms**

After running the Intel® Advisor analysis, the displayed report may contain information about your target's debug (symbol) information that is unexpected or does not make sense.

Details

When debug information is not available, the ability to use binary-to-source correlation prevents the display of source code. One or more of the following might occur only for the calls into third-party library routine code for which library sources are not available to the project:

- After running the Survey tool, a message may appear near the top of the **Survey Report** window indicating **Some target modules do not contain debug information**. After viewing the message and writing down module names that lack debug information, you can click the red ✖ in the top-right corner to close it.
- When viewing a **Report** window, a column that should contain a **Source** location or a **function name** instead contains the target's *executable-name* in square brackets, `[Unknown]`, a question mark `?`, or is blank. Also, a broken or missing icon can indicate that sources are not available, such as , , , or  for a Survey loop.
- When viewing a **Source** window, a column that should contain source code or a source location instead contains **source is not available** or **Intel Advisor cannot show source code for this location** message instead of the expected data. Also, a broken or missing icon can indicate that sources are not available (listed above).

- When viewing a **Source** window, the **Call Stack** indicates that sources are not available for the starting (top) line, such as containing a broken icon (listed above).
- In a context menu, the items **View Source** or **Edit Source** may appear as dimmed.

However, if your application calls library functions, you should expect to see some symptoms about sources not being available. For example, the C/C++ sample application `stats` calls certain library functions to calculate standard deviation or similar values. Because the source code for the library functions is not available, you will see that source code is not available within the called library functions, but is available for the related project source files. In this case, see the help topic [Sources Not Available](#).

Cause

The most common causes are:

- The build option(s) did not request debug information when building the target executable. Debug information must be present for Intel® Advisor tools to display source information. When building native code targets, specify the appropriate compiler and linker options to ensure the target executable contains debug information.
- The appropriate project properties in the Intel® Advisor GUI did not provide the correct binary/symbol or source locations.

Possible Solution

- Do the following:
 1. Check the build settings for the target to ensure they specify debug information option(s).
 2. Adjust your makefile, Microsoft Visual Studio* project properties, or build script to specify debug information option(s).
 3. Rebuild the target.
 4. Run the Intel® Advisor analysis tool(s) again.
- If you are using the Intel® Advisor GUI, check the **Project Properties** dialog to make sure that:
 - Binary and symbol files are specified in the **Binary/Symbol Search** tab.
 - Source locations are specified in the **Source Search** tab.
- Investigate other possible causes, such as the compiler not generating debug information for a source line or the source file, the linker not including debug information in the debug information database, or whether the debug information database was not being found during the finalization step by an Intel® Advisor analysis tool. For example, the last issue can occur if the debug information database was not moved to the location with the target executable.

Tip

For the most current information on optimal C/C++ and Fortran build settings, see [Build Your Target Application](#).

[Binary/Symbol Search Tab](#)

[Source Search Tab](#)

[No Annotations Found](#)

[Sources Not Available](#)

Problem: No Data

Symptoms

A **No Data** message appears when you click the **Survey Report**, **Suitability Report**, or the **Dependencies Report** button and you have not yet run these tools for the currently selected project or startup project (Windows* OS).

This message also appears if Intel® Advisor annotations were not executed by the Suitability or Dependencies tools. When using the Intel Advisor GUI, this message may appear if the Suitability or Dependencies tools could not find the source files using the specified project properties.

To help you add annotations to your sources, the **No Data** message is accompanied by the annotation assistant pane.

Cause

After you run the Suitability or Dependencies tools, the data collected populates the corresponding **Suitability Report** and **Dependencies Report** window, and the list of annotations displayed in the **Annotation Report** window is updated.

- To use the Suitability or Dependencies tools, your project/startup project must execute Intel Advisor parallel site and task annotations.
- When using the Intel Advisor GUI, this message appears when the specified project properties do not provide a correct path to the source location(s).
- This message can appear with the Survey tool if your target executes quickly and you clicked the **Started Paused** button (or equivalent option or Pause Collection annotation) in the side command toolbar. That is, you paused data collection so that data collection did not start until after the target's execution completed, but the target executes too quickly for the Survey tool to analyze.

Possible Solution

- Use the **Advisor Workflow** tab to guide you through the steps needed to run these tools, which analyze your running program.
- Windows* OS only: If you selected the wrong startup project, select and build the correct startup project and run the tool again.
- If your project/startup project did not execute Intel Advisor annotations, make sure you have added parallel site and task annotations to your program and that they get executed when you run the startup project (Windows* OS) / target executable (Linux* OS).
- When using the Intel Advisor GUI, open the project and then open **Project Properties** dialog box. After checking that the path and target file name of the **Application** is correct in the **Analysis Target** tab, click the **Source Search** tab. Insert one or more new rows to specify the path to the source location(s). In this case, you do not need to rebuild your application.
- If your target executes quickly and you clicked the **Started Paused** button (or used the equivalent option or Pause Collection annotation) in the side command toolbar, click **Collect Survey Data** instead. Otherwise, the Survey tool cannot analyze your target because it executes too quickly.

Problem: Source Not Available

Symptoms





After running the Intel® Advisor tools to analyze your running application's target, the displayed report may contain information about your target's source code that is unexpected or does not make sense.

Details

NOTE

When you see that source code is not available, be aware that improper build settings can also cause this symptom. Missing debug information symbols disable the binary-to-source correlation that allows the normal display of source code. If the source code for annotations is not displayed in **Report** and **Source** windows, this may indicate missing debug information symbols, as explained in the help topic **Troubleshooting Debug Information Not Available**.

One or more of the following might occur only for the calls into third-party library routine code for which library sources are not available to the project:

- When viewing a **Report** window, a column that should contain either a source location or a function name instead contains a question mark (?), [Unknown], the target's *executable-name* (rather than a source file name), is blank, or contains a broken icon, such as , , , or  for a Survey loop.
- When viewing a **Source** window, you may see a message **Intel Advisor cannot show source code for this location** instead of the collected data. Also the **Call Stack** pane or a **Function** column may contain ?, [Unknown], or a broken icon (listed above). For example, if the top (starting) function line in a Call Stack pane contains source information, but calls to library routines lower in the calls stack do not contain source information.
- When using the Intel Advisor GUI while viewing the **Summary** window, you see a message **No annotations detected in your project sources** when your sources do contain Intel Advisor annotations.

If your application's target calls libraries for which sources are not found by the project, you may see some symptoms about source code not being available. For example, the C/C++ sample application *stats* calls certain library routines to calculate standard deviation or similar values. Because the library's source code for those functions is not available, the symptoms like a broken icon or the message **Intel Advisor cannot show source code for this location** are expected.

Cause

The most common causes are:

- The target's execution paths called library functions for which sources are not available.
- The cause may be that debug information symbols are not available for the main part of the program and annotated parallel sites and their tasks.

Possible Solution

- Do the following:
 1. If source and debug information symbols are available for the annotated parallel sites and their tasks, but not for calls into library code for which sources are not available, this is expected and is not a problem. If you were considering adding annotations to the library code, instead add site/task annotations to the code that calls the library routines.
 2. If source and debug information symbols are not available for the main part of the program and annotated parallel sites and their tasks, see the help topic [Troubleshooting Debug Information Not Available](#).

Tip

For the most current information on optimal C/C++ and Fortran build settings, see [Build Your Target Application](#).

See Also

[Survey Tool does not Display Survey Report](#)
[Debug Information Not Available](#)

Problem: Stack in the Top-Down Tree Window Is Incorrect

Symptoms

After you run any Intel® Advisor analysis, the Top-Down view in a displayed report shows application stack that is unexpected or does not make sense.

Cause

The target was built with an optimization level that removed stack information from the binary.

Possible Solution

Do one of the following:

- Change the stackwalk mode from offline (after collection) to online (during collection):
 - From the Intel Advisor GUI: Go to **Project Properties > Analysis Target > Survey Hotspots Analysis > Advanced > Stack unwinding mode and select** and select **During collection** drop-down option.
 - From the command line interface (CLI): Use the `--stackwalk-mode=online` option. For example:

```
advisor --collect=survey --project-dir=./myAdvisorProj --stackwalk-mode=online -- ./bin/
myTargetApplication
```

This option increases analysis overhead.

- Decrease the optimization level of your project and rebuild the target. For example, with the Intel® C++ Compiler Classic or the Intel® oneAPI DPC++/C++ Compiler, use the following options:
 - Request moderate optimization:
 - On Linux* OS and macOS*: use `O2` or lower
 - On Windows* OS: use `/O2` or lower
 - Disable compiler inlining:
 - On Linux OS and macOS: use `-fno-inline`
 - On Windows OS: use `/Ob0`
 - Disable interprocedural optimization:
 - On Linux OS and macOS: use `-no-ipo -no-ip`
 - On Windows OS: use `/Qipo- /Qip-`

Consider also using the following options:

- Select the maximum level of debug information:
 - On Linux OS and macOS: use `-g[n]`
 - On Windows OS: use `Zi, Z7, ZI`

To generate additional debug information in the object file, use `-g3` or `ZI`.

- Set a maximum number of times to unroll loops:
 - On Linux OS and macOS: use `-unroll[=n]`
 - On Windows OS: use `/Qunroll[:n]`

To tell the compiler not to unroll loops, use `-unroll=0` or `/Qunroll:0`.

Build Target Application

Problem: Survey Tool does not Display Survey Report

Symptoms

After you run the Survey tool, a message appears instead of the **Survey Report**. The message indicates that the specified target runs too quickly for the Survey tool to analyze or that the target does not contain debug symbol information.

Cause

After you run the Intel® Advisor Survey tool, if the specified target runs too quickly for the Survey tool to analyze using the sampling interval, there is insufficient data collected to provide a complete and meaningful **Survey Report**. So a message appears instead.

This message can also appear under certain conditions when the target was built without the required Debug symbol information.

Possible Solution

If the cause is that the specified target runs too quickly:

- Modify the target to increase its workload or data set so it executes longer, rebuild, and run the Survey tool again.
- Specify a different target that executes longer, build, and run the Survey tool again.
- When using the Intel® Advisor GUI, if you cannot increase the target's workload or the data set to increase its execution time, consider specifying the **Project Properties** in the **Analysis Target** dialog box **Advanced** fields to slightly reduce the **Sampling interval** for this target. However, if you reduce the sampling interval so sampling occurs too often, this can cause the sampling activity itself to be measured (noise), reducing the accuracy of the measurements. The default Sampling interval works for most targets, but can be modified for specific targets (see Advanced options in the help topic Dialog Box: Project Properties - Analysis Target).
- Windows* OS only: If you selected the wrong Visual Studio startup project, select and build the correct startup project and run the tool again.

If you suspect that the target executable does not contain debug symbol information, please check your target's build settings and compare them with the recommended Build Settings for your language. Source- or target-related information may be missing from the Intel Advisor tools reports if the target executable does not contain debug symbol information.

Tip

For the most current information on optimal C/C++ and Fortran build settings, see [Build Your Target Application](#).

Problem: Unexpected C/C++ Compilation Errors After Adding Annotations

Symptoms

After adding Intel® Advisor annotations, you see unexpected compiler messages when building your C/C++ target executable.

Details

After you add the annotations and the `#include` line to include the Intel® Advisor annotation definition file, you see unexpected C/C++ compiler messages.

Cause

Possible causes:

- Type and debug symbol conflicts.
- `windows.h` file issues.

Possible Solution

Do the following:

1. Read the help topics starting with Tips for Annotation Use with C/C++ Programs to help you decide how to modify your sources, such as Handling Compilation Issues that Appear After Adding advisor-annotate.h.
2. Modify sources.
3. Rebuild your target.

Problem: Unexpected Unmatched Annotations in the Dependencies Report

Symptoms

When running Intel® Advisor Dependencies tool analysis, you see unmatched problems reported that are caused by unmatched annotations execution that you did not expect.

Details

The **Dependencies Report** window lists the problems and messages reported by Dependencies tool analysis in the **Problems and Messages** pane. You may see some unexpected problems related to unmatched annotations, such as the following problem types: Dangling Lock, Missing Begin Site, Missing Begin Task, Missing End Site, Missing End Task, or Orphaned Task. For example, within a parallel site, there is no annotation to mark the end of the parallel site for all possible the code execution paths.

Cause

Possible causes:

- You placed annotations inside macros.
- For Linux* OS: You placed parallel sites and their related annotations outside the project.
- For Windows* OS: You placed parallel sites and their related annotations outside a set of projects that the startup project depends on.
- Your sources contain huge source files that contain annotations. As only the first 8 MB of each file are parsed for annotations (for performance reasons), this could possibly cause mismatched annotations messages.

Possible Solution

Do the following:

1. Use the Dependencies tool to view the code region(s) causing the problem. Investigate whether some sites or tasks may have multiple exit points and whether end annotations cover all exit points. For example, code that returns or branches around an end annotation, or throws an exception. If you suspect the problem is caused by adding annotations inside macros, remove the annotations from the macros and add them to the final location in the sources - similar to the way breakpoints do not work in macros. Rebuild your target and run the Dependencies tool again.
2. **Windows* OS only:** Use the Dependencies tool to investigate by viewing the code region(s) causing the problem. If you suspect that parallel sites and their related annotations that are placed outside the set of projects that the startup project depends on, consider using the Visual Studio* Project Dependencies context menu item to add appropriate dependencies to cause Intel® Advisor to scan sources in the additional project(s). Rebuild and run the Dependencies tool again.

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

3. If your sources include huge source files that contain annotations (more than 8 MB per file), consider breaking each huge source file into several source files.

See Also

[Annotating Code for Deeper Analysis](#)

[Annotations](#)

[Annotation General Characteristics](#)

Warning: Analysis of Debug Build

Symptoms

A message appears when you start the Survey or Suitability tools and the current build options selected for the project is a Debug build.

Details

When measuring performance, using a version of the program for analysis that is close to the version that will be provided to customers provides the most accurate data. If you will provide a Release build for customers, run the Survey or Suitability tools with a Release build. You can run this tool with a Debug build if you will run it again with a Release build. To produce the best results, your build settings should specify debug information and moderate optimization.

When a tool is waiting for your input (click either **Continue** or **Cancel**), the result name has a **[!]** prefix. If you do not respond within several minutes, the tool implicitly chooses the Cancel button.

Cause

For the Survey or Suitability tools, you should use a Release build of your program, not a Debug build.

Possible Solution

- Before you respond to the message, change to Release build settings and build the target executable. When it completes, click **Continue** to run the analysis.
- Click **Continue** and ignore this message. Later, run this tool again with a target built using a Release build.
- Click **Cancel**. Change your build settings to use a Release build and build the target executable. Then run the Intel® Advisor tool.

Tip

For the most current information on optimal C/C++ and Fortran build settings, see [Build Your Target Application](#).

Warning: Analysis of Release Build

Symptoms

A message appears when you start the Dependencies tool and the current build options selected for the project is a Release build.

Details

To produce the best results, your build settings for the Dependencies tool should specify debug information and no optimization. If possible, use a minimal data set for the Dependencies tool.

When a tool is waiting for your input (click either **Continue** or **Cancel**), the result name has a **[!]** prefix. If you do not respond within several minutes, the tool implicitly chooses the Cancel button.

Cause

For the Dependencies tool, you should use a Debug build of your program, not a Release build.

Possible Solution

- Before you respond to the message, change to use Debug build settings and build the target executable. When it completes, click **Continue** to run the analysis.
- Click **Continue** and ignore this message. Later, run this tool again with a target built from Debug build settings.
- Click **Cancel**. Change to use Debug build settings and build the target executable. Then run the Dependencies tool.

Tip

For the most current information on optimal C/C++ and Fortran build settings, see [Build Your Target Application](#).

See Also

[Debug Information Not Available](#)
[No Annotations Found](#)

Reference

Data Reference

The following sections describe the contents of data columns in reports for each perspective.

- [CPU and Memory Metrics](#) - review the metrics reported in the CPU / Memory Roofline Insights, Vectorization and Code Insights, and Threading perspectives.
- [Accelerator Metrics](#) - review the metrics reported in the Offload Modeling and GPU Roofline Insights perspectives.

CPU Metrics

*This reference section describes the contents of data columns in **Survey** and **Refinement Reports** of the Vectorization and Code Insights, CPU / Memory Roofline Insights, and Threading perspectives.*

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [XYZ](#)

A

- [Access Pattern](#)
- [Access Type](#)
- [Address Range](#)
- [Average](#)

Access Pattern

Description: Summary of access types.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

Access Type

Description: Memory access type: Read, Write, Read/Write

Collected during [Memory Access Patterns Analysis](#) and **found** in **Memory Access Patterns Report**.

Address Range

Description: Instruction address range in memory.

Interpretation: A wide range indicates one or more of the following:

- The application uses too much memory.
- Memory usage is not optimal.

Average

Description: Loop trip count average.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in Loop Information Pane (Survey Report).

Prerequisites for collection/display: Enabled **Collect Trip Counts** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about Loop Trip Counts** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

B

C

- [Cache Line Utilization](#)
- [Cache Misses](#)
- [Call Count](#)
- [Compiler Estimated Gain](#)

Cache Line Utilization

Description: Simulated cache line utilization for data transfer operations.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

Cache Misses

Description: Number of memory load operations served by memory subsystem higher than cache. Calculated for the first instance of the loop (assuming *cold* CPU cache). Value is a result of virtual cache modeling, which might not match exact counter reported by hardware for this analysis run.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

Call Count

Description: Number of times loop/function was invoked.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Prerequisites for collection/display: Enabled **Collect Trip Counts** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about Loop Trip Counts** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

Interpretation: A high number means there is an outer loop in the selected loop call chain with high trip count values. If the loop has a low trip count value, the outer loop could be a better candidate for parallelization (threading/vectorization).

Compiler Estimated Gain

Description: Theoretical compiler estimate of relative loop performance speedup achieved or achievable due to vectorization.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Comparison with similar metrics: **Gain Estimate** is Intel Advisor-calculated estimate of relative loop performance speedup achieved due to vectorization.

D

- [Data Types](#)
- [Description](#)
- [Dirty Evictions](#)

Data Types

Description: Data types provided by binary static analysis.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Interpretation: Bold indicates primary data type used for vectorization.

Description

Description: Code location classification.

Collected during [Dependencies Analysis](#) and **found** in Dependencies Report.

Dirty Evictions

Description: Number of evicted cache lines with a modified state introducing upstream memory traffic to a higher memory subsystem.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

E

- [Efficiency](#)
- [Elapsed Time](#)

Efficiency

Description: Intel Advisor-calculated performance estimated gain compared to maximum achievable gain from vectorization.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report).

Interpretation: Normally means how effectively vectorization was applied, compared to maximum possible gain (higher is better).

Calculation/Aggregation: (Estimated gain/Vector length) * 100%

Interpretation: Hover mouse over data cell for more information.

Elapsed Time

Description: Elapsed (wall-clock) application time.

Collected during [Survey Analysis](#), and **found** in Filters banner.

F

- [First Instance Site Footprint](#)
- [Function](#)
- [Function Call Sites and Loops](#)

First Instance Site Footprint

Description: For each memory access instruction for the first instance of a loop, the Intel Advisor:

- Tracks the minimum and maximum access addresses.
- Displays the maximum range in this metric.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

Comparison with similar metrics: This metric is more reliable than the **Maximum Per-Instruction Address Range** metric.

	Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memory Footprint
Number of threads analyzed for loop/site	1	1	1
Number of loop instances analyzed	All instances, but with some memory access instruction filtering	1	Depends on loop call count limit: <ul style="list-style-type: none"> • GUI: Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Loop call count limit • CLI action option: -loop-call-count limit
Awareness of overlap between address ranges accessed in loop	No	Yes	Yes
Suitability for code with random memory access	No	No	Yes

Function

Description: Function name.






Collected during [Dependencies Analysis](#) and **found in** Dependencies Report.

Function Call Sites and Loops

Description: Information about parent function, source file, and line where site/loop begins in **Loop Information Pane (Survey Report)**, and top-down call tree of target functions and loops in **Loop Information Pane (Survey Report)**

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Interpretation:

-  - Scalar function.
-  - Vectorized function.
-  - Scalar loop. Vectorization might be possible.
-  - Vectorized loop. Optimization might be possible.
-  - Scalar inner loop within vectorized outer loop. Optimization might be possible.

G

- [Gain Estimate](#)

Gain Estimate

Description: Intel Advisor-calculated estimate of relative loop performance speedup achieved due to vectorization.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report).

Comparison with similar metrics: **Compiler Estimated Gain** is the theoretical compiler estimate of relative loop performance speedup achieved or achievable due to vectorization.

H

I

- [Instruction Address](#)
- [Instruction Sets](#)
- [Iteration Duration](#)

Instruction Address

Description: Instruction address in memory.

Collected during [Dependencies Analysis](#) and **found** in Dependencies Report.

Instruction Sets

Description: Instruction Set Architecture (ISA) usage for individual instructions.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Iteration Duration

Description: Average loop iteration time.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Prerequisites for collection/display: Enabled **Collect Trip Counts** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about Loop Trip Counts** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

J

K

L

- [Loop Instance Total Time](#)
- [Loop-Carried Dependencies](#)

Loop Instance Total Time

Description: Average loop instance total time.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display: Enabled **Collect Trip Counts** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about Loop Trip Counts** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

Loop-Carried Dependencies

Description: Dependencies summary across iterations

Collected during [Dependencies Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

Possible values:

- **RAW** (Read after Write) - Flow dependency
- **WAR** (Write after Read) - Anti dependency
- **WAW** (Write after Write) - Output dependency

M

- [Max](#)
- [Max Site Footprint](#)
- [Maximum Per-Instruction Address Range](#)
- [Memory Access Footprint](#)
- [Memory Loads](#)
- [Memory Stores](#)
- [Memory, GB](#)
- [Min](#)
- [Module/Modules](#)
- [Multi-Pumping Factor](#)

Max

Description: Loop trip count maximum.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display: Enabled **Collect Trip Counts** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about Loop Trip Counts** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

Max Site Footprint

Description: Maximum distance (among all instances of the loop) between the minimum and maximum memory address values.

Maximum Per-Instruction Address Range

Description: For most memory access instructions for all instances of a loop, the Intel Advisor:

- Tracks the minimum and maximum access addresses.
- Displays the maximum range in this metric.

The value may be imprecise because the Intel Advisor filters some memory access instructions while analyzing all instances of a loop. Unreliable values are displayed in gray.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)** and Memory Access Patterns Report.

Comparison with similar metrics: This metric is less reliable than the **First Instance Site Footprint** metric.

	Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memory Footprint
Number of threads analyzed for loop/site	1	1	1
Number of loop instances analyzed	All instances, but with some memory access instruction filtering	1	Depends on loop call count limit: <ul style="list-style-type: none"> • GUI: Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Loop call count limit • CLI action option: -loop-call-count limit
Awareness of overlap between address ranges accessed in loop	No	Yes	Yes
Suitability for code with random memory access	No	No	Yes

Memory Access Footprint

Description: Maximum distance (among all instances of the loop) between minimum and maximum memory address values, accessed by the instructions, generated from the current source line.

Memory Loads

Description: Number of memory load operations in first instance of the loop.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**].

Memory Stores

Description: Number of memory store operations in first instance of the loop.

Collected during [Memory Access Patterns Analysis](#)] and **found** in **Loop Information Pane (Refinement Reports)**.

Memory, GB

Description: Number of data transfers, in GB, between the CPU and memory subsystem.

Important

This is a core metric that is the basis of the arithmetic intensity (AI) calculation.

Min

Description: Loop trip count minimum.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display: Enabled **Collect Trip Counts** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about Loop Trip Counts** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

Module/Modules

Description: Executable or library name.

Collected during [Survey Analysis](#), [Dependencies Analysis](#), and [Memory Access Patterns Analysis](#); and **found** in **Loop Information Pane (Survey Report)**, **Advanced View Pane (Survey Report)**, **Dependencies Report**, and **Memory Access Patterns Report**.

Multi-Pumping Factor

Description: The number of times the compiler applied a pumping optimization to extend vector length.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

N

- [Nested Function](#)

Nested Function

Description: Name of the function (invoked from the site) where the stride diagnostic was detected.

Collected during [Memory Access Patterns Analysis](#) and **found** in **Memory Access Patterns Report**.

O

- [Optimization Details](#)

Optimization Details

Description: Compiler optimization details.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

P

- [Performance Issues](#)

- [Problem Severity](#)

Performance Issues

Description: Performance issues found.

Collected during [Survey Analysis](#), and [Memory Access Patterns Analysis](#), and **found** in **Loop Information Pane (Survey Report)** and **Memory Access Patterns Analysis**.




Interpretation: Click to display confidence level about issue root cause and recommended fixes.

Problem Severity

Description: Seriousness of a detected problem.

Collected for during [Dependencies Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

Possible values:

-  - Error.
-  - Warning.
-  - Informational.

Q

R

- [RFO Cache Misses](#)

RFO Cache Misses

Description: Number of cache lines loaded to cache due to a modification request (Request for Ownership).

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

S

- [Self AI](#)
- [Self Elapsed Time](#)
- [Self GFLOP](#)
- [Self GFLOPS](#)
- [Self Giga OP](#)
- [Self Giga OPS](#)
- [Self GINTOP](#)
- [Self GINTOPS](#)
- [Self INT AI](#)
- [Self Memory \(GB\)](#)
- [Self Memory \(GB/s\)](#)
- [Self Overall AI](#)
- [Self Time](#)
- [Simulated Memory Footprint](#)
- [Site Location](#)
- [Site Name](#)
- [Source/Source Location/Sources](#)
- [State](#)
- [Stride](#)

- [Strides Distribution](#)

Self AI

Description: Ratio of **Self GFLOPS** to self L1 transferred bytes.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).
- Selected **Show Floating-Point Operation Columns** for column setting.

Instruction types counted for FLOP calculation:

- FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND

Self Elapsed Time

Description: **Self Time**-based wall time from beginning to end of loop/function execution, excluding time for callees.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Comparison with similar metrics: **Total Elapsed Time** is **Total Time**-based wall time from beginning to end of loop/function execution, including time for callees.

Interpretation: Same as **Self Time** for single-threaded applications

Self GFLOP

Description: Giga floating-point operations, excluding GFLOP for callees.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).
- Selected **Show Floating-Point Operation Columns** for column setting.

Instruction types counted for FLOP calculation:

- FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND

Self GFLOPS

Description: Ratio of **Self GLOP** to **Self Elapsed Time**.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

- Selected **Show Floating-Point Operation Columns** for column setting.

Instruction types counted for FLOP calculation:

- FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND

Self Giga OP

Description: Giga floating-point operations plus giga integer operations, excluding giga floating-point and integer operations for callees.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).
- Selected **Show Sum of Integer and Floating-Point Operation Columns** for column setting.

Instruction types counted for FLOP calculation:

- FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND

Instruction types counted for INTOP calculation (default):

- ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates

Self Giga OPS

Description: Ratio of **Self GFLOP** plus **Self GINTOP** to **Self Elapsed Time**.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).
- Selected **Show Sum of Integer and Floating-Point Operation Columns** for column setting.

Instruction types counted for FLOP calculation:

- FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND

Instruction types counted for INTOP calculation (default):

- ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates

Self GINTOP

Description: Giga integer operations, excluding giga integer operations for callees.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

- Selected **Show Integer Operation Columns** for column setting.

Instruction types counted for INTOP calculation (default):

- ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates

Self GINTOPS

Description: Ratio of **Self GINTOP** to **Self Elapsed Time**.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).
- Selected **Show Integer Operation Columns** for column setting.

Instruction types counted for INTOP calculation (default):

- ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates

Self INT AI

Description: Ratio of **Self GINTOPS** to self L1 transferred bytes.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).
- Selected **Show Integer Operation Columns** for column setting.

Instruction types counted for INTOP calculation (default):

- ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates

Self Memory (GB)

Description: Data transfers between CPU and memory subsystem (total traffic, including caches and DRAM) in gigabytes, excluding transfers for callees.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display: Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

Self Memory (GB/s)

Description: Data transfers between CPU and memory subsystem (total traffic, including caches and DRAM) in gigabytes per second, excluding transfers for callees.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display: Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).

Calculation/Aggregation: Self GBs / Self Elapsed Time

Self Overall AI

Description: Ratio of **Self GFLOPS** plus **Self GINTOPS** to self L1 transferred bytes.

Collected during [Trip Counts Analysis](#) (Characterization), and **found** in **Loop Information Pane** (Survey Report).

Prerequisites for collection/display:

- Enabled **Collect FLOP** option of the **Characterization** step on Analysis Workflow tab or enabled **Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage** on **Trip Counts and FLOP Analysis** tab of [Project Properties Dialog Box](#).
- Selected **Show Sum of Integer and Floating-Point Operation Columns** for column setting.

Instruction types counted for FLOP calculation:

- FMA, ADD, SUB, DIV, DP, MUL, ATAN, FPREM, TAN, SIN, COS, SQRT, SUB, RCP, RSQRT, EXP, VSCALE, MAX, MIN, ABS, IMUL, IDIV, FIDIVR, CMP, VREDUCE, VRND

Instruction types counted for INTOP calculation (default):

- ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shifts, rotates

Self Time

Description: Time actively executing a function/loop, excluding time for callees.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Comparison with similar metrics: **Total Time** is time actively executing a function/loop, including time for callees.

Simulated Memory Footprint

Description: The summarized and overlap-aware memory footprint across all instances of a loop.

Collected during [Memory Access Patterns Analysis](#) and found in **Loop Information Pane (Refinement Reports)**.

Prerequisites for collection/display:

In the GUI [Project Properties Dialog Box](#):

- Enable **Enable CPU cache simulation**.
- In the **Cache simulation mode** drop-down list, choose **Model cache misses and loop footprint**.
- Tweak other **Enable CPU cache simulation** parameters as necessary.

CLI example:

```
advisor -collect map
-mark-up-list=1,2,7,17,26
-enable-cache-simulation
-cachesim-mode=footprint
-project-dir C:\my_advisor_project
-- my_application.exe
```

Comparison with similar metrics:

	Max. Per-Instruction Addr. Range	First Instance Site Footprint	Simulated Memory Footprint
Number of threads analyzed for loop/site	1	1	1
Number of loop instances analyzed	All instances, but with some memory access instruction filtering	1	Depends on loop call count limit: <ul style="list-style-type: none"> GUI: Project Properties > Analysis Target > Memory Access Patterns Analysis > Advanced > Loop call count limit CLI action option: -loop-call-count limit
Awareness of overlap between address ranges accessed in loop	No	Yes	Yes
Suitability for code with random memory access	No	No	Yes

Calculation/Aggregation: Number of unique cache lines accessed during cache simulation * Cache line size.

For performance reasons, not all accesses and cache lines are simulated. Instead the Intel Advisor tracks a subset and then scales up to the whole cache size to determine the final footprint value.

Site Location

Description: Information about parent function, source file, and line where site/loop begins.

Collected during [Dependencies Analysis](#) and [Memory Access Patterns Analysis](#), and **found** in **Loop Information Pane (Refinement Reports)**.

Site Name

Description: Site name if using source annotations; sequence ID if marking loops for deeper analysis in **Survey Report**.

Collected during [Dependencies Analysis](#) and [Memory Access Patterns Analysis](#), and **found** in **Loop Information Pane (Refinement Reports)**, **Dependencies Report**, and **Memory Access Patterns Report**.

Source/Source Location/Sources

Description: Source file name(s) and line number(s).

Collected during [Survey Analysis](#), [Dependencies Analysis](#) and [Memory Access Patterns Analysis](#); and **found** in **Loop Information Pane (Survey Report)**, **Advanced View Pane (Survey Report)**, **Dependencies Report**, and **Memory Access Report**.

State

Description: State of most severe problem in problem set.

Collected during [Dependencies Analysis](#) and **found** in **Dependencies Report**.

Possible values:

- **Regression]** - Not investigated. Set by the Intel Advisor.

Issue requires more investigation because it was marked as **Fixed** in baseline result but still appears.

- **New]** - Not investigated. Set by the Intel Advisor or user.

Issue did not appear in the baseline result, or there is no older result from which the Intel Advisor can propagate state information.

- **Not Fixed]** - Not investigated. Set by user.

Issue appeared in the baseline result and still requires investigation.

- **Confirmed]** - Investigated. Set by user.

Issue requires fixing but has not yet been fixed.

- **Fixed]** - Investigated. Set by user.

Issue requires fixing and has been fixed.

- **Not a problem]** - Investigated. Set by user.

Issue does not require fixing.

- **Deferred]** - Investigated. Set by user.

You are postponing further investigation on an issue that may or may not require fixing.

Stride

Description: Distance, in elements, between memory accesses in two consequent iterations.

Collected during [Memory Access Patterns Analysis](#) and found in Memory Access Patterns Report.

Strides Distribution

Description: Stride ratio in following format: Unit%/Constant%/Variable%

Collected during [Memory Access Patterns Analysis](#) and **found** in **Loop Information Pane (Refinement Reports)**.

T

- [Total Elapsed Time](#)
- [Total Time](#)
- [Traits](#)
- [Transformations](#)
- [Type](#)

Total Elapsed Time

Description: **Total Time**-based wall time from beginning to end of loop/function execution, including time for callees

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Comparison with similar metrics: **Self Elapsed Time** is **Self Time**-based wall time from beginning to end of loop/function execution, excluding time for callees.

Interpretation: Same as **Total Time** for single-threaded applications.

Total Time

Description: Time actively executing a function/loop, including time for callees.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Comparison with similar metrics: **Self Time** is time actively executing a function/loop, not including time for callees.

Traits

Description: Scalar and vectorization characteristics that may impact performance.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Possible values:

Trait	Detected ASM Instructions
Divisions	*DIV*
Square Roots	*SQRT*
Type Conversions	*CVT*
NT-stores	*MOVNT*
Gathers	*GATHER*
Scatters	*SCATTER*
Shuffles	*SHUF*
Permutates	*PERM*
Blends	*BLEND*
Packs	*PACK*
Unpacks	*UNPCK*
Inserts	*INSERT*
Extracts	*EXTRACT*
Masked Stores	*MASKMOV*
Shifts	*PROR*, *PROL*, *PSLL*, *PSRA*, *PSRL*
FMA	*FMADD*, *FMSUB*, *FNMADD*, *FNMSUB*
Mask Manipulations	*KADD*, *KTEST*, *KAND*, *KOR*, *KXOR*, *KXNOR*, *KNOT*, *KUNPCK*, *KMOV*, *KSHIFT*
Conflict Detections	*VPCONFLICT*
Exponent extractions	*VGETEXP*
Mantissa extractions	*VGETMANT*
Expands	*EXPAND*
Compresses	*COMPRESS*
VNNI	*VNNI*

Transformations

Description: Loop transformations applied by compiler.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).






Type

Collected during [Survey Analysis](#), [Dependencies Analysis](#), and [Memory Access Patterns Analysis](#); and **found** in **Loop Information Pane (Survey Report)**, **Advanced View Pane (Survey Report)**, [Dependencies Report](#), and [Memory Access Patterns Report](#).

Possible Survey Report values:

- **Peeled/Remainder** - For loops that have child loops. Appears only when scalar peeled loop and/or remainder loop executed.
- **Threaded** - For loops that have child loops. Appears when some parallel framework (OpenMP* or automatically by Intel compiler) is used in the loop.
- **Vectorized (<loop part(s)>)** - For vectorized parent and child loops. Appears when a parent loop has *any* of the following parts executed: peeled, body, remainder. Also appears for child loops that have *one* of the following parts executed: peeled, body, remainder.
- **Peeled** - For small, (usually) compiler-generated loops created to align the memory accesses inside the loop body and maximize its efficiency.
- **Body** - For vectorized loops (compiler-generated from a source loop). Most loop iterations should execute in body, as body normally processes more data than peeled or remainder loops. Vector length in the body is usually larger than in peeled and/or remainder loops, which means body is the most efficient place for performance.
- **Remainder** - For (usually) compiler-generated loops created to clean up any remaining iterations that do not fit within the scope of the loop body.
- **[Not Executed]** - Mark that appears next to any other loop metric when a loop was not executed.
- **Scalar** - Appears when non-vectorized loops executed.
- **Completely Unrolled** - Appears when the loop body was copied several times (equal to trip counts value) by the compiler.
- **Inside vectorized** - Appears when the inner loop was vectorized in addition to the outer loop.
- **Inlined Function** - Appears when the function body was inlined into the loop/function body.
- **Vector Function** - Appears when a SIMD-enabled version of the function executed. (See Intel compiler documentation for details).
- **Function** - Appears when a scalar version of the function executed.

Possible Memory Access Patterns Report values:

-  **Uniform stride 0** - Instruction accesses the same memory from iteration to iteration.
Represents the ideal situation and does not require any improvements.
-  **Unit stride (stride 1)** - Instruction accesses memory that consistently changes by one element from iteration to iteration.
Represents the ideal situation and does not require any improvements.
-  **Constant stride (stride N)** - Instruction accesses memory that consistently changes by N elements (N>1) from iteration to iteration.
Code uses more memory than is ideal and requires more cache lines. Consider studying recommendations on AOS/SOA optimization.
-  **Irregular stride** - Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration.
Might limit vectorization or even make vectorization impossible.
-  **Gather (irregular) stride** - Detected for v(p)gather* instructions on AVX2 Instruction Set Architecture (ISA).
The compiler vectorized code with an irregular memory access pattern. Consider improving the code to use a more constant memory access pattern.

Possible Dependencies Report values - See [Problem and Message Types](#).

U

- [Unroll Factor](#)

Unroll Factor

Description: Loop unroll factor applied by the compiler.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

V

- [Variable References](#)
- [Vector ISA](#)
- [Vector Widths](#)
- [Vectorization Details](#)
- [VL \(Vector Length\)](#)

Variable References

Description: Name of the variable for which the dependency or memory access stride is detected.

Collected during [Dependencies Analysis](#) and [Memory Access Patterns Analysis](#), and **found** in Dependencies Report and Memory Access Patterns Report.

Vector ISA

Description: The highest vector Instruction Set Architecture used for individual instructions.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Comparison with similar metrics: An ISA higher than the ISA of your current hardware appears when you add corresponding codepaths with `x`, `Qx` / `ax`, `Qax` compiler options. To see the ISA of non-executed codepaths, enable the **Analyze non-executed codepaths** option in **Project Properties**.

Vector Widths

Description: Vector register width in bits.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Possible values: Combination of values, including 32, 64, 128, 256, 512, delimited by a slash or semi-colon (/ or ;).

Vectorization Details

Description: Compiler notes on vectorization.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

VL (Vector Length)

Description: The number of elements processed in a single iteration of vector loops, or the number of elements processed in individual vector instructions.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Calculation/Aggregation: Estimated by binary static analysis or the Intel compiler.

W

- [Why No Vectorization?](#)

Why No Vectorization?

Description: The reason the compiler did not vectorize the loop.

Collected during [Survey Analysis](#) and **found** in **Loop Information Pane** (Survey Report) and **Advanced View Pane** (Survey Report).

Interpretation: Click to display the issue root cause and recommended fixes.

X, Y, Z

Accelerator Metrics

This reference section describes the contents of data columns in reports of the Offload Modeling and GPU Roofline Insights perspectives.

| [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [XYZ](#)

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

#

- [2 FPU's Active](#)

2 FPU's Active

Description: Average percentage of time when both floating-point units (FPU) are used.

Collected during the **Survey** analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **EU Instructions** column group.

A

- [Active](#)
- [Advanced Diagnostics](#)
- [Allocation Time](#)
- [Atomic Accesses](#)
- [Atomic Throughput](#)
- [Atomic Throughput per Cycle](#)
- [Average Time \(GPU Roofline\)](#)
- [Average Time \(Offload Modeling\)](#)
- [Average Trip Count](#)

Active

Description: Percentage of cycles actively executing instructions on all execution units (EU) or vector engines (XVE).

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **EU Array** column group or **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Array** column group (for code running on the Intel® Arc™ graphics code-named Alchemist or newer).

Advanced Diagnostics

Description: Additional information about a code region that might help to understand the achieved performance.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane.

Allocation Time

Description: Total time spent on memory allocation.

Collected during the Characterization analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Prerequisites for display: Expand the **Data Transferred** column group.

Atomic Accesses

Description: Total number of atomic memory accesses.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Atomics** column group.

Atomic Throughput

Description: Average atomic throughput for a kernel, in operations per seconds.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Atomics** column group.

Prerequisites for display: Expand the **Atomics** column group.

Atomic Throughput per Cycle

Description: Average atomic throughput for a kernel, in operations per cycle.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Atomics** column group.

Prerequisites for display: Expand the **Atomics** column group.

Average Time (GPU Roofline)

Description: Average time spent executing one task instance.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Kernel Details** column group.

Prerequisites for display: Expand the **Kernel Details** column.

Average Time (Offload Modeling)

Description: Average time spent executing one task instance. This metric is only available for the GPU-to-GPU modeling.

Collected during the Survey analysis with enabled GPU profiling in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column.

Average Trip Count

Description: Average number of times a loop/function is executed.

Collected during the Trip Counts (Characterization) in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

B

- [Bandwidth, GB/s \(GPU Memory\)](#)
- [Bandwidth, GB/s \(L3 Shader\)](#)
- [Bandwidth, GB/s \(SLM\)](#)
- [Baseline Device](#)
- [Bounded By](#)

Bandwidth, GB/s (GPU Memory)

Description: Rate at which data is transferred to and from GPU, chip uncore (LLC), and main memory, in gigabytes.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Memory** column group.

Prerequisite for display: Expand the **GPU Memory** column. This metric is also shown in the collapsed **GPU Memory** column.

Bandwidth, GB/s (L3 Shader)

Description: Rate at which data is transferred between execution units or vector engines and L3 caches, in gigabytes per second.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **L3 Shader** column group.

Prerequisite for display: Expand the **L3 Shader** column. This metric is also shown in the collapsed **L3 Shader** column.

Bandwidth, GB/s (SLM)

Description: Rate at which data is transferred to and from shared local memory (SLM), in gigabytes per second.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** report > **GPU Kernels** pane > **SLM** column group.

Prerequisites for display: Expand the **SLM** column. This metric is also shown in the collapsed **SLM** column.

Baseline Device

Description: Host platform that application is executed on.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisite for display: Expand the **Measured** column group.

Bounded By

Description: List of main factors that limit the estimated performance of a code region offloaded to a target device.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Basic Estimated Metrics** column group.

Interpretation: This metric shows one or more bottleneck(s) in a code region.

Category	Bottleneck	Description
Algorithmic	Dependencies	Data dependencies limit the parallel execution efficiency. Fix the dependencies to offload this code region.
	Kernel Decomposition	The workload decomposition strategy does not allow to schedule enough parallel threads to use all execution units or vector engines on a selected target device.
	Trip Counts	The number of loop iterations is not enough to use all execution units or vector engines on a selected target device.
Taxes	Data Transfer	Data transfer tax is greater than the <i>sum</i> of the maximum throughput time and latencies time.
	Launch Tax	Kernel launch tax is greater than the <i>sum</i> of the maximum throughput time and latencies time.
Throughput	Compute	The code region uses full target device capabilities, but the compute time is still high. The time is greater than all other execution time components on a target device.
	Global Atomics	Global atomics bandwidth time is greater than all other execution time components on a target device.

Category	Bottleneck	Description
	Memory Sub-System bandwidth (BW): for example, L3 BW, LLC BW, DRAM BW	Memory sub-system bandwidth time is greater than all other execution time components on a target device.
Latencies	Latencies	Instruction latency is greater than the maximum throughput time.

Resulting estimated time is calculated as a sum of the four factors: throughput, latency, and taxes, which include data transfer taxes and submission tax:

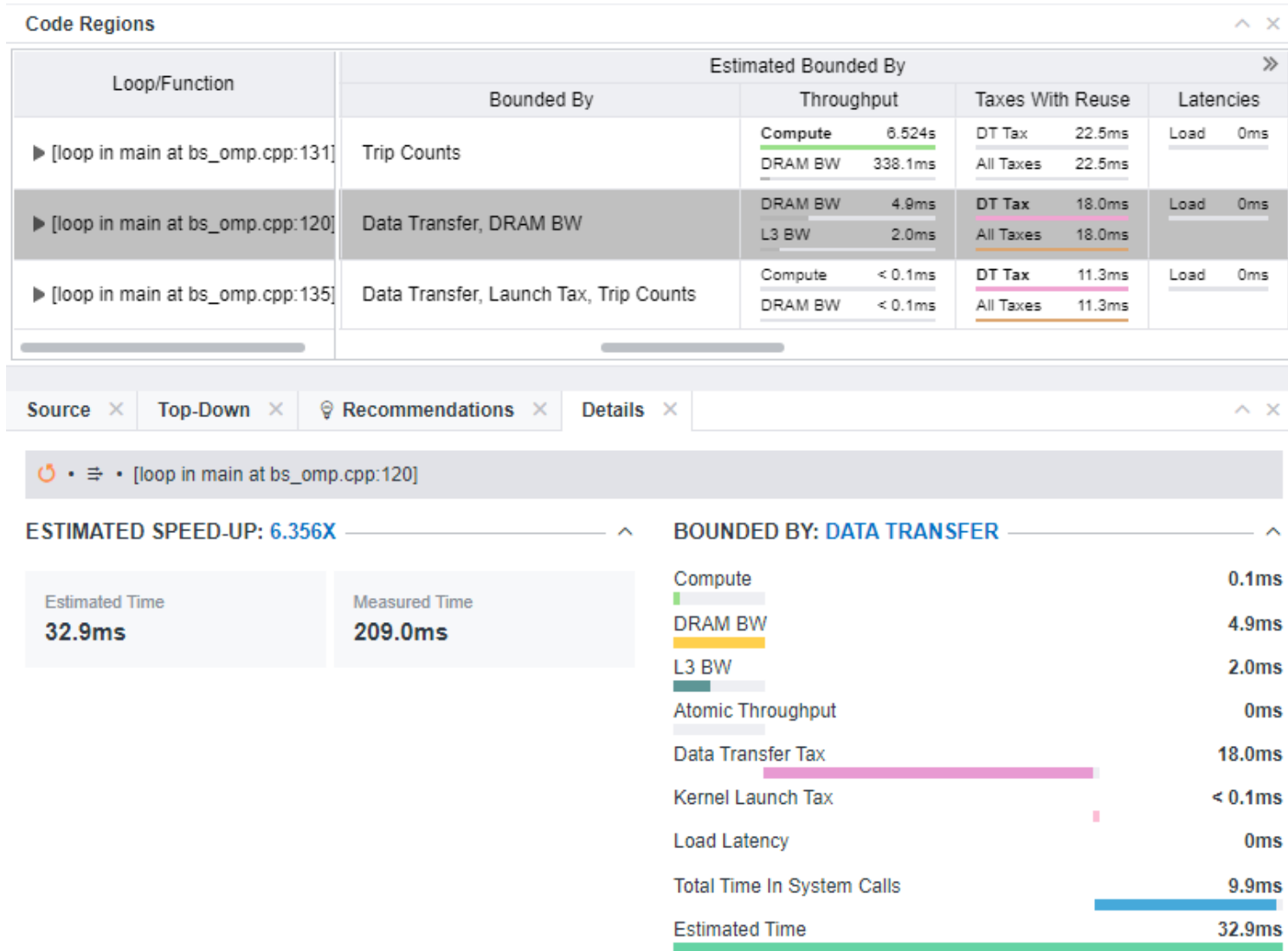
$$\text{Time} = \text{max_throughput_bottleneck_time} + \text{non_overlapped_latency} + \text{data_transfer_time} + \text{kernel_submission_taxes_time}$$

The model assumes that throughput-defined times are fully "overlapped" and chooses only a "maximum" throughput bottleneck to show in the column. If the impact of other components is comparable to the throughput component, top bottlenecks of *all* four factors (one for throughput, one for latency, and one for data transfer/submission) are shown in this column. This means the code region is limited by this combination of factors, which is ordered by the *impact* on the region performance.

Otherwise, for example, if the relative throughput impact is much higher than the latency and data transfer ones, *only* the maximum throughput bottleneck is shown as dominating over others. If the maximum *throughput time* is compute, Intel Advisor assumes the algorithmic factors (dependencies, kernel decomposition, trip counts) limit offloading a code region.

For example, the combined **Data Transfer, DRAM BW** value means the following:

- The main limiting factor for the code region is *data transfer tax*. The tax is greater than the sum of the maximum throughput time and latencies time for this region.
- The second limiting factor for the code region is the *DRAM bandwidth time*. The time is greater than other execution time components on a target device.



C

- [Cache Line Utilization](#)
- [Call Count](#)
- [CARM, GB](#)
- [Compute](#)
- [Computing Threads Started](#)

Cache Line Utilization

Description: Fraction of global memory traffic used by execution units or vector engines.

Collected during the Survey analysis with GPU profiling enabled in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **L3 Shader** column group.

Prerequisites for display: Expand the **L3 Shader** column group. This metric is also shown in the collapsed **L3 Shader** column.

Calculation: Ratio of global memory traffic to the observed cache traffic, where:

- Global memory traffic is traffic between execution units or vector engines and cache data ports, in cache-line granularity transactions.
- Observed cache traffic is traffic between a data port and caches, in cache-line granularity transactions.

Interpretation: If you see a low value, it may indicate that the kernel has an inefficient or not GPU-friendly memory access pattern.

Call Count

Description: Number of times a loop/function was invoked.

Collected during the Trip Counts (Characterization) in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

CARM, GB

Description: Total data transferred to and from execution units or vector engines, in gigabytes..

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Compute

Description: Estimated execution time assuming an offloaded loop is bound only by compute throughput.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Estimated Bounded By** column group in the **Accelerated Regions** tab > **Code Regions** pane.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

Computing Threads Started

Description: Total number of threads started across all execution units or vector engines for a computing task.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

D

- [Data Transfer Tax](#)
- [Data Transfer Tax without Reuse](#)
- [Data Reuse Gain](#)
- [Dependency Type](#)
- [Device-to-Host Size](#)
- [Device-to-Host Time](#)
- [DRAM](#)
- [DRAM BW \(Estimated Bounded By\)](#)
- [DRAM BW \(Memory Estimations\)](#)
- [DRAM BW Utilization](#)
- [DRAM Read Traffic](#)
- [DRAM Traffic](#)
- [DRAM Write Traffic](#)

Data Transfer Tax

Description: Estimated time cost, in milliseconds, for transferring loop data between host and target platform. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data is reused between code regions on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

Data Transfer Tax without Reuse

Description: Estimated time cost, in milliseconds, for transferring loop data between host and target platforms considering data is not reused. This metric is available only if you enabled the data reuse analysis for the Performance Modeling.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** under **Characterization** to **Full** and enable the **Data Reuse Analysis** checkbox under **Performance Modeling**.
- CLI: Use the `--data-transfer=full` action option with the `--collect=tripcounts` action and the `--data-reuse-analysis` option with the `--collect=tripcounts` and `--collect=projection` actions.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

Data Reuse Gain

Description: Difference between data transfer time estimated with data reuse and without data reuse, in milliseconds. This option is available only if you enabled the data reuse analysis for the Performance Modeling.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** under **Characterization** to **Full** and enable the **Data Reuse Analysis** checkbox under **Performance Modeling**.
- CLI: Use the `--data-transfer=full` action option with the `--collect=tripcounts` action and the `--data-reuse-analysis` option with the `--collect=tripcounts` and `--collect=projection` actions.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

Dependency Type

Description: Dependency absence or presence in a loop across iterations.

Collected during the Survey and Dependencies analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisite for display: Expand the **Measured** column group.

Possible values:

- **Parallel: Explicit** - The loop does not have dependencies because it is explicitly vectorized or threaded on CPU.
- **Parallel: Proven** - A compiler did not detect dependencies in the loop at the compile time but did not vectorize the loop automatically for a certain reason.

- **Parallel: Kernel** (GPU-to-GPU modeling only) - The kernel is executed on a GPU.
- **Parallel: Programming Model** - The loop does not have dependencies because it is parallelized for execution on a target platform using a performance model (for example, OpenMP*, oneAPI Treading Building Blocks, Intel® oneAPI Data Analytics Library, SYCL).
- **Parallel: Workload** - Intel Advisor did not find dependencies in the loop based on the workload analyzed during the Dependencies analysis.
- **Parallel: User** - The loop is marked as not having dependencies with the `--set-parallel=<string>` option.
- **Parallel: Assumed** - Intel Advisor does not have information about loop dependencies but it assumed all such loops are parallel (that is, not having dependencies).
- **Dependency: <dependency-type>** - Intel Advisor found dependencies of specific types in the loop during the Dependencies analysis. Possible dependency types are RAW (read after write), WAR (write after read), WAW (write after read), Reduction.
- **Dependency: User** - The loop is marked as having dependencies with the `--set-dependency=<string>` option.
- **Dependency: Assumed** - Intel Advisor does not have information about dependencies for this loops but it assumes all such loops have dependencies.

Prerequisites for collection/display:

Some values in this column can appear only if you select specific options when collecting data or run the Dependencies analysis:

For **Parallel: Workload** and **Dependency: <dependency-type>**:

- GUI: Enable Dependencies analysis in the **Analysis Workflow** pane.
- CLI: Run `advisor --collect=dependencies --project-dir=<project-dir> [<options>] --<target>`. See [advisor Command Option Reference](#) for details.

For **Parallel: User**:

- GUI: Go to **Project Properties > Performance Modeling**. In the **Other parameters** field, enter a `--set-parallel=<string>` and a comma-separated list of loop IDs and/or source locations to mark them as parallel.
- CLI: Specify a comma-separated list of loop IDs and/or source locations with the `--set-parallel=<string>` option when modeling performance with `advisor --collect=projection`.

For **Dependency: User**:

- GUI: Go to **Project Properties > Performance Modeling**. In the **Other parameters** field, enter a `--set-dependency=<string>` and a comma-separated list of loop IDs and/or source locations to mark them as having dependencies.
- CLI: Specify a comma-separated list of loop IDs and/or source locations with the `--set-dependency=<string>` option when modeling performance with `advisor --collect=projection`.

For **Parallel: Assumed**:

- GUI: Disable **Assume Dependencies** under Performance Modeling analysis in the **Analysis Workflow** pane.
- CLI: Use the `--no-assume-dependencies` option when modeling performance with `advisor --collect=projection`.

For **Dependencies: Assumed**:

- GUI: Enable **Assume Dependencies** under Performance Modeling analysis in the **Analysis Workflow** pane.
- CLI: Use the `--assume-dependencies` option when modeling performance with `advisor --collect=projection`.

Interpretation:

- Loops with *no* real dependencies (**Parallel: Explicit**, **Parallel: Proven**, **Parallel: Programming Model**, and **Parallel: User** if you know that marked loops are parallel) can be safely offloaded to a target platform.
- If many loops have **Parallel: Assumed** or **Dependencies: Assumed** value, you are recommended to run the Dependencies analysis. See [Check How Assumed Dependencies Affect Modeling](#) for details.

Device-to-Host Size

Description: Total data transferred from device to host.

Collected during the FLOP analysis (Characterization) in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Prerequisites for display: Expand the **Data Transferred** column group.

Device-to-Host Time

Description: Total time spent on transferring data from device to host.

Collected during the FLOP analysis (Characterization) in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Prerequisites for display: Expand the **Data Transferred** column group.

DRAM

Description: Summary of estimated DRAM memory usage, including DRAM bandwidth, in gigabytes per second, and total DRAM traffic calculated as sum of read and write traffic.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

DRAM BW (Estimated Bounded By)

Description: DRAM Bandwidth. Estimated time, in seconds, spent on reading from DRAM memory and writing to DRAM memory assuming a maximum DRAM memory bandwidth is achieved.

Collected during the Trip Counts analysis (Characterization) and the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

DRAM BW (Memory Estimations)

Description: DRAM Bandwidth. Estimated rate at which data is transferred to and from the DRAM, in gigabytes per second.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.

- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

DRAM BW Utilization

Description: Estimated DRAM bandwidth utilization, in per cent.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Memory Estimations** column group in the Code Regions pane of the Accelerated Regions tab.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

Calculation: Ratio of average bandwidth to a maximum theoretical bandwidth.

DRAM Read Traffic

Description: Total estimated data read from the DRAM memory.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

DRAM Traffic

Description: Estimated sum of data read from and written to the DRAM memory.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

DRAM Write Traffic

Description: Total estimated data written to the DRAM memory.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

E

- [Elapsed Time](#)
- [EM Active](#)
- [Estimated Data Transfer with Reuse](#)
- [Estimated Time on Device](#)
- [EU Threading Occupancy](#)

Elapsed Time

Description: Wall-clock time from beginning to end of computing task execution.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

EM Active

Description: Average percentage of time when an extended math (EM) pipeline is processed. This metric is available for code running on the Intel® Arc™ graphics code-named Alchemist or newer.

Collected during the **Survey** analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Instructions** column group.

Estimated Data Transfer with Reuse

Description: Summary of data read from a target platform and written to the target platform. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on the target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Estimated Time on Device

Description: Estimated elapsed wall-clock time from beginning to end of loop execution estimated on a target platform after offloading without offload overhead and time for non-offloaded code.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Basic Estimated Metrics** column group.

Prerequisite for display: Expand the **Basic Estimated Metrics** column group.

EU Threading Occupancy

Description: Percentage of cycles on all execution units (EU) and thread slots when a slot has a thread scheduled.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

F

- [Fill Count per Thread](#)
- [FLOAT Operations](#)
- [FLOP AI \(Global Memory\)](#)
- [FP AI](#)
- [FPU Active](#)
- [FPU and EM Active](#)
- [FPU and Matrix Engine Active](#)
- [Fraction of Offloads](#)
- [From Target](#)

Fill Count per Thread

Description: Number of fill instructions used to read data values spilled from GRF into memory (L3 cache).

Collected during the Characterization analysis with Trip Counts and GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Register Spilling** column group.

Prerequisites for display: Expand the **Register Spilling** column group.

Interpretation: A high number of memory spill/fill (or load/store) operations significantly increases memory traffic and decreases the performance.

FLOAT Operations

Description: Summary of floating-point operations in a kernel.

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Compute Performance** column group.

Aggregation:

- [GOp](#) - Number of giga floating-point operations.
- [GOp/s](#) - Number of giga floating-point operations per second.
- [AI](#) - Ratio of floating-point operations to the bytes transferred to GPU memory.

You can hover over each value in the cell to see the value description.

FLOP AI (Global Memory)

Description: Estimated arithmetic intensity for floating-point operations (FLOP), in operations per byte.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

Calculated as ratio of floating-point operations to total bytes transferred to global memory (DRAM, HBM, or GDDR6).

FP AI

Description: Ratio of floating-point operations to bytes transferred to GPU memory.

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Compute Performance** column group.

Prerequisites for display: Expand the **GPU Compute Performance** column group. This metric is also shown in the collapsed **FLOAT Operations** column.

FPU Active

Description: Average percentage of time when an floating-point unit (FPU) pipeline is processed. This metric is available for code running on the Intel® Arc™ graphics code-named Alchemist or newer.

Collected during the **Survey** analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Instructions** column group.

FPU and EM Active

Description: Average percentage of time when floating-point unit (FPU) and extended math (EM) unit pipelines are processed. This metric is available for code running on the Intel® Arc™ graphics code-named Alchemist or newer.

Collected during the **Survey** analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Instructions** column group.

FPU and Matrix Engine Active

Description: Average percentage of time when floating-point unit (FPU) and matrix engine pipelines are processed. This metric is available for code running on the Intel® Arc™ graphics code-named Alchemist or newer.

Collected during the **Survey** analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Instructions** column group.

Fraction of Offloads

Description: Percentage of time spent in code regions profitable for offloading in relation to the total execution time of the region.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Basic Estimated Metrics** column group.

Prerequisites for display: Expand the **Basic Estimated Metrics** column group.

Interpretation: 100% means there are no non-offloaded child regions, calls to parallel runtime libraries, or system calls in the region.

From Target

Description: Estimated data transferred from a target platform to a shared memory by a loop, in megabytes. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfer with Reuse** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Prerequisite for display: Expand the **Estimated Data Transfer with Reuse** column group.

G

- [GFLOP \(GPU Roofline\)](#)
- [GFLOP \(Offload Modeling\)](#)
- [GFLOP32](#)

- GFLOP64
- GFLOPS (GPU Roofline)
- GFLOPS (Offload Modeling)
- GFLOPS32
- GFLOPS64
- GINT32
- GINT64
- GINTOP (GPU Roofline)
- GINTOP (Offload Modeling)
- GINTOPS (GPU Roofline)
- GINTOPS (Offload Modeling)
- GINTOPS32
- GINTOPS64
- Global
- Global Size (Compute Estimates)
- Global Size (Measured)
- GPU Memory
- GPU Shader Atomics
- GPU Shader Barriers
- GTI
- GTI BW (Estimated Bounded By)
- GTI BW (Memory Estimations)
- GTI BW Utilization
- GTI Read Traffic
- GTI Traffic
- GTI Write Traffic

GFLOP (GPU Roofline)

Description: Number of giga floating-point operations.

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Compute Performance** column group.

Instruction types counted: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Prerequisites for display: Expand the **GPU Compute Performance** column group. This metric is also shown in the collapsed **FLOAT Operations** column.

GFLOP (Offload Modeling)

Description: Estimated number of giga floating-point operations.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

GFLOP32

Description: Estimated number of 32-bit giga floating-point operations.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

Prerequisites for display: Expand the **Estimated FLOAT Operations** column group.

GFLOP64

Description: Estimated number of 64-bit giga floating-point operations.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

Prerequisites for display: Expand the **Estimated FLOAT Operations** column group.

GFLOPS (GPU Roofline)

Description: Number of giga floating-point operations per second.

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Compute Performance** column group.

Instruction types counted: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Prerequisites for display: Expand the **GPU Compute Performance** column group. This metric is also shown in the collapsed **FLOAT Operations** column.

GFLOPS (Offload Modeling)

Description: Estimated number of giga floating-point operations per second.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

GFLOPS32

Description: Estimated number of 32-bit giga floating-point operations per second.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

Prerequisites for display: Expand the **Estimated FLOAT Operations** column group.

GFLOPS64

Description: Estimated number of 64-bit giga floating-point operations per second.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

Prerequisites for display: Expand the **Estimated FLOAT Operations** column group.

GINT32

Description: Estimated number of 32-bit giga integer operations.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated INT Operations** column group.

Prerequisites for display: Expand the **Estimated INT Operations** column group.

GINT64

Description: Estimated number of 64-bit giga integer operations.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated INT Operations** column group.

Prerequisites for display: Expand the **Estimated INT Operations** column group.

GINTOP (GPU Roofline)

Description: Number of giga integer operations.

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Compute Performance** column group.

Instruction types counted: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Prerequisites for display: Expand the **GPU Compute Performance** column group. This metric is also shown in the collapsed **INT Operations** column.

GINTOP (Offload Modeling)

Description: Estimated number of giga integer operations.

Collected during the Performance Modeling analysis enabled in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated INT Operations** column group.

GINTOPS (GPU Roofline)

Description: Number of giga integer operations per second.

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Compute Performance** column group.

Instruction types counted: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Prerequisites for display: Expand the **GPU Compute Performance** column group. This metric is also shown in the collapsed **INT Operations** column.

GINTOPS (Offload Modeling)

Description: Estimated number of giga integer operations per second.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated INT Operations** column group.

GINTOPS32

Description: Estimated number of 32-bit giga integer operations per second.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated INT Operations** column group.

Prerequisite for display: Expand the **Estimated INT Operations** column group.

GINTOPS64

Description: Estimated number of 64-bit giga integer operations per second.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated INT Operations** column group.

Prerequisite for display: Expand the **Estimated INT Operations** column group.

Global

Description: Total number of work items in all work groups.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Work Size** column group.

Global Size (Compute Estimates)

Description: Total estimated number of work items in a loop executed after offloaded on a target platform.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisite for display: Expand the **Compute Estimates** column group.

Global Size (Measured)

Description: Total number of work items in a kernel instance on a baseline device. This metric is only available for the GPU-to-GPU modeling.

Collected during the Survey analysis with enabled GPU profiling in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisite for display: Expand the **Measured** column group.

GPU Memory

Description: Summary of GPU memory usage in a kernel. GPU memory is data transferred to and from GPU, chip uncore (LLC), and main memory.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Aggregation: The column reports the following metrics:

- [Total GPU memory traffic](#), in gigabytes
- [GPU memory bandwidth](#), in gigabytes per second

You can hover over each value in the cell to see the value description.

GPU Shader Atomics

Description: Total number of shader atomic memory accesses.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

GPU Shader Barriers

Description: Total number of shader barrier messages.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

GTI

Description: Summary of estimated GTI memory usage, including GTI bandwidth, in gigabytes per second, and total GTI traffic calculated as sum of read and write traffic.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

GTI BW (Estimated Bounded By)

Description: Graphics technology interface (GTI) Bandwidth. Estimated time, in seconds, spent on reading from and writing to GTI memory assuming a maximum GTI memory bandwidth is achieved.

Collected during the Trip Counts analysis (Characterization) and the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

GTI BW (Memory Estimations)

Description: Graphics technology interface (GTI) Bandwidth. Estimated rate at which data is transferred to and from the GTI, in gigabytes per second.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

GTI BW Utilization

Description: Graphics technology interface (GTI) bandwidth utilization. Estimated GTI bandwidth utilization, in per cent.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Memory Estimations** column group in the Code Regions pane of the Accelerated Regions tab.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

Calculation: Ratio of average bandwidth to a maximum theoretical bandwidth.

GTI Read Traffic

Description: Total estimated data read from the GTI memory.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

GTI Traffic

Description: Estimated sum of data read from and written to the GTI memory.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

GTI Write Traffic

Description: Total estimated data written to the GTI memory.

Collected during the Trip Counts (Characterization) and Performance Modeling analyses in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

H

- [Hardware Events per SEND](#)
- [Host-to-Device Size](#)
- [Host-to-Device Time](#)

Hardware Events per SEND

Description: Average number of atomic accesses generated by one atomic SEND instruction.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Atomics** column group.

Prerequisites for display: Expand the **Atomics** column group.

Host-to-Device Size

Description: Total data transferred from host to device.

Collected during the Characterization analysis with FLOP in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Prerequisites for display: Expand the **Data Transferred** column group.

Host-to-Device Time

Description: Total time spent on transferring data from host to device.

Collected during the Characterization analysis with FLOP in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Prerequisites for display: Expand the **Data Transferred** column group.

I

- [Idle](#)
- [Ignored Time](#)
- [Instances \(GPU Roofline\)](#)
- [Instances \(Offload Modeling - Compute Estimates\)](#)
- [Instances \(Offload Modeling - Measured\)](#)
- [INT AI \(GPU Roofline\)](#)
- [INT AI \(Global Memory\)](#)
- [INT Operations](#)
- [IPC Rate](#)
- [Iteration Space](#)

Idle

Description: Percentage of cycles on all execution units (EU) or vector engines (XVE) when no threads are scheduled on an EU or XVE.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** report > **GPU Kernels** pane > **EU Array** column group or **GPU Roofline Regions** report > **GPU Kernels** pane > **XVE Array** column group (for code running on the Intel® Arc™ graphics code-named Alchemist or newer).

Ignored Time

Description: Time spent in system calls and calls to ignored modules or parallel runtime libraries in the code regions recommended for offloading.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Non-User Code Metrics** column group.

Prerequisites for collection: From CLI, run the `--collect=projection` action with the `--ignore=<code-to-ignore>` action option. For example, to ignore MPI and OpenMP* calls, use the flag as follows: `--ignore=MPI,OMP`.

Prerequisite for display: Expand the **Time in Non-User Code** column group.

Interpretation: Time in the ignored code parts is not used for the : estimations. It does not affect time estimated for offloaded code regions.

Instances (GPU Roofline)

Description: Total number of times a task executes on a GPU.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Kernel Details** column group.

Prerequisite for display: Expand the **Kernel Details** column group.

Instances (Offload Modeling - Compute Estimates)

Description: Total estimated number of times a loop executes on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisite for display: Expand the **Compute Estimates** column group.

Instances (Offload Modeling - Measured)

Description: Total number of times a loop executes on a baseline GPU device.

Collected during the \ Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

INT AI (GPU Compute Performance)

Description: Ratio of integer operations to transferred bytes.

Collected during the Characterization with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** report > **GPU Kernels** pane > **GPU Compute Performance** column group.

Instruction types counted: BASIC COMPUTE, FMA, BIT, DIV, POW, MATH.

Prerequisites for display: Expand the **GPU Compute Performance** column group. This metric is also shown in the **INT Operations** column when the group is collapsed.

INT AI (Global Memory)

Description: Estimated arithmetic intensity for integer operations, in operations per byte.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated FLOAT Operations** column group.

Calculated as ratio of integer operations to total bytes transferred to global memory (DRAM, HBM, or GDDR6).

INT Operations

Description: Summary of integer operations used in a kernel.

Collected during the Characterization analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Compute Performance** column group.

Aggregation:

- [GOp](#) - Number of giga integer operations.
- [GOp/s](#) - Number of giga integer operations per second.
- [AI](#) - Ratio of integer operations to the bytes transferred to GPU memory.

You can hover over each value in the cell to see the value description.

IPC Rate

Description: Average rate of instructions per cycle (IPC) calculated for two floating-point unit (FPU) pipelines. For code running on the Intel® Arc™ graphics code-named Alchemist or newer, IPC rate is calculated for extended math (EM) unit and floating-point unit (FPU) pipelines.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **EU Instructions** column group or **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Instructions** column group (for code running on the Intel® Arc™ graphics code-named Alchemist or newer).

Iteration Space

Description: Summary of iteration metrics measured on a baseline device.

Collected during the Characterization analysis with Trip Counts (for CPU regions) or the Survey analysis with GPU profiling (for GPU regions) in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Aggregation: For the CPU-to-GPU modeling, this column reports the following metrics:

- [Call Count](#) (CC) - The number of times a loop/function was invoked.
- [Trip Counts](#) (TC) - The average number of times a loop/function was executed.

For the GPU-to-GPU modeling, this column reports the following metrics:

- [Global](#) - Total number of work items in all work groups.
- [Local](#) - The number of work items in one work group.

J

K

- [Kernel](#)
- [Kernel Launch Tax](#)
- [Kernel Type](#)

Kernel

Description: Kernel name.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Kernel Launch Tax

Description: Total estimated time cost for invoking a kernel when offloading a loop to a target platform. *Does not include data transfer costs.*

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

Kernel Type

Description: Action that a kernel performs.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Possible values:

- Compute
- Transfer
- Transfer in
- Transfer out
- Synchronization
- Allocate memory

L

- [Latencies](#)
- [L3 BW](#)
- [L3 Cache](#)
- [L3 Cache BW](#)
- [L3 Cache BW Utilization](#)
- [L3 Cache Read Traffic](#)
- [L3 Cache Traffic](#)
- [L3 Cache Write Traffic](#)
- [L3 Shader](#)
- [LLC](#)
- [LLC BW \(Estimated Bounded By\)](#)

- LLC BW (Memory Estimations)
- LLC BW Utilization
- LLC Read Traffic
- LLC Traffic
- LLC Write Traffic
- Load Latency
- Local
- Local Memory Size
- Local Size (Compute Estimates)
- Local Size (Measured)
- Loop/Function

Latencies

Description: Top uncovered latency in a loop/function, in milliseconds.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

L3 BW

Description: L3 Bandwidth. Estimated time, in seconds, spent on reading from L3 cache and writing to L3 cache assuming a maximum L3 cache bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

L3 Cache

Description: Summary of estimated L3 cache usage, including L3 cache bandwidth (in gigabytes per second) and L3 cache traffic calculated as sum of read and write traffic.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

L3 Cache BW

Description: Average estimated rate at which data is transferred to and from the L3 cache, in gigabytes per second.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

L3 Cache BW Utilization

Description: Estimated L3 cache bandwidth utilization, in per cent, calculated as ratio of average bandwidth to a maximum theoretical bandwidth.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

L3 Cache Read Traffic

Description: Total estimated data read from the L3 cache.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

L3 Cache Traffic

Description: Estimated sum of data read from and written to the L3 cache.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

L3 Cache Write Traffic

Description: Total estimated data written to the L3 cache.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

L3 Shader

Description: Summary of L3 cache usage in a kernel.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Aggregation: The column reports the following metrics:

- [Total L3 traffic](#), in gigabytes
- [L3 bandwidth](#), in gigabytes per second
- [Cache line utilization](#), in per cent. This metric is shown only if **CARM** is collected. If the kernel uses only a small portion of the transferred bytes, the value is highlighted in red.

You can hover over each value in the cell to see the value description and interpretation hints.

LLC

Description: Estimated last-level cache (LLC) usage, including LLC cache bandwidth (in gigabytes per second) and total LLC cache traffic, which is a sum of read and write traffic.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

LLC BW (Estimated Bounded By)

Description: Last-level cache (LLC) bandwidth. Estimated time, in seconds, spent on reading from LLC and writing to LLC assuming a maximum LLC bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

LLC BW (Memory Estimations)

Description: Estimated rate at which data is transferred to and from the LLC cache, in gigabytes per second.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

LLC BW Utilization

Description: Estimated LLC cache bandwidth utilization, in per cent.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

Calculation: Ratio of average bandwidth to a maximum theoretical bandwidth.

LLC Read Traffic

Description: Total estimated data read from the LLC cache.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

LLC Traffic

Description: Estimated sum of data read from and written to the LLC cache.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

LLC Write Traffic

Description: Total estimated data written to the LLC cache.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

Load Latency

Description: Uncovered cache or memory load latencies uncovered in a code region, in milliseconds.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: **Estimated Bounded By** column group.

Local

Description: Number of work items in one work group.

Collected during the **Survey** analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** report > **GPU Kernels** pane > **Work Size** column group.

Local Memory Size

Description: Local memory size used by each thread group.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Kernel Details** column group.

Prerequisite for display: Expand the **Kernel Details** column group.

Local Size (Compute Estimates)

Description: Total estimated number of work items in one work group of a loop executed after offloaded on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisite for display: Expand the **Compute Estimates** column group.

Local Size (Measured)

Description: Total number of work items in one work group of a kernel. This metric is only available for the GPU-to-GPU modeling.

Collected during the Survey analysis with enabled GPU profiling in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisite for display: Expand the **Measured** column group.

Loop/Function

Description: Name and source location of a loop/function in a region, where region is a sub-tree of loops/functions in a call tree.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane.

M

- [Matrix Engine Active](#)
- [Memory Footprint, B](#)
- [Memory Impact](#)
- [Module](#)

Matrix Engine Active

Description: Average percentage of time when a matrix engine pipeline is processed. This metric is available for code running on the Intel® Arc™ graphics code-named Alchemist or newer.

Collected during the **Survey** analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Instructions** column group.

Memory Footprint, B

Description: Size of unique data (variables) spilled from general register file (GRF) per thread, in bytes.

Collected during the Characterization analysis with Trip Counts and GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Register Spilling** column group.

Prerequisites for display: Expand the **Register Spilling** column group. This metric is also shown in the collapsed **Register Spilling** column.

Interpretation: Higher value indicates that register spilling decreases performance.

Memory Impact

Description: Total memory traffic between general register file (GRF) and L3 caused by the register spilling, in percentage of total traffic.

Collected during the Characterization analysis with Trip Counts and GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Register Spilling** column group.

Prerequisites for display: Expand the **Register Spilling** column group.

Interpretation: The lower the ratio is, the better the kernel is optimized. If you see a high value, it means that spill/fill traffic takes up a big part of total traffic and may significantly decrease kernel performance.

Calculation: Ratio of total spill/fill traffic to the total observed cache traffic.

Module

Description: Program module name.

Collected during the Survey in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Location** column group.

Prerequisites for display: Expand the **Location** column group.

N

O

- [Offload Tax](#)
- [Offload Summary](#)
- [Overall Non-Accelerable Time](#)

Offload Tax

Description: Total time spent for transferring data and launching kernel, in milliseconds.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

Offload Summary

Description: Conclusion that indicates whether a code region is profitable for offloading to a target platform. In the Top-Down pane, it also reports the node position, such as offload child loops and child functions.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Basic Estimated Metrics** column group.

Overall Non-Accelerable Time

Description: Total estimated time spent in non-offloaded parts of offloaded code regions.

Collected during the Survey and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Calculation: This column is a sum of the following metrics:

- [Total Time in DAAL Calls](#)
- [Total Time in SYCL Calls](#)
- [Total Time in MPI Calls](#)
- [Total Time in OpenCL Calls](#)
- [Total Time in OpenMP Calls](#)
- [Total Time in System Calls](#)
- [Total Time in TBB Calls](#)

Interpretation: These code parts are located inside offloaded regions, but the performance model assumes these parts are executed on a baseline device. Examples of such code parts are OpenMP* code parts, SYCL runtimes, and system calls.

P

- [Parallel Factor](#)
- [Parallel Threads](#)
- [Performance Issues \(GPU Roofline\)](#)
- [Performance Issues \(Offload Modeling\)](#)
- [Private](#)
- [Private Memory Size](#)

Parallel Factor

Description: Number of loop iterations or kernel work items executed in parallel on a target device for a loop/function.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Parallel Threads

Description: Estimated number of threads scheduled simultaneously on *all* execution units (EU) or vector engines (XVE).

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisites for display: Expand the **Compute Estimates** column group.

Performance Issues (GPU Roofline)

Description: Performance issues and recommendations for optimizing code regions executed on a GPU.

Collected during the Survey, Characterization, and Performance Modeling analyses in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane .

Interpretation: Click to view the full recommendation text with code examples and recommended fixes in the Recommendations pane of the **GPU Roofline Regions** tab.

Performance Issues (Offload Modeling)

Description: Recommendations for offloading code regions with estimated performance summary and/or potential issues with optimization hints.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane.

Interpretation: Click to view the full recommendation text with examples of using SYCL and OpenMP* programming modeling to offload the code regions and/or fix the performance issue in the

Recommendations pane of the **Accelerated Regions** tab.

Private

Description: Total estimated data transferred to a private memory from a target platform by a loop. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfers with Reuse** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Prerequisite for display: Expand the **Estimated Data Transfers with Reuse** column group.

Private Memory Size

Description: Private memory size allocated by a compiler to each thread.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Kernel Details** column group.

Prerequisite for display: Expand the **Kernel Details** column group.

Q

R

- [Read](#)
- [Read, GB \(GPU Memory\)](#)
- [Read, GB \(Register Spilling\)](#)
- [Read, GB \(SLM\)](#)
- [Read, GB/s \(GPU Memory\)](#)
- [Read, GB/s \(SLM\)](#)
- [Read without Reuse](#)
- [Region](#)
- [Register Spilling](#)
- [Repetitions](#)

Read

Description: Estimated data read from a target platform by an offload region, in megabytes. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform.

Collected during the Characterization analysis with Trip Counts analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfers with Reuse** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Prerequisite for display: Expand the **Estimated Data Transfers with Reuse** column group.

Read, GB (GPU Memory)

Description: Total data read from GPU, chip uncore (LLC), and main memory, in gigabytes.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Memory** column group.

Prerequisites for display: Expand the **GPU Memory** column group.

Read, GB (Register Spilling)

Description: Total data read, or filled, from L3 memory due to register spilling, in gigabytes.

Collected during the Characterization analysis with Trip Counts and GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Register Spilling** column group.

Prerequisites for display: Expand the **Register Spilling** column group.

Read, GB (SLM)

Description: Total data read from the shared local memory (SLM), in gigabytes.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **SLM** column group.

Prerequisites for display: Expand the **SLM** column group.

Read, GB/s (GPU Memory)

Description: Rate at which data is read from GPU, chip uncore (LLC), and main memory, in gigabytes per second.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Memory** column group.

Prerequisites for display: Expand the **GPU Memory** column group.

Read, GB/s (SLM)

Description: Rate at which data is read from shared local memory (SLM), in gigabytes per second.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** report > **GPU Kernels** pane > **SLM** column group.

Prerequisites for display: Expand the **SLM** column group.

Read without Reuse

Description: Estimated data read from a target platform by a code region considering no data is reused between kernels, in megabytes. This metric is available only if you enabled the data reuse analysis for the Performance Modeling.

Collected during the Characterization analysis with Trip Counts) and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfers with Reuse** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** under **Characterization** to **Full** and enable the **Data Reuse Analysis** checkbox under **Performance Modeling**.
- CLI: Use the `--data-transfer=full` action option with the `--collect=tripcounts` action and the `--data-reuse-analysis` option with the `--collect=tripcounts` and `--collect=projection` actions.

Prerequisite for display: Expand the **Estimated Data Transfers with Reuse** column group.

Region

Description: Programming model used in a code region.

Collected during the **Survey** analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Register Spilling

Description: Summary of register spilling impact on kernel performance

Collected during the Characterization analysis with Trip Counts and GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Interpretation: Register spilling occurs when a thread block (or work item) needs more space in the general register file (GRF) than is available, and data is loaded, or *spilled*, into memory through L3 cache. Next this data is needed, application has to read, or *fill*, it from the L3 cache memory, which causes more memory operation. As a result, when register spilling occurs in a kernel, it decreases its performance.

For the best performance, there should be no spills in the kernel.

Aggregation:

- **Footprint** - Size of unique data (variables) spilled from GRF per thread, in bytes. Higher value indicates register spilling decreases performance.
- **Traffic** - Total size of data spilled to (spill traffic) and filled from (fill traffic) L3 cache memory due to register spilling, in gigabytes. Higher value indicates register spilling decreases performance.
- **Impact** (in per cent) - Ratio between total spill/fill traffic and total L3 traffic. It indicates how much traffic is not caused by data exchanged in the kernel algorithm. Higher value indicates register spilling decreases performance.

Repetitions

Description: Average repetitions of atomic SEND instructions.

If a GPU does not support an atomic operation, an additional Compare-And-Swap (CAS) atomic is called. It loads data to a register, operates on it, and compares the result with a previous value. If the values do not match, it means another thread has changed the value, and the current result is invalid. To recalculate, the CAS atomic repeats the process: it loads data, operates on it, and compares. The number of such repetitions on average for atomic SEND instructions is reported in the *repetitions* metric.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Atomics** column group.

Prerequisites for display: Expand the **Atomics** column group.

Calculated as *Shader atomics / Expected atomics*, where:

- Shader atomics is the number of atomics called in a kernel as measured on a hardware.
- Expected atomics is the expected number of atomics called in a kernel without repetitions. It is calculated as *Static Atomics * Work Items / Average execution size*.

Static atomics is the number of atomics defined in a source code. Each static atomic is transformed to an atomic SEND instruction with a certain average execution size, which is the number of elements that the instruction can process in parallel (SIMD model). The ratio of work items to the average execution size is the estimated number of atomic SEND instructions for one source static atomic.

S

- [Send Active](#)
- [SIMD Width \(GPU Roofline\)](#)
- [SIMD Width \(Offload Modeling - Compute Estimates\)](#)
- [SIMD Width \(Offload Modeling - Measured\)](#)
- [SLM \(GPU Roofline\)](#)
- [SLM \(Offload Modeling\)](#)
- [SLM BW \(Estimated Bounded by\)](#)
- [SLM BW \(Memory Estimations\)](#)
- [SLM BW Utilization](#)
- [SLM Read Traffic](#)
- [SLM Traffic](#)
- [SLM Write Traffic](#)
- [Source Location \(GPU Roofline\)](#)
- [Source Location \(Offload Modeling\)](#)
- [Spill Count per Thread](#)
- [Stalled](#)
- [SVM Usage Type](#)
- [Speed-Up](#)
- [Synchronization Time](#)

Send Active

Description: Percentage of cycles on all execution units (EU) or vector engines (XVE) when a send pipeline is actively processed.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **EU Instructions** column group or **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Instructions** column group (for code running on the Intel® Arc™ graphics code-named Alchemist or newer).

SIMD Width (GPU Roofline)

Description: Number of work items processed by a single GPU thread.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** report > **GPU Kernels** pane > **Kernel Details** column group.

Prerequisites for display: Expand the **Kernel Details** column group.

SIMD Width (Offload Modeling - Compute Estimates)

Description: Estimated number of work items processed by a single thread on a target platform.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisites for display: Expand the **Compute Estimates** column group.

SIMD Width (Offload Modeling - Measured)

Description: Number of work items processed by a single thread on a baseline device. This metric is only available for the GPU-to-GPU modeling.

Collected during the Survey analysis with enabled GPU profiling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

SLM (GPU Roofline)

Description: Summary of shared local memory (SLM) usage in a kernel.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Aggregation: The column reports the following metrics:

- [Total SLM traffic](#), in gigabytes
- [SLM bandwidth](#), in gigabytes per second

You can hover over each value in the cell to see the value description.

SLM (Offload Modeling)

Description: Summary of estimated SLM usage, including SLM bandwidth, in gigabytes per second, and SLM traffic calculated as sum of read and write traffic.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

SLM BW (Estimated Bounded by)

Description: Shared Local Memory (SLM) bandwidth. Estimated time, in seconds, spent on reading from SLM and writing to SLM assuming a maximum SLM bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Prerequisite for display: Expand the **Estimated Bounded By** column group.

SLM BW (Memory Estimations)

Description: Shared Local Memory (SLM) bandwidth. Average estimated rate at which data is transferred to and from the SLM. This is a dynamic value, and depending on the bandwidth value, it can be measured in bytes per second, kilobytes per second, megabytes per second, and so on.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for display: Expand the **Memory Estimations** column group.

SLM BW Utilization

Description: Estimated shared local memory (SLM) bandwidth utilization, in per cent.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for display: Expand the **Memory Estimations** column group.

Calculation: Ratio of average bandwidth to a maximum theoretical bandwidth.

SLM Read Traffic

Description: Total estimated data read from the SLM.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for display: Expand the **Memory Estimations** column group.

SLM Traffic

Description: Estimated sum of data read from and written to the shared local memory (SLM).

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for display: Expand the **Memory Estimations** column group.

SLM Write Traffic

Description: Total estimated data written to shared local memory (SLM).

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for display: Expand the **Memory Estimations** column group.

Source Location (Offload Modeling)

Description: Source file name and line number.

Collected during the Survey in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Location** column group.

Interpretation: Use this column to understand where a code region is located.

Source Location (GPU Roofline)

Description: Source file name and line number.

Collected during the Survey in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Interpretation: Use this column to understand where a kernel is located.

Spill Count per Thread

Description: Number of spill instructions used to load data values from general register file (GRF) into memory (L3 cache).

Collected during the Characterization analysis with Trip Counts and GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Register Spilling** column group.

Prerequisites for display: Expand the **Register Spilling** column group.

Interpretation: A high number of memory spill/fill (or load/store) operations significantly increases memory traffic and decreases the performance.

Stalled

Description: Percentage of cycles on all execution units (EU) or vector engines (XVE) when at least one thread is scheduled, but the EU or XVE is stalled.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **EU Array** column group or **GPU Roofline Regions** tab > **GPU Kernels** pane > **XVE Array** column group (for code running on the Intel® Arc™ graphics code-named Alchemist or newer).

SVM Usage Type

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Kernel Details** column group.

Prerequisites for display: Expand the **Kernel Details** column group.

Speed-Up

Description: Estimated speedup for a loop offloaded to a target device, in comparison to the original elapsed time.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Basic Estimated Metrics** column group.

Interpretation: If the speedup is more than 1, the code region is recommended for offloading to a target device. If the speedup is equal to or less than 1, the code region is not recommended for offloading.

Synchronization Time

Description: Total time spent on synchronization tasks.

Collected during the Characterization analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Prerequisites for display: Expand the **Data Transferred** column group.

T

- [Taxes with Reuse](#)
- [Thread Occupancy \(Compute Estimates\)](#)
- [Thread Occupancy \(Measured\)](#)
- [Threads per EU](#)
- [Throughput](#)
- [Time \(Estimated\)](#)
- [Time \(Measured\)](#)
- [Time by DRAM BW](#)
- [Time by GTI BW](#)
- [Time by L3 Cache BW](#)
- [Time by LLC BW](#)
- [Time by SLM BW](#)
- [To Target](#)
- [ToFrom Target](#)
- [Total](#)
- [Total, GB \(GPU Memory\)](#)
- [Total, GB \(L3 Shader\)](#)
- [Total, GB \(SLM\)](#)
- [Total, GB/s](#)
- [Total Size](#)
- [Total Time \(Data Transferred\)](#)

- Total Time (Kernel Details)
- Total Time in DAAL Calls
- Total Time in SYCL Calls
- Total Time in MPI Calls
- Total Time in OpenCL Calls
- Total Time in OpenMP Calls
- Total Time in System Calls
- Total Time in TBB Calls
- Total Traffic, GB (Register Spilling)
- Total Trip Count
- Total without Reuse

Taxes with Reuse

Description: The highest estimated time cost and a sum of all other costs for offloading a loop from host to a target platform. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform. A *triangle* icon in a table cell indicates that this region reused data.

This decreases the estimates data transfer tax.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Thread Occupancy (Compute Estimates)

Description: Average percentage of thread slots occupied on all execution units or vector engines estimated on a target device.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisites for display: Expand the **Compute Estimates** column group.

Thread Occupancy (Measured)

Description: Average percentage of thread slots occupied on all execution units or vector engines measured on a baseline device. This metric is only available for the GPU-to-GPU modeling.

Collected during the Survey analysis with GPU profiling in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

Threads per EU

Description: Estimated number of threads scheduled simultaneously *per execution unit (EU)*.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisites for display: Expand the **Compute Estimates** column group.

Threads per XVE

Description: Estimated number of threads scheduled simultaneously *per vector engine (XVE)*. This metric is available if you model performance for the Intel® Arc™ graphics code-named Alchemist, which is **XeHPG 256** and **XeHPG 512** target device configurations in the Intel Advisor, or newer. This metric is equivalent to the **Threads per EU** metric for legacy terminology.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Compute Estimates** column group.

Prerequisites for display: Expand the **Compute Estimates** column group.

Throughput

Description: Top two factors that a loop/function is bounded by, in milliseconds.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Bounded By** column group.

Time (Estimated)

Description: Estimated elapsed wall-clock time from beginning to end of loop execution estimated on a target platform after offloading, including offload overhead, with percentage to the total time.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Basic Estimated Metrics** column group.

Time (Measured)

Description: Elapsed wall-clock time from beginning to end of loop execution measured on a host platform.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Time by DRAM BW

Description: Estimated time, in seconds, spent on reading from DRAM memory and writing to DRAM memory assuming a maximum DRAM memory bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

Time by GTI BW

Description: Estimated time, in seconds, spent on reading from graphics technology interface (GTI) and writing to GTI assuming a maximum GTI bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for display: Expand the **Memory Estimations** column group.

Time by L3 Cache BW

Description: Estimated time, in seconds, spent on reading from L3 cache and writing to L3 cache assuming a maximum L3 cache bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

Time by LLC BW

Description: Estimated time, in seconds, spent on reading from last-level cache (LLC) and writing to LLC assuming a maximum LLC bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, enable the **Cache Simulation** checkbox.
- CLI: Run the `--collect=tripcounts` action with the `--enable-cache-simulation` and `--target-device=<device>` action options.

Prerequisites for display: Expand the **Memory Estimations** column group.

Time by SLM BW

Description: Estimated time, in seconds, spent on reading from shared local memory (SLM) and writing to SLM assuming a maximum SLM bandwidth is achieved.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Memory Estimations** column group.

Prerequisites for display: Expand the **Memory Estimations** column group.

To Target

Description: Estimated data transferred to a target platform from a shared memory by a loop, in megabytes. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfer with Reuse** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Prerequisite for display: Expand the **Estimated Data Transfer with Reuse** column group.

ToFrom Target

Description: Sum of estimated data transferred both to/from a shared memory to/from a target platform by a loop, in megabytes. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfer with Reuse** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Prerequisite for display: Expand the **Estimated Data Transfer with Reuse** column group.

Total

Description: Sum of the total estimated traffic incoming to a target platform and the total estimated traffic outgoing from the target platform, for an offload loop, in megabytes.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfer with Reuse** column group.

Prerequisites for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** to **Light**, **Medium**, or **Full**.
- CLI: Run the `--collect=tripcounts` action with the `--data-transfer=[full | medium | light]` action options.

Prerequisite for display: Expand the **Estimated Data Transfer with Reuse** column group.

Calculation: $(MappedTo + MappedFrom + 2 * MappedToFrom)$. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform.

Total, GB (GPU Memory)

Description: Total data transferred to and from GPU, chip uncore (LLC), and main memory, in gigabytes.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Memory** column group.

Prerequisite for display: Expand the **GPU Memory** column. This metric is also shown in the collapsed **GPU Memory** column.

Total, GB (L3 Shader)

Description: Total data transferred between execution units or vector engines and L3 cache, in gigabytes.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **L3 Shader** column group.

Prerequisites for display: Expand the **L3 Shader** column. This metric is also shown in the collapsed **L3 Shader** column.

Total, GB (SLM)

Description: Total data transferred to and from the shared local memory (SLM).

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **SLM** column group.

Prerequisites for display: Expand the **SLM** column. This metric is also shown in the collapsed **SLM** column.

Total, GB/s

Description: Average data transfer bandwidth between CPU and GPU.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Prerequisites for display: Expand the **Data Transferred** column group.

Interpretation: In some cases, such as `clEnqueueMapBuffer`, data transfers might generate high bandwidth because memory is not copied but shared using L3 cache.

Total Size

Description: Total data processed on a GPU.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Total Time (Data Transferred)

Description: Total time for transferring data from host to device and from device to host.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Data Transferred** column group.

Total Time (Kernel Details)

Description: Total time spent executing a task.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Kernel Details** column group.

Prerequisites for display: Expand the **Kernel Details** column group.

Total Time in DAAL Calls

Description: Total time spent in Intel® Data Analytics Acceleration Library (Intel® DAAL) calls in an offloaded code region, in seconds.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Prerequisites for display: Expand the **Time in Non-User Code** column group.

Interpretation: If the value in the column is more than 0, the code region contains Intel DAAL calls.

Total Time in SYCL Calls

Description: Total time spent in SYCL calls in an offloaded code region, in seconds.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Prerequisites for display: Expand the **Time in Non-User Code** column group.

Interpretation: If the value in the column is more than 0, the code region contains SYCL calls.

Total Time in MPI Calls

Description: Total time spent in MPI calls in an offloaded code region, in seconds.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Interpretation: If the value in the column is more than 0, the code region contains MPI calls.

Total Time in OpenCL Calls

Description: Total time spent in OpenCL™ calls in an offloaded code region, in seconds.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Prerequisites for display: Expand the **Time in Non-User Code** column group.

Interpretation: If the value in the column is more than 0, the code region contains OpenCL calls.

Total Time in OpenMP Calls

Description: Total time spent in OpenMP* calls in an offloaded code region, in seconds.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Prerequisites for display: Expand the **Time in Non-User Code** column group.

Interpretation: If the value in the column is more than 0, the code region contains OpenMP calls.

Total Time in System Calls

Description: Total time spent in system calls in an offloaded code region, in seconds.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Prerequisites for display: Expand the **Time in Non-User Code** column group.

Interpretation: If the value in the column is more than 0, the code region contains system calls.

Total Time in TBB Calls

Description: Total time spent in Intel® oneAPI Threading Building Blocks (oneTBB) calls in an offloaded code region, in seconds.

Collected during the Survey analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Time in Non-User Code** column group.

Prerequisites for display: Expand the **Time in Non-User Code** column group.

Interpretation: If the value in the column is more than 0, the code region contains oneTBB calls.

Total Traffic, GB (Register Spilling)

Description: Total data spilled to and filled from L3 memory due to register spilling, in gigabytes.

Collected during the Trip Counts analysis with GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Register Spilling** column group.

Prerequisites for display: Expand the **Register Spilling** column group.

Interpretation: High value indicates that spill/fill traffic might take a big part of the total data traffic in the kernel and decrease its performance. See the **Memory Impact** column to understand how much of total traffic it is.

Calculation: A sum of data spilled from general register file (GRF) to L3 and filled from L3 to GRF.

Total Trip Count

Description: Total number of times a loop/function is executed.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

Total without Reuse

Description: Sum of the total estimated traffic incoming to a target platform and the total estimated traffic outgoing from the target platform considering no data is reused, in megabytes. This metric is available only if you enabled the data reuse analysis for the Performance Modeling.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfer with Reuse** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** under **Characterization** to **Full** and enable the **Data Reuse Analysis** checkbox under **Performance Modeling**.
- CLI: Use the `--data-transfer=full` action option with the `--collect=tripcounts` action and the `--data-reuse-analysis` option with the `--collect=tripcounts` and `--collect=projection` actions.

Prerequisite for display: Expand the **Estimated Data Transfer with Reuse** column group.

Calculation: $(MappedTo + MappedFrom + 2 * MappedToFrom)$.

U

- [Unroll Factor](#)

Unroll Factor

Description: Loop unroll factor applied by the compiler.

Collected during the Survey in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

V

- [Vector ISA](#)
- [Vector Length](#)

Vector ISA

Description: The highest vector instruction set architecture (ISA) used for individual instructions.

Collected during the Survey in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

Vector Length

Description: Number of elements processed in a single iteration of vector loops or the number of elements processed in individual vector instructions determined by a binary static analysis or an Intel® Compiler.

Collected during the Survey in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Measured** column group.

Prerequisites for display: Expand the **Measured** column group.

W

- [Why Not Offloaded](#)

- Write
- Write, GB (GPU Memory)
- Write, GB (Register Spilling)
- Write, GB (SLM)
- Write, GB/s (GPU Memory)
- Write, GB/s (SLM)
- Write without Reuse

Why Not Offloaded

Description: Reason why a code region is not recommended for offloading to a target GPU device.

Collected during the Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Basic Estimated Metrics** column group.

Interpretation: See [Investigate Non-Offloaded Code Regions](#) for details about available reasons.

Write

Description: Estimated data written to a target platform by a loop. If you enabled the data reuse analysis for the Performance Modeling, the metric value is calculated considering data reuse between code regions on a target platform.

Collected during the Characterization analysis with Trip Counts in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfer with Reuse** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** under **Characterization** to **Light**, **Medium**, or **Full**.
- CLI: Use the `--data-transfer=[full | medium | light]` option with the `--collect=tripcounts` action.

Prerequisite for display: Expand the **Estimated Data Transfer with Reuse** column group.

Write, GB (GPU Memory)

Description: Total data written to GPU, chip uncore (LLC), and main memory, in gigabytes.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Memory** column group.

Prerequisites for display: Expand the **GPU Memory** column group.

Write, GB (Register Spilling)

Description: Total data written, or spilled, to L3 memory due to register spilling, in gigabytes.

Collected during the Characterization analysis with Trip Counts and GPU profiling in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **Register Spilling** column group.

Prerequisites for display: Expand the **Register Spilling** column group.

Write, GB (SLM)

Description: Total data written to the shared local memory (SLM), in gigabytes.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **SLM** column group.

Prerequisites for display: Expand the **SLM** column group.

Write, GB/s (GPU Memory)

Description: Rate at which data is written to GPU, chip uncore (LLC), and main memory, in gigabytes per second.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **GPU Memory** column group.

Prerequisites for display: Expand the **GPU Memory** column group.

Write, GB/s (SLM)

Description: Rate at which data is written to shared local memory (SLM), in gigabytes per second.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane > **SLM** column group.

Prerequisites for display: Expand the **SLM** column group.

Write without Reuse

Description: Estimated data written to a target platform by a code region considering no data is reused, in megabytes. This metric is available only if you enabled the data reuse analysis for the Performance Modeling.

Collected during the Characterization analysis with Trip Counts and Performance Modeling analysis in the [Offload Modeling](#) perspective and **found** in the **Accelerated Regions** tab > **Code Regions** pane > **Estimated Data Transfer with Reuse** column group.

Prerequisite for collection:

- GUI: From the **Analysis Workflow** pane, set the **Data Transfer Simulation** under **Characterization** to **Full** and enable the **Data Reuse Analysis** checkbox under **Performance Modeling**.
- CLI: Use the `--data-transfer=full` action option with the `--collect=tripcounts` action and the `--data-reuse-analysis` option with the `--collect=tripcounts` and `--collect=projection` actions.

Prerequisite for display: Expand the **Estimated Data Transfer with Reuse** column group.

X, Y, Z

- [XVE Threading Occupancy](#)

XVE Threading Occupancy

Description: Percentage of cycles on all vector engines (XVE) and thread slots when a slot has a thread scheduled. This metric is available for code running on the Intel® Arc™ graphics code-named Alchemist or newer. This metric is equivalent to the **EU Threading Occupancy** metric for legacy terminology.

Collected during the Survey analysis in the [GPU Roofline Insights](#) perspective and **found** in the **GPU Roofline Regions** tab > **GPU Kernels** pane.

Dependencies Problem and Message Types

The Intel® Advisor Dependencies analysis identifies various data sharing problems and messages. This reference section describes these problems and messages, and offers possible correction strategies.

Problem Type Name	Severity and Cause
Dangling Lock	Error. Occurs when a task does not release a lock before the task ends.

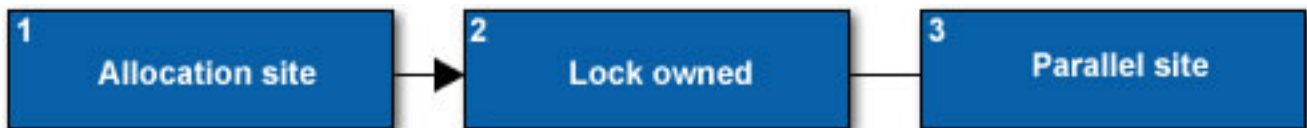
Problem Type Name	Severity and Cause
Data Communication	Error. Occurs when a task writes a value that a different task reads. If not fixed prior to conversion to parallel code, a Data Communication problem could result in a data race.
Data Communication, Child Task	Error. Occurs when a task writes a value that a different (child) task reads. If not fixed prior to conversion to parallel code, a Data Communication problem could result in a data race.
Inconsistent Lock Use	Warning. Occurs when a task execution accesses a memory location more than once, under the control of different locks.
Lock Hierarchy Violation	Warning. Occurs when two or more locks are acquired in a different order in two task executions, potentially leading to a deadlock when the program's tasks execute in parallel.
Memory Reuse	Error. Occurs when two tasks write to a shared memory location. That is, a task writes to a variable with a new value but does not read the same value generated by a prior task. If not fixed prior to conversion to parallel code, this Memory Reuse problem could result in a data race.
Memory Reuse, Child Task	Error. Occurs when two tasks write to a shared memory location, where a parent task overwrites a variable with a new value that was read by a previously executed child task in the same site. If not fixed prior to conversion to parallel code, this Memory Reuse, Child Task problem could result in a data race.
Memory Watch	Remark. Occurs when a task accesses a memory location marked by an <code>ANNOTATE_OBSERVE_USES</code> annotation. In this case, this <i>problem</i> provides informational feedback only and no action is required. This is useful for finding uses of specified memory locations while a task is executing.
Missing End Site	Error. Occurs when a site-begin annotation is executed but the corresponding site-end annotation is not executed before the thread or application exits.
Missing End Task	Error. Occurs when a task-begin annotation is executed but the corresponding task-end annotation is not executed before the site, thread, or application exits.
Missing Start Site	Error. Occurs when an end-site annotation is executed but there is no active site.
Missing Start Task	Error. Occurs when an end task annotation is executed but there is no active task.
No tasks in parallel site	Warning. Occurs when a parallel site was executed but no task annotations were executed in the dynamic extent of the active parallel site.
One task instance in parallel site	Warning. Occurs when a parallel site was executed but annotations for only one task instance were executed in the dynamic extent of the active parallel site. This may be the expected behavior, or it may indicate an error in the placement of annotations or a data set that is not well suited for parallelism.
Orphaned Task	Error. Occurs when a task-begin annotation is executed that is not within an active parallel site.

Problem Type Name	Severity and Cause
Parallel Site Information	Remark. Occurs when execution enters a parallel site. This confirms that your program and its data are executing the annotations you inserted during execution of the Dependencies tool analysis. In this case, this <i>message</i> provides informational feedback only and no action is required.
Thread Information	Remark. In this case, this <i>message</i> provides informational feedback and no action is required.
Unhandled Application Exception	Error. Occurs when an unhandled exception is detected that causes the application program to crash.

Dangling Lock

Occurs when a task does not release a lock before the task ends.

Syntax



ID	Code Location	Description
1	Allocation site	If present, represents the location and associated call stack when the lock was created.
2	Lock owned	If present, represents the location and its associated call stack when the lock was last acquired.
3	Parallel site	If present, represents the location and associated call stack of the site-begin annotation of the parallel site containing the task that acquired the lock.

Example

```

void problem()
{
    ANNOTATE_SITE_BEGIN(dangle_site1);          // Parallel site
    ANNOTATE_TASK_BEGIN(task1);
    ANNOTATE_LOCK_ACQUIRE(&dangle);           // Lock owned
    ANNOTATE_TASK_END();
    // ...
    ANNOTATE_SITE_END();
}
  
```

In this example:

- There is a parallel site that contains a task.
- The lock is acquired in the task.
- The lock is not released before the end of the task - the `ANNOTATE_LOCK_RELEASE()` annotation is missing.

Possible Correction Strategies

- Make sure that an `ANNOTATE_LOCK_RELEASE(address)` annotation is executed on every flow path from an `ANNOTATE_LOCK_ACQUIRE(address)` annotation to the end of the task. If a code region has multiple exit flow paths, make sure the lock is released on all the paths.
- Consider putting the lock-acquire and lock-release in the constructor and destructor of a static object, so that lock release happens automatically.
- Consider putting a lock release in an exception handler if a the locked region might exit from an exception.

Data Communication

Occurs when a task writes a value that a different task reads.

Syntax



ID	Code Location	Description
1	Allocation site	If present, represents the location and associated call stack when the memory block was allocated.
2	Parallel site	If present, represents the location and associated call stack of the parallel site containing the Data Communication problem.
3	Write	Represents the instruction and associated call stack where the memory was written.
4	Read	Represents the instruction and associated call stack where the memory was read in a different task execution.

Example

In this example, the write to the heap variable in task1 might occur either before or after the read in task2.

```

void problem()
{
    int* pointer = new int;           // Allocation site
    ANNOTATE_SITE_BEGIN(datacomm_site1); // Begin parallel site
    ANNOTATE_TASK_BEGIN(task1);
        *pointer = 999;               // Write
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(task2);
        assert(*pointer == 999);      // Read
    ANNOTATE_TASK_END();
    ANNOTATE_SITE_END();
}
  
```

In this example, each task execution reads the variable `communication`, adds one to its value, and writes the result back to the variable. The write in each task execution might occur either before or after the read in the other task instance execution.

```

void data_communication()
{
    ANNOTATE_SITE_BEGIN(site);       // Parallel site
  
```

```

for (int i = 0; i < 2; i++) {
    ANNOTATE_TASK_BEGIN(task);    // Write and Read in different task execution
    communication++;             /* data communication */
    ANNOTATE_TASK_END();
}
ANNOTATE_SITE_END();
}

```

Possible Correction Strategies

- If two accesses to the same memory must occur in a specific order, then the accesses must not be in different task executions in a single site execution. You will have to change the structure of your sites and tasks.
- If the order of memory modifications in two task executions is not important, but the executions of the modifications must not occur simultaneously, use locks to synchronize them.
- Induction and reduction annotations can tell the Dependencies tool about programs where the program behavior will be correct, regardless of the order of the accesses.

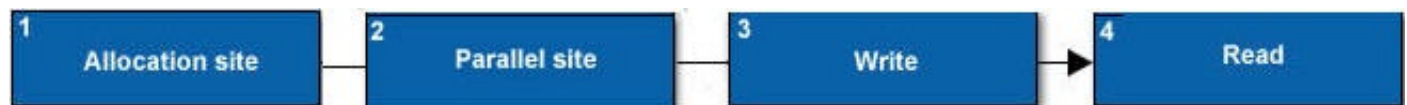
See Also

Data Sharing Problems

Data Communication, Child Task

Occurs when a task writes a value that a different (child) task reads. A child task is a task nested inside another task.

Syntax



ID	Code Location	Description
1	Allocation site	If present, represents the location and associated call stack when the memory block was allocated.
2	Parallel site	Represents the location and associated call stack of the parallel site containing the Data Communication problem.
3	Write	Represents the instruction and associated call stack where the memory was written.
4	Read	Represents the instruction and associated call stack where the memory was read in a different task execution.

Example

```

void problem()
{
    int* pointer = new int;           // Allocation site
    ANNOTATE_SITE_BEGIN(datacomm_site1); // Begin parallel site
    ANNOTATE_TASK_BEGIN(task1);
        *pointer = 999;               // Write
    ANNOTATE_TASK_END();
    assert(*pointer == 999);          // Read
    ANNOTATE_SITE_END();
}

```

In this example, one task writes a heap-allocated `int`, then an ancestor task reads it.

```
void data_communication()
{
    ANNOTATE_SITE_BEGIN(data_communication_site); // Parallel site
    {
        for (int i=0; i<N; i++) {
            ANNOTATE_TASK_BEGIN(data_communication_task1);
            {
                communication++; /* write in child */ // Write
            }
            ANNOTATE_TASK_END();
            printf("%d\n", communication); /* read in parent */ // Read
        }
        ANNOTATE_SITE_END();
    }
}
```

In this example, the incremented variable is read after each task. This creates a serial dependence.

Possible Correction Strategies

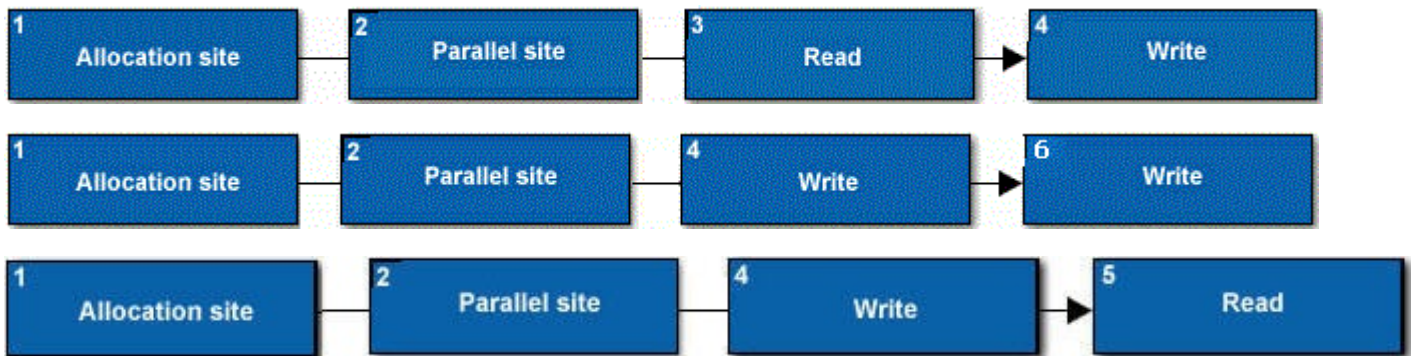
If you can preserve the application's integrity, consider moving the reads by the parent task into the child task. In the example above, this would result in non-deterministic output. If moving the read is not possible, you may need to use a different strategy, such as pipelining the loop.

Inconsistent Lock Use

Occurs when a task execution accesses a memory location more than once, under the control of different locks.

Syntax

One of the following has occurred:



ID	Code Location	Description
1	Allocation site	If present, represents the location and associated call stack when the memory was allocated.
2	Parallel site	If present, represents the location and associated call stack of the parallel site containing the Inconsistent Lock Use problem.
3	Read	Represents the location and associated call stack of the first access if it is a memory read.
4	Write	Represents the location and associated call stack of the second access if it is a memory write.

ID	Code Location	Description
5	Read	Represents the location and associated call stack of the second access if it is a memory read.
6	Write	Represents the location and associated call stack of the first access if it is a memory write.

Example

```
// Parallel site
ANNOTATE_TASK_BEGIN(task);
for (int i = 0; i < N; i++) {
    ANNOTATE_LOCK_ACQUIRE(1);
    a[i][j0]++;           // Read and/or Write
    ANNOTATE_LOCK_RELEASE(1);
}
for (int j = 0; i < N; i++) {
    ANNOTATE_LOCK_ACQUIRE(2);
    a[i0][j]++;           // Read and/or Write
    ANNOTATE_LOCK_RELEASE(2);
}
ANNOTATE_TASK_END();
```

In this example, `a[i0][j0]` is accessed under lock 1 in the first loop and under lock 2 in the second loop. It is likely that an access in another task will not have the right combination of locks to avoid conflicting with both these accesses.

Possible Correction Strategies

Lock all accesses to the same memory location with the same lock.

Lock Hierarchy Violation

Occurs when two or more locks are acquired in a different order in two task executions, potentially leading to a deadlock when the program's tasks execute in parallel.

A **Lock hierarchy violation** problem indicates the following timeline:

- | | |
|--------|--|
| Task 1 | <ol style="list-style-type: none"> 1. Acquire lock A. 2. Acquire lock B. 3. Release lock B. 4. Release lock A. |
| Task 2 | <ol style="list-style-type: none"> 1. Acquire lock B. 2. Acquire lock A. 3. Release lock A. 4. Release lock B. |

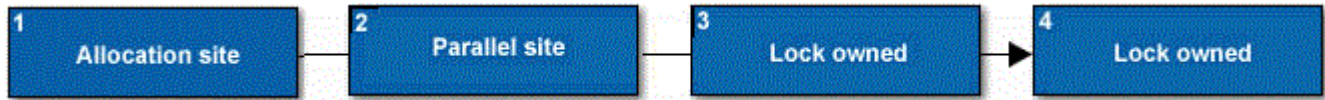
If these time lines are interleaved when the two tasks execute in parallel, a **Deadlock** occurs:

1. Task 1: Acquire lock A.
2. Task 2: Acquire lock B.
3. Task 1: Try to acquire lock B; wait until task 2 releases it.
4. Task 2: Try to acquire lock A; wait until task 1 releases it.

The Dependencies tool reports a **Lock hierarchy violation** as multiple problems in a problem set. Each problem shows a portion of the **Lock hierarchy violation** from the perspective of a single thread.

Lock hierarchy violation problems are the most common cause of **Deadlock** problems, and a report of a **Lock hierarchy violation** problem indicates a **Deadlock** problem might occur when the target executes in parallel.

Syntax



ID	Code Location	Description
1	Allocation site	If present, represents the location and its associated call stack where the synchronization object acquired by a thread (usually the object acquired first) was created.
2	Parallel site	If present, represents the location and associated call stack of the parallel site containing the Lock Hierarchy Violation problem.
3	Lock owned	Represents the location and associated call stack where a task acquired a lock.
4	Lock owned	Represents the location and associated call stack where a task acquired a second lock while the task still held the first lock.

Example

```

// in task 1
ANNOTATE_LOCK_ACQUIRE(&lahv_lock1);
ANNOTATE_LOCK_ACQUIRE(&lahv_lock2); /* lock hierarchy violation */
ANNOTATE_LOCK_RELEASE(&lahv_lock2);
ANNOTATE_LOCK_RELEASE(&lahv_lock1);

// in task 2
ANNOTATE_LOCK_ACQUIRE(&lahv_lock2);
ANNOTATE_LOCK_ACQUIRE(&lahv_lock1); /* lock hierarchy violation */
ANNOTATE_LOCK_RELEASE(&lahv_lock1);
ANNOTATE_LOCK_RELEASE(&lahv_lock2);
  
```

Possible Correction Strategies

Determine if interleaving is possible, or whether some other synchronization exists that might prevent interleaving. If interleaving is possible, consider the following options.

Use a single lock instead of multiple locks:

```

// in task 1
ANNOTATE_LOCK_ACQUIRE(&lahv_lock1);
a++;
b += a;
ANNOTATE_LOCK_RELEASE(&lahv_lock1);

// in task 2
ANNOTATE_LOCK_ACQUIRE(&lahv_lock2);
b += x[i];
a -= b;
ANNOTATE_LOCK_RELEASE(&lahv_lock2);
  
```

Try to define a consistent order for your locks, so that any task that acquires the same set of locks, will acquire them in the same order:

```
// in task 1
ANNOTATE_LOCK_ACQUIRE(&lahv_lock1);
a++;
b += a;
ANNOTATE_LOCK_RELEASE(&lahv_lock1);

// in task 2
ANNOTATE_LOCK_ACQUIRE(&lahv_lock2);
b += x[i];
a -= b;
ANNOTATE_LOCK_RELEASE(&lahv_lock2);
```

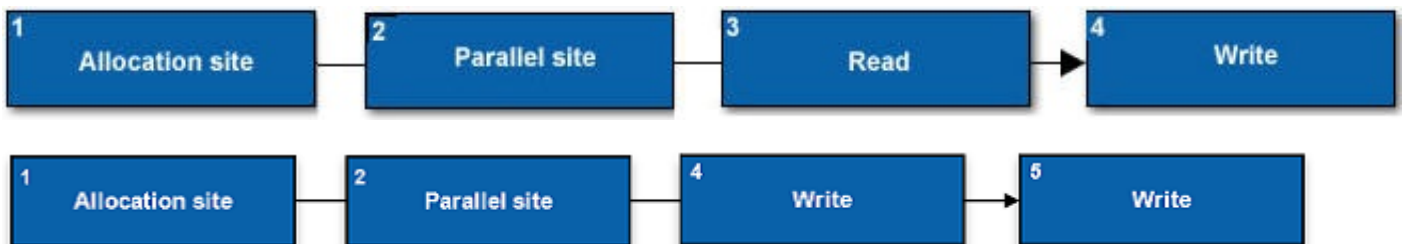
When a task acquires multiple locks, make sure that it always releases them in the opposite order that it acquired them.

Memory Reuse

Occurs when two tasks write to a shared memory location. That is, a task writes to a variable with a new value but does not read the same value generated by a prior task.

Syntax

One of the following has occurred:



ID	Code Location	Description
1	Allocation site	If present, and if the memory involved is heap memory, represents the location and associated call stack when the memory was allocated.
2	Parallel site	If present, represents the location and associated call stack of the parallel site containing the Memory Reuse problem.
3	Read	Represents the instruction and associated call stack of the first access if it is a memory read.
4	Write	Represents the instruction and associated call stack of the second access if it is a memory write.
5	Write	Represents the instruction and associated call stack of the first access if it is a memory write.

Example

```
int global;
void main()
{
    ANNOTATE_SITE_BEGIN(reuse_site);    // Begin parallel site
    ANNOTATE_TASK_BEGIN(task111);
```

```

    global = 111;           // Read and/or Write
    assert(global == 111);
    ANNOTATE_TASK_END();
    ANNOTATE_TASK_BEGIN(task222);
    global = 222;           // Write
    assert(global == 222);
    ANNOTATE_TASK_END();
    ANNOTATE_SITE_END();
}

```

In this example, two tasks use the same `global` variable. Each task does not read or communicate the value produced by the other task.

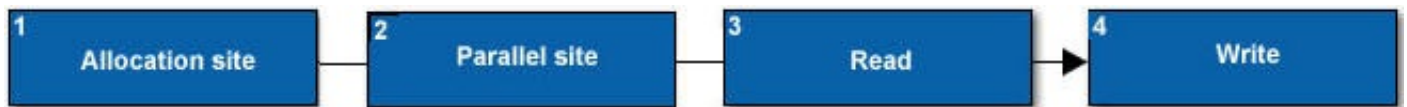
Some Possible Correction Strategies

Change the tasks to have their own private variables rather than sharing a variable.

Memory Reuse, Child Task

Occurs when two tasks write to a shared memory location, where a parent task overwrites a variable with a new value that was read by a previously executed child task. A child task is a task nested inside another task.

Syntax



ID	Code Location	Description
1	Allocation site	If present, and if the memory involved is heap memory, represents the location and associated call stack when the memory was allocated.
2	Parallel site	If present, represents the location and associated call stack of the parallel site containing the Memory Reuse, Child Task problem.
3	Read	Represents the instruction and associated call stack of the first access if it is a memory read.
4	Write	Represents the instruction and associated call stack of the second access if it is a memory write.

Example

```

int global;
void main()
{
    ANNOTATE_SITE_BEGIN(reuse_site); // Begin parallel site
    ANNOTATE_TASK_BEGIN(task111);
    assert(global == 111);           // Read
    ANNOTATE_TASK_END();
    global = 222;                     // Write
    ANNOTATE_SITE_END();
}

```

In this example, a parent task is writing to a shared variable after a task that reads that same variable.

Some Possible Correction Strategies

Create a private copy of the variable before executing the child task. Use the private copy in the child task.

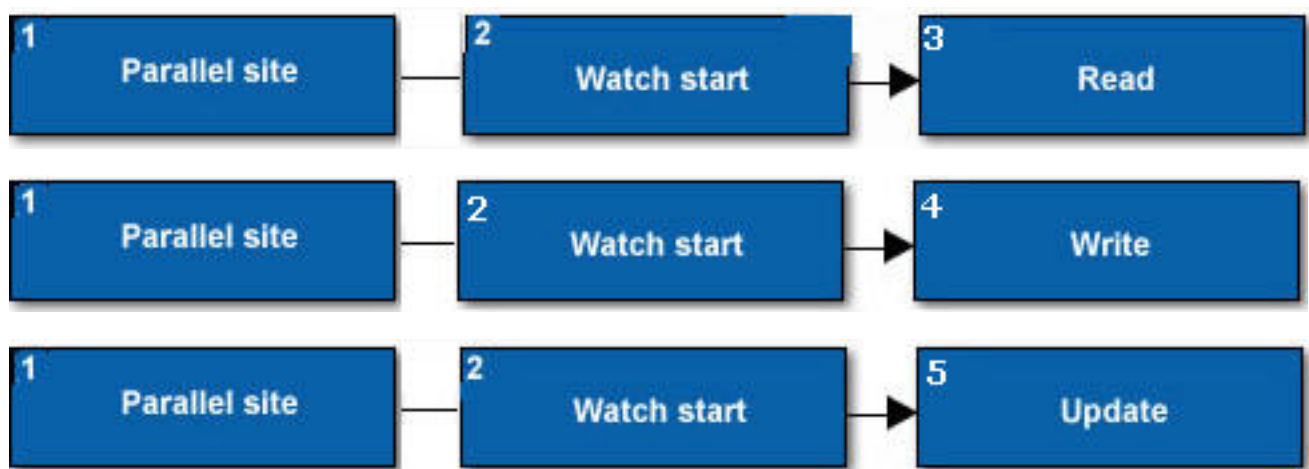
Memory Watch

Occurs when a task accesses a memory location marked by an `ANNOTATE_OBSERVE_USES` annotation.

In this case, this problem provides informational feedback only and no action is required. This is useful for finding uses of specified memory locations while a task is executing.

Syntax

One of the following has occurred:



ID	Code Location	Description
1	Parallel site	If present, represents the location and associated call stack of the parallel site containing the Memory Watch problem.
2	Watch start	Represents the location and its associated call stack where an <code>ANNOTATE_OBSERVE_USES()</code> annotation marks a memory location.
3	Read	Represents the location and associated call stack where a task read the watched memory location.
4	Write	Represents the location and associated call stack where a task wrote the watched memory location.
5	Update	Represents the location and associated call stack where a task read and wrote the watched memory location.

Example

```

void watch_memory()
{
    ANNOTATE_OBSERVE_USES(&watch, sizeof(watch)); // Watch start
    ANNOTATE_SITE_BEGIN(watch_site);               // Parallel site
    {
        ANNOTATE_TASK_BEGIN(watch_task1);
    }
}
  
```

```

{
    ANNOTATE_LOCK_ACQUIRE(&watch);
    watch++; /* watch memory */           // Read and/or Write
    ANNOTATE_LOCK_RELEASE(&watch);
}
ANNOTATE_TASK_END();
ANNOTATE_TASK_BEGIN(watch_task2);
{
    ANNOTATE_LOCK_ACQUIRE(&watch);
    watch++; /* watch memory */           // Read and/or Write
    ANNOTATE_LOCK_RELEASE(&watch);
}
ANNOTATE_TASK_END();
}
ANNOTATE_SITE_END();
ANNOTATE_CLEAR_USES(&watch);
}

```

This example reports all places that use the memory location referenced by `watch` during the call to `watch_memory()`.

Possible Correction Strategies

To use `ANNOTATE_OBSERVE_USES` to help you correct an incidental sharing problem, do the following to mark places where you may be able replace uses of a shared memory location with uses of a non-shared memory location:

1. Add an `ANNOTATE_OBSERVE_USES` annotation to the task.
2. Find all uses of the shared memory location in the dynamic extent of the task.

Missing End Site

Occurs when a site-begin annotation is executed but the corresponding site-end annotation is not executed before the thread or application exits.

Syntax



ID	Code Location	Description
1	Start site	Represents the location and the associated call stack when the parallel site execution began.

Example

```

void main()
{
    ANNOTATE_SITE_BEGIN(site1); // Begin parallel site
    return;
    ANNOTATE_SITE_END();
}

```

This example's execution skips the end-site annotation, `ANNOTATE_SITE_END()`.

Possible Correction Strategies

Always execute an `ANNOTATE_SITE_END()` annotation after executing an `ANNOTATE_SITE_BEGIN(sitename)` annotation. This omission can be caused by `throw` exceptions, `return`, `break`, `continue`, and `goto` statements or keywords. All control flow paths out of a site need to use the `ANNOTATE_SITE_END()` annotations.

Missing End Task

Occurs when a task-begin annotation is executed but the corresponding task-end annotation is not executed before the site, thread, or application exits.

Syntax



ID	Code Location	Description
1	Task start	Represents the location and associated call stack when the task began execution.
2	Parallel site	If present, represents the location and associated call stack of the beginning of the parallel site that contained the task.

Example

```

void main()
{
    ANNOTATE_SITE_BEGIN(site_name);
    ANNOTATE_TASK_BEGIN(taskname1);
    ANNOTATE_SITE_END();
}
  
```

This example lacks an end-task annotation, `ANNOTATE_TASK_END()`.

NOTE

An error also occurs if your code branches around a single `ANNOTATE_TASK_END()` annotation.

Possible Correction Strategies

Always execute an `ANNOTATE_TASK_BEGIN(taskname)` annotation before executing an `ANNOTATE_SITE_END()` annotation. This omission can be caused by `throw` exceptions, `return`, `break`, `continue`, and `goto` statements or keywords. All control flow paths out of a site need to use the `ANNOTATE_TASK_END()` annotations.

Missing Start Site

Occurs when an end-site annotation is executed but there is no active site.

Syntax



ID	Code Location	Description
1	End site	Represents the location and associated call stack when the end site annotation was executed.

Example

```

void main()
{
    ANNOTATE_SITE_END(); // End parallel site
}
  
```

This example executes an `ANNOTATE_SITE_END()` annotation before executing the corresponding (in this case, missing) `ANNOTATE_SITE_BEGIN(sitename)` annotation.

Possible Correction Strategies

Always execute an `ANNOTATE_SITE_BEGIN()` annotation before executing an `ANNOTATE_SITE_END()` annotation.

Missing Start Task

Occurs when an end task annotation is executed but there is no active task.

Syntax



ID	Code Location	Description
1	Task end	Represents the location and associated call stack when the task end annotation was executed.
2	Parallel site	If present, represents the location and associated call stack of the beginning of the parallel site that contained the task end annotation.

Example

```

void main()
{
    ANNOTATE_SITE_BEGIN(name_site1);
    ANNOTATE_TASK_END();
    ANNOTATE_SITE_END();
}
  
```

This example lacks an `ANNOTATE_TASK_BEGIN(taskname)` annotation.

NOTE

This error also occurs if your code branches around an `ANNOTATE_TASK_BEGIN(taskname)` annotation.

Possible Correction Strategies

Always execute an `ANNOTATE_TASK_BEGIN(taskname)` annotation before executing an `ANNOTATE_TASK_END()` annotation.

No Tasks in Parallel Site

Occurs when a parallel site was executed but no task annotations were executed in the dynamic extent of the active parallel site.

Syntax

1 Parallel site

ID	Code Location	Description
1	Parallel site	Represents the location and associated call stack of the parallel site. No task annotations were executed in the dynamic extent of the active parallel site.

Example

```
int global;
void main()
{
    ANNOTATE_SITE_BEGIN(reuse_site); // Parallel site
    assert(global == 111);
    global = 222;
    ANNOTATE_SITE_END();
}
```

In this example, the site begin and site end annotations are present, but the execution paths within the parallel site do not execute any task annotations.

Some Possible Correction Strategies

Check the execution paths within the parallel site and add task annotations to mark at least one task.

One Task Instance in Parallel Site

Occurs when a parallel site was executed but annotations for only one task instance were executed in the dynamic extent of the active parallel site. This may be the expected behavior, or it may indicate an error in the placement of annotations or a data set that is not well suited for parallelism.

Syntax

1 Parallel site

ID	Code Location	Description
1	Parallel site	Represents the location and associated call stack of the parallel site. Occurs when a parallel site was executed but annotations for only one instance of a task's code were executed in the dynamic extent of the active parallel site. The warning is based on the site and task annotations detected during program execution. This may be the expected behavior. In other cases, this warning may indicate an error in the placement of annotations or a data set that is not well suited for parallelism (a single instance of a task may not contribute to parallel execution speed-up).

Example

```
int global;
extern arg_map parse_args(int argc, char ** argv);
void main(int argc, char * argv[])
{
    int x;
    parse_args(argc, argv);
    int y = arg_map.get("iterations"); //command line specifies 1 iteration

    ANNOTATE_SITE_BEGIN(loopsite);      // Parallel site
    for (x=0; x<y; x++)
    {
        ANNOTATE_ITERATION_TASK(task);
        ...
    }
    ANNOTATE_SITE_END();
}
```

In this example, the selected data set results in only a single iteration of the loop. No dependencies will be found between multiple iterations of the loop.

Some Possible Correction Strategies

Check the execution paths within the parallel site and confirm that you intended to have only one task for this parallel site. If needed, examine the loop structure and its scaling characteristics (reported by the Suitability tool) to ensure that this parallel site does not need additional tasks. If the problem is caused by using too small a data set, increase the size of your data set.

Orphaned Task

Occurs when a task-begin annotation is executed that is not within an active parallel site.

Syntax



ID	Code Location	Description
1	Task start	Represents the location and associated call stack when the task began execution.

Example

```
void main()
{
    ANNOTATE_TASK_BEGIN(name_task1); // Begin task
    ANNOTATE_TASK_END();
}
```

This example does not execute a `ANNOTATE_SITE_BEGIN(sitename)/ANNOTATE_SITE_END()` annotation pair required to wrap the execution of a task `ANNOTATE_TASK_BEGIN(taskname)/ANNOTATE_TASK_END()` annotation pair.

Possible Correction Strategies

Always execute an `ANNOTATE_SITE_BEGIN(sitename)` annotation before executing an `ANNOTATE_TASK_BEGIN(taskname)` annotation. You may need to add an `ANNOTATE_SITE_BEGIN(sitename) /ANNOTATE_SITE_END()` annotation pair in the same function, or in some calling function.

An orphaned task is effectively ignored by the Suitability and Dependencies tool analysis, so you should fix the orphaned task code and run the Suitability and Dependencies tools again.

Parallel Site Information

Occurs when execution enters a parallel site. This confirms that your program and its data are executing the annotations you inserted during execution of the Dependencies tool analysis. In this case, this message provides informational feedback only and no action is required.

Syntax



ID	Code Location	Description
1	Parallel site	Represents the location and associated call stack of the parallel site.

Example

```
ANNOTATE_SITE_BEGIN(name_site1); // Begin parallel site
for (i = 0; i < n; ++i)
{
    ANNOTATE_ITERATION_TASK(name_task1);
    process(i);
}
ANNOTATE_SITE_END(); // End parallel site
```

See Also

[Dependencies Tool Overview](#)

Thread Information

Occurs when a thread is created. In this case, this message provides informational feedback and no action is required.

Expect at least one such message when the main program's thread is created by the operating system. If you are running the Dependencies tool on a partially parallelized target, expect additional messages for each thread the program creates.

The creation site of the main program thread is the point where the `main()` function - or other standard entry point line `_wmain()` - is called from the startup initialization code.

Syntax



ID	Code Location	Description
1	Creation site	Represents the location and call stack where a thread was created.

Example

```
ANNOTATE_SITE_BEGIN(name1);
for (int i = 0; i < n; ++i)
{
    ANNOTATE_ITERATION_TASK(task_process_array);
    process(array1[i]);
}
ANNOTATE_SITE_END();
...
// create thread using parallel framework code or CreateThread()
for (int i = 0; i < n; ++i)
{
    process(array2[i]);
}
```

Unhandled Application Exception

Occurs when an unhandled exception is detected that causes the application program to crash.

Syntax



ID	Code Location	Description
1	Exception	Represents the instruction that threw the exception.

Example

```
void problem1(int *y)
{
    *y = 5;
}

void problem2()
{
    int x = new int;
}
```

In these (simplified) example functions, two exceptions are possible:

- Variable `y` may not reference a valid memory location and therefore the write may cause an exception to be thrown. If that exception is not properly handled, the Dependencies Report will show an **Unhandled application exception** pointing to the write of `y`.
- If the process is out of memory, the allocation will throw an exception. If the exception is not handled, the Dependencies Report will show an **Unhandled application exception** associated with the allocation.

Because of the abnormal process termination (crash), the Dependencies tool may also report a **Missing end task** and **Missing end site** problem.

Possible Correction Strategies

This problem usually exposes an existing bug in your application that appears when the application is run with the Dependencies tool.

See Also

[Dependencies Tool Overview](#)

Recommendation Reference

Explore Intel® Advisor recommendations to get hints on how to optimize your application and achieve better performance.

The following sections describe Intel® Advisor recommendations that you can find in the **Recommendations** tab of the result window:

- [Recommendations for C++](#)
- [Recommendations for Fortran](#)

Vectorization Recommendations for C++

Ineffective Peeled/Remainder Loop(s) Present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops to the loop body.

Align Data

One of the memory accesses in the source loop does not start at an optimally aligned address boundary. To fix: Align the data and tell the compiler the data is aligned.

Align dynamic data using a 64-byte boundary and tell the compiler the data is aligned:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);
// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
_mm_free(array);
```

Align static data using a 64-byte boundary:

```
__declspec(align(64)) float array[ARRAY_SIZE]
```

See also:

- [align](#)
- [Data Alignment to Assist Vectorization](#) and [Vectorization Resources for Intel® Advisor Users](#)

Parallelize The Loop with Both Threads and SIMD Instructions

The loop is threaded and auto-vectorized; however, the trip count is not a multiple of vector length. To fix: Do all of the following:

- Use the `#pragma omp parallel for simd` directive to parallelize the loop with both threads and SIMD instructions. Specifically, this directive divides loop iterations into chunks (subsets) and distributes the chunks among threads, then chunk iterations execute concurrently using SIMD instructions.
- Add the `schedule(simd: [kind])` modifier to the directive to guarantee the chunk size (number of iterations per chunk) is a multiple of vector length.

Original code sample:

```
void f(int a[], int b[], int c[])
{
    #pragma omp parallel for schedule(static)
    for (int i = 0; i < n; i++)
    {
        a[i] = b[i] + c[i];
    }
}
```

Revised code sample:

```
void f(int a[], int b[], int c[])
{
    #pragma omp parallel for simd schedule(simd:static)
    for (int i = 0; i < n; i++)
    {
        a[i] = b[i] + c[i];
    }
}
```

See also:

- [OpenMP Application Programming Interface](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Force Scalar Remainder Generation

The compiler generated a masked vectorized remainder loop that contains too few iterations for efficient vector processing. A scalar loop may be more beneficial. To fix: Force scalar remainder generation using a directive: `#pragma vector novectorremainder`.

```
void add_floats(float *a, float *b, float *c, float *d, float *e, int n)
{
    int i;
    // Force the compiler to not vectorize the remainder loop
    #pragma vector novectorremainder
    for (i=0; i<n; i++)
    {
        a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
    }
}
```

See also:

- [vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Force Vectorized Remainder

The compiler did not vectorize the remainder loop, even though doing so could improve performance. To fix: Force vectorization using a directive: `#pragma vector vecremainder`.

```
void add_floats(float *a, float *b, float *c, float *d, float *e, int n)
{
    int i;
    // Force the compiler to vectorize the remainder loop
    #pragma vector vecremainder
    for (i=0; i<n; i++)
    {
        a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
    }
}
```

See also:

- [vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Specify The Expected Loop Trip Count

The compiler cannot detect the trip count statically. To fix: Specify the expected number of iterations using a directive: `#pragma loop_count`.

```
#include <stdio.h>

int mysum(int start, int end, int a)
{
    int iret=0;
    // Iterate through a loop a minimum of three, maximum of ten, and average of five times
    #pragma loop_count min(3), max(10), avg(5)
    for (int i=start; i<=end; i++)
        iret += a;
    return iret;
}

int main()
{
    int t;
    t = mysum(1, 10, 3);
    printf("t1=%d\r\n", t);
    t = mysum(2, 6, 2);
    printf("t2=%d\r\n", t);
    t = mysum(5, 12, 1);
    printf("t3=%d\r\n", t);
}
```

See also:

- [loop_count](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Change The Chunk Size

The loop is threaded and vectorized using the `#pragma omp parallel for simd` directive, which parallelizes the loop with both threads and SIMD instructions. Specifically, the directive divides loop iterations into chunks (subsets) and distributes the chunks among threads, then chunk iterations execute concurrently using SIMD instructions. In this case, the chunk size (number of iterations per chunk) is not a multiple of vector length. To fix: Add a `schedule (simd: [kind])` modifier to the `#pragma omp parallel for simd` directive.

```
void f(int a[], int b[], int c[])
{
    // Guarantee a multiple of vector length.
    #pragma omp parallel for simd schedule(simd: static)
    for (int i = 0; i < n; i++)
    {
        a[i] = b[i] + c[i];
    }
}
```

See also:

- [OpenMP Application Programming Interface](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Add Data Padding

The trip count is not a multiple of vector length . To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

See also:

- [loop_count](#)
- [Utilizing Full Vectors](#) and [Vectorization Resources for Intel® Advisor Users](#)

Collect Trip Counts Data

The Survey Report lacks trip counts data that might generate more precise recommendations.

Disable Unrolling

The trip count after loop unrolling is too small compared to the vector length . To fix: Prevent loop unrolling or decrease the unroll factor using a directive: `#pragma nounroll` or `#pragma unroll`.

```
void nounroll(int a[], int b[], int c[], int d[])
{
    // Disable automatic loop unrolling using
    #pragma nounroll
    for (int i = 1; i < 100; i++)
    {
        b[i] = a[i] + 1;
        d[i] = c[i] + 1;
    }
}
```

See also:

- [unroll/nounroll](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use A Smaller Vector Length

The compiler chose a vector length of , but the trip count might be smaller than the vector length. To fix: Specify a smaller vector length using a directive: `#pragma omp simd simdlen`.

```
void f(int a[], int b[], int c[], int d[])
{
    // Specify vector length using
    #pragma omp simd simdlen(4)
    for (int i = 1; i < 100; i++)
    {
        b[i] = a[i] + 1;
        d[i] = c[i] + 1;
    }
}
```

In Intel Compiler version 19.0 and higher, there is a new vector length clause that allows the compiler to choose the best vector length based on cost: `#pragma vector vectorlength(vl1, vl2, ..., vln)` where `vl` is an integer power of 2.

```
void f(int a[], int b[], int c[], int d[])
{
    // Specify list of vector lengths
    #pragma vector vectorlength(2, 4, 16)
    for (int i = 1; i < 100; i++)
    {
        b[i] = a[i] + 1;
        d[i] = c[i] + 1;
    }
}
```

See also:

- `omp simd` in [OpenMP Pragmas Summary](#), [vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Disable Dynamic Alignment

The compiler automatically peeled iterations from the vector loop into a scalar loop to align the vector loop with a particular memory reference; however, this optimization may not be ideal. To possibly achieve better performance, disable automatic peel generation using the directive: `#pragma vector nodynamic_align`.

```
...
#pragma vector nodynamic_align
for (int i = 0; i < len; i++)
...
void f(float * a, float * b, float * c, int len)
{
    #pragma vector nodynamic_align
    for (int i = 0; i < len; i++)
    {
        a[i] = b[i] * c[i];
    }
}
```

See also:

- [vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Serialized User Function Call(s) Present

User-defined functions in the loop body are not vectorized.

Enable Inline Expansion

Inlining of user-defined functions is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

See also:

- [inline-level, Ob](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vectorize Serialized Function(s) Inside Loop

```
#pragma omp declare simd
int f (int x)
{
    return x+1;
}
#pragma omp simd
for (int k = 0; k < N; k++)
{
    a[k] = f(k);
}
```

See also:

- `omp simd`, `omp declare simd` in [OpenMP Pragmas Summary](#), [vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Scalar Math Function Call(s) Present

Math functions in the loop body are preventing the compiler from effectively vectorizing the loop. Improve performance by enabling vectorized math call(s).

Enable Inline Expansion

Inlining is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

Alternatively use `#include <mathimf.h>` header instead of the standard `#include <math.h>` header to call highly optimized and accurate mathematical functions commonly used in applications that rely heavily on floating point computations.

See also:

- [inline-level, Ob](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vectorize Math Function Calls Inside Loops

Your application calls serialized versions of math functions when you use the precise floating point model. To fix: Do one of the following:

- Add `fast-transcendentals` compiler option to replace calls to transcendental functions with faster calls.
-

Caution This may reduce floating point accuracy.

- Enforce vectorization of the source loop using a directive: `#pragma omp simd`

```
void add_floats(float *a, float *b, float *c, float *d, float *e, int n)
{
    int i;
    #pragma omp simd
    for (i=0; i<n; i++)
    {
        a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
    }
}
```

See also:

- [fast-transcendentals, Qfast-transcendentals; `omp simd` in OpenMP Pragmas Summary, vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Change The Floating Point Model

Your application calls serialized versions of math functions when you use the `strict` floating point model. To fix: Do one of the following:

- Use the `fast` floating point model to enable more aggressive optimizations or the `precise` floating point model to disable optimizations that are not value-safe on fast transcendental functions.

Caution This may reduce floating point accuracy.

- Use the `precise` floating point model and enforce vectorization of the source loop using a directive: `#pragma omp simd`

```
gcc program.c -O2 -fopenmp -fp-model precise -fast-transcendentals
#pragma omp simd collapse(2)
for (i=0; i<N; i++)
{
    a[i] = b[i] * c[i];
    for (i=0; i<N; i++)
    {
        d[i] = e[i] * f[i];
    }
}
```

See also:

- [fast-transcendentals, Qfast-transcendentals; `omp simd` in OpenMP Pragmas Summary, vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use a Glibc Library with Vectorized SVML Functions

Your application calls scalar instead of vectorized versions of math functions. To fix: Do all of the following:

- Upgrade the Glibc library to version 2.22 or higher. It supports SIMD directives in OpenMP* 4.0 or higher.
- Upgrade the GNU* gcc compiler to version 4.9 or higher. It supports vectorized math function options.
- Use the `-fopenmp` and `-ffast-math` compiler options to enable vector math functions.
- Use appropriate OpenMP SIMD directives to enable vectorization.

NOTE Also use the `-I/path/to/glibc/install/include` and `-L/path/to/glibc/install/lib` compiler options if you have multiple Glibc libraries installed on the host.

```
gcc program.c -O2 -fopenmp -ffast-math -lrt -lm -mavx2 -I/opt/glibc-2.22/include -L/opt/glibc-2.22/lib -Wl,--dynamic-linker=/opt/glibc-2.22/lib/ld-linux-x86-64.so.2
#include "math.h"
#include "stdio.h"
#define N 100000

int main()
{
    double angles[N], results[N];
    int i;
    srand(86456);

    for (i = 0; i < N; i++)
    {
        angles[i] = rand();
    }

    #pragma omp simd
    for (i = 0; i < N; i++)
    {
        results[i] = cos(angles[i]);
    }

    return 0;
}
```

See also:

- [Glibc wiki/libmvec](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use The Intel Short Vector Math Library for Vector Intrinsics

Your application calls scalar instead of vectorized versions of math functions. To fix: Do all of the following:

- Use the `-mveclibabi=svml` compiler option to specify the Intel short vector math library ABI type for vector intrinsics.
- Use the `-ftree-vectorize` and `-funsafe-math-optimizations` compiler options to enable vector math functions.
- Use the `-L/path/to/intel/lib` and `-lsvml` compiler options to specify an SVML ABI-compatible library at link time.

```
gcc program.c -O2 -ftree-vectorize -funsafe-math-optimizations -mveclibabi=svml -L/opt/intel/lib/intel64 -lm -lsvml -Wl,-rpath=/opt/intel/lib/intel64
#include "math.h"
#include "stdio.h"
#define N 100000

int main()
{
    double angles[N], results[N];
    int i;
    srand(86456);

    for (i = 0; i < N; i++)
    {
```

```

    angles[i] = rand();
}

// the loop will be auto-vectorized
for (i = 0; i < N; i++)
{
    results[i] = cos(angles[i]);
}

return 0;
}

```

See also:

- [GCC Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Inefficient Gather/Scatter Instructions Present

The compiler assumes indirect or irregular stride access to data used for vector operations. Improve memory access by alerting the compiler to detected regular stride access patterns, such as:

Refactor code with detected regular stride access patterns

The Memory Access Patterns Report shows the following regular stride access(es):

See details in the Memory Access Patterns Report Source Details view.

To improve memory access: Refactor your code to alert the compiler to a regular stride access. Sometimes, it might be beneficial to use the `ipo/Qipo` compiler option to enable interprocedural optimization (IPO) between files.

An array is the most common type of data structure containing a contiguous collection of data items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). Detected constant stride might be the result of AoS implementation. While this organization is excellent for encapsulation, it can hinder effective vector processing. To fix: Rewrite code to organize data using SoA instead of AoS.

However, the cost of rewriting code to organize data using SoA instead of AoS may outweigh the benefit. To fix: Use Intel SIMD Data Layout Templates (Intel SDLT), introduced in version 16.1 of the Intel compiler, to mitigate the cost. Intel SDLT is a C++11 template library that may reduce code rewrites to just a few lines.

Refactor for vertical invariant pattern.

```

// main.cpp
int a[8] = {1,0,5,7,4,2,6,3};

// gather.cpp
void test_gather(int* a, int* b, int* c, int* d)
{
    int i, k;
    // inefficient access
    #pragma omp simd
    for (i = 0; i < INNER_COUNT; i++)
        d[i] = b[a[i%8]] + c[i];

    int b_alt[8];
    for (k = 0; k < 8; ++k)
        b_alt[k] = b[a[k]];

    // more effective version

```

```

    for (i = 0; i < INNER_COUNT/8; i++)
    {
#pragma omp simd
        for(k = 0; k < 8; ++k)
            d[i*8+k] = b_alt[k] + c[i*8+k];
    }
}

```

Also make sure vector function clauses match arguments in the calls within the loop (if any).

NOTE You may use several `#pragma declare simd` directives to tell the compiler to generate several vector variants of a function.

Compare function calls with their declarations.

```

// functions.cpp
#pragma omp declare simd
int foo1(int* arr, int idx) { return 2 * arr[idx]; }

#pragma omp declare simd uniform(arr) linear(idx)
int foo2(int* arr, int idx) { return 2 * arr[idx]; }

#pragma omp declare simd linear(arr) uniform(idx)
int foo3(int* arr, int idx) { return 2 * arr[idx]; }

// gather.cpp
void test_gather(int* a, int* b, int* c)
{
    int i, k;

    // Loop will be vectorized, for complex access patterns gathers could be used for function call.
    #pragma omp simd
        for (i = 0; i < INNER_COUNT; i++) a[i] = b[i] + foo1(c,i);

    // Loop will be vectorized with vectorized call
    #pragma omp simd
        for (i = 0; i < INNER_COUNT; i++) a[i] = b[i] + foo2(c,i);

    // Loop will be vectorized with serialized function call
    #pragma omp simd
        for (i = 0; i < INNER_COUNT; i++) a[i] = b[i] + foo3(c,i);
}

```

See also:

- [ipo, Qipo; omp simd, omp declare simd in OpenMP* Pragmas Summary](#)
- [Case study: Comparing Arrays of Structures and Structures of Arrays Data Layouts for a Compute-Intensive Loop](#)
- [Introduction to the Intel® SIMD Data Layout Templates \(Intel®SDLT\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vector Register Spinning Possible

Possible register spilling was detected and all vector registers are in use. This may negatively impact performance, because the spilled variable must be loaded to and unloaded from main memory. Improve performance by decreasing vector register pressure.

Decrease Unroll Factor

The current directive unroll factor increases vector register pressure. To fix: Decrease unroll factor using a directive: `#pragma nounroll` or `#pragma unroll`.

```
void nounroll(int a[], int b[], int c[], int d[])
{
    #pragma nounroll
    for (int i = 1; i < 100; i++)
    {
        b[i] = a[i] + 1;
        d[i] = c[i] + 1;
    }
}
```

See also:

- [unroll/nounroll](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Split Loop into Smaller Loops

Possible register spilling along with high vector register pressure is preventing effective vectorization. To fix: Use the directive `#pragma distribute_point` or rewrite your code to distribute the source loop. This can decrease register pressure as well as enable www pipelining and improve both instruction and data cache use.

```
#define NUM 1024
void loop_distribution_pragma2(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
    int i;
    // After distribution or splitting the loop.
    for (i=0; i< NUM; i++)
    {
        a[i] = a[i] +i;
        b[i] = b[i] +i;
        c[i] = c[i] +i;
        #pragma distribute_point
        x[i] = x[i] +i;
        y[i] = y[i] +i;
        z[i] = z[i] +i;
    }
}
```

See also:

- [distribute_point](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Assumed Dependency Present

The compiler assumed there is an anti-dependency (Write after read - WAR) or a true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Confirm Dependency Is Real

There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a [Dependencies analysis](#).

Enable Vectorization

The Dependencies analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a directive:

```
#pragma ivdep
for (i = 0; i < n - 4; i += 4)
{
    // Here another line of comments for demonstration of
    // easy to use code sample...
    a[i + 4] = a[i] * c;
}
```

See also:

- [ivdep, omp simd in OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Proven (Real) Dependency Is Present

The compiler assumed there is an anti-dependency (Write after read - WAR) or true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Resolve Dependency

The Dependencies analysis shows there is a real (proven) dependency in the loop. To fix: Do one of the following:

- If there is an anti-dependency, enable vectorization using the directive `#pragma omp simd safelen(length)`, where `length` is smaller than the distance between dependent iterations in anti-dependency.

```
#pragma omp simd safelen(4)
for (i = 0; i < n - 4; i += 4)
{
    a[i + 4] = a[i] * c;
}
```

- If there is a reduction pattern dependency in the loop, enable vectorization using the directive `#pragma omp simd reduction(operator:list)`.

```
#pragma omp simd reduction(+:sumx)
for (k = 0; k < size2; k++)
{
    sumx += x[k]*b[k];
}
```

- Rewrite the code to remove the dependency. Use programming techniques such as variable privatization.

See also:

- [omp simd in OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Data Type Conversions Present

There are multiple data types within loops. Utilize hardware vectorization support more effectively by avoiding data type conversion.

Use The Smallest Data Type

The source loop contains data types of different widths. To fix: Use the smallest data type that gives the needed precision to use the entire vector register width.

Example: If only 16-bits are needed, using a short rather than an int can make the difference between eight-way or four-way SIMD parallelism, respectively.

User Function Call(s) Present

User-defined functions in the loop body are preventing the compiler from vectorizing the loop.

Enable Inline Expansion

Inlining of user-defined functions is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

See also:

- [inline-level, Ob](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vectorize User Function(s) Inside Loop

These user-defined function(s) are not vectorized or inlined by the compiler: `my_calc()` To fix: Do one of the following:

- Enforce vectorization of the source loop by means of SIMD instructions and/or create a SIMD version of the function(s) using a directive:

Target	Directive
Source loop	<code>#pragma omp simd</code>
Inner function definition or declaration	<code>#pragma omp declare simd</code>

```
#pragma omp declare simd
int f (int x)
{
    return x+1;
}
#pragma omp simd
for (int k = 0; k < N; k++)
{
    a[k] = f(k);
}
```

See also:

- `omp simd`, `omp declare simd` in [OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Compiler Lacks Sufficient Information to Vectorize Loop

Cause: You are using a non-Intel compiler or an outdated Intel compiler. Nevertheless, it appears there are no issues preventing vectorization and vectorization may be profitable.

Explore Vectorization Opportunities

You compiled with auto-vectorization enabled; however, the compiler did not vectorize the code. Explore vectorization opportunities:

- Run a Dependencies analysis to identify real data dependencies that could make forced vectorization unsafe.
- Microsoft Visual C++* compiler: Use the `Qvec-report` compiler option (i.e. `/Qvec-report:2`)
- Auto-Vectorizer Reporting Level to output missed optimization opportunities.
- GNU* gcc compiler, do one of the following:
 - Use the `fopt-info-vec-missed` compiler option to output missed optimization opportunities.
 - Use the OpenMP* `omp simd` directive to tell the compiler it is safe to vectorize.
 - Use additional auto-vectorization directives.

See also:

- [Visual Studio 2015/Visual C++ Compiler Options Listed Alphabetically](#)
- [GCC online documentation](#)
- [OpenMP Resources](#)

Enable Auto-Vectorization

You compiled with auto-vectorization disabled; enable auto-vectorization:

- Intel compiler 14.x or below: Increase the optimization level to O2 or O3.
- Microsoft Visual C++* compiler: Increase the optimization level to O2 or O3.
- GNU* gcc compiler, do one of the following:
 - Increase the optimization level to O2 or O3.
 - Use the `fvec`-vectorize compiler option.

See also:

- [Visual Studio 2015/Visual C++ Compiler Options Listed Alphabetically](#)
- [GCC online documentation](#)

System Function Call(s) Present

System function call(s) in the loop body are preventing the compiler from vectorizing the loop.

Remove System Function Call(s) Inside Loop

Typically system function or subroutine calls cannot be vectorized; even a print statement is sufficient to prevent vectorization. To fix: Avoid using system function calls in loops.

OpenMP* Function Call(s) Present

OpenMP* function call(s) in the loop body are preventing the compiler from effectively vectorizing the loop.

Move OpenMP Call(s) Outside The Loop Body

OpenMP calls prevent automatic vectorization when the compiler cannot move the calls outside the loop body, such as when OpenMP calls are not invariant. To fix:

1. Split the OpenMP parallel loop directive into two directives.
2. Move the OpenMP calls outside the loop when possible.

Original code example:

```
#pragma omp parallel for private(tid, nthreads)
for (int k = 0; k < N; k++)
{
    tid = omp_get_thread_num(); // this call inside loop prevents vectorization
    nthreads = omp_get_num_threads(); // this call inside loop prevents vectorization
    ...
}
```

Revised code example:

```
#pragma omp parallel private(tid, nthreads)
{
    // Move OpenMP calls here
    tid = omp_get_thread_num();
    nthreads = omp_get_num_threads();

    #pragma omp for nowait
    for (int k = 0; k < N; k++)
    {
```

```

    ...
}
}

```

See also:

- *omp for, omp parallel recommendations* in [OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Remove OpenMP Lock Functions

Locking objects slows loop execution. To fix: Rewrite the code without OpenMP lock functions.

Allocating separate arrays for each thread and then merging them after a parallel recommendation may improve speed (but consume more memory).

Original code example:

```

int A[n];
list<int> L;
...
omp_lock_t lock_obj;
omp_init_lock(&lock_obj);
#pragma omp parallel for shared(L, A, lock_obj) default(none)
for (int i = 0; i < n; ++i)
{
    // A[i] calculation
    ...
    if (A[i]<1.0)
    {
        omp_set_lock(&(lock_obj));
        L.insert(L.begin(), A[i]);
        omp_unset_lock(&(lock_obj));
    }
}
omp_destroy_lock(&lock_obj);

```

Revised code example:

```

int A[n];
list<int> L;
omp_set_num_threads(nthreads_all);
...
vector<list<int>> L_by_thread(nthreads_all); // separate list for each thread
#pragma omp parallel shared(L, L_by_thread, A) default(none)
{
    int k = omp_get_thread_num();
    #pragma omp for nowait
    for (int i = 0; i < n; ++i)
    {
        // A[i] calculation
        ...
        if (A[i]<1.0)
        {
            L_by_thread[k].insert(L_by_thread[k].begin(), A[i]);
        }
    }
}

// merge data into single list
for (int k = 0; k < L_by_thread.size(); k++)

```

```
{
    L.splice(L.end(), L_by_thread[k]);
}
```

See also:

- Calling Functions on the CPU to Modify the Coprocessor's Execution Environment; Lock Routines recommendation in [OpenMP Run-time Library Routines](#); *omp for*, *omp parallel* recommendations in [OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Potential Inefficient Memory Access Patterns Present

Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.

Confirm Inefficient Memory Access Patterns

There is no confirmation inefficient memory access patterns are present. To fix: Run a [Memory Access Patterns](#) analysis.

Inefficient Memory Access Patterns Present

There is a high of percentage memory instructions with irregular (variable or random) stride accesses. Improve performance by investigating and handling accordingly.

Reorder Loops

This loop has less efficient memory access patterns than a nearby outer loop. To fix: Reorder the loops if possible.

Original code example:

```
void matmul(float *a[], float *b[], float *c[], int N) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Revised code example:

```
void matmul(float *a[], float *b[], float *c[], int N) {
    for (int i = 0; i < N; i++)
        for (int k = 0; k < N; k++)
            for (int j = 0; j < N; j++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Interchanging is not always possible because of dependencies, which can lead to different results.

Use Intel SDLT

The cost of rewriting code to organize data using SoA instead of AoS may outweigh the benefit. To fix: Use Intel SIMD Data Layout Templates (Intel SDLT), introduced in version 16.1 of the Intel compiler, to mitigate the cost. Intel SDLT is a C++11 template library that may reduce code rewrites to just a few lines.

Using SDLT instead of STL containers may improve the memory access pattern for more efficient vector processing.

Original code example:

```

struct kValues {
    float Kx;
    float Ky;
    float Kz;
    float PhiMag;
};

std::vector<kValues> dataset(count);

// Initialization step
for(int i=0; i < count; ++i) {
    kValues[i].Kx = kx[i];
    kValues[i].Ky = ky[i];
    kValues[i].Kz = kz[i];
    kValues[i].PhiMag = phiMag[i];
}

// Calculation step
for (indexK = 0; indexK < numK; indexK++) {
    expArg = PIx2 * (kValues[indexK].Kx * x[indexX] +
        kValues[indexK].Ky * y[indexX] +
        kValues[indexK].Kz * z[indexX]);
    cosArg = cosf(expArg);
    sinArg = sinf(expArg);
    float phi = kValues[indexK].PhiMag;
    QrSum += phi * cosArg;
    QiSum += phi * sinArg;
}

```

Revised code example:

```

#include <sdl/sdl.h>

struct kValues {
    float Kx;
    float Ky;
    float Kz;
    float PhiMag;
};
SDLT_PRIMITIVE(kValues, Kx, Ky, Kz, PhiMag)

sdl::soald_container<kValues> dataset(count);

// Initialization step
auto kValues = dataset.access();
for (k = 0; k < numK; k++) {
    kValues [k].Kx() = kx[k];
    kValues [k].Ky() = ky[k];
    kValues [k].Kz() = kz[k];
    kValues [k].PhiMag() = phiMag[k];
}

// Calculation step
auto kVals = dataset.const_access();
#pragma omp simd private(expArg, cosArg, sinArg) reduction(+:QrSum, QiSum)
for (indexK = 0; indexK < numK; indexK++) {
    expArg = PIx2 * (kVals[indexK].Kx() * x[indexX] +
        kVals[indexK].Ky() * y[indexX] +

```

```

    kVals[indexK].Kz() * z[indexX]);
    cosArg = cosf(expArg);
    sinArg = sinf(expArg);
    float phi = kVals[indexK].PhiMag();
    QrSum += phi * cosArg;
    QiSum += phi * sinArg;
}

```

See also:

- [Introduction to the Intel® SIMD Data Layout Templates \(Intel® SDLT\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use SoA Instead of AoS

An array is the most common type of data structure containing a contiguous collection of data items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). While AoS organization is excellent for encapsulation, it can hinder effective vector processing. To fix: Rewrite code to organize data using SoA instead of AoS.

See also:

- [Programming Guidelines for Vectorization](#)
- [Case study: Comparing Arrays of Structures and Structures of Arrays Data Layouts for a Compute-Intensive Loop](#) and [Vectorization Resources for Intel® Advisor Users](#)

Potential Underutilization of FMA Instructions

Your current hardware supports the AVX2 instruction set architecture (ISA), which enables the use of fused multiply-add (FMA) instructions. Improve performance by utilizing FMA instructions.

Force Vectorization If Possible

The loop contains FMA instructions (so vectorization could be beneficial), but is not vectorized. To fix, review:

- Corresponding compiler diagnostic to check if vectorization enforcement is possible and profitable
- The Dependencies analysis to distinguish between compiler-assumed dependencies and real dependencies

See also:

- [Vectorization Resources for Intel® Advisor Users](#)

Explicitly Enable FMA Generation When Using The Strict Floating-Point Model

Static analysis presumes the loop may benefit from FMA instructions available with the AVX2 ISA, but the `strict` floating-point model disables FMA instruction generation by default. To fix: Override this behavior using the `fma` compiler option.

Windows OS	Linux OS
/Qfma	-fma

See also:

- [fma, Qfma](#)
- [Floating-point Operations](#) and [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Target The Higher ISA

Although static analysis presumes the loop may benefit from FMA instructions available with the AVX2 or higher ISA, no FMA instructions executed for this loop. To fix: Use the following compiler options:

- `xCORE-AVX2` to compile for machines with and without AVX2 support
- `axCORE-AVX2` to compile for machines with AVX2 support only

- `xCOMMON-AVX512` to compile for machines with AVX-512 support only
- `axCOMMON-AVX512` to compile for machines with and without AVX-512 support

NOTE The compiler options may vary depending on the CPU microarchitecture.

See also:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Target A Specific ISA Instead of Using The xHost Option

Although static analysis presumes the loop may benefit from FMA instructions available with the AVX2 or higher ISA, no FMA instructions executed for this loop. To fix: Instead of using the xHost compiler option, which limits optimization opportunities by the host ISA, use the following compiler options:

- `xCORE-AVX2` to compile for machines with and without AVX2 support
- `axCORE-AVX2` to compile for machines with AVX2 support only
- `xCOMMON-AVX512` to compile for machines with AVX-512 support only
- `axCOMMON-AVX512` to compile for machines with and without AVX-512 support

NOTE The compiler options may vary depending on the CPU microarchitecture.

Windows OS	Linux OS
<code>/QxCORE-AVX2</code> or <code>/QaxCORE-AVX2</code>	<code>-xCORE-AVX2</code> or <code>-axCORE-AVX2</code>
<code>/QxCOMMON-AVX512</code> or <code>/QaxCOMMON-AVX512</code>	<code>-xCOMMON-AVX512</code> or <code>-axCOMMON-AVX512</code>

See also:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Indirect Function Call(s) Present

Indirect function call(s) in the loop body are preventing the compiler from vectorizing the loop. Indirect calls, sometimes called indirect jumps, get the callee address from a register or memory; direct calls get the callee address from an argument. Even if you force loop vectorization, indirect calls remain serialized.

Improve Branch Prediction

For 64-bit applications, branch prediction performance can be negatively impacted when the branch target is more than 4 GB away from the branch. This is more likely to happen when the application is split into shared libraries. To fix: Do the following:

- Upgrade the Glibc library to version 2.23 or higher.
- Set environment variable export `LD_PREFER_MAP_32BIT_EXEC=1`.

See also:

- [Glibc 2.23 release notes](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Remove Insirect Call(s) Inside The Loop

Indirect function or subroutine calls cannot be vectorized. To fix: Avoid using indirect calls in loops.

Replace Calls to Virtual Methods with Direct Calls

Calls to virtual methods are always indirect because the function address is calculated during runtime. Do the following to fix:

- Force vectorization of the source loop using SIMD instructions and/or create a SIMD version of the function(s) using a directive:
- Update to Intel Compiler 17.x or higher. Or replace the virtual method with a direct function call.

Original code example:

```
struct A {
    virtual double foo(double x) { return x+1; }
};

struct B : public A {
    double foo(double x) override { return x-1; }
};

...

A* obj = new B();

double sum = 0.0;
#pragma omp simd reduction(+:sum)
for (int k = 0; k < N; ++k) {
    // virtual indirect call
    sum += obj->foo(a[k]);
}
...
```

Revised code example:

```
struct A {
    // Intel Compiler 17.x or higher could vectorize call to virtual method
    #pragma omp declare simd
    virtual double foo(double x) { return x+1; }
};

...

sum = 0.0;
#pragma omp simd reduction(+:sum)
for (int k = 0; k < N; ++k) {
    // step for Intel Compiler 16.x or lower:
    // if you know the method to be called,
    // replace virtual call with direct one
    sum += ((B*)obj)->B::foo(a[k]);
}
...
```

See also:

- *omp simd*, *omp declare simd* in [OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vectorize Calls to Virtual Method

Force vectorization of the source loop using SIMD instructions and/or generate vector variants of the function(s) using a directive:

Original code example:

```
struct A {
    virtual double foo(double x) { return x+1; }
};

struct B : public A {
    double foo(double x) override { return x-1; }
};

...

A* obj = new B();

double sum = 0.0;
#pragma omp simd reduction(+:sum)
for (int k = 0; k < N; ++k) {
    // indirect call to virtual method
    sum += obj->foo(a[k]);
}
...
```

Revised code example:

```
struct A {
    #pragma omp declare simd
    virtual double foo(double x) { return x+1; }
};

...
```

See also:

- [omp simd, omp declare simd in OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Inefficient Processing of SIMD-enabled Functions Possible

Vector declaration defaults for your SIMD-enabled functions may result in extra computations or ineffective memory access patterns. Improve performance by overriding defaults.

Target a Specific Processor Type(s)

The default instruction set architecture (ISA) for SIMD-enabled functions is inefficient for your host processor because it could result in extra memory operations between registers. To fix: Add one of the following to tell the compiler to generate an extended set of vector functions.

Windows OS	Linux OS
processor(cpuid) to #pragma omp declare simd	processor(cpuid) to #pragma omp declare simd
processor(cpuid) to _declspec(vector())	processor(cpuid) to _attribute_(vector())
/Qvecabi:cmdtarget Note: Vector variants are created for targets specified for targets specified by compiler	-vecabi=cmdtarget Note: Vector variants are created for targets specified for targets specified by compiler options
options /Qx or /Qax	-x or -ax

See also:

- [cpu_specific; SIMD-Enabled Functions; vecabi, Qvecabi; vector; omp declare simd in OpenMP Pragmas Summary](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Enforce the Compiler to Ignore Assumed Vector Dependencies

No real dependencies were detected, so there is no need for conflict-detection instructions. To fix: Tell the compiler it is safe to vectorize using a directive `#pragma ivdep`.

NOTE This fix may be unsafe in other scenarios; use with care to avoid incorrect results.

```
#pragma ivdep
for (i = 0; i < n; i++)
{
    a[index[i]] = b[i] * c;
}
```

See also:

- [ivdep](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Opportunity for Outer Loop Vectorization

This is outer (non-innermost) loop. Normally outer loops are not targeted by auto-vectorization. Outer loop vectorization is also possible and sometimes more profitable, but requires explicit vectorization using OpenMP* API or Intel® Cilk™ Plus.

Collect Trip Counts Data

The Survey Report lacks trip counts data that might prove profitability for outer loop vectorization. To fix: Run a [Trip Counts analysis](#).

Check Dependencies for Outer Loop

It is not safe to force vectorization without knowing that there are no dependencies. **Disable inner vectorization before check Dependency**. To check: Run a [Dependencies analysis](#).

Check Memory Access Patterns for Outer Loop

To ensure that outer loop has optimal memory access patterns run a [Memory Access Patterns analysis](#).

Consider Outer Loop Vectorization

The compiler never targets loops other than innermost ones, so it vectorized the inner loop while did not vectorize the outer loop. However outer loop vectorization could be more profitable because of better Memory Access Pattern, higher Trip Counts or better Dependencies profile.

To enforce outer loop vectorization:

Target	Directive
Outer Loop	<code>#pragma omp simd</code>
Inner Loop	<code>#pragma novector</code>

Given issue is only about opportunity to vectorize outer loop, to prove profitability you need perform deeper dive analysis (MAP, Trip Counts, Dependencies)

```
#pragma omp simd
for(i=0; i<N; i++)
{
    #pragma novector
    for(j=0; j<N; j++)
    {
        sum += A[i]*A[j];
    }
}
```

See also:

- *omp simd* in [OpenMP Pragmas Summary](#), [novector](#)
- [Outer Loop Vectorization](#), [Vectorization Resources for Intel® Advisor Users](#)

Consider Outer Loop Vectorization.

The compiler did not vectorize the loop as the code exceeds the compilers complexity criteria. You might get higher performance if you enforce the loop vectorization. Use a directive right before your loop block in the source code.

ICL/ICC/ICPC Directive
#pragma omp simd

See also:

- *omp simd* in [OpenMP Pragmas Summary](#)
- [Outer Loop Vectorization](#), [Vectorization Resources for Intel® Advisor Users](#)

Consider Outer Loop Vectorization

The compiler did not vectorize the inner loop due to potential dependencies detected. You might vectorize outer loop if it has no dependency. Use a directive right before your loop block in the source code.

ICL/ICC/ICPC Directive
#pragma omp simd

See also:

- *omp simd* in [OpenMP Pragmas Summary](#)
- [Outer Loop Vectorization](#), [Vectorization Resources for Intel® Advisor Users](#)

STL Algorithms Present

STL algorithms are algorithmically optimized. Improve performance with algorithms that are both algorithmically and programmatically optimized by using Parallel STL. Parallel STL is an implementation of C++ standard library algorithms for the next version of the C++ standard, commonly called C++17, that supports execution policies and is specifically optimized for Intel® processors. Pass one of the following values as the first parameter in an algorithm call to specify the desired execution policy.

Execution Policy	Meaning
seq	Execute sequentially.
unseq	Use SIMD. (Requires SIMD-safe functions.)
par	Use multithreading. (Requires thread-safe functions.)
par_unseq	Use SIMD and multithreading. (Requires SIMD-safe and thread-safe functions.)

Parallel STL supports SIMD and multithreading execution policies for a subset of algorithms if random access iterators are provided. Execution remains sequential for all other algorithms.

Use Parallel STL Alternative to `std::any_of`

The `std::any_of` algorithm runs sequentially. To run in parallel, use the Parallel STL alternative with the following execution policy: `std::execution::unseq`

```
#include "pstl/execution"
#include "pstl/algorithm"
void foo(float* a, int n)
{
    std::any_of(std::execution::unseq, a, a+n, [](float elem)
    {
```

```
        return elem > 100.f;
    });
}
```

See also:

- [Get Started with Parallel STL](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use Parallel STL Alternative to `std::copy_if`

The `std::copy_if` algorithm runs sequentially. To run in parallel, use the Parallel STL alternative with one of the following execution policies:

- `std::execution::par`
- `std::execution::par_unseq`

```
#include "pstl/execution"
#include "pstl/algorithm"
void foo(float* a, float* b, int n)
{
    std::copy_if(std::execution::par_unseq, a, a+n, b, [](float elem)
    {
        return elem > 10.f;
    });
}
```

See also:

- [Get Started with Parallel STL](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use Parallel STL Alternative to `std::for_each`

The `std::for_each` algorithm runs sequentially. To run in parallel, use the Parallel STL alternative with one of the following execution policies:

- `std::execution::par`
- `std::execution::par_unseq`

```
#include "pstl/execution"
#include "pstl/algorithm"
void foo(float* a, int n)
{
    std::for_each(std::execution::par_unseq, a, a+n, [](float elem)
    {
        ...
    });
}
```

See also:

- [Get Started with Parallel STL](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use Parallel STL Alternative to `std::sort`

The `std::any_of` algorithm runs sequentially. To run in parallel, use the Parallel STL alternative with the following execution policy: `std::execution::par`

```
#include "pstl/execution"
#include "pstl/algorithm"
void foo(float* a, int n)
```

```
{
    std::sort(std::execution::par, a, a+n);
}
```

See also:

- [Get Started with Parallel STL](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Potential Underutilization of Approximate Reciprocal Instructions

Your current hardware supports Advanced Vector Extensions 512 (AVX-512) instructions that enable the use of approximate reciprocal and reciprocal square root instructions both for single- and double-precision floating-point calculations. Improve performance by utilizing these instructions.

Force Vectorization If Possible

The loop contains SQRT/DIV instructions (so vectorization could be beneficial), but is not vectorized. To fix, review:

- Corresponding compiler diagnostic to check if vectorization enforcement is possible and profitable
- The Dependencies analysis to distinguish between compiler-assumed dependencies and real dependencies

See also:

- [Vectorization Resources for Intel® Advisor Users](#)

Target the AVX-512 ISA

Static analysis presumes the loop may benefit from AVX-512 approximate reciprocal instructions, but these instructions were not used. To fix: Use one of the following compiler options:

- `xCOMMON-AVX512` - tells the compiler which processor features to target, including instructions sets and optimizations it may generate, including AVX-512.
- `axCOMMON-AVX512` - tells the compiler to generate multiple, feature-specific, auto-dispatch code for Intel processors if there is a performance benefit.

Windows OS	Linux OS
/QxCOMMON-AVX512 or /QaxCOMMON-AVX512	-xCOMMON-AVX512 or -axCOMMON-AVX512

See also:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Target the AVX-512 Exponential and Reciprocal Instructions ISA

Static analysis presumes the loop may benefit from AVX-512 Exponential and Reciprocal (AVX-512ER) instructions currently supported only on Intel® Xeon Phi™ processors, but these instructions were not used. To fix: Use one of the following compiler options:

- `xMIC-AVX512` - tells the compiler which processor features to target, including instructions sets and optimizations it may generate, including AVX-512ER.
- `axMIC-AVX512` - tells the compiler to generate multiple, feature-specific, auto-dispatch code for Intel processors if there is a performance benefit.

Windows OS	Linux OS
/QxMIC-AVX512 or /QaxMIC-AVX512	-xMIC-AVX512 or -axMIC-AVX512

See also:

- [ax, Qax; x, Qx](#)

- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Enable the Use of Approximate Reciprocal Instructions by Fine-Tuning Precision and Floating-Point Model Compiler Options

Static analysis presumes the loop may benefit from using approximate reciprocal instructions, but the precision and floating-point model settings may prevent the compiler from using these instructions. To fix: Fine-tune your usage of the following compiler options:

Windows OS	Linux OS	Comment
/fp	-fp-model	-fp-model=precise prevents the use of approximate reciprocal instructions.
/Qimf-precision	-fimf-precision	Consider using -fimf-precision=medium or -fimf-precision=low.
/Qimf-accuracy-bits	-fimf-accuracy-bits	Consider decreasing this setting.
/Qimf-max-error	-fimf-max-error	Consider increasing this setting.
		There is a similar option: -fimf-absolute-error. Avoid using both options at the same time or tune them together.
/Qimf-absolute-error	-fimf-absolute-error	Consider using -fimf-max-error instead and set -fimf-absolute-error=0 (default) or increase this setting together with -fimf-max-error.
/Qimf-domain-exclusion	-fimf-domain-exclusion	Consider increasing this setting. More excluded classes enable more optimized code. USE WITH CAUTION. This option may cause incorrect behavior if your calculations involve excluded domains.
/Qimf-arch-consistency	-fimf-arch-consistency	-fimf-arch-consistency=true may prevent the use of approximate reciprocal instructions.
/Qprec-div	-prec-div	-prec-div prevents the use of approximate reciprocal instructions.
/Qprec-sqrt	-prec-sqrt	-prec-sqrt prevents the use of approximate reciprocal instructions.

See also:

- [Floating-point Operations](#) and [Floating-point Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Possible Inefficient Conflict-Detection Instructions Present

Stores with indirect addressing caused the compiler to assume a potential dependency.

This resulted in the use of conflict-detection instructions during SIMD processing, such as the AVX-512 vpmconflict instruction, which detects duplicate values within a vector and creates conflict-free subsets. Improve performance by removing the need for conflict-detection instructions.

Enforce the Compiler to Ignore Assumed Vector Dependencies

No real dependencies were detected, so there is no need for conflict-detection instructions. To fix: Tell the compiler it is safe to vectorize using a directive `#pragma ivdep`.

NOTE This fix may be unsafe in other scenarios; use with care to avoid incorrect results.

```
#pragma ivdep
for (i = 0; i < n; i++)
{
    a[index[i]] = b[i] * c;
}
```

See also:

- [ivdep](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Unoptimized Floating-Point Operation Processing Possible

Improve performance by enabling approximate operations instructions.

Enable the Use of Approximate Division Instructions

Static analysis presumes the loop may benefit from using approximate calculations. Independent divisors will be pre-calculated and replaced with multipliers. To fix: Fine-tune your usage of the following compiler option:

Windows OS	Linux OS	Comment
/Qprec-div	-no-prec-div	-no-prec-div enables the use of approximate division optimizations.

See also:

- [prec-div, Qprec-div](#)
- [Floating-point Operations](#) and [Floating-point Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Enable the Use of Approximate sqrt Instructions

Static analysis presumes the loop may benefit from using approximate sqrt instructions, but the precision and floating-point model settings may prevent the compiler from using these instructions. To fix: Fine-tune your usage of the following compiler option:

Windows OS	Linux OS	Comment
/Qprec-sqrt	-no-prec-sqrt	-no-prec-sqrt enables the use of approximate sqrt optimizations.

See also:

- [prec-sqrt, Qprec-sqrt](#)
- [Floating-point Operations](#) and [Floating-point Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Potential Excessive Caching Present

Enable Non-Temporal Store

Enable non-temporal store using `#pragma vector nontemporal`. The `nontemporal` clause instructs the compiler to use non-temporal (that is, streaming) stores on systems based on all supported architectures, unless specified otherwise; optionally takes a comma-separated list of variables.

When this pragma is specified, it is your responsibility to also insert any fences as required to ensure correct memory ordering within a thread or across threads. One typical way to do this is to insert a `_mm_sfence` intrinsic call just after the loops (such as the initialization loop) where the compiler may insert streaming store instructions.

Streaming stores may cause significant performance improvements over non-streaming stores for large numbers on certain processors. However, the misuse of streaming stores can significantly degrade performance.

```
float a[1000];
void foo(int N)
{
    int i;
    #pragma vector nontemporal
    for (i = 0; i < N; i++)
    {
        a[i] = 1;
    }
}
```

See also:

- [vector](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Misaligned Loop Code Present

Current placement of the loop in memory may result in inefficient use of the CPU front-end. Improve performance by aligning loop code.

Force the Compiler to Align Loop Code

Caution Excessive code alignment may increase application binary size and decrease performance.

Static analysis shows the loop may benefit from code alignment. To fix: Force the compiler to align the loop to a power-of-two byte boundary using a compiler directive for finer-grained control: `#pragma code_align (n)`

Align inner loop to 32-byte boundary:

```
for (i = 0; i < n; i++)
{
    #pragma code_align 32
    for (j = 0; j < m; j++)
    {
        a[i] *= b[i] + c[j];
    }
}
```

You may also need the following compiler option:

Windows OS	Linux OS and Mac OS
/Qalign-loops[:n]	-falign-loops[=n]

where n = a power of 2 between 1 and 4096, such as 1, 2, 4, 8, 16, 32, etc. n = 1 performs no alignment. If n is not present, the compiler uses an alignment of 16 bytes. Suggestion: Try 16 and 32 first.

/Qalign-loops- and -fno-align-loops, the default compiler option, disables special loop alignment.

Vectorization Recommendations for Fortran

Ineffective Peeled/Remainder Loop(s) Present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops to the loop body.

Align Data

One of the memory accesses in the source loop does not start at an optimally aligned address boundary. To fix: Align the data and tell the compiler the data is aligned. To align data, use `__declspec(align())`. To tell the compiler the data is aligned, use `__assume_aligned()` before the source loop.

See also:

- [Data Alignment to Assist Vectorization](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Parallelize The Loop with Both Threads and SIMD Instructions

The loop is threaded and auto-vectorized; however, the trip count is not a multiple of vector length. To fix: Do all of the following:

- Use the `!$omp parallel do simd` directive to parallelize the loop with both threads and SIMD instructions. Specifically, this directive divides loop iterations into chunks (subsets) and distributes the chunks among threads, then chunk iterations execute concurrently using SIMD instructions.
- Add the `schedule(simd: [kind])` modifier to the directive to guarantee the chunk size (number of iterations per chunk) is a multiple of vector length.

Original code sample:

```
!$omp parallel do schedule(static)
do i = 1,1000
    c(i) = a(i)*b(i)
end do
!$omp end parallel do
```

Revised code sample:

```
!$omp parallel do simd schedule(simd: static)
do i = 1,1000
    c(i) = a(i)*b(i)
end do
!$omp end parallel do simd
```

See also:

- [OpenMP Application Programming Interface](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Force Scalar Remainder Generation

The compiler generated a masked vectorized remainder loop that contains too few iterations for efficient vector processing. A scalar loop may be more beneficial. To fix: Force scalar remainder generation using a directive: `!DIR$ VECTOR NOVECREMAINDER`.

```
subroutine add(A, N, X)
    integer N, X
    real    A(N)
    ! Force the compiler to not vectorize the remainder loop
    !DIR$ VECTOR NOVECREMAINDER
    do i=x+1, n
        a(i) = a(i) + a(i-x)
    enddo
end
```

See also:

- [VECTOR and NOVECTOR](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Force Vectorized Remainder

The compiler did not vectorize the remainder loop, even though doing so could improve performance. To fix: Force vectorization using a directive: `!DIR$ VECTOR VECREMAINDER`.

```
subroutine add(A, N, X)
  integer N, X
  real    A(N)
  ! Force the compiler to vectorize the remainder
  !DIR$ VECTOR VECREMAINDER
  do i=x+1, n
    a(i) = a(i) + a(i-x)
  enddo
end
```

See also:

- [VECTOR and NOVECTOR](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Specify The Expected Loop Trip Count

The compiler cannot detect the trip count statically. To fix: Specify the expected number of iterations using a directive: `!DIR$ LOOP COUNT`.

Iterate through a loop a maximum of ten, minimum of three, and average of five times:

```
!DIR$ LOOP COUNT MAX(10), MIN(3), AVG(5)
do i =1, m
  b(i) = a(i) + 1
  d(i) = c(i) + 1
enddo
```

See also:

- [LOOP COUNT](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Change The Chunk Size

The loop is threaded and vectorized using the `!$omp parallel for simd` directive, which parallelizes the loop with both threads and SIMD instructions. Specifically, the directive divides loop iterations into chunks (subsets) and distributes the chunks among threads, then chunk iterations execute concurrently using SIMD instructions. In this case, the chunk size (number of iterations per chunk) is not a multiple of vector length. To fix: Add a `schedule (simd: [kind])` modifier to the `!$omp parallel for simd` directive.

Guarantee a maximum vector length.

```
!$omp parallel do simd schedule(simd: static)
do i = 1,1000
  c(i) = a(i)*b(i)
end do
!$omp end parallel do simd
```

See also:

- [OpenMP Application Programming Interface](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Add Data Padding

The trip count is not a multiple of vector length . To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

See also:

- [LOOP COUNT](#)
- [Utilizing Full Vectors and Vectorization Resources for Intel® Advisor Users](#)

Collect Trip Counts Data

The Survey Report lacks trip counts data that might generate more precise recommendations.

Disable Unrolling

The trip count after loop unrolling is too small compared to the vector length . To fix: Prevent loop unrolling or decrease the unroll factor using a directive: `!DIR$ NOUNROLL` or `!DIR$ UNROLL`.

Disable automatic loop unrolling using `!DIR$ NOUNROLL`.

```
!DIR$ NOUNROLL
do i = 1, m
    b(i) = a(i) + 1
    d(i) = c(i) + 1
enddo
```

See also:

- [UNROLL and NOUNROLL](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use A Smaller Vector Length

The compiler chose a vector length of , but the trip count might be smaller than the vector length. To fix: Specify a smaller vector length using a directive: `!$OMP SIMD SIMDLLEN`.

```
!$OMP SIMD SIMDLLEN(4)
do i = 1, m
    b(i) = a(i) + 1
    d(i) = c(i) + 1
enddo
```

In Intel Compiler version 19.0 and higher, there is a new vector length clause that allows the compiler to choose the best vector length based on cost: `!DIR$ VECTOR VECTORLENGTH (v11, v12, ..., v1n)` where `v1` is an integer power of 2.

```
!DIR$ VECTOR VECTORLENGTH(2, 4, 16)
do i = 1, m
    b(i) = a(i) + 1
    d(i) = c(i) + 1
enddo
```

See also:

- [SIMD Directive \(OpenMP* API\), VECTOR and NOVECTOR](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Disable Dynamic Alignment

The compiler automatically peeled iterations from the vector loop into a scalar loop to align the vector loop with a particular memory reference; however, this optimization may not be ideal. To possibly achieve better performance, disable automatic peel generation using the directive: `!DIR$ VECTOR NODYNAMIC_ALIGN`.

```
...
!DIR$ VECTOR NODYNAMIC_ALIGN
do i = 1, len
    a(i) = b(i) * c(i)
enddo
```

See also:

- [VECTOR and NOVECTOR](#)

- [Vectorization Resources for Intel® Advisor Users](#)

Serialized User Function Call(s) Present

User-defined functions in the loop body are not vectorized.

Enable Inline Expansion

Inlining of user-defined functions is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

See also:

- [inline-level, Ob](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vectorize Serialized Function(s) Inside Loop

- Enforce vectorization of the source loop by means of SIMD instructions and/or create a SIMD version of the function(s) using a directive:

Target	Directive
Source Loop	!\$OMP SIMD
Inner function definition or declaration	!\$OMP DECLARE SIMD

- If using the `Ob` or `inline-level` compiler option to control inline expansion with the `1` argument, use an `inline` keyword to enable inlining or replace the `1` argument with `2` to enable inlining of any function at compiler discretion.

```
real function f (x)
  !DIR$ OMP DECLARE SIMD
  real, intent(in), value :: x
  f= x + 1
end function f

!DIR$ OMP SIMD
do k = 1, N
  a(k) = f(k)
enddo
```

See also:

- [DECLARE SIMD, SIMD Directive \(OpenMP* API\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Scalar Math Function Call(s) Present

Math functions in the loop body are preventing the compiler from effectively vectorizing the loop. Improve performance by enabling vectorized math call(s).

Enable Inline Expansion

Inlining is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

Alternatively use `#include <mathimf.h>` header instead of the standard `#include <math.h>` header to call highly optimized and accurate mathematical functions commonly used in applications that rely heavily on floating point computations.

See also:

- [inline-level, Ob](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vectorize Math Function Calls Inside Loops

Your application calls serialized versions of math functions when you use the precise floating point model. To fix: Do one of the following:

- Add `fast-transcendentals` compiler option to replace calls to transcendental functions with faster calls.

Caution This may reduce floating point accuracy.

- Enforce vectorization of the source loop using a directive: `!$OMP SIMD`

```
subroutine add(A, N, X)
  integer N, X
  real    A(N)
  !DIR$ OMP SIMD
  do i=x+1, n
    a(i) = a(i) + a(i-x)
  enddo
end
```

See also:

- [fast-transcendentals, Qfast-transcendentals; SIMD Directive \(OpenMP* API\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Change The Floating Point Model

Your application calls serialized versions of math functions when you use the `strict` floating point model. To fix: Do one of the following:

- Use the `fast` floating point model to enable more aggressive optimizations or the `precise` floating point model to disable optimizations that are not value-safe on fast transcendental functions.

Caution This may reduce floating point accuracy.

- Use the `precise` floating point model and enforce vectorization of the source loop using a directive: `!$OMP SIMD`

```
gfortran program.for -O2 -fopenmp -fp-model precise -fast-transcendentals
!DIR$ OMP SIMD COLLAPSE(2)
do i = 1, N
  a(i) = b(i) * c(i)
  do j = 1, N
    d(j) = e(j) * f(j)
  enddo
enddo
```

See also:

- [fast-transcendentals, Qfast-transcendentals; SIMD Directive \(OpenMP* API\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use a Glibc Library with Vectorized SVML Functions

Your application calls scalar instead of vectorized versions of math functions. To fix: Do all of the following:

- Upgrade the Glibc library to version 2.22 or higher. It supports SIMD directives in OpenMP* 4.0 or higher.
- Upgrade the GNU* gcc compiler to version 4.9 or higher. It supports vectorized math function options.

- Use the `-fopenmp` and `-ffast-math` compiler options to enable vector math functions.
- Use appropriate OpenMP SIMD directives to enable vectorization.

NOTE Also use the `-I/path/to/glibc/install/include` and `-L/path/to/glibc/install/lib` compiler options if you have multiple Glibc libraries installed on the host.

```
gfortran PROGRAM.FOR -O2 -fopenmp -ffast-math -lrt -lm -mavx2
program main
  parameter (N=100000000)
  real*8 angles(N), results(N)
  integer i
  call srand(86456)

  do i=1,N
    angles(i) = rand()
  enddo

  !$OMP SIMD
  do i=1,N
    results(i) = cos(angles(i))
  enddo

end
```

See also:

- [Glibc wiki/libmvec](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use The Intel Short Vector Math Library for Vector Intrinsics

Your application calls scalar instead of vectorized versions of math functions. To fix: Do all of the following:

- Use the `-mveclibabi=svml` compiler option to specify the Intel short vector math library ABI type for vector intrinsics.
- Use the `-ftree-vectorize` and `-funsafe-math-optimizations` compiler options to enable vector math functions.
- Use the `-L/path/to/intel/lib` and `-lsVML` compiler options to specify an SVML ABI-compatible library at link time.

```
gfortran PROGRAM.FOR -O2 -ftree-vectorize -funsafe-math-optimizations -mveclibabi=svml -L/opt/
intel/lib/intel64 -lm -lsVML -Wl,-rpath=/opt/intel/lib/intel64
program main
  parameter (N=100000000)
  real*8 angles(N), results(N)
  integer i
  call srand(86456)

  do i=1,N
    angles(i) = rand()
  enddo

  ! the loop will be auto-vectorized
  do i=1,N
    results(i) = cos(angles(i))
  enddo

end
```

See also:

- [The GNU Fortran Compiler](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Inefficient Gather/Scatter Instructions Present

The compiler assumes indirect or irregular stride access to data used for vector operations. Improve memory access by alerting the compiler to detected regular stride access patterns, such as:

Refactor code with detected regular stride access patterns

The Memory Access Patterns Report shows the following regular stride access(es):

See details in the Memory Access Patterns Report Source Details view.

To improve memory access: Refactor your code to alert the compiler to a regular stride access. Sometimes, it might be beneficial to use the `ipo/Qipo` compiler option to enable interprocedural optimization (IPO) between files.

See also:

- [ipo, Qipo](#)
- [Case study: Comparing Arrays of Structures and Structures of Arrays Data Layouts for a Compute-Intensive Loop](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vector Register Spinning Possible

Possible register spilling was detected and all vector registers are in use. This may negatively impact performance, because the spilled variable must be loaded to and unloaded from main memory. Improve performance by decreasing vector register pressure.

Decrease Unroll Factor

The current directive unroll factor increases vector register pressure. To fix: Decrease unroll factor using a directive: `!DIR$ NOUNROLL` or `!DIR$ UNROLL`.

```
!DIR$ UNROLL
do i = 1, m
    b(i) = a(i) + 1
    d(i) = c(i) + 1
enddo
```

See also:

- [UNROLL and NOUNROLL](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Split Loop into Smaller Loops

Possible register spilling along with high vector register pressure is preventing effective vectorization. To fix: Use the directive `!DIR$ DISTRIBUTE POINT` or rewrite your code to distribute the source loop. This can decrease register pressure as well as enable software pipelining and improve both instruction and data cache use.

```
!DIR$ DISTRIBUTE POINT
do i = 1, m
    b(i) = a(i) + 1
    ...
    c(i) = a(i) + b(i) ! Compiler will decide
    ! where to distribute.
    ! Data dependencies are observed
```

```

...
d(i) = c(i) + 1
enddo
do i =1, m
  b(i) = a(i) + 1
  ...
  !DIR$ DISTRIBUTE POINT
  call sub(a, n)! Distribution will start here,
  ! ignoring all loop-carried dependencies
  c(i) = a(i) + b(i)
  ...
  d(i) = c(i) + 1
enddo

```

See also:

- [DISTRIBUTE POINT](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Assumed Dependency Present

The compiler assumed there is an anti-dependency (Write after read - WAR) or a true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Confirm Dependency Is Real

There is no confirmation that a real (proven) dependency is present in the loop. To confirm: Run a [Dependencies analysis](#).

Enable Vectorization

The Dependencies analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a directive:

```

!DIR$ IVDEP
do i = 1, N-4, 4
  a(i+4) = b(i) * c
enddo

```

See also:

- [IVDEP; SIMD Directive \(OpenMP* API\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Proven (Real) Dependency Is Present

The compiler assumed there is an anti-dependency (Write after read - WAR) or true dependency (Read after write - RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Resolve Dependency

The Dependencies analysis shows there is a real (proven) dependency in the loop. To fix: Do one of the following:

- If there is an anti-dependency, enable vectorization using the directive `!$OMP SIMD SAFELEN(length)` , where `length` is smaller than the distance between dependent iterations in anti-dependency.

```

!$OMP SIMD SAFELEN(4)
do i = 1, N-4, 4
  a(i+4) = b(i) * c
enddo

```


- If there is a reduction pattern dependency in the loop, enable vectorization using the directive `!$OMP SIMD REDUCTION(operator:list)`.

```
!$OMP SIMD REDUCTION(+:SUMX)
do k = 1, size2
    sumx = sumx + x(k) * b(k)
enddo
```

- Rewrite the code to remove the dependency. Use programming techniques such as variable privatization.

See also:

- [SIMD Directive \(OpenMP* API\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Data Type Conversions Present

There are multiple data types within loops. Utilize hardware vectorization support more effectively by avoiding data type conversion.

Use The Smallest Data Type

The source loop contains data types of different widths. To fix: Use the smallest data type that gives the needed precision to use the entire vector register width.

Example: If only 16-bits are needed, using a short rather than an int can make the difference between eight-way or four-way SIMD parallelism, respectively.

User Function Call(s) Present

User-defined functions in the loop body are preventing the compiler from vectorizing the loop.

Enable Inline Expansion

Inlining of user-defined functions is disabled by compiler option. To fix: When using the `Ob` or `inline-level` compiler option to control inline expansion, replace the `0` argument with the `1` argument to enable inlining when an `inline` keyword or attribute is specified or the `2` argument to enable inlining of any function at compiler discretion.

See also:

- [inline-level, Ob](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Vectorize User Function(s) Inside Loop

These user-defined function(s) are not vectorized or inlined by the compiler: `my_calc()` To fix: Do one of the following:

- Enforce vectorization of the source loop by means of SIMD instructions and/or create a SIMD version of the function(s) using a directive:

Target	Directive
Source loop	<code>!\$OMP SIMD</code>
Inner function definition or declaration	<code>!\$OMP DECLARE SIMD</code>

```
real function f (x)
    !DIR$ OMP DECLARE SIMD
    real, intent(in), value :: x
    f= x + 1
end function f

!DIR$ OMP SIMD
```

```
do k = 1, N
  a(k) = f(k)
enddo
```

See also:

- [DECLARE SIMD; SIMD Directive \(OpenMP* API\)](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Convert to Fortran SIMD-Enabled Functions

Passing an array/array recommendation to an `ELEMENTAL` function/subroutine is creating a dependency that prevents vectorization. To fix:

- Enforce vectorization of the source loop using SIMD instructions and/or create a SIMD version of the function(s) using a directive:

Target	Directive
Source loop	!\$OMP SIMD
Inner function definition or declaration	!\$OMP DECLARE SIMD

- Call from a `DO` loop.

Original code example:

```
elemental subroutine callee(t,q,r)
  real, intent(in) :: t, q
  real, intent(out) :: r
  r = t + q
end subroutine callee
...
do k = 1,nlev
  call callee(a(:,k), b(:,k), c(:,k))
end do
...
```

Revised code example:

```
subroutine callee(t,q,r)
  !$OMP DECLARE SIMD(callee)
  real, intent(in) :: t, q
  real, intent(out) :: r
  r = t + q
end subroutine callee
...
do k = 1,nlev
  !$OMP SIMD
  do i = 1,n
    call callee(a(i,k), b(i,k), c(i,k))
  end do
end do
...
```

See also:

- [DECLARE SIMD; SIMD Directive \(OpenMP* API\)](#)
- [Explicit Vector Programming in Fortran; Vectorization Resources for Intel® Advisor Users](#)

Compiler Lacks Sufficient Information to Vectorize Loop

Cause: You are using a non-Intel compiler or an outdated Intel compiler. Nevertheless, it appears there are no issues preventing vectorization and vectorization may be profitable.

Explore Vectorization Opportunities

You compiled with auto-vectorization enabled; however, the compiler did not vectorize the code. Explore vectorization opportunities:

- Run a Dependencies analysis to identify real data dependencies that could make forced vectorization unsafe.
- Auto-Vectorizer Reporting Level to output missed optimization opportunities.
- GNU* Fortran compiler, do one of the following:
 - Use the `fopt-info-vec-missed` compiler option to output missed optimization opportunities.
 - Use the OpenMP* `omp simd` directive to tell the compiler it is safe to vectorize.
 - Use additional auto-vectorization directives.

See also:

- [GCC online documentation](#)
- [OpenMP Resources](#)

Enable Auto-Vectorization

You compiled with auto-vectorization disabled; enable auto-vectorization:

- Intel compiler 14.x or below: Increase the optimization level to O2 or O3.
- GNU* Fortran compiler, do one of the following:
 - Increase the optimization level to O2 or O3.
 - Use the `fvec-vectorize` compiler option.

See also:

- [GCC online documentation](#)

System Function Call(s) Present

System function call(s) in the loop body are preventing the compiler from vectorizing the loop.

Remove System Function Call(s) Inside Loop

Typically system function or subroutine calls cannot be vectorized; even a print statement is sufficient to prevent vectorization. To fix: Avoid using system function calls in loops.

OpenMP* Function Call(s) Present

OpenMP* function call(s) in the loop body are preventing the compiler from effectively vectorizing the loop.

Move OpenMP Call(s) Outside The Loop Body

OpenMP calls prevent automatic vectorization when the compiler cannot move the calls outside the loop body, such as when OpenMP calls are not invariant. To fix:

1. Split the OpenMP parallel loop directive into two directives.
2. Move the OpenMP calls outside the loop when possible.

Original code example:

```
!$OMP PARALLEL DO PRIVATE(tid, nthreads)
do k = 1, N
    tid = omp_get_thread_num() ! this call inside loop prevents vectorization
    nthreads = omp_get_num_threads() ! this call inside loop prevents vectorization
    ...
enddo
```

Revised code example:

```
!$OMP PARALLEL PRIVATE(tid, nthreads)
! Move OpenMP calls here
tid = omp_get_thread_num()
nthreads = omp_get_num_threads()

!$OMP DO NOWAIT
do k = 1, N
    ...
enddo
!$OMP END PARALLEL
```

See also:

- [NOWAIT Clause; PARALLEL SECTIONS](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Remove OpenMP Lock Functions

Locking objects slows loop execution. To fix: Rewrite the code without OpenMP lock functions.

Allocating separate arrays for each thread and then merging them after a parallel recommendation may improve speed (but consume more memory).

See also:

- Lock Routines recommendation in [OpenMP Run-time Library Routines;PARALLEL SECTIONS](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Potential Inefficient Memory Access Patterns Present

Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.

Confirm Inefficient Memory Access Patterns

There is no confirmation inefficient memory access patterns are present. To fix: Run a [Memory Access Patterns](#) analysis.

Inefficient Memory Access Patterns Present

There is a high of percentage memory instructions with irregular (variable or random) stride accesses. Improve performance by investigating and handling accordingly.

Reorder Loops

This loop has less efficient memory access patterns than a nearby outer loop. To fix: Reorder the loops if possible.

Original code example:

```
subroutine matrix_multiply(arrSize, aMatrix, bMatrix, cMatrix)
  implicit none
  real, intent(inout) :: cMatrix(:, :)
  real, intent(in)    :: aMatrix(:, :), bMatrix(:, :)
  integer, intent(in) :: arrSize
  integer :: i, j, k;

  do j=1, arrSize
    do i=1, arrSize
      do k=1, arrSize
        cMatrix(i, j) = cMatrix(i, j) + aMatrix(i, k) * bMatrix(k, j)
      end do
    end do
  end do
```

```

    end do
  end do

end subroutine matrix_multiply

```

Revised code example:

```

subroutine matrix_multiply(arrSize, aMatrix, bMatrix, cMatrix)
  implicit none
  real, intent(inout) :: cMatrix(:, :)
  real, intent(in)    :: aMatrix(:, :), bMatrix(:, :)
  integer, intent(in) :: arrSize
  integer :: i, j, k;

  do j=1, arrSize
    do k=1, arrSize
      do i=1, arrSize
        cMatrix(i, j) = cMatrix(i, j) + aMatrix(i, k) * bMatrix(k, j)
      end do
    end do
  end do

end subroutine matrix_multiply

```

Interchanging is not always possible because of dependencies, which can lead to different results.

Use the Fortran 2008 CONTIGUOUS Attribute

The loop is multi-versioned for unit and non-unit strides in assumed-shape arrays or pointers, but marked versions of the loop have unit stride access only. The CONTIGUOUS attribute specifies the target of a pointer or an assumed-shape array is contiguous. It can make it easier to enable optimizations that rely on the memory layout of an object occupying a contiguous block of memory.

```

real, pointer, contiguous :: ptr(:)
real, contiguous :: arrayarg(:, :)

```

When multiple calling routines are involved, to tell the compiler assumed-shape arrays and/or pointers are always contiguous in memory, use the following options available in Version 18 and higher of the Intel® Fortran Compiler:

Type	Windows* OS	Linux* OS
assumed-shape array	/assume:contiguous_assumed_shape	-assume contiguous_assumed_shape
pointer	/assume:contiguous_pointer	-assume contiguous_pointer

NOTE Results are indeterminate and could result in incorrect code and segmentation faults if the user assertion is wrong and the data is not contiguous at runtime. To check at runtime if targets of contiguous pointer assignments are indeed contiguous in memory, use the following options available in Version 18 and higher of the Intel® Fortran Compiler:

Windows OS	Linux OS
/check:contiguous	-check contiguous

```
$ ifort -DCONTIG -check contiguous -traceback
```

```

forrtl: severe (408): fort: (32): A pointer with the CONTIGUOUS attributes is being made to a
non-contiguous target.

```

In this example, the compiler detects the assignment of a contiguous pointer to a non-contiguous target. The `-traceback` (Linux* OS) / `/traceback` (Windows* OS) option identifies the function and source file line number at which the incorrect assignment occurs. It is not necessary to compile with the debugging option `-g` (Linux* and macOS* OS) / `/zi` (Windows* OS) to get this traceback.

See also:

- [assume, check, CONTIGUOUS](#)
- [Fortran Array Data and Arguments and Vectorization](#)
- [Vectorization and Array Contiguity with the Intel® Fortran Compiler](#)
- [Contiguity Checking for Pointer Assignments in the Intel® Fortran Compiler](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Use SoA Instead of AoS

An array is the most common type of data structure containing a contiguous collection of data items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). While AoS organization is excellent for encapsulation, it can hinder effective vector processing. To fix: Rewrite code to organize data using SoA instead of AoS.

See also:

- [Programming Guidelines for Vectorization](#)
- [Case study: Comparing Arrays of Structures and Structures of Arrays Data Layouts for a Compute-Intensive Loop](#) and [Vectorization Resources for Intel® Advisor Users](#)

Potential Underutilization of FMA Instructions

Your current hardware supports the AVX2 instruction set architecture (ISA), which enables the use of fused multiply-add (FMA) instructions. Improve performance by utilizing FMA instructions.

Force Vectorization If Possible

The loop contains FMA instructions (so vectorization could be beneficial), but is not vectorized. To fix, review:

- Corresponding compiler diagnostic to check if vectorization enforcement is possible and profitable
- The Dependencies analysis to distinguish between compiler-assumed dependencies and real dependencies

See also:

- [Vectorization Resources for Intel® Advisor Users](#)

Explicitly Enable FMA Generation When Using The Strict Floating-Point Model

Static analysis presumes the loop may benefit from FMA instructions available with the AVX2 ISA, but the `strict` floating-point model disables FMA instruction generation by default. To fix: Override this behavior using the `fma` compiler option.

Windows OS	Linux OS
<code>/Qfma</code>	<code>-fma</code>

See also:

- [fma, Qfma](#)
- [Floating-point Operations](#) and [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Target The AVX2 ISA

Although static analysis presumes the loop may benefit from FMA instructions available with the AVX2 or higher ISA, no FMA instructions executed for this loop. To fix: Use the following compiler options:

- `xCORE-AVX2` to compile for machines with and without AVX2 support
- `axCORE-AVX2` to compile for machines with AVX2 support only

- `xCOMMON-AVX512` to compile for machines with AVX-512 support only
- `axCOMMON-AVX512` to compile for machines with and without AVX-512 support

NOTE The compiler options may vary depending on the CPU microarchitecture.

Windows OS	Linux OS
<code>/QxCORE-AVX2</code> or <code>/QaxCORE-AVX2</code>	<code>-xCORE-AVX2</code> or <code>-axCORE-AVX2</code>
<code>/QxCOMMON-AVX512</code> or <code>/QaxCOMMON-AVX512</code>	<code>-xCOMMON-AVX512</code> or <code>-axCOMMON-AVX512</code>

See also:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Target A Specific ISA Instead of Using The xHost Option

Although static analysis presumes the loop may benefit from FMA instructions available with the AVX2 or higher ISA, no FMA instructions executed for this loop. To fix: Instead of using the `xHost` compiler option, which limits optimization opportunities by the host ISA, use the following compiler options:

- `xCORE-AVX2` to compile for machines with and without AVX2 support
- `axCORE-AVX2` to compile for machines with AVX2 support only
- `xCOMMON-AVX512` to compile for machines with AVX-512 support only
- `axCOMMON-AVX512` to compile for machines with and without AVX-512 support

NOTE The compiler options may vary depending on the CPU microarchitecture.

Windows OS	Linux OS
<code>/QxCORE-AVX2</code> or <code>/QaxCORE-AVX2</code>	<code>-xCORE-AVX2</code> or <code>-axCORE-AVX2</code>
<code>/QxCOMMON-AVX512</code> or <code>/QaxCOMMON-AVX512</code>	<code>-xCOMMON-AVX512</code> or <code>-axCOMMON-AVX512</code>

See also:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Indirect Function Call(s) Present

Indirect function call(s) in the loop body are preventing the compiler from vectorizing the loop. Indirect calls, sometimes called indirect jumps, get the callee address from a register or memory; direct calls get the callee address from an argument. Even if you force loop vectorization, indirect calls remain serialized.

Improve Branch Prediction

For 64-bit applications, branch prediction performance can be negatively impacted when the branch target is more than 4 GB away from the branch. This is more likely to happen when the application is split into shared libraries. To fix: Do the following:

- Upgrade the Glibc library to version 2.23 or higher.
- Set environment variable `export LD_PREFER_MAP_32BIT_EXEC=1`.

See also:

- [Glibc 2.23 release notes](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Remove Indirect Call(s) Inside The Loop

Indirect function or subroutine calls cannot be vectorized. To fix: Avoid using indirect calls in loops.

Inefficient Processing of SIMD-enabled Functions Possible

Vector declaration defaults for your SIMD-enabled functions may result in extra computations or ineffective memory access patterns. Improve performance by overriding defaults.

Specify the Value of the Underlying Reference as Linear

In Fortran applications, by default, scalar arguments are passed by reference. Therefore, in SIMD-enabled functions, arguments are passed as a short vector of addresses instead of a single address. The compiler then gathers data from the vector of addresses to create a short vector of values for use in subsequent vector arithmetic. This gather activity negatively impacts performance. To fix: Add a `LINEAR` clause with a `REF` modifier (introduced in OpenMP* 4.5) to your vector declaration. Specifically, add `LINEAR (REF(linear-list[: linear-step]))` to your `!$OMP DECLARE SIMD` directive.

See also:

- [DECLARE SIMD; LINEAR Clause](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Target a Specific Processor Type(s)

The default instruction set architecture (ISA) for SIMD-enabled functions is inefficient for your host processor because it could result in extra memory operations between registers. To fix: Add one of the following to tell the compiler to generate an extended set of vector functions.

Windows OS	Linux OS
PROCESSOR(cpuid) to !\$OMP DECLARE SIMD /Qvecabi:cmdtarget Note: Vector variants are created for targets specified for targets specified by compiler options /Qx or /Qax	PROCESSOR(cpuid) to !\$OMP DECLARE SIMD -vecabi=cmdtarget Note: Vector variants are created for targets specified for targets specified by compiler options -x or -ax

See also:

- [DECLARE SIMD; PROCESSOR Clause; vecabi, Qvecabi](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Enforce the Compiler to Ignore Assumed Vector Dependencies

No real dependencies were detected, so there is no need for conflict-detection instructions. To fix: Tell the compiler it is safe to vectorize using a directive `!DIR$ IVDEP`.

NOTE This fix may be unsafe in other scenarios; use with care to avoid incorrect results.

```
!DIR$ IVDEP
do i = 1, N
  a(index(i)) = b(i) * c
enddo
```

See also:

- [IVDEP](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Opportunity for Outer Loop Vectorization

This is outer (non-innermost) loop. Normally outer loops are not targeted by auto-vectorization. Outer loop vectorization is also possible and sometimes more profitable, but requires explicit vectorization using OpenMP* API or Intel® Cilk™ Plus.

Collect Trip Counts Data

The Survey Report lacks trip counts data that might prove profitability for outer loop vectorization. To fix: Run a [Trip Counts analysis](#).

Check Dependencies for Outer Loop

It is not safe to force vectorization without knowing that there are no dependencies. **Disable inner vectorization before check Dependency.** To check: Run a [Dependencies analysis](#).

Check Memory Access Patterns for Outer Loop

To ensure that outer loop has optimal memory access patterns run a [Memory Access Patterns analysis](#).

Consider Outer Loop Vectorization

The compiler never targets loops other than innermost ones, so it vectorized the inner loop while did not vectorize the outer loop. However outer loop vectorization could be more profitable because of better Memory Access Pattern, higher Trip Counts or better Dependencies profile.

To enforce outer loop vectorization:

Target	Directive
Outer Loop	!\$OMP SIMD
Inner Loop	!\$OMP NOVECTOR

```
!$OMP SIMD
DO I=1,N
  !$OMP NOVECTOR
  DO J=1,N
    SUM = SUM + A(i)*A(j)
  ENDDO
ENDDO
```

See also:

- [SIMD Directive \(OpenMP* API\); VECTOR and NOVECTOR](#)
- [Outer Loop Vectorization, Vectorization Resources for Intel® Advisor Users](#)

Consider Outer Loop Vectorization.

The compiler did not vectorize the loop as the code exceeds the compilers complexity criteria. You might get higher performance if you enforce the loop vectorization. Use a directive right before your loop block in the source code.

ICL/ICC/ICPC Directive
!\$OMP SIMD

See also:

- [SIMD Directive \(OpenMP* API\)](#)
- [Outer Loop Vectorization, Vectorization Resources for Intel® Advisor Users](#)

Consider Outer Loop Vectorization

The compiler did not vectorize the inner loop due to potential dependencies detected. You might vectorize outer loop if it has no dependency. Use a directive right before your loop block in the source code.

ICL/ICC/ICPC Directive
!\$OMP SIMD

See also:

- [SIMD Directive \(OpenMP* API\)](#)
- [Outer Loop Vectorization, Vectorization Resources for Intel® Advisor Users](#)

Potential Underutilization of Approximate Reciprocal Instructions

Your current hardware supports Advanced Vector Extensions 512 (AVX-512) instructions that enable the use of approximate reciprocal and reciprocal square root instructions both for single- and double-precision floating-point calculations. Improve performance by utilizing these instructions.

Force Vectorization If Possible

The loop contains SQRT/DIV instructions (so vectorization could be beneficial), but is not vectorized. To fix, review:

- Corresponding compiler diagnostic to check if vectorization enforcement is possible and profitable
- The Dependencies analysis to distinguish between compiler-assumed dependencies and real dependencies

See also:

- [Vectorization Resources for Intel® Advisor Users](#)

Target the AVX-512 ISA

Static analysis presumes the loop may benefit from AVX-512 approximate reciprocal instructions, but these instructions were not used. To fix: Use one of the following compiler options:

- `xCOMMON-AVX512` - tells the compiler which processor features to target, including instructions sets and optimizations it may generate, including AVX-512.
- `axCOMMON-AVX512` - tells the compiler to generate multiple, feature-specific, auto-dispatch code for Intel processors if there is a performance benefit.

Windows OS	Linux OS
<code>/QxCOMMON-AVX512</code> or <code>/QaxCOMMON-AVX512</code>	<code>-xCOMMON-AVX512</code> or <code>-axCOMMON-AVX512</code>

See also:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Target the AVX-512 Exponential and Reciprocal Instructions ISA

Static analysis presumes the loop may benefit from AVX-512 Exponential and Reciprocal (AVX-512ER) instructions currently supported only on Intel® Xeon Phi™ processors, but these instructions were not used. To fix: Use one of the following compiler options:

- `xMIC-AVX512` - tells the compiler which processor features to target, including instructions sets and optimizations it may generate, including AVX-512ER.
- `axMIC-AVX512` - tells the compiler to generate multiple, feature-specific, auto-dispatch code for Intel processors if there is a performance benefit.

Windows OS	Linux OS
<code>/QxMIC-AVX512</code> or <code>/QaxMIC-AVX512</code>	<code>-xMIC-AVX512</code> or <code>-axMIC-AVX512</code>

See also:

- [ax, Qax; x, Qx](#)
- [Code Generation Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Enable the Use of Approximate Reciprocal Instructions by Fine-Tuning Precision and Floating-Point Model Compiler Options

Static analysis presumes the loop may benefit from using approximate reciprocal instructions, but the precision and floating-point model settings may prevent the compiler from using these instructions. To fix: Fine-tune your usage of the following compiler options:

Windows OS	Linux OS	Comment
/fp	-fp-model	-fp-model=precise prevents the use of approximate reciprocal instructions.
/Qimf-precision	-fimf-precision	Consider using -fimf-precision=medium or -fimf-precision=low.
/Qimf-accuracy-bits	-fimf-accuracy-bits	Consider decreasing this setting.
/Qimf-max-error	-fimf-max-error	Consider increasing this setting.
		There is a similar option: -fimf-absolute-error. Avoid using both options at the same time or tune them together.
/Qimf-absolute-error	-fimf-absolute-error	Consider using -fimf-max-error instead and set -fimf-absolute-error=0 (default) or increase this setting together with -fimf-max-error.
/Qimf-domain-exclusion	-fimf-domain-exclusion	Consider increasing this setting. More excluded classes enable more optimized code. USE WITH CAUTION. This option may cause incorrect behavior if your calculations involve excluded domains.
/Qimf-arch-consistency	-fimf-arch-consistency	-fimf-arch-consistency=true may prevent the use of approximate reciprocal instructions.
/Qprec-div	-prec-div	-prec-div prevents the use of approximate reciprocal instructions.
/Qprec-sqrt	-prec-sqrt	-prec-sqrt prevents the use of approximate reciprocal instructions.

See also:

- [Floating-point Operations and Floating-point Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Possible Inefficient Conflict-Detection Instructions Present

Stores with indirect addressing caused the compiler to assume a potential dependency.

This resulted in the use of conflict-detection instructions during SIMD processing, such as the AVX-512 vpconflict instruction, which detects duplicate values within a vector and creates conflict-free subsets. Improve performance by removing the need for conflict-detection instructions.

Enforce the Compiler to Ignore Assumed Vector Dependencies

No real dependencies were detected, so there is no need for conflict-detection instructions. To fix: Tell the compiler it is safe to vectorize using a directive `!DIR$ IVDEP`.

NOTE This fix may be unsafe in other scenarios; use with care to avoid incorrect results.

```
!DIR$ IVDEP
do i = 1, N
  a(index(i)) = b(i) * c
enddo
```

See also:

- [IVDEP](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Unoptimized Floating-Point Operation Processing Possible

Improve performance by enabling approximate operations instructions.

Enable the Use of Approximate Division Instructions

Static analysis presumes the loop may benefit from using approximate calculations. Independent divisors will be pre-calculated and replaced with multipliers. To fix: Fine-tune your usage of the following compiler option:

Windows OS	Linux OS	Comment
/Qprec-div	-no-prec-div	-no-prec-div enables the use of approximate division optimizations.

See also:

- [prec-div, Qprec-div](#)
- [Floating-point Operations](#) and [Floating-point Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Enable the Use of Approximate sqrt Instructions

Static analysis presumes the loop may benefit from using approximate sqrt instructions, but the precision and floating-point model settings may prevent the compiler from using these instructions. To fix: Fine-tune your usage of the following compiler option:

Windows OS	Linux OS	Comment
/Qprec-sqrt	-no-prec-sqrt	-no-prec-sqrt enables the use of approximate sqrt optimizations.

See also:

- [prec-sqrt, Qprec-sqrt](#)
- [Floating-point Operations](#) and [Floating-point Options](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Potential Excessive Caching Present

Enable Non-Temporal Store

Enable non-temporal store using `!DIR$ vector nontemporal`. The nontemporal clause directs the compiler to use non-temporal (that is, streaming) stores, optionally takes a comma-separated list of variables.

Streaming stores may cause significant performance improvements over non-streaming stores for large numbers on certain processors. However, the misuse of streaming stores can significantly degrade performance.

```
!DIR$ vector nontemporal
do i=1,N
  arr1(i) = 0
end do
```

See also:

- [VECTOR and NOVECTOR](#)
- [Vectorization Resources for Intel® Advisor Users](#)

Misaligned Loop Code Present

Current placement of the loop in memory may result in inefficient use of the CPU front-end. Improve performance by aligning loop code.

Force the Compiler to Align Loop Code

Caution Excessive code alignment may increase application binary size and decrease performance.

Static analysis shows the loop may benefit from code alignment. To fix: Force the compiler to align the loop to a power-of-two byte boundary using a compiler directive for finer-grained control: `!DIR$ CODE_ALIGN [:n]`

Align inner loop to 32-byte boundary:

```
!DIR$ CODE_ALIGN :64
do i = 1, n, 1
  do j = 1, m, 1
    a(i) = a(i) * (b(i) + c(j))
  enddo
enddo
```

You may also need the following compiler option:

Windows OS	Linux OS and Mac OS
/Qalign-loops[:n]	-falign-loops[=n]

where n = a power of 2 between 1 and 4096, such as 1, 2, 4, 8, 16, 32, etc. n = 1 performs no alignment. If n is not present, the compiler uses an alignment of 16 bytes. Suggestion: Try 16 and 32 first.

/Qalign-loops- and -fno-align-loops, the default compiler option, disables special loop alignment.

User Interface Reference

This section provides context-sensitive reference topics for Intel® Advisor user interface elements, typically

accessed via **Learn More** link,  Context Help button, or F1 button.



Dialog Box: Corresponding Command Line

Purpose

Use this dialog box to generate command lines for perspective or analysis configuration and copy the lines to a clipboard to run from a terminal/command prompt. For more information, see [Generating Command Lines from GUI](#).

Location

To access this dialog box, do one of the following:

- To generate a command line for the entire perspective, click the  button at the top of the **Analysis Workflow** pane.
- To generate a command line for a specific analysis, click the  button next to an analysis type in the **Analysis Workflow** pane.

Controls

Use This	To Do This
Command Line text box	View the full command line for running the required analysis type with the selected options.
Copy button	Copy the command line into clipboard.

Use This	To Do This
Close button	Close the dialog box.
Hide knobs with default values checkbox	Show/hide default options in the generated command line.
Generate command line for MPI checkbox	Generate command line for running analysis on an MPI application.

Dialog Box: Create a Project

Purpose

Use the **Create a Project** dialog box to create and configure your new Intel® Advisor project.

Location

To open the **Create a Project** dialog box, do one of the following:

- Click the  **Create Project** button on the Intel Advisor toolbar.
- Click the **New Project** button on the **Welcome** pane.
- Click **New > Project** in the **File** menu.
- Press **Ctrl+Shift+N**.

Controls

Use This	To Do This
Project name field	Specify the name of your Intel Advisor project. This might be similar to the target executable name. The project name is used for the project directory name: <ul style="list-style-type: none"> • A project file that identifies the target to be analyzed and a set of configurable attributes for running the target. • Results that allows you to view the collected data.
Location field and Browse button	Choose or create a directory to contain the project directory. Click the Browse button to browse to and select a directory where the project directory will be created. Project files should be located in a different directory than your source directories, such as a directory above the source directories or in a separate projects directory. You must have write permission to the specified directory and its subdirectories.
Create project button	After entering the Project name and specifying its Location , click Create project to create the project and its directory and open the Project Properties dialog box and configure your project .

Dialog Box: Create a Result Snapshot

Purpose



Intel® Advisor stores only the most recent analysis result. Use this dialog box to save a read-only result snapshot you can view any time.

Tip

- Visually comparing one or more snapshots to each other or to the most recent analysis result can be an effective way to judge performance improvement progress.
- To view a snapshot, choose **File > Open > Result...**
- Snapshots are identified by a different icon in the Visual Studio* Solution Explorer and the Intel® Advisor **Project Navigator**. The words **(read-only)** appear after the snapshot name in a result tab.

Location

To open the dialog box, do one of the following:

- Click the  button in the analysis result.
- Click the  button in the main toolbar.

Controls

Use This	To Do This
Result name field	Specify the name of the read-only result snapshot. Provide a unique name, perhaps by adding an identifying suffix within the result name.
Cache sources checkbox	Enable source code availability in the resulting snapshot.
Cache binaries checkbox	Enable binary availability in the resulting snapshot.
Pack into archive checkbox	Create a one-file archive with all snapshot data inside.
Result path text box	Specify the path to the resulting snapshot archive. Use the Browse... button to specify the address. Disabled by default. Enable by selecting the Pack into archive checkbox.

Dialog Box: Options - Assembly**Purpose**

Use this tab to set assembly code style.

Location

To access this tab:

- In the Intel® Advisor GUI, choose **File > Options..** On the left of the **Options** dialog box, choose **Assembly** property page.

Controls

Use radio buttons on the **Assembly** property page to set style for your assembly code. The following options are available:

- **Default Syntax MASM style for Windows*, GAS style for Unix.**
- **GAS Style Syntax** - use strictly GAS syntax.

- MASM Style Syntax** - use strictly MASM syntax.

Enable the checkbox under assembly code style radio buttons to enclose the Intel® AVX-512 write mask in curly braces that omits the `k0` register.

Editor Tab

Purpose

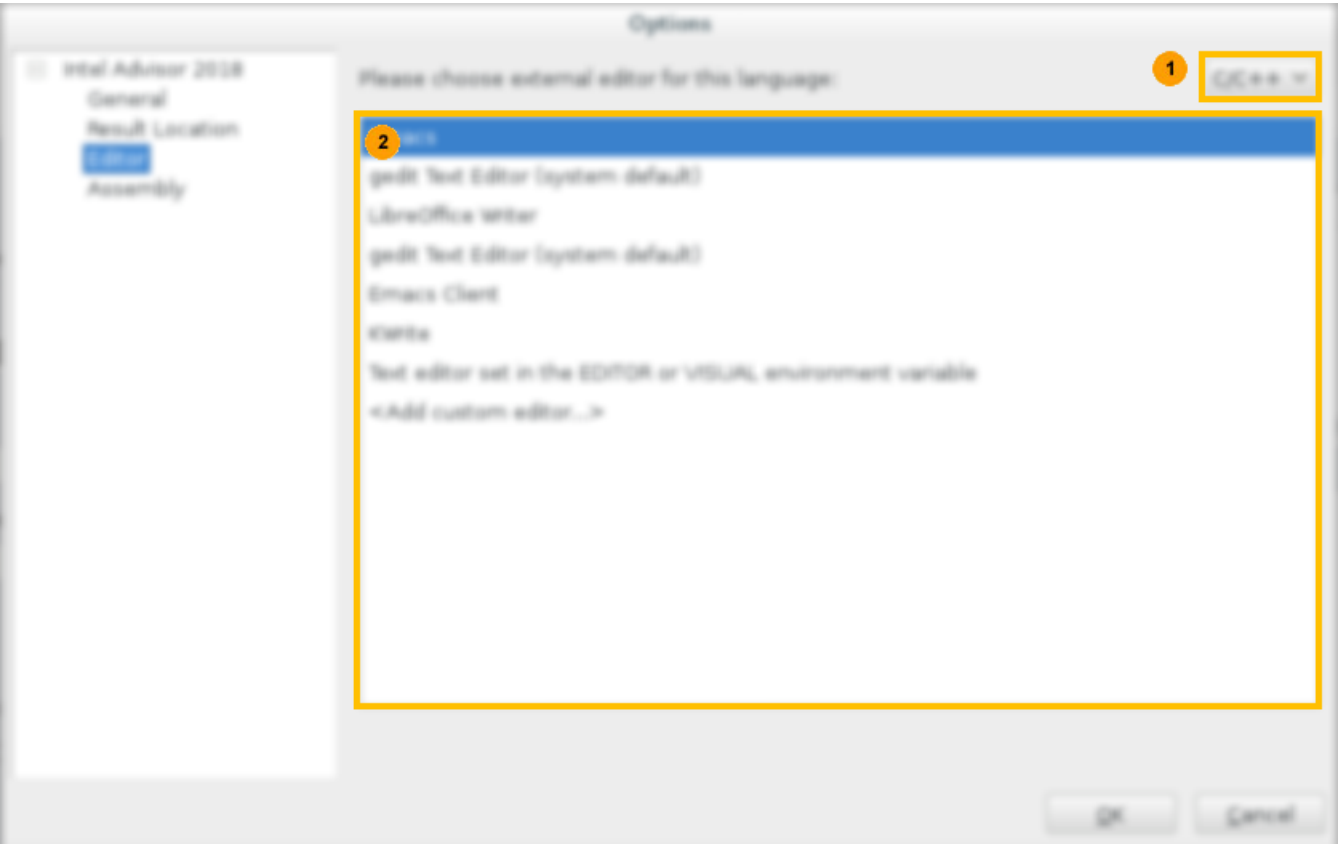
NOTE
The **Editor** tab is available on Linux* OS only.

Use this tab in the **Options** dialog box to specify the editor in which the Intel® Advisor displays source files when you double-click a line in an Intel Advisor source region.


Location

To access this tab: From the Intel Advisor GUI, choose **File > Options > Editor**.

Controls



Use This	To Do This
1 External editor for this language: drop-down menu	Select the language for which you will choose an editor: C/C++ , Fortran , or Other .

Use This	To Do This
 list of available editors on this system	<p>Select the editor (such as <code>gedit</code>) to be associated with the selected language , or choose to allow selection using an environment variable with Text editor set in EDITOR or VISUAL environment variable. Repeat to associate an editor with each language you use.</p> <p>For example, if you choose Text editor set in EDITOR or VISUAL environment variable, you can set the VISUAL environment variable by typing: <code>export VISUAL="/usr/bin/vi -n"</code></p> <p>When done, click OK.</p>

Dialog Box: Options - General

Purpose

Use this tab to configure default behavior, enable/disable warning messages, and set modeling assumptions, result locations.

Location

To access this tab:

- From the Intel® Advisor GUI, choose **File > Options > General**.
- From the Visual Studio* menu, choose **Tools > Options**. In the **Options** dialog box, expand the **Intel Advisor** program folder and choose the **General** page.

Controls

Use This	To Do This
When displaying a window, show its explanation tip checkbox	Determine whether a help snippet explanation for the current window appears when the Result is opened.
Show the Advisor Workflow tab when a new collection starts checkbox	Control whether the Advisor Workflow tab automatically opens when you run a tool analysis.
Show build settings warning before a new collection starts checkbox	Control whether a message appears when the build settings do match the suggested settings for the selected analysis: <ul style="list-style-type: none"> Analysis using the Debug build settings message appears when you run Survey or Suitability tool analysis with Debug build options. Analysis using the Release build settings message appears when you run Dependencies tool analysis with Release build options.
Show missing debug information warning checkbox	Control whether a message appears near the top of the Survey Report window when the target executable does not contain debug information after running the Survey tool.
Show incorrect compiler options warning checkbox	Control whether a message appears near the top of the Survey Report window when the current compiler options are incorrect.

Use This	To Do This
Show incorrect compiler version warning checkbox	Control whether a message appears near the top of the Survey Report window when a higher compiler version should be installed.
Show higher ISA available warning checkbox	Control whether a message appears near the top of the Survey Report window when a higher Instruction Set Architecture should be used.
Show inline debug information warning checkbox	Control whether a message appears near the top of the Survey Report window when debug information is in the code.
Modeling Assumptions drop-down lists	<p>Set the default values for Modeling Assumptions that appear in the Suitability Report window, such as the scalability graph.</p> <ul style="list-style-type: none"> • Maximum CPU Count - specify the maximum number of CPUs to model. This value limits the size you can set for the CPU Count; it also sets the size of the scalability graph CPU Count (X axis). You can set values using power-of-two integers 2, 4, 8, ... up to 8192. • CPU Count - specify the number of CPUs to model for the target system(s). Set a value using power-of-two integers from 2 up to the chosen Maximum CPU Count. This value sets the default for the Suitability Report window. • Threading Mode - choose either Intel TBB, OpenMP, Microsoft TPL, or Other.
Application output destination radio button	<p>On Windows* OS systems, control whether console output for a program is displayed in the:</p> <ul style="list-style-type: none"> • Separate console window. • Microsoft Visual Studio Output window. • Application Output window (in the Intel Advisor result tab). This also enables application output to be viewed after collection by clicking a link in the Summary window to display the Application Output window. <p>The next time you run a tool analysis, the application output appears in the selected output destination.</p> <p>On Linux* OS systems, control whether the console output from the target is displayed in the Intel Advisor Application Output window (in the Result tab) or in a separate command terminal window.</p> <hr/> <p>NOTE</p> <p>If you must interact with the application during execution, choose the separate console window option or use <code>stdin</code> redirection on the command line.</p> <hr/>
Font Settings group box	Select the font size to use in the Intel Advisor Interface. Select the Use system default to use the same font size your operating system uses.

Dialog Box: Options - Result Location

Purpose

Use this tab to specify the storage directory for future Intel® Advisor result files.

Location

To access this tab, do one of the following:

- From the Intel Advisor GUI, choose **File > Options > Result Location**.
- From the Visual Studio* menu, choose **Tools > Options....** In the **Options** dialog box, expand the **Intel Advisor** program folder and choose the **Result Location** page.

Controls

Use This	To Do This
Result location radio button	Determine whether new result files are saved in a subdirectory within each Microsoft Visual Studio* or Intel Advisor GUI project's directory, or in a custom, central location that you specify. If you select the Save all results in this directory: option, either type the path or click the Browse button to navigate to the desired custom directory location. The subdirectory name is the result name, such as e000. When done, click OK .

Dialog Box: Project Properties - Analysis Target

Purpose

Use this tab to specify the target executable, set important project properties, and review current project properties.

Tip

Always check project property values before analyzing a new target.

Location

Analysis Target tab is located in the **Project Properties** dialog box.

To access the **Project Properties** dialog box, do one of the following:

- Click the  button on the main toolbar.
- Choose **File > Project Properties....**
- Press **Ctrl+P**.

Controls

In the **Analysis Target** tab, select an analysis type from list (on the left) to display and configure project properties.

The following table covers project properties applicable to all analysis types. To view controls applicable only to a specific analysis type, use the links immediately below:

- [Survey Analysis Controls](#)
- [Trip Counts and FLOPS Controls](#)
- [Suitability Analysis Controls](#)
- [MAP Analysis Controls](#)
- [Dependencies Analysis Controls](#)

Common Controls

The following controls are common for all analysis types. Specify the properties in the **Survey Hotspot Analysis** tab and check that the **Inherit Settings from the Survey Hotspots Analysis Type** checkbox is enabled in other tabs to share the properties for all analyses.

Use This	To Do This
Target type drop-down	<ul style="list-style-type: none"> Analyze an executable or script (choose Launch Application). Analyze a process (choose Attach to Process). <p>If you choose Attach to Process, you can either inherit settings from the Survey Hotspots Analysis Type or specify the needed settings.</p>
Inherit settings from Visual Studio project checkbox and field (Visual Studio* IDE only)	<p>Inherit Intel Advisor project properties from the Visual Studio* startup project (enable).</p> <p>If enabled, the Application, Application parameters, and Working directory fields are pre-filled and cannot be modified.</p> <hr/> <p>NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.</p> <hr/>
Application field and Browse... button	<p>Select an analysis target executable or script.</p> <p>If you specify a script in this field, consider specifying the executable in the Advanced > Child application field (required for Dependencies analysis).</p>
Application parameters field and Modify... button	Specify runtime arguments to use when performing analysis (equivalent to command line arguments).
Use application directory as working directory checkbox	Automatically use the value in the Application directory to pre-fill the Working directory value (enable).
Working directory field and Browse... button	Select the working directory.
User-defined environment variables field and Modify... button	Specify environment variables to use during analysis.
Managed code profiling mode drop-down	<ul style="list-style-type: none"> Automatically detect the type of target executable as Native or Managed, and switch to that mode (choose Auto). Collect data for native code and do not attribute data to managed code (choose Native). Collect data for both native and managed code, and attribute data to managed code as appropriate (choose Mixed). Consider using this option when analyzing a native executable that makes calls to managed code. Collect data for both native and managed code, resolve samples attributed to native code, and attribute data to managed source only (choose Managed). The call stack in the analysis result displays data for managed code only.

Use This	To Do This
Child application field	<p>Analyze a file that is not the starting application. For example: Analyze an executable (identified in this field) called by a script (identified in the Application field).</p> <p>Invoking these properties could decrease analysis overhead.</p> <hr/> <p>NOTE For the Dependencies Analysis Type: If you specify a script file in the Application field, you must specify the target executable in the Child application field.</p> <hr/>
Modules radio buttons, field, and Modify... button	<ul style="list-style-type: none"> Analyze specific modules and disable analysis of all other modules (click the Include only the following module(s) radio button and choose the modules). Disable analysis of specific modules and analyze all other modules (click the Exclude only the following module(s) radio button and choose the modules). <p>Including/excluding modules could minimize analysis overhead.</p>
Use MPI launcher checkbox	<p>Generate a command line (enable) that appears in the Get command line field based on the following parameters:</p> <ul style="list-style-type: none"> Select MPI Launcher - Intel or another vendor Number of ranks - Number of instances of the application Profile ranks - All or a range of ranks to profile
Automatically stop collection after (sec) checkbox and field	<p>Stop collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could minimize analysis overhead.</p>

Survey Analysis-Specific Controls

Use This	To Do This
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <hr/> <p>Tip The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p> <hr/>
Sampling Interval selector	<p>Set the wait time between each analysis collection CPU sample while your target application is running.</p> <p>Increasing the wait time could decrease analysis overhead.</p>

Use This	To Do This
Collection data limit, MB selector	Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses. Decreasing the limit could decrease analysis overhead.
Callstack unwinding mode drop-down list	Set to After collection if: <ul style="list-style-type: none"> Survey analysis runtime overhead exceeds 1.1x. A large quantity of data is allocated on the stack, which is a common case for Fortran applications or applications with a large number of small, parallel, OpenMP* regions. Otherwise, set to During Collection .
Stitch stacks checkbox	Restore a logical call tree for Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP* applications by catching notifications from the runtime and attaching stacks to a point introducing a parallel workload (enable). Disable if Survey analysis runtime overhead exceeds 1.1x.
Analyze MKL Loops and Functions checkbox	Show Intel® oneAPI Math Kernel Library loops and functions in Intel Advisor reports (enable). Enabling could increase analysis overhead.
Analyze Python loops and functions checkbox	Show Python* loops and functions in Intel Advisor reports (enable). Enabling could increase analysis overhead.
Analyze loops that reside in non-executed code paths checkbox	Collect a variety of data during analysis for loops that reside in non-executed code paths, including loop assembly code, instruction set architecture (ISA), and vector length (enable). Enabling could increase analysis overhead. NOTE Analyzing non-executed code paths in binaries that target multiple ISAs (contain multiple code paths) is available only for binaries compiled using the <code>-ax</code> (Linux* OS) / <code>Qax</code> (Windows* OS) option with an Intel compiler.
Enable registry spill/fill analysis checkbox	Calculate the number of consecutive load/store operations in registers and related memory traffic (enable). Enabling could increase analysis overhead.
Enable static instruction mix analysis checkbox	Statically calculate the number of specific instructions present in the binary (enable). Enabling could increase analysis overhead.
Source caching drop-down list	<ul style="list-style-type: none"> Delete source code cache from a project with each analysis run (default; choose Clear cached files). Keep source code cache within the project (choose Keep cached files).

Trip Counts and FLOP Analysis-Specific Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Automatically resume collection after (sec) checkbox and field	Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds). Invoking this property could decrease analysis overhead. Tip The corresponding CLI action option is <code>--resume-after=<integer></code> , where the integer argument is in milliseconds, not seconds.
Collect information about Loop Trip Counts checkbox	Measure loop invocation and execution (enable).
Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage checkbox	Measure floating-point operations, integer operations, and memory traffic (enable).
Collect callstacks checkbox	Collect call stack information when performing analysis (enable). Enabling could increase analysis overhead.
Capture metrics for dynamic loops and functions checkbox	Collect metrics for dynamic Just-In-Time (JIT) generated code regions.
Capture metrics for stripped binaries checkbox	Collect metrics for stripped binaries. Enabling could increase analysis overhead.
Enable Memory-Level Roofline with cache simulation checkbox	Model multiple levels of cache for data, such as counts of loaded or stored bytes for each loop, to plot the Roofline chart for all memory levels (enable). Enabling could increase analysis overhead.
Cache simulator configuration field	Specify a cache hierarchy configuration to model (enable and specify hierarchy). The hierarchy configuration template is: <code>[num_of_level1_caches]:[num_of_ways_level1_connected]:</code> <code>[level1_cache_size]:[level1_cacheline_size]/</code> <code>[num_of_level2_caches]:[num_of_ways_level2_connected]:</code> <code>[level2_cache_size]:[level2_cacheline_size]/</code> <code>[num_of_level3_caches]:[num_of_ways_level3_connected]:</code> <code>[level3_cache_size]:[level3_cacheline_size]</code> For example: <code>4:8w:32k:64l/4:4w:256k:64l/1:16w:6m:64l</code> is the hierarchy configuration for:

Use This	To Do This
	<ul style="list-style-type: none"> Four eight-way 32-KB level 1 caches with line size of 64 bytes Four four-way 256-KB level 2 caches with line size of 64 bytes One sixteen-way 6-MB level 3 cache with line size of 64 bytes
Data transfer simulation mode drop-down	<p>Select a level of details for data transfer simulation:</p> <ul style="list-style-type: none"> Off - Disable data transfer simulation analysis. Light - Model data transfers between host and device memory. Full - Model data transfers, attribute memory objects to loops that accessed the objects, and track accesses to stack memory.

Suitability Analysis-Specific Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	<p>Copy similar settings from Survey analysis properties (enable).</p> <p>When enabled, this option disables application parameters controls.</p>
Automatically resume collection after (sec) checkbox and field	<p>Start running your target application with collection paused, then resume collection after a specified number of seconds (enable and specify seconds).</p> <p>Invoking this property could decrease analysis overhead.</p> <hr/> <p>Tip The corresponding CLI action option is <code>--resume-after=<integer></code>, where the integer argument is in milliseconds, not seconds.</p> <hr/>
Sampling Interval selector	<p>Set the wait time between each analysis collection sample while your target application is running.</p> <p>Increasing the wait time could decrease analysis overhead.</p>
Collection data limit, MB selector	<p>Set the amount of collected raw data if exceeding a size threshold could cause issues. Not available for hardware event-based analyses.</p> <p>Decreasing the limit could decrease analysis overhead.</p>

Memory Access Patterns Analysis-Specific Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	<p>Copy similar settings from Survey analysis properties (enable).</p> <p>When enabled, this option disables application parameters controls.</p>
Suppression mode group box	<ul style="list-style-type: none"> Report possible memory issues in system modules (choose the Show problems in system modules radio button). Do not report possible memory issues in system modules (choose the Suppress problems in system modules radio button).

Use This	To Do This
Loop call count limit selector	Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances. Supplying a non-zero value could decrease analysis overhead.
Instance of interest selector	Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes. Supplying a non-zero value could decrease analysis overhead.
Report stack variables checkbox	Report stack variables for which memory access strides are detected (enable). Enabling could increase analysis overhead.
Report heap allocated variables checkbox	Report heap-allocated variables for which memory access strides are detected (enable). Enabling could increase analysis overhead.
Enable CPU cache simulation checkbox	Model cache misses, cache misses and cache line utilization, or cache misses and loop footprint (enable and select desired options). Enabling could increase analysis overhead.
Cache associativity drop-down list	Set the cache associativity for modeling CPU cache behavior. You can set the value to the following power-of-two integers: 1, 2, 4, 8, 16.
Cache sets drop-down list	Set the cache set size (in bytes) for modeling CPU cache behavior. You can set the value to the following power-of-two integers: 256, 512, 1024, 2048, 4096, 8192.
Cache line size drop-down list	Set the cache line size (in bytes) to model CPU cache behavior. You can set the value to the following power-of-two integers: 4, 8, 16, 32, ..., up to 65536.
Cache simulation mode drop-down list	Set the focus for modeling CPU cache behavior: <ul style="list-style-type: none"> • Model cache misses only. • Model cache misses and memory footprint of a loop. Calculation: Cache line size x Number of unique cache lines accessed during simulation. • Model cache misses and cache line utilization.

Dependencies Analysis Controls

Use This	To Do This
Inherit settings from the Survey Hotspots Analysis Type checkbox	Copy similar settings from Survey analysis properties (enable). When enabled, this option disables application parameters controls.
Suppression mode radio buttons	<ul style="list-style-type: none"> • Report possible dependencies in system modules (choose the Show problems in system modules radio button). • Do not report possible dependencies in system modules (choose the Suppress problems in system modules radio button).

Use This	To Do This
Loop call count limit selector	Choose the maximum number of instances each marked loop is analyzed. 0 = analyze all loop instances. Supplying a non-zero value could decrease analysis overhead.
Instance of interest selector	Analyze the nth child process, where 1 = the first process of the specified name in the application process tree. 0 = analyze all processes. Supplying a non-zero value could decrease analysis overhead.
Analyze stack variables checkbox	Analyze parallel data sharing for stack variables (enable). Enabling could increase analysis overhead.
Filter stack variables by scope checkbox	Enable to report: <ul style="list-style-type: none"> Variables initiated inside the loop as potential dependencies (warning) Variables initialized outside the loop as dependencies (error) Enabling could increase analysis overhead.
Filter reduction variables checkbox	Mark all potential reductions by a specific diagnostic (enable). Enabling could increase analysis overhead.

Performance Modeling-Specific Controls

Use This	To Do This
Device configuration	Select a pre-defined hardware configurations from a drop-down list to model application performance on.
Other parameters	Enter a space-separated list of command-line parameters. For a full list of available options, see analyse.py Options .

Dialog Box: Project Properties - Binary/Symbol Search

Purpose

Use this tab to specify non-standard directories for the supporting files needed to execute and analyze the target. With Visual Studio* on Windows* OS, you can instead use the Visual Studio solution and project capabilities to search for specific directories.





Location

Binary/Symbol Search tab is located in the **Project Properties** dialog box.

To access the **Project Properties** dialog box, do one of the following:

- Click the  button on the main toolbar.
- Choose **File > Project Properties....**
- Press **Ctrl+P**.

Controls

Use This	To Do This
 button	On a row containing Add new search location , click to browse for directories to include in the search list. You can also type directly in the row. In addition to local directories, you can specify a symbol server URL.
 and  buttons	Change the search order of the selected directory by moving it up or down. To select multiple rows, use the Ctrl or Shift keys.
 button	Delete a selected directory row(s).
Search recursively checkbox	Enable to search the specified location subdirectories. To use recursive search, the lines must provide only a directory name and omit a file name. Using a recursive search for multiple directories may slow processing and could lead to unexpected results.

Dialog Box: Project Properties - Source Search

Purpose

Use this tab to specify the source search locations needed to execute and analyze the target. With Visual Studio*, some source locations are pre-populated from the Visual Studio startup project into the internal representation of Intel® Advisor project properties, so you may not need to add new row(s).

Tip

For Threading perspective only: Intel® Advisor does not automatically populate source locations after you create a project using the Intel® Advisor GUI, so you must specify one or more locations to find application annotations. View the **Annotation Report** to verify all project annotations are found.





Location


Source Search tab is located in the **Project Properties** dialog box.

To access the **Project Properties** dialog box, do one of the following:

- Click the  button on the main toolbar.
- Choose **File > Project Properties....**
- Press **Ctrl+P**.

Controls

Use This	To Do This
 button	On a row containing Add new search location , click to browse for directories to include in the search list. You can also type directly in the row.
 and  buttons	Change the search order of the selected directory by moving it up or down. To select multiple rows, use the Ctrl or Shift keys.
 button	Delete a selected directory row(s).

Use This	To Do This
Search recursively checkbox	Enable to search the specified location subdirectories. To use recursive search, the lines must provide only a directory name and omit a file name. Using a recursive search for multiple directories may slow processing and could lead to unexpected results.
Mask text box	Specify the file name mask pattern(s) to ignore (skip) using wildcard characters, such as an asterisk (*). For example, you can skip certain file suffixes.
File text box	Specify the file(s) to ignore (skip) using an absolute path. To delete a row, use the  button.

Pane: Advanced View

Use this pane to get detailed information about a specific function/loop.

Location

View the Advanced view pane at the bottom of the Survey Report and CPU Roofline report windows.

Source Tab

Use this tab to view source code for a selected function/loop.

Double click a code line to open source code in a code editor. Right-click a code line to open a context menu with the following options:

- **Edit Source** - open source code in a code editor.
- **Copy to Clipboard** - copy source code for a selected function/loop.
- **What Should I Do Next?** - open information about your next steps in a web-browser.

Top Down Tab

Use this tab to view the hierarchy of functions/loops, locate a selected function/loop in call chains, and view analysis results for a selected function/loop.

View the function/loop hierarchy in a stack, the source code associated with a specific function or loop, and more. Each function or loop appears on a separate grid line. Loops are identified with an icon and the word loop, followed by the function or procedure name that executes it and the source location.

There are two main regions in the **Top Down** tab:

- **Function Call Sites and Loops:** View a hierarchical listing of functions and loops in your code. You can expand and collapse entries, or double-click the name of a function or loop to view its source code.
- **Table Columns:** View additional information about functions and loops in the grid, such as CPU time, type (function, scalar, etc.), compute performance statistics, the instruction sets and extensions (such as VNNI) used, and trip counts. See Data Reference for descriptions of the data columns in Survey and Refinement Reports.

Controls

You can customize the columns shown in the Top Down grid. Click the **Customize View** button in the upper right of the application to display the **View Layout** drop-down list and the **Settings** control (gear icon) in the upper-right of the Top Down tab.

Select a column layout from the **View Layout** drop-down list to change the columns to match an existing column layout.

You can modify a column layout. Select the **Settings** control next to the View Layout drop-down list to open the **Configure Columns** dialog box, then:

1. Choose an existing view layout in the **Configuration** drop-down list.
2. Enable/disable columns to show/hide.

Outcome: A new view layout is added to the **Configuration** drop-down list, with Copy n added to the name of the original layout.

3. Click the **Rename** button and supply an appropriate name for the customized view layout.
4. Click **OK** to save the customized view layout.

You can also right-click the name of a column in the grid to **Hide Column**, **Show All Columns**, or **Configure Column Layouts**. You can choose to display one layout in the main Survey Report grid, and choose another layout for the Top Down tab.

NOTE Hiding or showing columns in a column layout will apply your changes to any grid (the Survey Report grid or the Top Down tab) that is currently using the layout. However, you can rearrange columns in one grid without affecting another grid.

Code Analytics Tab

Use this tab to view the most important statistics for a selected function/loop.

There are several regions available in the **Code Analytics** tab:

- **Summary:** View a quick list of basic information about the loop, such as the code source, whether the loop is scalar or vector, instruction set (and whether extensions, such as VNNI, are used), total time, self-time and the static and dynamic instruction mix.
- **Traits:** View additional scalar and vectorization characteristics that may impact performance. For a list of possible traits, see the Data Reference.
- **Trip Counts:** View information about the number of times the loop is invoked (trip count), such as the minimum and maximum trip count, the average loop iteration time, etc.
- **Statistics for <operations type>:** Click the drop-down list at the top of this section to choose to display performance statistics for a specific operation type: **FLOP**, **INTOP**, **INT + FLOAT**, or **All Operations**. Click the toggle control to switch between displaying performance statistics using self or total loop metrics.
- **Code Optimizations:** View a list of code optimizations applied to the loop by the compiler, as well as information on which compiler was used and what version. This information is only available for binaries compiled by the Intel® C, C++, or Fortran Compilers.
- **Roofline:** View a more detailed roofline chart that summarizes recommendations and information from the Roofline Conclusions section, such as whether the loop is compute bound, memory bound, or both. This chart features:
 - The labeled distance between the loop and the performance roof limiting it.
 - The metrics used to plot the loop on the chart, Giga OPS (operations per second) and AI (arithmetic intensity).

If you have collected the Roofline for all memory levels, you can use the **Memory Level/CARM** selector to switch between Roofline guidance views. When you set the selector to **Memory Level**, the chart features X marks representing memory levels for the loop and an arrowed line indicating the memory level that bounds the loop.

Assembly Tab

Use this tab to view assembly representation for a selected loop.


Assistance Tab (Threading Perspective Only)

Use this tab to get recommendations on making annotations in your source code.

Controls

Use a drop-down menu on the top right of this pane to view examples of annotated source code for different task code structures and recommended build settings for the language in use. The following options are available:

- **Iteration Loop, Single Task** - View and copy an annotation code snippet for a simple loop structure, where the task's code includes the entire loop body. Use this common task structure when only a single task is needed within a parallel site.
- **Loop, One or More Tasks** - View and copy an annotation code snippet for a loops where the task code does not include all of the loop body, or for complex loops or code that requires specific task begin-end boundaries, including multiple task end annotations. Also use this structure when multiple tasks are needed within a parallel site.
- **Function, One or More Tasks** - View and copy an annotation code snippet for code that calls multiple functions (task parallelism). Use this structure when multiple tasks are needed within a parallel site.
- **Pause/Resume Collection** - View and copy an annotation code snippet whose annotations temporarily pause data collection and later resume it. This lets you skip uninteresting parts of the target program's execution to minimize the data collected and speed up the analysis of large applications. Add these annotations outside a parallel site.
- **Build Settings** - View and copy build (compiler and linker) settings. The Build Settings are specific to the language in use.

View annotated code samples in the text display area. To copy the text lines to the clipboard, right-click and select **Copy to Clipboard** from the context menu, or use the  **Copy to Clipboard** button on the upper-right of the Assistance tab..

Recommendations Tab

Use this tab to explore code specific recommendations how to fix vectorization issues.

Controls

You can view a list of all performance issues detectable in Intel Advisor. Next to **All Advisor-detectable issues**, click to display either **C++** or **Fortran** issues.

Issues and recommendations are displayed in a list. Under each issue, you'll see an explanation of the issue, recommendations for how to resolve it, and code samples that you can expand or collapse. You may also see **Read More** links with additional information about the topic.

You can jump to a specific issue by clicking its name in the list of detected issues to the right.

Pane: Analysis Workflow

Use the **Analysis Workflow** pane to set up and control execution of your Intel® Advisor perspectives. This pane allows you to select and run a perspective, choose data collection accuracy level, control both the execution of the entire perspective and of each analysis separately.

To open the **Analysis Workflow** pane, click the  **Show My Result and Workflow** button on the main toolbar.

Select a Perspective

Use a drop-down list at the top of the **Analysis Workflow** pane to select a perspective you are going to run.

Intel Advisor allows you to analyze application performance using the following perspectives:

- [Vectorization and Code Insights](#)
- [CPU / Memory Roofline Insights](#)
- [Threading](#)
- [Offload Modeling](#)
- [GPU Roofline Insights](#)

Select Data Collection Accuracy Level




View the accuracy level options in the **Accuracy** pane under perspective execution controls. You can select **Low**, **Medium**, **High**, and **Custom** accuracy level depending on the analysis types you want to perform. Data collection accuracy level affects the potential overhead. View the potential overhead in the **Overhead** indicator under the **Accuracy** pane. For more information about managing overhead, see [Minimize Analysis Overhead](#).

NOTE Choosing analysis types manually automatically sets the data collection accuracy level to **Custom**.

Control Perspective Execution


Select a perspective using the drop-down menu on top of the Analysis Workflow pane. Control the execution of the entire perspective using the pane on top of the Analysis Workflow pane. There are two sets of controls available to you depending on the status of your perspective.

When the perspective is not started or paused, you can:





- Run the perspective using the  button.
- Resume its execution if the perspective is paused using the  button.
- Get a corresponding command line for your perspective to access it from CLI using the  button.

When the perspective is running, you can pause  it, stop  it, or cancel  its execution.

Control Analysis Execution

Control the execution of a specific analysis. To do this, click the  button near the analysis type you want to perform. There are two sets of controls available to you depending on the status of your analysis.

When the analysis is not started or paused, you can:

- Run it from scratch using the  button.
- Resume its execution if the analysis is paused using the  button.
- Get a corresponding command line for your analysis to access it from CLI using the  button.
- Open analysis properties pane to configure your analysis using the  button.

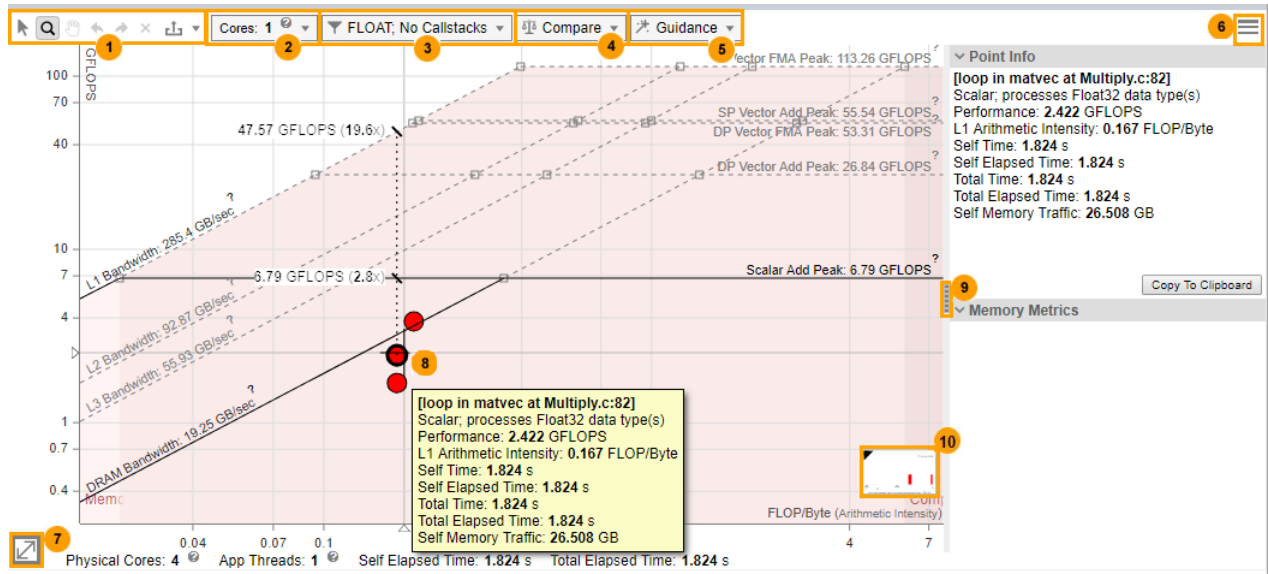
When the analysis is running, pause  it, stop  it, and cancel  its execution.

Pane: Roofline Chart

Use GPU Roofline chart to visualize actual performance of your GPU kernels against hardware-imposed performance ceilings. For more information about investigating GPU Roofline results, see [Examine Bottlenecks on CPU Roofline Chart](#).

Controls

There are several controls to help you focus on the **Roofline** chart data most important to you, including the following.



- 1
 - **Select Loops by Mouse Rect:** Select one or more loops/functions by tracing a rectangle with your mouse.
 - **Zoom by Mouse Rect:** Zoom in and out by tracing a rectangle with your mouse. You can also zoom in and out using your mouse wheel.
 - **Move View By Mouse:** Move the chart left, right, up, and down.
 - **Undo or Redo:** Undo or redo the previous zoom action.
 - **Cancel Zoom:** Reset to the default zoom level.
 - **Export as x:** Export the chart as a dynamic and interactive HTML or SVG file that does not require the Intel Advisor viewer for display. Use the arrow to toggle between the options.

- 2 Use the **Cores** drop-down toolbar to:
 - Adjust rooflines to see practical performance limits for your code on the host system.
 - Build roofs for single-threaded applications (or for multi-threaded applications configured to run single threaded, such as one thread-per-rank for MPI applications. (You can use Intel Advisor filters to control the loops displayed in the **Roofline** chart; however, the **Roofline** chart does not support the **Threads** filter.)

Choose the appropriate number of CPU cores to scale roof values up or down:

- 1 – if your code is single-threaded
- Number of cores equal or close to the number of threads – if your code has fewer threads than available CPU cores
- Maximum number of cores – if your code has more threads than available CPU cores

By default, the number of cores is set to the number of threads used by the application (even values only).

You'll see the following options if your code is running on a multisocket PC:

- Choose **Bind cores to 1 socket** (default) if your application binds memory to one socket. For example, choose this option for MPI applications structured as one rank per socket.

NOTE This option may be disabled if you choose a number of CPU cores exceeding the maximum number of cores available on one socket.

- Choose **Spread cores between all n sockets** if your application binds memory to all sockets. For example, choose this option for non-MPI applications.

- 3**
- Toggle the display between floating-point (FLOP), integer (INT) operations, and mixed operations (floating-point and integer).
 - *If you collected Roofline with Calltacks:* Enable the display of Roofline with Callstacks additions to the **Roofline** chart.

- 4**
- Display **Roofline** chart data from other Intel Advisor results or non-archived snapshots for comparison purposes.

Use the drop-down toolbar to:

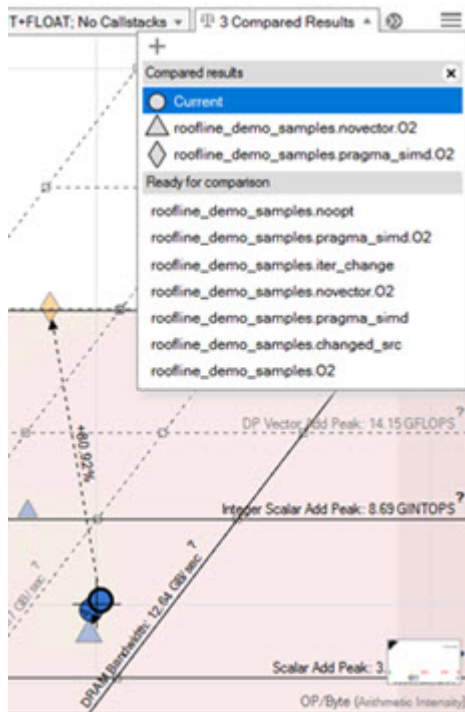
- Load a result/snapshot and display the corresponding filename in the **Compared Results** region.
- Clear a selected result/snapshot and move the corresponding filename to the **Ready for comparison** region.

Note: Click a filename in the **Ready for comparison** region to reload the result/snapshot.

- Save the comparison itself to a file.

NOTE The arrowed lines showing the relationship among loops/functions do not reappear if you upload the comparison file.

Click a loop/function dot in the current result to show the relationship (arrowed lines) between it and the corresponding loop/function dots in loaded results/snapshots.



- 5 Add visual indicators to the Roofline chart to make the interpretation of data easier, including performance limits and whether loops/functions are memory bound, compute bound, or both.

Use the drop-down toolbar to:

- Show a vertical line from a loop/function to the nearest and topmost performance ceilings by enabling the **Display roof rulers** checkbox. To view the ruler, hover the cursor over a loop/function. Where the line intersects with each roof, labels display hardware performance limits for the loop/function.
- *If you collected Roofline for All Memory Levels:* Visually emphasize the relationships among displayed memory levels and roofs and for a selected loop/function dot by enabling the **Show memory level relationships** checkbox.
- Color the roofline zones to make it easier to see if enclosed loops/functions are fundamentally memory bound, compute bound, or bound by compute and memory roofs by enabling the **Show Roofline boundaries** checkbox.

The preview picture is updated as you select guidance options, allowing you to see how changes will affect the Roofline chart's appearance. Click **Apply** to apply your changes, or **Default** to return the Roofline chart to its original appearance.

Once you have a loop/function's dots highlighted, you can zoom and fit the Roofline chart to the dots for the selected loop/function by once again double-clicking the loop/function or pressing **SPACE** or **ENTER** with the loop/function selected. Repeat this action to return to the original Roofline chart view.

To hide the labeled dots, select another loop/function, or double-click an empty space in the Roofline chart.

- 6
- **Roofline View Settings:** Adjust the default scale setting to show:
 - The optimal scale for each **Roofline** chart view
 - A scale that accommodates all **Roofline** chart views
 - **Roofs Settings:** Change the visibility and appearance of roofline representations (lines):

- Enable calculating roof values based on single-threaded benchmark results instead of multi-threaded.
- Click a **Visible** checkbox to show/hide a roofline.
- Click a **Selected** checkbox to change roofline appearance: display a roofline as a solid or a dashed line.
- Manually fine-tune roof values in the **Value** column to set hardware limits specific to your code.
- **Loop Weight Representation:** Change the appearance of loop/function weight representations (dots):
 - **Point Weight Calculation:** Change the **Base Value** for a loop/function weight calculation.
 - **Point Weight Ranges:** Change the **Size**, **Color**, and weight **Range (R)** of a loop/function dot. Click the + button to split a loop weight range in two. Click the - button to merge a loop weight range with the range below.
 - **Point Colorization:** color loop/function dots by weight ranges or by type (vectorized or scalar). You can also change the color of loop with no self time.

You can save your Roofs Settings or Point Weight Representation configuration to a JSON file or load a custom configuration.

7 Zoom in and out using numerical values.

8 Click a loop/function dot to:

- Outline it in black.
- Display metrics for it.
- Display corresponding data in other window tabs.

Right-click a loop/function dot or a blank area in the **Roofline** chart to perform more functions, such as:

- Further simplify the **Roofline** chart by filtering out (temporarily hiding a dot), filtering in (temporarily hiding all other dots), and clearing filters (showing all originally displayed dots).
- Copy data to the clipboard.

9 Show/hide the metrics pane:

- Review the basic performance metrics in the **Point Info** pane.
- *If you collected the Roofline for All Memory Levels:* Review how efficiently the loop/function uses cache and what memory level bounds the loop/function in the **Memory Metrics** pane.

10 Display the number and percentage of loops in each loop weight representation category.

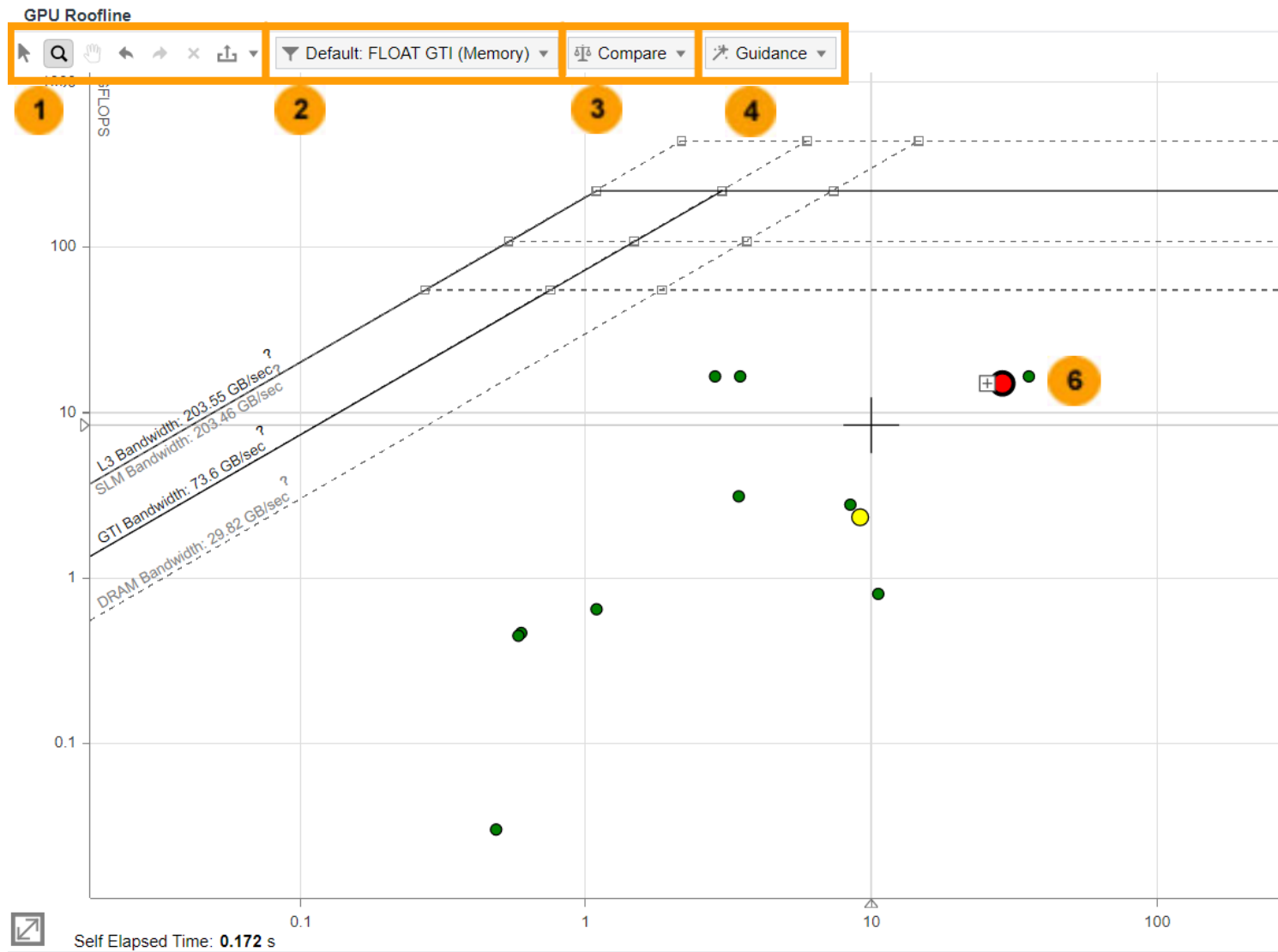
Pane: GPU Roofline Chart

Use GPU Roofline chart to visualize actual performance of your GPU kernels against hardware-imposed performance ceilings. For more information about investigating GPU Roofline results, see [Examine Bottlenecks on GPU Roofline Chart](#).

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

GPU Roofline Chart Controls

There are several controls to help you focus on the GPU Roofline chart data most important to you, including the following.



1

- **Select by Mouse Rect:** Select one or more kernels by tracing a rectangle with your mouse.
- **Zoom by Mouse Rect:** Zoom in and out by tracing a rectangle with your mouse. You can also zoom in and out using your mouse wheel.
- **Move View by Mouse:** Move the chart left, right, up, and down.
- **Undo or Redo:** Undo or redo the previous zoom action.


2

Use the filter drop-down to choose, which functions/loops to display on a Roofline chart. The following controls are available:

3

- Use **Operations** pane to filter kernels by type of operations: INT or FLOAT.
- Use the **Memory Level** pane to show results for each kernel in the chart: CARM, L3, SLM, GTI.

Use the **Compare** drop-down to plot results from another Roofline chart on results of your current project.

Click the  button to add results for comparison.

View and switch between the files that are currently compared in the **Compared Results** pane.

After comparison, the recent results are saved. You can view the list of recent results in the **Ready for Comparison** pane.

4

Add visual **Guidance** to the GPU Roofline chart to make the interpretation of data easier, including performance limits and whether kernels are memory bound, compute bound, or both.

In the **Guidance** drop-down toolbar, use the **Display roof rulers** checkbox to enable showing a vertical line from a kernel to the nearest and topmost performance ceilings. To view the ruler, hover the cursor over a kernel dot. Where the line intersects with each roof, labels display hardware performance limits for the kernel.

The preview picture is updated as you select guidance options, allowing you to see how changes will affect the GPU Roofline chart's appearance. Click **Apply** to apply your changes or **Default** to return the GPU Roofline chart to its original appearance.

5

- **Roofline View Settings:** Change the default scale setting to show:
 - The **optimal** scale (default), which adjusts to a chosen GPU Roofline chart view.
 - A **constant** scale, which adjusts to the tallest or widest view and does not change when a different GPU Roofline chart view is chosen.
- **Roof Settings:** Change the visibility and appearance of roofline representations (lines):
 - Click a **Visible** checkbox to show/hide a roof line.
 - Click a **Selected** checkbox to change a roof line appearance: display the roof line as a solid or a dashed line.
 - Manually fine-tune roof values in the **Value** column to set hardware limits specific to your code.

6

- **Loop Weight Representation:** Change the appearance of dots:
 - **Point Weight Calculation:** Change the **Base Value** for a point weight calculation.

NOTE For a GPU Roofline chart, only **Self Elapsed Time** is available as a base value.

- **Point Weight Ranges:** Change the **Size**, **Color**, and weight **Range** of a dot. Click the **+** button to split a point weight range in two. Click the **-** button to merge a point weight range with the range below.
- **Point Colorization:** Color dots by *weight ranges* or by *type* (vectorized or scalar). You can also change the color of loop with no self time.

- Hover your mouse over a dot to display metrics and, if enabled, a roof ruler for it.
- By default, Intel Advisor generates a roofline for *GTI (Memory)*, which reports memory traffic, in bytes, generated by all execution units.

Double-click a dot or select a dot and press **SPACE** or **ENTER** to display labeled dots representing memory levels for the selected kernel. Lines connect the dots to indicate that they correspond to the selected kernel.

NOTE If you have chosen to display only *some* memory levels in the chart using the **Memory Level** toolbar, unselected memory levels are displayed with **X** marks.

To hide the labeled dots, do one of the following:

- Select another kernel.
- Double-click an empty space in the GPU Roofline chart.
- Press **SPACE** or **ENTER**.
- Click the **+** button next to a dot on a chart to break it into smaller dots representing groups of instances of the same source kernel. Instances differ by global and local size.
 - Hover over each instance to view its performance metrics.
 - Select a dot representing an instance to highlight it in the **GPU** pane and view detailed information about its performance and memory usage in the **GPU Details** tab.

- Double-click a dot representing an instance to view how it utilizes each memory level.
- Right-click a kernel dot or a blank area in the Roofline chart to perform more functions, such as:
 - Further simplify the GPU Roofline chart by filtering out (temporarily hiding a dot), filtering in (temporarily hiding all other dots), and clearing filters (showing all originally displayed dots).
 - Show/hide a side panel that displays metrics for a selected dot.
 - Add visual guidance to the GPU Roofline chart to make the interpretation of data easier. These options are the same as in the **Guidance** toolbar.

Project Navigator Pane

Purpose

Use this pane to view, modify, and open existing Intel® Advisor results.

Location

To open the **Project Navigator** pane, do one of the following:

- Click the  button on the main toolbar.
- Choose **View > Project Navigator**










Controls

Use This	To Do This
Title bar	Drag to move the Project Navigator pane. Drag to a window edge to dock the Project Navigator pane.
Path to project directory	View the location of the currently opened project. Right-click the path to access the directory context menu.
Project name	Double-click to open the project. Right-click to access the project context menu. NOTE Opening a project closes the currently opened project.
Result name	Double-click to open the result. Right-click to access the result context menu.

Use This	To Do This
	<p>NOTE</p> <p>Opening a result opens the associated project.</p>

Toolbar: Intel Advisor




Use the Intel® Advisor toolbar to run the Intel Advisor perspectives and open certain panes or windows.

Use This Icon	To Do This
 Run Perspective	Run a perspective and open results of the latest perspective execution.
 Show My Result and Workflow	Open the Result window and Analysis Workflow pane to view the results of the latest project and run a new perspective.
 Perspective Selector	Open the Perspective Selector window to switch between the available perspectives and view their short descriptions.
 Project Properties	Open the Project Properties dialog box to specify the target executable, set up search directories for supporting files and source search location needed to analyze the target. Configure common project properties and analysis-specific properties.
 Snapshot	Create a snapshot of your project results.
 Create Project	Open Create Project dialog box to create and set up your project.
 Open Project	Open an existing project and view it in Project Navigator.
 Project Navigator	Open the Project Navigator pane to manage your existing Intel Advisor projects or create a new one.
 Help	Open the installed Help or view the Intel Advisor User Guide in your web browser.

Annotation Report

The **Annotation Report** window lists all annotations found during source scanning or running the Suitability and Dependencies tools. It lists the annotation type, source location, and annotations label in a table-like grid format, where each annotation appears on a separate row. Intel® Advisor updates the listed annotations when changes occur to the specified source directories. For example, when you save a source file with a code editor.

To sort the grid using a column's values, click on the column's heading. The columns of the grid are the following:

Use This Column	To Do This
Annotation	<p>View the type of annotation, such as Site, Task, or Lock.</p> <p>To show or hide a code snippet showing the annotation, click the  icon next to its name.</p> <p>For information about each annotation type, see the help topic Summary of Annotation Types.</p> <p>To view the source associated with an annotation in your code editor, double-click its name or a line in the code snippet (or right-click and select Edit Source from the context menu) in this column.</p> <ul style="list-style-type: none"> On Windows* OS: <ul style="list-style-type: none"> When using Visual Studio, the Visual Studio code editor appears with the file open at the corresponding location. When using the Intel® Advisor GUI, the file type association (or Open With dialog box) determines the editor used. On Linux* OS: When using the Intel® Advisor GUI, the editor defined by the Options > Editor dialog box appears with the file open at the corresponding location.
Source Location	<p>View the name of the source file that contains the annotation and the line number.</p> <p>Icons indicate where source is available  or not available .</p> <p>To view the source, double-click its name (or right-click and select Edit Source) in this column. The code editor appears.</p>
Annotation Label	<p>View the annotation's label (name).</p> <p>To view the source associated with an annotation, double-click its name (or right-click and select Edit Source) in this column. The code editor appears.</p>

Window: Dependencies Source

Code Locations Pane







Use this pane to view details about the code location for a selected problem in the Dependencies Report window.

Location

Bottom left of Dependencies Source window.

Controls

Use This	To Do This
Title bar	View the problem type.

Use This	To Do This
Code location data row(s)	Review related code locations: <ul style="list-style-type: none"> ID - Code location identifier Description - What happens at this code location Source - The source file associated with this code location. Function - Function name. Modules - The executable associated with this problem. State - Indicates whether the problem has been fixed or not. To change the state, use the context menu.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> Whether this is a related code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> Whether this is the focus code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	Shows if code location source code is available for viewing and editing.
Column labels	Click a column heading to sort the data grid rows in either ascending or descending order.
Pane border	Resize the pane (drag).
Right click a row to display a context menu	Display a context menu to: set this code location as the focus or related code location, copy the selected data row(s) to the clipboard, mark the state as fixed or not fixed, or display context-sensitive help.



Focus Code Locations Pane

Use this pane to explore the source code associated with focus code location in the Dependencies Source window.

Location

Top left of Dependencies Source window.

Controls

Use This	To Do This
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> Whether this is the focus code location. If code location source code is available for viewing and editing.
Pane border	Resize the pane (drag).
Source code	<ul style="list-style-type: none"> Explore source code associated with the focus code location Display the code editor at the corresponding source file by double-clicking a data row or by using the corresponding context menu item.

Use This	To Do This
Right click a row to display a context menu	Display a context menu to: open the code editor to the corresponding source line, copy the selected data row(s) to the clipboard, or display context-sensitive help.




Call Stack Pane

Use this pane to select which source code appears in the Focus Code Location pane in the Dependencies Source window.

Location

Top right of the Dependencies Source window.

Controls

Use This	To Do This
 or  icon	View whether: <ul style="list-style-type: none"> Source code is available for viewing and editing. An  icon indicates that source code is not available.
Click a row in the Call Stack pane	Displays source code for the specified call stack entry.
Pane border	Resize the pane (drag).
Right click a row in the Call Stack pane	Customize the call stack presentation by using the Call Stack context menu.







Relationship Diagram Pane

Use this pane to view the relationships among code locations for the selected problem.

Location

Bottom right of Dependencies Source window.

Controls

Use This	To Do This
Title bar	View the problem type.
 icon,  icon, or no icon in the Source column	View: <ul style="list-style-type: none"> Whether this is a related code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	View: <ul style="list-style-type: none"> Whether this is the focus code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	View if code location source code is available for viewing and editing.

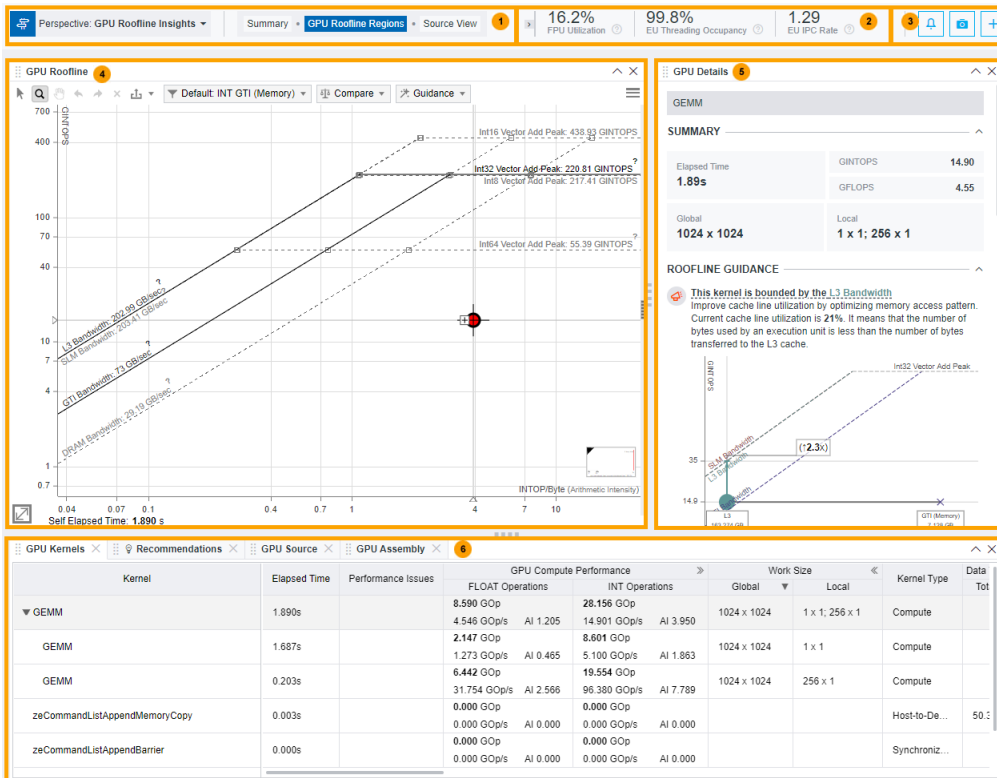
Use This	To Do This
Pane border	Resize the pane (drag).
Diagram	View the relationship among code locations in a problem: <ul style="list-style-type: none"> Each box in a diagram represents a code location in a problem. A diagram with a single box is a trivial problem with no related code locations. Boxes arranged left-to-right with connecting arrows indicate a time ordering. Boxes with connecting lines indicate association.

Window: GPU Roofline Regions

Use the **GPU Roofline Regions** window to view GPU metrics for your kernels in the grid, visualize kernel performance and identify room for optimization using a GPU Roofline chart, and view detailed information about how well a specific kernel utilizes compute and memory bandwidth.

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

Review the controls available in the main report of the GPU Roofline Insights perspective of Intel® Advisor. In the **GPU Roofline Regions** and **Summary** tabs, you can drag-and-drop, close/open, collapse/expand panes to change the report layout.



- Switch between perspectives using a drop-down menu in the top left corner.
- Switch between **Summary**, **GPU Roofline Regions**, and **Source View**.

2

3

Review the summary metrics for parts of your application executed on an accelerator.

- Expand/collapse a top slider with per-program



recommendations using the button:

- Expand or collapse each recommendation.
- Pin recommendations pane by clicking the



button.

- Expand/collapse a top slider with collection



event log using the button. The top slider enables you to view the following:

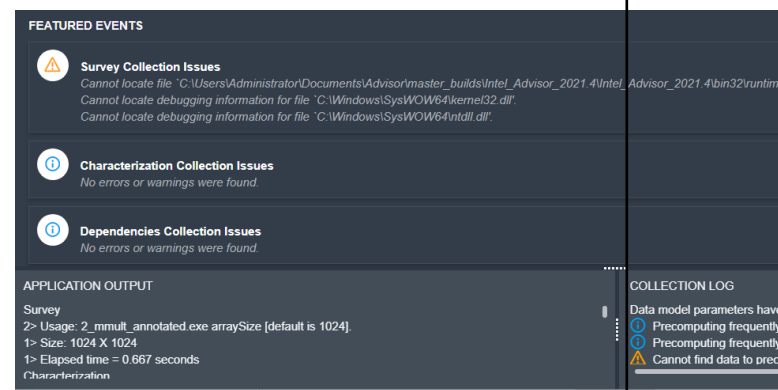
- Main data collection events and issues in the **Featured Events** pane.

Expand/collapse each featured event in the log to view details.

- Application Output.** Use a toggle in the upper right corner of the view to show/hide the application output.
- Full execution log of your application in the **Collection Log** pane.

Click the button to collapse the top slider or drag it to maximize the event log.

Tip Collection event log top slider appears automatically when you run a perspective. You can track collection using the green progress bar at the top and view collection events online.



- Create a snapshot for the current project results



using the button. For details, see [Create a Read-only Result Snapshot](#).

- Click a + button to open previously closed panes. With this button, you can add the following panes:

4

- **CPU Roofline** pane that enables you to view the actual performance of functions/loops executed on a CPU against hardware-imposed performance ceilings visualized on a Roofline chart.

For details about interpretation, see [Examine Bottlenecks on CPU Roofline Chart](#).

- **CPU** pane that enables you to review performance metrics of your application performance on a CPU and compare them with performance metrics on an accelerator. For details, see [CPU Metrics](#).

Review the actual performance of GPU kernels in your application against hardware-imposed performance ceilings using the **GPU Roofline** chart.

For details about interpretation, see [Examine Bottlenecks on GPU Roofline Chart](#).

See detailed description of GPU Roofline chart controls in [Pane: GPU Roofline Chart](#).

5

Use the **GPU Details** tab to view the detailed information about the execution of a selected kernel:

- View program metrics for a selected kernel in the **Summary** pane.
- Identify the memory level your selected kernel is bounded by using the **Roofline Guidance** pane.
- Explore the compute operations count and memory level utilization metrics in the **OP/S and Bandwidth** pane. Use the drop-downs to view the operations count, memory traffic, and arithmetic intensity (AI) for floating-point and integer operations at different memory levels.
- View how the selected kernel impacts each memory level and explore the amount of data passed through each memory level using the **Memory Metrics** pane.
- Explore the ratio of compute, memory and other instructions grouped by types in the **Instruction Mix** pane.
- Get detailed overview of instruction types used during the execution of your application using the **Instruction Mix Details** pane. Use the drop-downs to expand each instruction category and view the included instruction types and instruction count. For compute category, Intel Advisor determines the data type. The dominating data type in the entire kernel is highlighted blue. Filter instructions by type and dominating data type using a filter button.
- View how the loops in a selected kernel utilize the execution unit (EU) in the **Performance Characteristics** pane.

Switch between **GPU Source** and **GPU Assembly** tabs to:

7

- Examine the source code and offload details for each source line. Select a loop in the **GPU** table or a dot in the **GPU Roofline** to focus on the corresponding parts of source and assembly code.
- Review GPU assembly representation for a selected kernel. Select a code line to highlight the corresponding part in source code.

For details about interpreting GPU Roofline Insights perspective results, see [Explore GPU Roofline Results](#)

Use the **Recommendations** tab to view actionable recommendations helping you improve performance of the currently selected kernel. Expand a recommendation to view more information and a code snippet.

Review performance metrics of your application performance on a GPU accelerator. For details about metrics, see [Accelerator Metrics](#).

Window: GPU Roofline Insights Summary

After running [GPU Roofline Insights Perspective](#), use the GPU Roofline Insights Summary window to view the most important information about the execution of your code on a GPU and on a CPU devices.

Customize the window layout using drop-downs in the upper-right corner of each pane.



Create a snapshot of your GPU Roofline result using the button. For details, see [Create a Read-only Result Snapshot](#).

NOTE Families of Intel® Xe graphics products starting with Intel® Arc™ Alchemist (formerly DG2) and newer generations feature GPU architecture terminology that shifts from legacy terms. For more information on the terminology changes and to understand their mapping with legacy content, see [GPU Architecture Terminology for Intel® Xe Graphics](#).

Program Metrics Pane

View the most important metrics for parts of your application executed on a GPU and on a CPU. This pane tells you how well your application uses the GPU resources and how much space for improvement your application has. This pane is broken into the following sub-sections:

- **GPU Time:** view total elapsed time of all compute tasks executed on a GPU device.
- **FPU Utilization:** view the average percentage of GPU time when both floating-point units (FPUs) are used.
- **EU Threading Occupancy:** view the percentage of cycles on all execution units (EUs) and thread slots when a slot has a thread scheduled.
- **EU IPC Rate:** view the average rate of instructions per cycle (IPC) for execution units when two FPUs are used.
- **CPU Time:** view total elapsed time for a part of your application executed on a CPU.
- **Thread Count:** view the number of threads used for execution of your application on a CPU.

Open the drop-down menus below the main program metrics to view detailed information about GFLOPS, GINTOPS, and arithmetic intensity for INT and FLOP operation types.

OP/S and Bandwidth Pane

View metrics for all compute tasks and functions/loops of your application against the hardware-imposed performance ceilings on preview Roofline charts for GPU and CPU. Explore how many FLOPS and INTOPS per second can be executed on different memory levels. For details about using GPU Roofline Chart, see [Examine Bottlenecks on GPU Roofline Chart](#).

Filter operations by type by switching between **INT** and **FLOAT** in the upper-right corner of preview Roofline charts.

Open the drop-down menus below the preview Roofline charts to view detailed information about GFLOPS, GINTOPS, and arithmetic intensity for INT and FLOP operation types on different memory levels (CARM, L3, SLM, GTI). For a GPU Roofline chart, view instruction mix diagram showing a total number of instructions united by their types (FLOAT, INT, STORE, LOAD, and MOVE).

Hover over a dot on a Roofline chart to view metrics for the selected function/loop. Click a dot to open it in source code and view it on a GPU Roofline chart.

Top Hotspots Pane

View key metrics (elapsed time, FLOPS, GINTOPS) for top five most time-consuming compute tasks on a GPU and functions/loops on a CPU that are the best candidates for optimization. Click the function name to open it in source code and view it on a GPU Roofline chart.

Performance Characteristics Pane

View the execution time details for GPU- and CPU-executed parts of your application. This pane can tell you how well your application uses GPU resources on each memory level. Hover over the histogram to see the fractions of active, stalled, and idle EU arrays.

Platform Information Pane

View the system information including software and hardware summary.

Collection Information Pane

View information about Survey and Characterization data collection. Use drop-downs to show/hide information for each analysis type.

Window: Memory Access Patterns Source

Details View Pane

Use this pane at the bottom right of the Memory Access Patterns Source window to examine details for a selected site.

Source View Pane

Middle of Memory Access Patterns Source window

Controls

Use This	To Do This
Source lines	To navigate to related source lines.
Double-click a source line	To open your code editor to the corresponding source file. The editor allows you to add annotations to your code (right-click to open the context menu). <ul style="list-style-type: none"> On Windows* OS:

Use This	To Do This
	<ul style="list-style-type: none"> When using Microsoft Visual Studio*, the Visual Studio code editor appears with the file open at the corresponding location. <hr/> <p>NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.</p> <hr/> <ul style="list-style-type: none"> When using the Intel® Advisor GUI, the file type association (or Open With dialog box) determines the editor used. On Linux* OS: When using the Intel Advisor GUI, the editor defined by the Options > Editor dialog box appears with the file open at the corresponding location.
Select multiple source lines	To view the accumulated details in the Details View pane.
Right click a source line or multiple source lines	Display a context menu to: open your code editor to the corresponding source line, copy the selected source line(s) to the clipboard, or display context-sensitive help relevant to the selected loop or function.

Assembly View Pane

Use this pane at the bottom of the Memory Access Patterns Source window to view assembly representation of your code.

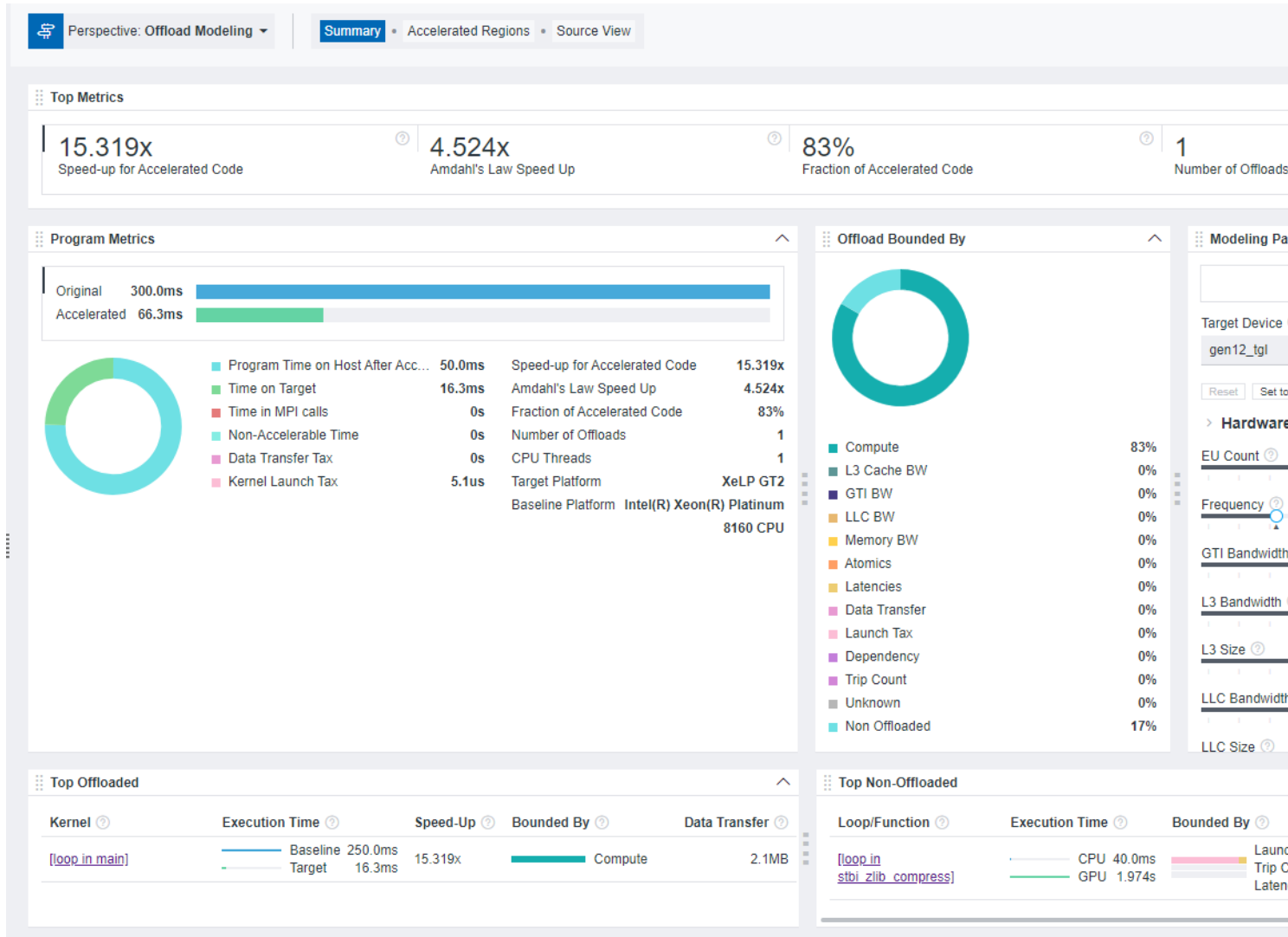
Controls

Use This	To Do This
Source lines	You can navigate to related source lines or explore assembly representation of the code by using the Call Stack with Loops pane.
Select multiple source lines	To view the accumulated time values for multiple source lines below the Self Time column, or enable you to copy multiple source lines using the context menu. Viewing accumulated time can help you decide how to divide the work.

Window: Offload Modeling Summary

After running [Offload Modeling perspective](#), use the **Summary** window to view the most important information about your code, total estimated speedup achieved from offloading, top offloaded and non-offloaded code regions, and more.

You can drag and drop, expand and collapse, and resize the panes to customize the Summary view to your needs.



Top Metrics

Use this pane to view information about estimated speedup of your code achieved by offloading. The following metrics are reported:

Speed-up for Accelerated Code

Estimated speedup of your code in relation to its original execution time in pane

Amdahl's Law Speed Up

Estimated speedup for the whole application estimated by the Amdahl's law, which states that the potential speedup from parallelizing one part of a program is limited by the portion of the program that still runs serially. This metric is available only for the CPU-to-GPU modeling.

Fraction of Accelerated Code

Fraction of accelerated code, in per cent, relative to the total time of the original program. This metric is available only for the CPU-to-GPU modeling.

Number of Offloads

Number of offloaded code regions

Program Metrics

This pane lists performance metrics estimated for the whole application, including original time before offloading and estimated time after offloading, break-down of estimated time spent on host and target devices, offload taxes, information about host and target platforms, and so on. This pane helps you to determine if your code is profitable to offload to a target device and compare time of original code before acceleration with estimated time of accelerated code.

Offload Bounded By

This pane lists the factors that prevent your code from achieving better performance on a target device. The information is shown in a list and in a pie chart that helps you visualize the results. The factor with the highest percentage indicates what you should optimize your application for on the target GPU to optimize its performance.

Modeling Parameters

This pane shows the current modeled target GPU and its parameters. The pane is interactive, and you can use it to:

- Examine device parameters that the application performance was modeled on to understand how they affect the estimated performance.
- Change the target device to compare the selected device configuration with the current modeled device.
- Adjust the parameters using sliders and remodel performance for a new device to experiment with parameters and see how they affect the performance on the GPU.

The pane has the following functionality:

Modeling Parameters

Save to Remodel

Target Device

 XeLP GT2

Reset
 Set to Hardware Default

Hardware Parameters

Frequency 2.7 GHz

EU Count 256

Memory Bandwidth 48 GB/s

PCIe Bandwidth 684.12 MB/s

SLM Bandwidth ∞

SLM Size OFF

L3 Bandwidth 518.4 GB/s

L3 Size 3.75 MB

LLC Bandwidth 172.8 GB/s

LLC Size 12 MB

GTI Bandwidth 172.8 GB/s

Changing values in the Modeling pane does not automatically update the result. Re-run Intel Advisor using this command line:


```
adviser -c=projection --custom-config=/config.toml --config=gen12_tgl -p
```

1

For CPU-to-GPU modeling in GUI and HTML report or for GPU-to-GPU modeling in HTML report: After you change the hardware parameters, click **Save to Remodel** to save the configuration file with your parameters and use it for remodeling. This does not update the modeling results automatically, but generates a configuration file with the device parameters you set.

2

For GPU-to-GPU modeling in GUI report: After you change the hardware parameters, click the button to rerun the Performance Modeling analysis for the custom device.

3

Select a target device for modeling to see its parameters and how they are different from the current device configuration.

Click the **Reset** button to change the slider positions back to the parameters used for the current modeling. This button activates after you change any slider position.

Click the **Set to Hardware Default** button to change the slider positions to the default target GPU parameters, for example, if your current modeled configuration is custom.

4

Move the sliders to change the parameter to a desired value for a custom device configuration. Hover over the ? icon near the parameter name to learn more about it.

- An arrow under a slider indicates the default value of the parameter for the selected device.
- Black line indicates the parameter value for the current modeled result.
- When you move a slider, a blue line indicates the difference between the new parameter value and the current modeled parameter.
- For bandwidth and size parameters, when you move a slider to a *maximum* value to the right, it sets the parameter to infinite meaning that the bandwidth/size is unlimited.
- For bandwidth and size parameters, when you move a slider to a *minimum* value to the left, it disables the parameter as it does not exist on a target device.

Notice that the parameter list might change depending on the target device selected. This might be due to differences between GPU architecture or terminology specifics.

5

This is available only for CPU-to-GPU modeling in GUI and HTML report or for GPU-to-GPU modeling in HTML report.

Copy the generated Performance Modeling command and run it from a terminal or a command prompt to remodel application performance for the custom target device. This command line is generated *after* you save the custom configuration with the **Save to Remodel** button. The command

already includes all necessary options and paths to the configuration file and project directory and is ready for copy and paste.

Top Offloaded

This pane lists the top five code regions that are the most profitable to offload to a target device with the following data per code region:

Loop/Function	For CPU-to-GPU modeling only. Source locations of top five offloaded loops/functions with the highest speedup. Click a loop/function name to switch to the Accelerated Regions tab and view information about it in more detail.
Kernel	For GPU-to-GPU modeling only. Source locations of top five kernels with the highest speedup. Click a kernel name to switch to the Accelerated Regions tab and view information about it in more detail.
Execution Time	Elapsed time measured on a baseline device before offloading and elapsed time estimated on a target device after offloading.
Speed-Up	Estimated speedup the code region can achieve on a target device after offloading.
Bounded By	Main factor(s) preventing the code region from achieving better performance. Hover over the diagram to see bounded-by time for each factor.
Data Transfer	Data transfer overhead for the selected code region.

Top Non-Offloaded

This pane lists the top five code regions not recommended for offloading to the current target device. This pane is available only if you run the CPU-to-GPU modeling and is empty for the GPU-to-GPU modeling as it assumes all kernels are offloaded ignoring their estimated speedup.

The pane shows the following data per code region:

Loop/Function	Source locations of top five non-offloaded loops/functions. Click a loop/function name to switch to the Accelerated Regions tab and view information about it in more detail.
Execution Time	Elapsed time measured on a baseline device before offloading and elapsed time estimated on a target device after offloading.
Speed-Up	Main factor(s) preventing the code region from achieving better performance. Hover over the diagram to see bounded-by time for each factor.
Bounded By	Main factor(s) preventing the code region from achieving better performance. Hover over the diagram to see bounded-by time for each factor.

Why Not Offloaded

Reason(s) why the code region is not recommended for offloading to the current target device. Switch to the **Accelerated Regions** tab to get a more detailed explanation.

Data Transfer

Data transfer overhead for the selected code region.

See Also

Model Offloading to a GPU Find high-impact opportunities to offload/run your code and identify potential performance bottlenecks on a target graphics processing unit (GPU) by running the Offload Modeling perspective.

Window: Offload Modeling Report - Accelerated Regions

Use the **Accelerated Regions** window to view detailed information about offloaded and non-offloaded loops/functions, view source code for your loops/functions.

Review the controls available in the main report of the Offload Modeling perspective of the Intel® Advisor. In the **Accelerated Regions** and **Summary** reports, you can drag-and-drop, close/open, collapse/expand panes to change the report layout.

The screenshot displays the Intel Advisor interface for the Offload Modeling perspective, specifically the Accelerated Regions tab. The main pane shows a table of code regions with the following columns: Loop/Function, Performance Issues, Time, Baseline Device, Dependency Type, Programming Model, Measured Iteration Space, and Average. The table lists several loops and functions, including 'loop in compute_flux_ser at euler3d_cpu_ser.cpp', 'loop in time_step_ser at euler3d_cpu_ser.cpp', and 'loop in TS9comp_flux at euler3d.cpp'. The 'Data Transfer Estimations' pane on the right provides a detailed breakdown of performance metrics, including Compute (2.888s), DRAM BW (< 0.1ms), L3 BW (6.311s), LLC BW (1.099s), Atomic Throughput (0ms), Data Transfer Tax (0ms), Kernel Launch Tax (30.0ms), Load Latency (2.999s), and Estimated Time (9.34s). The 'Source' pane at the bottom shows the C++ code for the selected loop, with a 'Yes' in the 'Is Offloaded' column and a speedup of 3.916.

Loop/Function	Performance Issues	Time	Baseline Device	Dependency Type	Programming Model	Measured Iteration Space	Average
loop in compute_flux_ser at euler3d_cpu_ser.cpp	Code re...	36.58s	x86	Parallel: Assumed		Call Count 6000 Trip Count 506	506
loop in time_step_ser at euler3d_cpu_ser.cpp	Code re...	516.0ms	x86	Parallel: Assumed		Call Count 2000 Trip Count 506	506
loop in time_step_ser at euler3d_cpu_ser.cpp	Code re...	456.0ms	x86	Parallel: Assumed		Call Count 2000 Trip Count 506	506
loop in time_step_ser at euler3d_cpu_ser.cpp	Code re...	432.0ms	x86	Parallel: Assumed		Call Count 2000 Trip Count 506	506
loop in compute_step_factor_ser at euler3d_cpu_ser.cpp	Code re...	844.0ms	x86	Parallel: Assumed		Call Count 2000 Trip Count 506	506
loop in verify_ser at euler3d_cpu_ser.cpp		252.0ms	x86	Parallel: Assumed		Call Count 1 Trip Count 97046	97046
loop in TS9comp_flux at euler3d.cpp		1.754s	x86	Parallel: Programming Model	DPCPP	Call Count 2742825 Trip Count 22	22
loop in TS13comp_timestep at euler3d.cpp		256.0ms	x86	Parallel: Programming Model	DPCPP	Call Count 2799013 Trip Count 22	22
loop in TS13comp_step_fac at euler3d.cpp		84.0ms	x86	Parallel: Explicit	DPCPP	Call Count 881423 Trip Count 21	21
loop in TS9comp_flux at euler3d.cpp		194.0ms	x86	Parallel: Programming Model	DPCPP	Call Count 293176 Trip Count 3	3

Metric	Value
Compute	2.888s
DRAM BW	< 0.1ms
L3 BW	6.311s
LLC BW	1.099s
Atomic Throughput	0ms
Data Transfer Tax	0ms
Kernel Launch Tax	30.0ms
Load Latency	2.999s
Estimated Time	9.34s

```

220  /*
221
222  void compute_flux_ser(int nplr, int* elements_surrounding_elements, float* normals, float* variables, float* fluxes, float* ff_variable,
223  {
224      const float smoothing_coefficient = float(0.2f);
225
226  for(int blk = 0; blk < nplr/block_length; ++blk)
227  {
228      int b_start = blk*block_length;
229      int b_end = (blk+1)*block_length > nplr ? nplr : (blk+1)*block_length;
230      for(int i = b_start; i < b_end; ++i)
231      {
232          float density_i = variables[i + VAR_DENSITY*nplr];
233          float3 momentum i;

```

1

- Switch between perspectives using a drop-down menu in the top left corner.
- Switch between **Summary**, **Accelerated Regions**, and **Source View**.

2


Review the summary offload characteristics for your application to decide if the application is recommended for offloading. The pane highlights total speedup, number of loops and functions offloaded, and a fraction of code accelerated.

3


Select code regions to show in the report based on offload type:

- Show *all* code regions in your code.
- Show only code regions *recommended* for offloading.
- Show only code regions *not recommended* for offloading.


4

Expand/collapse a top slider with per-program recommendations using the  button.

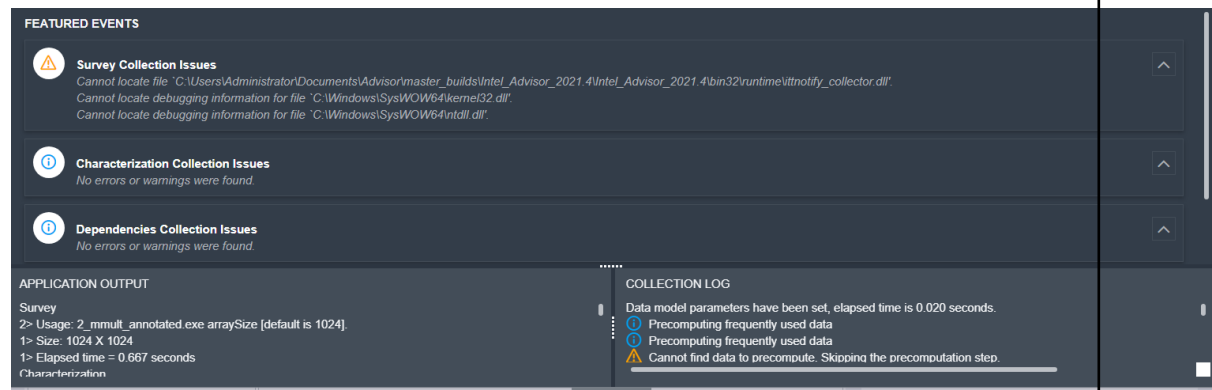
You can expand or collapse each recommendation.


Expand/collapse a top slider with collection event log using the  button. The top slider enables you to view the following:

- Main data collection events and issues in the **Featured Events** pane.
Expand/collapse each featured event in the log to view details.
- **Application Output**
- Full execution log of your application in the **Collection Log** pane.

Click the  button to collapse the top slider or drag it to maximize the event log.

Tip Collection event log top slider appears automatically when you run a perspective. You can track collection using the green progress bar at the top and view collection events online.



Create a snapshot for the current project results using the  button. For details, see [Create a Read-only Result Snapshot](#).

- Click a + button to open previously closed panes.

5

Review the detailed information about your application performance measured on a host platform and its performance modeled on a target platform. For details about metrics reported, see [Accelerator Metrics](#).

Depending on a perspective configuration, you might see different metrics reported and some metrics might be not accurate. Refer to the following topics for interpretation details:

- Low accuracy: [Examine Regions Recommended for Offloading](#)
- Medium accuracy: [Examine Data Transfers for Modeled Regions](#)
- High accuracy: [Check for Dependency Issues](#)

6

Switch between the **Data Transfer Estimations** tab and **Details** tab.

In the **Data Transfer Estimations** tab, examine details about estimated data transfers in a loop and memory objects tracked and review data transfer recommendations. Select a loop in the **Code Regions** table to see the data transfer estimations for it. Click a recommendation to expand it and view hints for offloading your code to another accelerator, code samples, and links to useful resources.

For details about how to read this pane, see [Examine Data Transfers for Modeled Regions](#).

NOTE You need to enable data transfer analysis before running the perspective to see metrics in this pane,

7

In the **Details** tab, view performance metrics for a selected function/loop.


Advanced view pane enables you to:

- Examine the source code for selected functions/loops in the **Source** tab. Select a loop in the **Code Regions** table to focus on the corresponding part of the source code.
Right-click a code line and click **View Source** to open the source view. Double click a code line in the source view to open it in code editor.
- View the execution details for a selected function/loop in the call stack using the **Top-Down** tab.
- Examine recommendations that provide guidance and code samples to resolve the issues found by Intel Advisor using the **Recommendations** tab. Use the drop-down to expand code snippets if you need them.

For details, see [Examine Regions Recommended for Offloading](#)

Window: Perspective Selector



Click the  button on the main toolbar to open the **Perspective Selector** window that enables you to switch between different Intel® Advisor perspectives.

Click a perspective to see a brief information about it.

Double-click a perspective to open it.

For details about Intel Advisor perspectives, see:

- [Vectorization and Code Insights Perspective](#)
- [CPU / Memory Roofline Insights Perspective](#)
- [Threading Perspective](#)
- [GPU Roofline Insights Perspective](#)
- [Offload Modeling Perspective](#)

Window: Refinement Reports

Intel® Advisor offers two *refinement* analyses:

- Dependencies analysis (optional) - For safety purposes, the compiler is often conservative when assuming data dependencies. Run a Dependencies analysis to check for real data dependencies in loops the compiler did not vectorize because of assumed dependencies. If real dependencies are detected, the analysis can provide additional details to help resolve the dependencies. Your objective: Identify and better characterize real data dependencies that could make forced vectorization unsafe. For more details, see [Check for Dependencies Issues](#).

- **Memory Access Patterns (MAP) analysis (optional)** - Run a MAP analysis to check for various memory issues, such as non-contiguous memory accesses and unit stride vs. non-unit stride accesses. Your objective: Eliminate issues that could lead to significant vector code execution slowdown or block automatic vectorization by the compiler. For more details, see [Investigate Memory Usage and Traffic](#)

Site Report Pane

The **Site Report** pane on top of Refinement Reports window comprises top-level information:

- **Site Location** lists names of the analyzed loops, names of the files with the source code, as well as the number of the line where the loop is invoked
- **Loop-Carried Dependencies** Summarizes presence or absence of dependencies across iterations (loop-carried dependencies). Dependency types:
 - RAW - read after write (flow dependency)
 - WAR - write after read (anti dependency)
 - WAW - write after write (output dependency)
- **Strides Distribution** Unit/Constant/Variable stride ratio for the selected site.
- **Access Pattern** information about stride types detected in the site.
- **Site Name** Site Name in case of using source annotations, or sequence id in case of marking loops for deeper analysis in survey report.

Double-click any line in the **Refinement Reports** top pane to see the loop source code.

The pane at the bottom of the Refinement Reports window contains the following elements:






- **Filters** pane - filter analysis data by a variety of criteria, such as module, loop/function, vectorized/non-vectorized.
- **Advanced View** pane - includes the following tabs:
 - [Memory Access Patterns Report](#) tab - view information about types of memory access inside selected loops/functions. (Vectorization and Code Insights perspective only.)
 - [Dependencies Report](#) tab - view any predicted data sharing problems and informational remark messages.
 - Recommendations tab - view memory-specific recommendations.

Tab: Dependencies Report

Problems and Messages Pane

Select the problems that you want to analyze by viewing their associated observations.

Controls








Use This	To Do This
Column labels	Click a column label to sort the data grid data rows in either ascending or descending order.
Selected data row	<p>Review the characteristics of each data row in the grid. The columns are:</p> <ul style="list-style-type: none"> • ID - Identifier for the problem. •  (severity) - The severity of the problem, such as error , warning , or an informational remark message . For example, the location of parallel sites executed are indicated by the message  Parallel Site. • Type - The problem type or message type. For more information about a problem, right click to display the context menu. • Site Name- The name of the site associated with this problem. • Sources - The source file associated with this problem.

Use This	To Do This
	<ul style="list-style-type: none"> Modules - The modules (executable) associated with this problem. State - Indicates whether the problem has been fixed or not. To change the state, use the context menu in this pane.
Pane border	Resize the pane (drag).
Right click a row to display a context menu	Display a context menu to: open the code editor to the corresponding source line, display the Dependencies Source window, copy the selected data row(s) to the clipboard, or display context-sensitive help for that problem or message.

Code Locations Pane

Use the Dependencies report to view each reported problem in its associated code locations.

Controls

Use This	To Do This
Title bar	View the problem type.
Code Location data row(s)	Review related code locations: <ul style="list-style-type: none"> ID - Code location identifier Description - What happens at this code location. Source - The source file for this code location. Function - Function name. Modules - The executable associated with this problem. State - Indicates whether the problem has been fixed or not. To change the state, use the context menu.
Click  to the left of a code location name	Display a code snippet associated with the selected code location.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> Whether this is a related code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	Shows: <ul style="list-style-type: none"> Whether this is the focus code location. If code location source code is available for viewing and editing.
 icon,  icon, or no icon in the Source column	Shows if code location source code is available for viewing and editing.
Double-click a code location data row or source line, or right-click and	Display the Dependencies Source window.






Use This	To Do This
select the View Source context menu item	
Right-click and select the Edit Source context menu item	Display a code editor with the corresponding source file. <ul style="list-style-type: none"> On Windows* OS: <ul style="list-style-type: none"> When using Visual Studio, the Visual Studio code editor appears with the file open at the corresponding location. When using the Intel Advisor GUI, the file type association (or Open With dialog box) determines the editor used. On Linux* OS: When using the Intel Advisor GUI, the editor defined by the Options > Editor dialog box appears with the file open at the corresponding location.
Column labels	Click a column heading to sort the data grid rows in either ascending or descending order.
Pane border	Resize the pane (drag).
Right click a row to display a context menu	Display a context menu to: expand or collapse all code snippets, open the Dependencies Source window, edit sources in the code editor, copy the selected data row(s) to the clipboard, mark the state as fixed or not fixed, or display context-sensitive help.

Tab: Memory Access Patterns Report

You can view the Memory Access Patterns analysis results in both the **Refinement Reports** top panel and in the **Memory Access Patterns Report** tab right beneath.

Controls

The Memory Access Patterns Report tab of the Refinement Reports window contains information about types of memory access inside selected loops, including the following:

- **ID** of the memory access location
- **Severity** stride type classification (icon of the stride type)
- **Stride** physical distance in elements between memory accesses in two consequent iterations
- **Type** of the memory access:
 - **Unit/Uniform types:**
 -  **Unit stride (stride 1)** instruction accesses memory that consistently changes by one element from iteration to iteration.
 -  **Uniform stride 0** instruction accesses the same memory from iteration to iteration.
 -  **Constant stride (stride N)** instruction accesses memory that consistently changes by N elements (N>1) from iteration to iteration.
 - **Variable stride types:**
 -  **Irregular stride** instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration.
 -  **Gather (irregular) stride**, which is detected for v(p)gather* instructions on AVX2 Instruction Set Architecture.
- **Source** provides info on the operation's source file name and code line where the memory access is issued.
- **Nested Function** function (invoked from site) where stride diagnostic was detected.

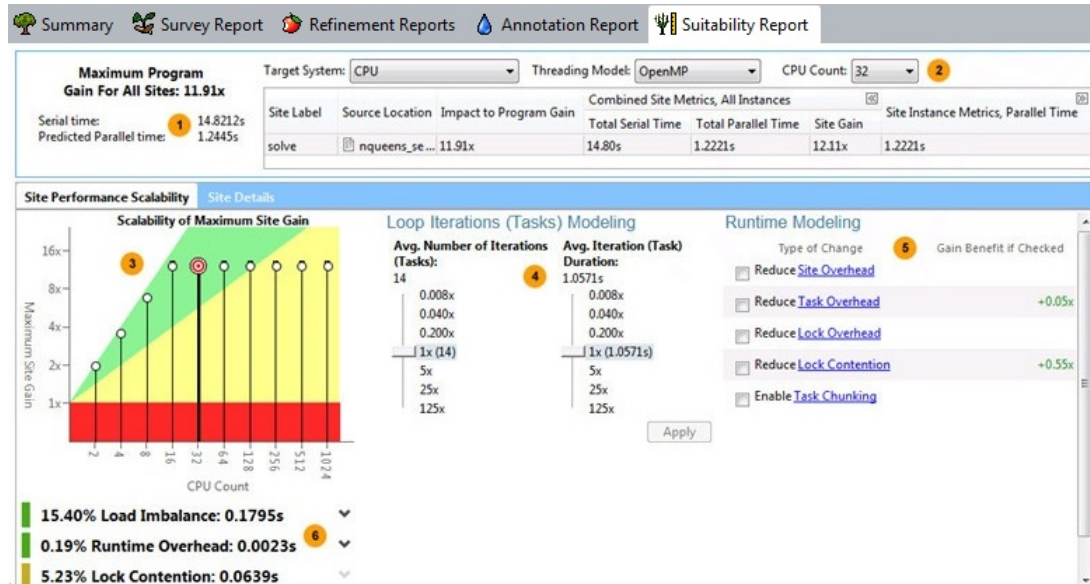
- **Modules** application modules, where the memory access is issued.
- **Variable references** name of the variable for which the memory access stride is detected.

Double-click any line in the **Memory Access Patterns Report** tab to see the selected operation's source code.

Window: Suitability Report





Use this window to review the parallel sites in the upper right area. Select a site and view its annotations and related characteristics. Use the list of sites as a to-do list: start at the top and work your way down.

Controls



This screen shows data based on a **Target System** of **CPU**. The screen shown on your system will differ.

<p>1</p>	<p>The upper-left area shows the Maximum Program Gain for All Sites in the program. Your overall goal of adding parallelism is to increase the Maximum Program Gain for All Sites so the parallel program will execute as fast as possible. The measured serial execution runtime, predicted parallel runtime, and any measured paused time are displayed below Maximum Program Gain for All Sites. Use the predicted Suitability gain values to help you make informed decisions about where to add parallelism.</p> <p>If the Suitability tool detects any annotation-related errors, they appear at the top of the Suitability Report window. If you see this type of error, the displayed Suitability data may not be reliable. Annotation-related errors may be caused when the correct sequence of annotations do not occur because of missing annotations, when unexpected execution paths occur, or if Suitability data collection was paused while the target was executing.</p>
<p>2</p>	<p>Use the upper-right row of modeling parameters to model performance. Choose a hardware configuration and threading model (parallel framework) values from the drop-down lists. If you select a Target System for Intel® Xeon Phi™ processors, an additional value for total Coprocessor Threads appears.</p> <p>Below this row is a grid of data that shows the estimated performance of each parallel site detected during program execution. The Site Label shows the argument to the site annotation. Examine the predicted Site Gain and Impact to Program Gain (higher values are better) to estimate how</p>

	<p>much each site contributes to the Maximum Program Gain for All Sites for all sites (described above). To expand the data under Combined Site Metrics or Site Instance Metrics, click the  icon to the right of that heading; to collapse data, click  to the right of that heading.</p> <p>To view source code for a selected parallel site, click its row to display the Suitability Source window.</p> <p>To show or hide the side command toolbar, click the  or  icon.</p>
3	<p>The Scalability of Maximum Site Gain graph summarizes performance for the selected site. The number of CPU processors or total number of coprocessor threads appears on the horizontal X axis and the target's predicted performance gain appears on the Y axis. To change the default CPU Count and the Maximum CPU Count, set the Options value.</p> <p>If you choose a Target System of CPU, to view detailed characteristics of the selected site as well as its tasks and locks, click the Site Details tab.</p>
4	<p>Use the Loop Iterations (Tasks) Modeling (or Tasks Modeling) modeling parameters to experiment with different loop structures, iteration counts, and instance durations that might improve the predicted parallel performance.</p> <p>For example, you might want to see the impact of modifying your nested change loop structure, modify the loop body code, or change number of iterations.</p> <p>If the task annotations indicate likely task parallelism, the title will appear as Task Modeling (instead of Loop Iterations (Task) Modeling for data parallelism).</p>
5	<p>Use the Runtime Modeling modeling parameters to learn which parallel overhead categories might have an impact on parallel overhead. If you agree to address a category later by using the chosen parallel framework's capabilities or by tuning the parallel code after you have implemented parallelism, check that category.</p> <p>If the chosen Target System is Intel Xeon Phi or Offload to Intel Xeon Phi, additional Intel® Xeon Phi™ Advanced Modeling options appear below the Runtime Modeling area. To expand this area, click the down arrow to the right of Intel Xeon Phi Advanced Modeling.</p>
6	<p>Below the graph is a list of issues that might be preventing better <i>predicted</i> performance gains as well as a summary of serial and predicted parallel time. To expand a line, click the down arrow to the right of the item's name. Most issues are related to the Runtime Modeling modeling parameters. Later, you can use other Analyzer tools like Intel® VTune™ Profiler to measure <i>actual</i> performance of your parallel program.</p>

Window: Suitability Source

Source View Pane

Use this pane to explore your source code.

Location

On the left of the Suitability Source window.

Controls

Use This	To Do This
Source code lines	You can view and navigate to related source code by using the Call Stack pane .
Double-click a source line	To open the code editor to the corresponding source line.

Use This	To Do This
	<ul style="list-style-type: none"> On Windows* OS: <ul style="list-style-type: none"> When using Visual Studio, the Visual Studio code editor appears with the file open at the corresponding location. When using the Intel® Advisor GUI, the file type association (or Open With dialog box) determines the editor used. On Linux* OS: When using the Intel Advisor GUI, the editor defined by the Options > Editor dialog box appears with the file open at the corresponding location. <p>To return to the Suitability Source or Suitability Report window, click the Result tab.</p>
Select multiple source lines	To copy source lines using the context menu.
Right click a source line or multiple source lines	Display a context menu to: open the code editor to the corresponding source file, copy the selected source line(s) to the clipboard, or display context-sensitive help relevant to the selected loop or function.





Call Stack Pane

Use this pane to locate and open your source code from a call stack.

Location

On the right of the Suitability Source Window.

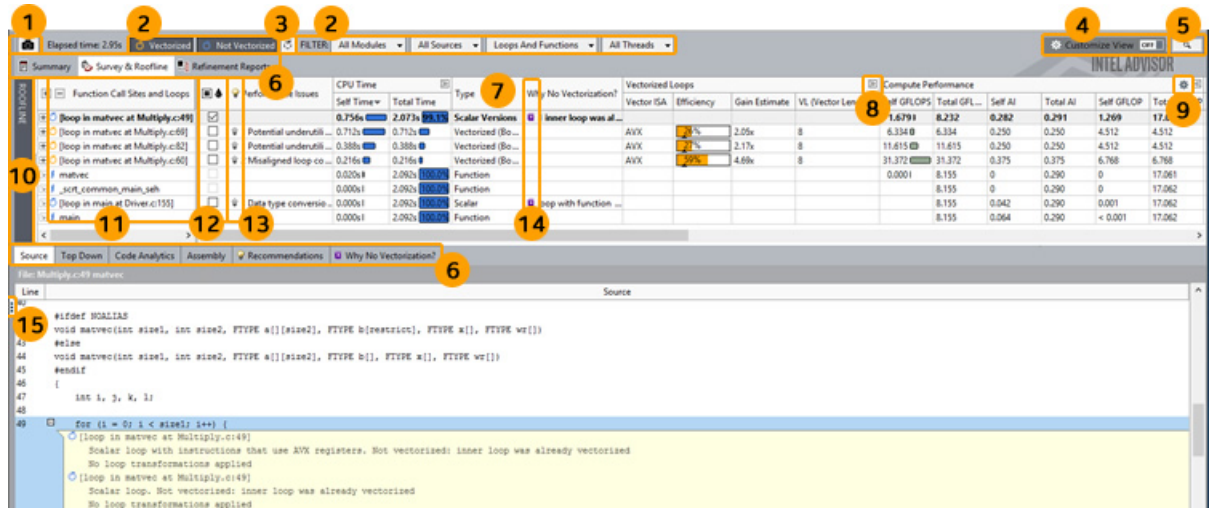
Controls

Use This	To Do This
 or  icon.	View whether: <ul style="list-style-type: none"> The row displayed is for a function . Source code is available for viewing and editing. An icon indicates whether source code is available  or not available .
Click a row in the Call Stack pane	Displays source code for the specified location in the call stack tree.
Pane border (drag)	Resize the pane.
Right click a row in the Call Stack pane	Customize call stack presentation by using the Call Stack context menu.

Window: Survey Report

After running the [Vectorization and Code Insights](#) perspective, Intel Advisor generates a Survey report that helps you identify top time-consuming and non- or under-vectorized functions/loops.

Controls



There are many controls available to help you focus on the data most important to you, including the following:

- 1 Click the control to save a read-only result snapshot you can view any time.
Intel Advisor stores only the most recent analysis result. Visually comparing one or more snapshots to each other or to the most recent analysis result can be an effective way to judge performance improvement progress.
To open a snapshot, choose **File > Open > Result...**
- 2 Click the various **Filter** controls to temporarily limit displayed data based on your criteria.
- 3 Click the control to view loops in non-executed code paths for various instruction set architectures (ISAs). Prerequisites:
 - Compile the target application for multiple code paths using the Intel compiler.
 - Enable the **Analyze loops in not executed code path** checkbox in **Project Properties > Analysis Target > Survey Hotspots Analysis**.
- 4 This toggle control currently combines two features: The **View Configurator** and the **Smart Mode** filter.
 - **View Configurator** - Toggle on the **Customize View** control to choose the view layout to display: **Default**, **Smart Mode**, or a customized view layout. To create a customized view layout you can apply to this and other projects:
 - 1 Click the **Settings** control next to the **View Layout** drop-down list to open the **Configure Columns** dialog box.
 - 2 Choose an existing view layout in the **Configuration** drop-down list.
 - 3 Enable/disable columns to show/hide.

Outcome: *Copy n* is added to the name of the selected view layout in the **Configuration** drop-down list.

 - 4 Click the **Rename** button and supply an appropriate name for the customized view layout.
 - 5 Click **OK** to save the customized view layout.

- **Smart Mode Filter** - Toggle on the **Customize View** control to temporarily limit displayed data to the top potential candidates for optimization based on **Total CPU Time** (the time your application spends actively executing a function/loop and its callees). In the **Top** drop-down list, choose one of the following:
 - The **Number** of top loops/functions to display
 - The **Percent of Total CPU Time** the displayed loops/functions must equal or exceed

5 Click the button to search for specific data.

6 Click the tab to open various Intel Advisor reports or views.

7 Right-click a column header to:

- Hide the associated report column.
- Resume showing all available report columns.
- Open the **Configure Columns** dialog box (see #4 for more information).

8 Click the toggle to show all available columns in a column set, and resume showing a limited number of preset columns in a column set.

9 Click the control to:





- Show options for customizing data in a column or column set.
- Open the **Configure Columns** dialog box (see #4 for more information).

For example, click the control in the **Compute Performance** column set to:

- Show data for floating-point operations only, for integer operations only, or for the sum of floating-point and integer operations.
- Determine what is counted as an integer operation in integer calculations:
 - Choose **Show Pure Compute Integer Operations** to count only ADD, MUL, IDIV, and SUB operations.
 - Choose **Show All Operations Processing Integer Data** to count ADD, ADC, SUB, MUL, IMUL, DIV, IDIV, INC/DEC, shift, and rotate operations.

10 Click the control to show/hide a chart that helps you visualize actual performance against hardware-imposed performance ceilings, as well as determine the main limiting factor (memory bandwidth or compute capacity), thereby providing an ideal roadmap of potential optimization steps.

11 Click a data row in the top of the **Survey Report** to display more data specific to that row in the bottom of the **Survey Report**. Double-click a loop data row to display a **Survey Source** window. To more easily identify data rows of interest:

-  = Vectorized function
-  = Vectorized loop
-  = Scalar function
-  = Scalar loop

12 Click a checkbox to mark a loop for deeper analysis.

13 If present, click the image to display code-specific *how-can-I-fix-this-issue?* information in the **Recommendations** pane.

14 If present, click the image to view the reason automatic compiler vectorization failed in the **Why No Vectorization?** pane.

15 Click the control to show/hide the **Workflow** pane.

Window: Survey Source

Source View Pane

Use this pane to view the user-visible source code representation of the selected site.

Location

Middle of the Survey Source window.

Controls

Use This	To Do This
Source lines	You can navigate to related source lines or explore your source code by using the Call Stack with Loops pane.
Double-click a source line	<p>To open your code editor to the corresponding source file. The editor allows you to add annotations to your code (right-click to open the context menu). You can use the annotation assistant pane to help you copy parallel site and task annotations.</p> <ul style="list-style-type: none"> On Windows* OS: <ul style="list-style-type: none"> When using Microsoft Visual Studio*, the Visual Studio code editor appears with the file open at the corresponding location. When using the Intel® Advisor GUI, the file type association (or Open With dialog box) determines the editor used. On Linux* OS: When using the Intel Advisor GUI, the editor defined by the Options > Editor dialog box appears with the file open at the corresponding location. <p>To return to the Survey Source or Survey Report window, click the Result tab.</p>
Select multiple source lines	To view the accumulated time values for multiple source lines below the Self Time column, or enable you to copy multiple source lines using the context menu. Viewing accumulated time can help you decide how to divide the work.
Right click a source line or multiple source lines	Display a context menu to: open your code editor to the corresponding source line, copy the selected source line(s) to the clipboard, or display context-sensitive help relevant to the selected loop or function.

Assembly View Pane

Use this pane to view assembly representation for a selected loop.

Location

Bottom of Survey Source window.

Controls

Use This	To Do This
Source lines	You can navigate to related source lines or explore assembly representation of the code by using the Call Stack with Loops pane.

Use This	To Do This
Select multiple source lines	To view the accumulated time values for multiple source lines below the Self Time column, or enable you to copy multiple source lines using the context menu. Viewing accumulated time can help you decide how to divide the work.










Call Stack Pane

View the call stack for a selected code region.

Location

On the right of the Survey Source window.

Controls

Use This	To Do This
 ,  , or  icon.	View whether: <ul style="list-style-type: none"> The row displayed is for a function  or a loop . A function  or loop  icon indicates that source code is available. Source code is available for viewing and editing. A  function or  loop icon indicates that source code is not available.
Click a row in the Call Stack pane	Displays source code for the specified location in the call stack tree.
Pane border (drag)	Resize the pane.
Right click a row in the Call Stack pane	Customize call stack presentation by using the Call Stack context menu.

Window: Threading Summary

After running [Threading perspective](#), review a results summary that includes the most important information about your code. Click the **Summary** tab after running an analysis to view results.

Program Metrics Pane


View the main performance metrics of your program, such as execution time statistics, vector instruction set (and whether extensions, such as VNNI, are used), and number of CPU threads utilized. The section is broken down into several sub-sections:

- Performance characteristics:** View execution time details, such as total CPU time and time spent in vectorized and scalar code.

If your application uses Intel® oneAPI Math Kernel Library, you will see the **MKL detail** button in the **Performance characteristics** section, which toggles two additional columns: the **User** column, which reports time spent in your code and corresponding compute metrics, and the **MKL** column, which reports time spent in the oneMKL code and corresponding compute metrics.
- Vectorization Gain/Efficiency:** View average estimated speedup of vectorized loops and total estimated program speedup.

NOTE

The vectorization efficiency data is available only for vectorized loops in modules compiled with an Intel® compiler version 16 or higher.

- **OP/S and Bandwidth:** View GFLOPS and GINTOPS usage and cache bandwidth metrics compared to hardware peak. Hover the mouse over the **Utilization** column and click the  button to select single-core or multicore benchmarks utilization metrics.

NOTE

The OP/S and bandwidth metrics are available after you run the Trip Counts and FLOP analysis.

Per Program Recommendations Pane

View suggested changes for your program that you might want to apply to achieve better performance.

Top Time-consuming Loops Pane

View top five time-consuming loops sorted by *total time* with performance metrics, such as execution time statistics and vectorization efficiency with comparison to original scalar loop efficiency.

Suitability and Dependencies Analysis Data Pane

View information about predicted sharing problems for annotated parallel sites. The **Maximum Site Gain** column summarizes potential performance improvement achieved through threading. The **Dependencies** column summarizes the predicted data sharing problems. To display the **Dependencies Report** window at the corresponding parallel site location, click a function link under the **Site Location** column.

NOTE

This information is available only after you run the Suitability or Dependencies analysis.

Recommendations Pane

View suggested changes with high confidence level for first five loops in the code that you might want to apply to achieve better performance. Click a recommendation link to access the recommendations texts.

Collection Details Pane

View execution statistics for each of the collectors, as well as the **Collection Log**, **Application Output**, and **Collection Command Line** links that lead to the corresponding report logs, command line and output details.

NOTE

Application Output is available if you set output destination to Application Output window. To do this, go to **File > Options > General > Application Output Destination** and choose **Application Output window**.

Platform Information Pane

View the system information including software and hardware summary.

Window: Vectorization Summary

After running the [Vectorization and Code Insights perspective](#), consider reviewing a results summary that includes the most important information about your code. Click the **Summary** tab after running an analysis to view results.


Program Metrics Pane

View the main performance metrics of your program, such as execution time statistics, vector instruction set (and whether extensions, such as VNNI, are used), and number of CPU threads utilized. The section is broken down into several sub-sections:

- **Performance characteristics:** View execution time details, such as total CPU time and time spent in vectorized and scalar code.
If your application uses Intel® oneAPI Math Kernel Library (oneMKL), you will see the **MKL detail** button in the **Performance characteristics** section, which toggles two additional columns: the **User** column, which reports time spent in your code and corresponding compute metrics, and the **MKL** column, which reports time spent in the oneMKL code and corresponding compute metrics.
- **Vectorization Gain/Efficiency:** View average estimated speedup of vectorized loops and total estimated program speedup.

NOTE

The vectorization efficiency data is available only for vectorized loops.

- **OP/S and Bandwidth:** View GFLOPS and GINTOPS usage and cache bandwidth metrics compared to hardware peak. Hover the mouse over the **Utilization** column and click the  button to select single-core or multicore benchmarks utilization metrics.

NOTE

The OP/S and bandwidth metrics are available after you run the Trip Counts and FLOP or the Roofline analysis.

Per Program Recommendations Pane

View suggested changes for your program that you might want to apply to achieve better performance.

Top Time-consuming Loops Pane

View top five time-consuming loops sorted by *self time* with performance metrics, such as execution time statistics and vectorization efficiency with comparison to original scalar loop efficiency.

Refinement Analysis Data Pane

View details about found dependencies and memory access patterns.

The **Dependencies** column summarizes the predicted data sharing problems collected by the Dependencies tool. To display the **Dependencies Report** window at the corresponding parallel site location, click a function link in the **Site Location** column.

The **Strides Distribution** column reports the memory access stride distribution within a loop in the ratio format in %: unit strides, constant strides, and variable strides.

NOTE

The information in the **Refinement analysis data** section is available only after you run the Memory Access Patterns or Dependencies analysis.

Recommendations Pane

View suggested changes with high confidence level for first five loops in the code that you might want to apply to achieve better performance. Click a recommendation link to access the recommendations texts.

Collection Details Pane

View execution statistics for each of the collectors, as well as the **Collection Log**, **Application Output**, and **Collection Command Line** links that lead to the corresponding report logs, command line and output details.

NOTE

Application Output is available if you set output destination to Application Output window. To do this, go to **File > Options > General > Application Output Destination** and choose **Application Output window**.


Platform Information Pane

View the system information including software and hardware summary.

Appendix

Data Sharing Problems

In a serial program, the order of the operations during program execution are known. However, when code executes as multiple parallel tasks, an operation can execute before, after, or simultaneously with an operation in the other task. For example, when parallel tasks access or modify a shared memory location, data sharing problems can occur.

The Intel® Advisor Dependencies tool  performs extensive analysis of your running serial program to help you predict data sharing problems. Use the **Dependencies Report** window and the topics introduced by this section to help you understand and decide how to fix the reported data sharing problems.

For each data sharing problem, you can either:

- Modify the sources to fix *incidental* or accidental data sharing by privatizing shared data use. This type of data sharing occurs when tasks use the same memory location, but do not communicate about using that memory location. If the data written by one is not needed by the other, each task could use a private copy of the data.
- Add lock annotations to implement synchronization for *independent updates*. This type of sharing occurs when multiple tasks contribute to determining the final value of a memory location.
- Recognize that the order of the operations cannot change, and consider modifying the chosen parallel sites and their tasks. When shared data access must occur in the original sequential order, this is called *true dependence*.

The following sections explain how to understand and fix sharing problems.

Data Sharing Problem Types

A data sharing problem happens when two tasks access the same memory location, and the behavior of the program depends on the order of accesses. This group of topics describes two common data access patterns - incidental sharing and independent updates - that result in data sharing problems that are relatively easy to fix. The fixes are described in [Problem Solving Strategies](#).

The task's code is called its *static extent*. You need to understand all the data accesses that might be executed during the execution of the task. You are interested in accesses to memory locations in the *dynamic extent* of a task, which includes all functions called from the task's static extent, all functions the called functions may in turn call, and so on.

Incidental Sharing

Sharing is *incidental* when tasks use the same memory location, but do not communicate any information using it.

The Basic Pattern

Suppose that a task always writes to a memory location before reading from it, and that the value that it writes is not read again outside the task. For example:

```
extern int x;
// ...
ANNOTATE_SITE_BEGIN(site1);
for (i = 0; i != n; ++i) {
    ANNOTATE_ITERATION_TASK(task1);
    x = a[i];
    b[i] = x * b[i];
}
ANNOTATE_SITE_END(site1);
```

The variable `x` is both read and written in the task, so there will be a sharing problem when multiple copies of the task execute at the same time. For example:

1. Task 0 sets `x` to `a[0]`.
2. Task 1 sets `x` to `a[1]`.
3. Task 0 computes `x * b[0]`.

What is interesting is that the sharing is incidental to the logic of the program. Each iteration of the loop uses `x`, but its use in each iteration is totally independent. Memory locations used in this way are called *privatizable*, because giving each task its own private memory location will eliminate the sharing without changing the program behavior.

Memory Allocators

The use of dynamically allocated memory is a special case of incidental sharing. Consider this task:

```
ANNOTATE_TASK_BEGIN(task2);
Type *ptr = allocate_Type();
// some code that uses the object pointed to by ptr
free_Type(ptr);
ANNOTATE_TASK_END();
```

If `allocate_Type()` returns the same address to one task that was used and freed by another task, then those tasks will both access the same memory location, but the sharing is incidental. The memory allocator will never return a pointer to memory that has been allocated and not freed, so the tasks will not use the same dynamically allocated memory location at the same time, and the appearance of sharing is an illusion.

The Dependencies tool understands the standard memory allocators such as C/C++ `new/delete` and `malloc/free`, but it does not know about any custom memory allocators that your program might have. If your code has custom memory allocators, you can mark their uses with the special-purpose C/C++ annotations `ANNOTATE_RECORD_ALLOCATION` and `ANNOTATE_RECORD_DEALLOCATION`.

See Also

[Independent Updates](#)

[Special-purpose Annotations](#)

Independent Updates

Independent updates can occur when multiple tasks contribute to determining the final value of a memory location.

The Basic Pattern

Suppose that multiple tasks write to a memory location, that the value written by each task is computed using the previous value in that location, and that the order in which the tasks update the memory location does not matter.

For example, consider a loop that sums all the values in an array:

```
extern int x;
// ...
ANNOTATE_SITE_BEGIN(site1);
for (i = 0; i != n; ++i) {
    ANNOTATE_ITERATION_TASK(task1);
    x = x + a[i];
}
ANNOTATE_SITE_END(site1);
printf("%d\n", x);
```

The sharing problem looks like this:

1. Task 0 reads `x`.
2. Task 1 reads `x`.
3. Task 0 adds `a[0]` to the value it read and stores the result back in `x`.
4. Task 1 adds `a[1]` to the value it read and stores the result back in `x`, overwriting the value stored by task 0.

The important fact is that the updates of `x` can be performed in any order. All you need to do is to make sure that no task can write to `x` between the read from `x` and the write to `x` in any other task; the uses of `x` in the tasks are otherwise independent.

Reductions

Reductions are a special case of the independent update pattern. The reduction pattern occurs when a loop combines a collection of values using a commutative, associative function.

In *Adding Parallelism to Your Program*, you will see that the Intel® oneAPI Threading Building Blocks and OpenMP* parallel frameworks have special features for writing parallel reductions.

Transactions

In a more general form of this pattern, there may be multiple memory locations which must be updated together.

```
void insert_node_in_list(T *new_node, T *insert_after)
{
    new_node->next = insert_after->next;
    new_node->prev = insert_after->next->prev;
```



```
insert_after->next->prev = new_node;
insert_after->next = new_node;
}
```

Two insertions must not occur simultaneously, but the insertions may occur in any order, as long as the final list order does not matter.

A collection of updates that must all occur together is referred to as a *transaction*.

Guard Variables

A special case is the use of a shared memory location to control some additional code. The update and the code that depends on it may be treated as a transaction.

```
bool initialized = false;
void do_something()
{
    if (!initialized) {
        do_the_initialization();
        initialized = true;
    }
    do_the_real_work();
}
```

If `do_something()` is called from multiple tasks, then the sharing problem is:

1. Task 0 reads `initialized`, which is false, and enters the body of the `if` statement.
2. Task 1 reads `initialized`, which is false, and enters the body of the `if` statement, so `do_the_initialization()` is called twice.

It does not matter which task the initialization occurs in, so your only problem is to make sure that other tasks wait until this initialization has happened.

Independent Writes

The simplest case occurs when the value that the tasks write to the memory location does not depend on its previous value:

```
bool found = false;
ANNOTATE_SITE_BEGIN(site1);
for (i = 0; i != n; ++i) {
    ANNOTATE_ITERATION_TASK(task1);
    if (a[i] == b) found = true;
}
if (found) printf("found\n");
ANNOTATE_SITE_BEGIN(site1);
```

There is no read to keep together with the write, and it does not matter what order the writes to `found` occur in, so the tasks are totally independent, and can execute concurrently without restrictions. If a task writes to `found` at all, it will write the value true.

Note that if the task body were the following, then this example would fit the reduction pattern:

```
found = found || (a[i] == b);
```

This is also called a benign race because the program will always compute the same value, regardless of which thread does the last write.

See Also

[Problem Solving Strategies](#)

[Adding Parallelism to Your Program](#)

Eliminating Incidental Sharing

Problem Solving Strategies

[Data Sharing Problem Types](#) describes the kinds of problems that can occur when tasks access the same memory locations. Two common strategies are used to deal with the following sharing problems:

- *Incidental sharing*: If a memory location is shared, but it is not used to communicate data between tasks, then you can eliminate the sharing by giving each task its own copy of the shared memory. This rarely causes significant increases in execution time or memory consumption. See [Eliminating Incidental Sharing](#).
- *Independent updates*: If the reads and writes of the memory location occur in updates which can be done in any order, then you can add synchronization code to guarantee that the updates and related code in different tasks cannot be intermingled. This can increase execution time because only one task at a time can be accessing the shared memory location. This limits parallel execution. See [Synchronizing Independent Updates](#).

If neither of these applies, you might be able to restructure your program to avoid the sharing problem; otherwise you may have to change your task structure. See [Difficult Problems: Choosing a Different Set of Tasks](#).

Eliminate Incidental Sharing

Sharing problems involving a task and a memory location are incidental if the memory location does not carry information into or out of the task. Therefore, if you replace all uses of the shared memory location in the task with uses of some non-shared memory location, you eliminate the sharing problem without changing the behavior of the program.

The following sections describe incidental sharing problems and their solutions.

Examine the Task's Static and Dynamic Extent

Consider the example of incidental sharing from the help topic [Incidental Sharing](#):

```
extern int x;
ANNOTATE_SITE_BEGIN(site1);
for (i = 0; i != n; ++i) {
    ANNOTATE_ITERATION_TASK(task1);
    x = a[i];
    b[i] = x * b[i];
}
ANNOTATE_SITE_END();
```

Examining the Static Extent of the Task

If you define a substitute variable inside the static extent, then each task will get its own private storage for it:

```
extern int x;
// ...
ANNOTATE_SITE_BEGIN(site2);
for (i = 0; i != n; ++i) {
    ANNOTATE_ITERATION_TASK(task2);
    int x_sub;
    x_sub = a[i];
    b[i] = x_sub * b[i];
}
ANNOTATE_SITE_END();
```

Examining the Dynamic Extent of the Task

In the simplest cases, like the example above, the task's dynamic extent is the same as its static extent - it does not contain any function calls. When it does contain function calls, all the functions that might be called while the task is executing are part of its dynamic extent, and you need to consider all reads and writes of the memory location in all of those functions.

So, you need to examine not only the static extent, but also the dynamic extent of a task.

See Also

[Verify Whether Incidental Sharing Exists](#)
[Data Sharing Problem Types](#)

Verify Whether Incidental Sharing Exists

Sharing is incidental only if the task writes to the memory location before any read of the memory location *anywhere in the dynamic extent of the task*. This is easy to check when the task is a few lines of code in a single function. It is much harder when the task is hundreds or thousands of lines of code, and involves calls to many functions in many source files.

Even worse, the sharing is not incidental if any code that might execute after the task completes, or in any other task that might run at the same time as the task, could read a value written by the task to that memory location.

There is no "magic bullet" to prove that the requirements are met, but there is a simple technique that you might find useful. Add statements that write a known bad value into the memory location immediately after the `ANNOTATE_ITERATION_TASK(taskname);`, and then test your serial program. If the sharing is incidental, these assignments will have no effect. If not, there is a good chance that the changes will change the program behavior. Of course, the effectiveness of this technique depends on how good your test system is at detecting the resulting bugs.

For example, if you want to confirm that the variable `x` is incidentally shared in `the_task()`:

```
extern int x;
// ...
ANNOTATE_SITE_BEGIN(site1);
for (i = 0; i != n; ++i) {
    ANNOTATE_ITERATION_TASK(task1);
    x = 0xdeadbeef;
    the_task();
    x = 0xdeadbeef;
}
ANNOTATE_SITE_BEGIN();
```

To identify stray memory references, consider using the C/C++ special-purpose annotations `ANNOTATE_OBSERVE_USES()` and `ANNOTATE_CLEAR_USES()`.

See Also

[Creating the Private Memory Location](#)
[Special-purpose Annotations](#)

Create the Private Memory Location

The important thing is that every execution of a task must get its own private memory location to take the place of the shared memory location in the original program. This involves:

- Creating the private memory location.
- Replacing all uses of the shared memory location with uses of the private memory location.

How you do this will depend on what kind of shared memory location you have.

Replacing a Local Variable

If the shared memory location is a local variable in the function containing the task's static extent, the fix is simple:

1. Add braces around the static extent, if necessary, to make sure that it is a block.
2. Define a new variable at the beginning of the block.
3. Replace every use of the shared variable in the static extent with a use of the new variable.

Now each occurrence of the task will have its own copy of the local variable.

Replacing a Static or Global Variable or Class Static Data Member

Using global variables is usually a bad idea in large-scale software design. Global variables often seem like the easiest solution to a design problem, but they create obscure dependencies between parts of a program, making it harder to understand the program and harder to make changes to it. When you convert a program to run in parallel, these problems are compounded, as global variables are a prolific source of data sharing problems.

Therefore, the changes that you will make to eliminate uses of global variables are not only necessary to fix sharing problems and allow your program to run correctly in parallel. You will probably find that they would make your program more understandable and maintainable, even if you were not parallelizing it.

For example:

```
extern int global;
// ...
ANNOTATE_SITE_BEGIN(site1);
    ANNOTATE_TASK_BEGIN(taskname);
        foo(i);
        bar(i);
    ANNOTATE_TASK_END();
// ...
ANNOTATE_SITE_END(site1);
void foo(int i)
{
    global = x*3 - a[i];
}
void bar(int i)
{
    b[i] = b[i] - global;
}
```

The approach to creating a private replacement for a static or global variable or a class static data member is the same as for a local variable. The important difference is that global variables may be accessed in other functions in the dynamic extent of the task, so you will have to make the private replacement variable accessible to those functions, too.

1. Define a local variable in the static extent to take the place of the shared variable, just as you do when replacing a local variable.
2. Replace all uses of the shared variable in the static extent with uses of the new local variable.
3. If the task calls other functions, add an additional reference parameter to each one, and pass the private variable to it. If you are programming in C, you will have to use a pointer parameter and pass the address of the variable to it.
4. Replace all uses of the shared variable in the called functions with uses of the reference parameter.

Applying these rules to the example above, we get:

```
// ...
ANNOTATE_SITE_BEGIN(site1);
ANNOTATE_TASK_BEGIN(taskname);
int replacement;
```

```

foo(i, replacement);
bar(i, replacement);
ANNOTATE_TASK_END();
ANNOTATE_SITE_END(site1);
// ...
void foo(int i, int& replacement)
{
    replacement = x*3 - a[i];
}
void bar(int i, int& replacement)
{
    b[i] = b[i] - replacement;
}

```

Mixed-caller functions: If there are functions that are called both from the dynamic extent of the task and from outside the task, steps 3 and 4 will fix the sharing problem in the task, but they will break the calls outside the task. There are two possible solutions:

- Modify all the calls to such functions from outside the task to pass the original variable to the new parameter.
- Make two copies of the function: the original version, to be called from outside the task, and the one with the new parameter, to be called from inside the task.

Variables whose address is taken: The special problems of variables that are accessed through pointers are discussed in the help topic [Pointer Dereferences](#).

More than one shared variable: The strategy described above is simple, but if you have a task with several incidentally shared variables, the multiple extra parameters are clumsy. A cleaner solution is to define a local structure variable with a field for each incidentally shared variable. Modify the functions and calls in the task's dynamic extent to pass the structure to them, and replace uses of the shared variables with uses of the appropriate field of the structure.

Creating a Task Class: If the functions in the task's dynamic extent are closely related, you might be able to create a new class which has the functions as member functions and the replacement shared variables as data members. Then the class's `this` pointer takes the place of the added reference parameter. Using the same example:

```

class TaskClass {
public:
    ANNOTATE_SITE_BEGIN(site1);
    void the_task()
    {
        ANNOTATE_TASK_BEGIN(taskname);
        foo(i);
        bar(i);
        ANNOTATE_TASK_END();
    }
    ANNOTATE_SITE_END(site1);
private:
    int replacement;
    void foo(int i)
    {
        replacement = x*3 - a[i];
    }
    void bar(int i)
    {
        b[i] = b[i] - replacement;
    }
};

```

Replacing a Structure Field

Sometimes you may have sharing problems with one or more fields in an object:

```
struct Point { float x, y, z; };
extern Point p;
// ...
ANNOTATE_SITE_BEGIN(site1);
    ANNOTATE_TASK_BEGIN(taskname);
    p.x = a[i].x * scale_x;
    p.y = a[i].y * scale_y;
    foo(i);
    ANNOTATE_TASK_END();
ANNOTATE_SITE_END(site1);
// ...
void foo(int i)
{
    b[i].x = b[i].x - p.x;
    b[i].y = b[i].y - p.y;
}
```

The most straightforward solution is to introduce a new local variable for the shared field:

```
struct Point { float x, y; };
extern Point p;
// ...
ANNOTATE_SITE_BEGIN(site1);
    ANNOTATE_TASK_BEGIN(taskname);
    float sub_p_x = a[i].x * scale_x;
    float sub_p_y = a[i].y * scale_y;
    foo(I, sub_p_x, sub_p_y);
    ANNOTATE_TASK_END();
ANNOTATE_SITE_END(site1);
// ...
void foo(int i, float& sub_p_x, float& sub_p_y)
{
    b[i].x = b[i].x - sub_p_x;
    b[i].y = b[i].y - sub_p_y;
}
```

If every shared field of the object is incidentally shared, then it will be simpler to make a single local replacement variable for the entire structure rather than a separate replacement variable for each shared field.

```
struct Point { float x, y; };
extern Point p;
//...
ANNOTATE_SITE_BEGIN(site1);
    ANNOTATE_TASK_BEGIN(taskname);
    Point sub_p;
    sub_p.x = a[i].x * scale_x;
    sub_p.y = a[i].y * scale_y;
    foo(I, sub_p);
    ANNOTATE_TASK_END();
ANNOTATE_SITE_END(site1);
// ...
void foo(int i, Point& sub_p)
{
    b[i].x = b[i].x - sub_p.x;
    b[i].y = b[i].y - sub_p.y;
}
```

See Also

Pointer Dereferences

Pointer Dereferences

It may be tedious to find all the uses of a shared variable in a task's dynamic extent, but at least it is relatively straightforward. The situation is much worse when the Dependencies tool reports that you have a sharing problem on a pointer dereference. In general:

- A dereference of a pointer expression may or may not refer to the same object as some other dereference of a pointer expression with the same type.
- Different executions of a pointer expression dereference may or may not refer to the same object.
- If you have a variable whose address is taken, a dereference of a pointer expression may or may not refer to that variable.

However, suppose that your program has an abstract data type whose objects are implemented as dynamically allocated data structures. You may be able to step back from the individual pointer dereferences involved in a sharing problem and say: "These are just implementation details in an access to an abstract object." If you can prove that the access pattern for the abstract object satisfies the incidental sharing pattern, you can apply the techniques from this topic:

- Within the task, create a private object of the abstract data type.
- Make a reference to the private object available throughout the task.
- Replace references to the original object with references to the private object.
- Destroy the private object before the task exits.

The point is to ignore the pointer dereferences, and solve the problem in terms of the abstraction that they are implementing.

Additional suggestions for dealing with sharing problems with pointer-accessed memory locations can be found in [Memory That is Accessed Through a Pointer](#).

See Also

Synchronizing Independent Updates

Memory That is Accessed Through a Pointer

Synchronize Independent Updates

In the independent update pattern, both of the following occur:

- Two tasks contain regions of code that update the same memory locations.
- It does not matter what order the code regions execute in, as long as the regions do not execute in parallel.

For example, suppose that multiple tasks call `do_something()`:

```
void do_something()
{
    static bool initialized = false;
    if (!initialized) {
        do_the_initialization();
        initialized = true;
    }
    do_the_real_work();
}
```

The function `do_something()` updates the variable `initialized` as well as the initialized memory locations. The function `do_the_real_work()` will never be called before the initialization happens; and the initialization will only happen once, regardless of which task calls `do_something()` first, as long as two tasks do not try to execute the `if` statement at the same time. If two tasks do try to execute the `if` statement at the same time, they could both see that `initialized` is false and both try to do the initialization.

The following sections describe several aspects of synchronizing independent updates, including explicit locking, assigning locks, and potential problems of using synchronization.

Synchronization

You can fix independent update sharing problems by synchronizing the execution of code that uses the same memory locations. The key idea is that when two or more tasks contain groups of operations which should not execute at the same time, there must be a lock which controls the execution of all of these groups of operations. Such a group of operations is called a *transaction*, and may be anything from a read/modify/write of a single variable to a collection of related modifications to multiple data structures.

Before beginning a transaction, a task must *acquire* the lock that controls it, and when the transaction is done, the task must *release* it. If one task has already acquired a lock, then another task that tries to acquire the same lock will stop executing until the first task has released it. This guarantees that two transactions controlled by the same lock cannot execute at the same time.

Use the Advisor lock annotations `ANNOTATE_LOCK_ACQUIRE` and `ANNOTATE_LOCK_RELEASE` to describe a transaction you intend to lock. Later, you will modify the lock annotations to actual code that implements a lock using the chosen parallel framework code:

```
void do_something()
{
    static bool initialized = false;
    ANNOTATE_LOCK_ACQUIRE(0);
    if (!initialized) {
        do_the_initialization();
        initialized = true;
    }
    ANNOTATE_LOCK_RELEASE(0);
    do_the_real_work();
}
```

Locks are identified by a lock address.

See Also

[Explicit Locking](#)

[Lock Annotations](#)

Explicit Locking

Use `ANNOTATE_LOCK_ACQUIRE` and `ANNOTATE_LOCK_RELEASE` to specify explicit locking. These annotations are simple executable statements you can put wherever is most convenient. For example:

```
if (synchronization_needed) ANNOTATE_LOCK_ACQUIRE(0);
x = f(x, a);
if (synchronization_needed) ANNOTATE_LOCK_RELEASE(0);
```

You must make sure you match the lock acquires and releases, and both occur in the same task. Your program will get synchronization errors if a task releases a lock that it does not own, or acquires a lock and fails to release it. You can acquire a lock in one function and release it in a different function, but it is a poor practice.

See Also

[Assigning Locks to Transactions](#)

[Lock Annotations](#)

Assign Locks to Transactions

A transaction updates a set of shared memory locations and is controlled by a lock. In general, you need to be sure that if two transactions both access the same memory location, they will not run simultaneously. What is the best way to associate locks with transactions to accomplish that? Consider:

```
// transaction 1
if (a > b) { a -= b; b = b / 2; }
```



```

...
// transaction 2
if (c > d) { c -= d; d = d / 2; }
...
// transaction 3
if (a > c) { a -= c; c = c / 2; }
...
// transaction 4
temp = x;
x = y;
y = temp;

```

You must ensure that if two transactions can access the same memory location, they are controlled by the same lock. The simplest way to do this is to assign locks to sets of memory locations, so that if a transaction accesses two or more memory locations, all of the memory locations accessed in the transaction have the same lock. Then a transaction must be controlled by the lock that is assigned to all of the variables it accesses.

In the example above, variables *a* and *b* are both accessed in transaction 1, so they must have the same lock. Variables *c* and *d* are both accessed in transaction 2, so they must have the same lock. Variables *a* and *c* are both accessed in transaction 3, so they must have the same lock, which must be the same as the locks for *b* and *d*. Transaction 4 accesses *x* and *y*, so they must have the same lock, which is different from the lock for *a*, *b*, *c*, and *d*:

```

int abcd_lock;
int xy_lock;
// ...
ANNOTATE_LOCK_ACQUIRE(&abcd_lock);
if (a > b) { a -= b; b = b / 2; }
ANNOTATE_LOCK_RELEASE(&abcd_lock);
ANNOTATE_LOCK_ACQUIRE(&abcd_lock);
if (c > d) { c -= d; d = d / 2; }
ANNOTATE_LOCK_RELEASE(&abcd_lock);
ANNOTATE_LOCK_ACQUIRE(&abcd_lock);
if (a > c) { a -= c; c = c / 2; }
ANNOTATE_LOCK_RELEASE(&abcd_lock);
ANNOTATE_LOCK_ACQUIRE(&xy_lock);
temp = x;
x = y;
y = temp;
ANNOTATE_LOCK_RELEASE(&xy_lock);

```

See Also

[Pitfalls from Using Synchronization](#)

Pitfalls from Using Synchronization

Synchronization is a relatively simple way to eliminate sharing problems, but it must be used very carefully.

Performance

The purpose of synchronization is to let tasks run safely in parallel, but it does this by not letting them run in parallel when it would be unsafe. A task that is waiting for a lock is not doing any work at all.

Also, acquiring and releasing locks take a non-trivial amount of time. It is easy to write tasks that spend more time doing synchronization than doing useful work.

Taken together, these two issues mean that you need to be careful how much you use synchronization. Synchronization should be used carefully to solve specific problems. If you find yourself synchronizing large portions of your tasks, you may need to rethink your task structure so that you can get useful tasks that can run safely without so much synchronization.

One strategy is for a task to synchronize its import of the data it needs into private memory locations, work on this private data, and then synchronize the export of the results.

Synchronization Errors

The final problem with synchronization is the danger of deadlocks. A deadlock happens when one or more threads cannot make progress. This can happen, for example, when a task has acquired one lock and is trying to acquire another, while another task has acquired this second lock and is trying to acquire the first. This situation is called *deadlock*, and it could cause a program to hang forever.

After adding synchronization, to see whether your changes have solved the problems, run the Dependencies tool again. This may reveal previously hidden and newly introduced problems. For example, after you add locks, run the Dependencies tool again to make sure you have not accidentally introduced deadlocks into your program or unbalanced pairs of annotations. The Dependencies tool can detect potential deadlocks because in addition to memory data accesses, it observes the lock events when the annotated program runs.

See Also

[Difficult Problems: Choosing a Different Set of Tasks](#)

Difficult Problems: Choosing a Different Set of Tasks

If you find a conflict that cannot be resolved using the above techniques, you should consider the following alternatives.

- Merge the two tasks involved in the conflict into a single task.
- Divide the tasks into smaller tasks and do the work preceding the conflict in parallel, the work involving the conflict serially, and the work after the conflict in parallel.
- Find a different site to introduce parallelism.
- Intel Advisor presents a simplified model of what is possible with parallel programming. Occasionally it will be beneficial to take advantage of more advanced techniques that are available in the Intel® oneAPI Threading Building Blocks (oneTBB), OpenMP*, or native threading APIs.

Any changes - other than lock annotations - you have made to fix incidental sharing problems should be left in the code. These changes will not harm performance, they have improved maintainability, and they may be useful at the new site or in future parallelization efforts.

See Also

[Fixing Problems in Code Used by Multiple Parallel Sites](#)

Fix Problems in Code Used by Multiple Parallel Sites

If the Dependencies tool reports a problem(s) in one or more common functions used by multiple parallel sites, you need to investigate and consider several options. In general, keep in mind that the performance impact for privatization is usually less than synchronization, and the performance impact for synchronization is usually less than not adding parallelism. Thus, the general approach is:

- Evaluate whether it is possible to privatize the data causing the Dependencies problem for both sites. For example, you can often use privatization if the cause is incidental (accidental) sharing. Usually, providing each task with its own private copy of a variable provides the best performance.
- If you cannot fix the problem by privatizing, consider using synchronization (such as using locks or mutexes). For example, synchronization is often needed if the Dependencies problem is caused by independent updates that have true dependence.
- In some cases, it may not be feasible to add parallelism to a site. That is, after you to modify the annotations for the parallel site or its tasks and check the Suitability and Dependencies again, you might find a parallel site has negative or minimal performance gain and/or complex data sharing problems. In this case, you may need to remove the site and task annotations and add a comment that states that this code location could not be parallelized.

If you cannot eliminate a Dependencies problem for a common function called by multiple parallel sites by using the approach described above, consider adding a cloned function. That is, one of the parallel sites calls the cloned function and the other parallel site calls the original function. This allows you to implement different fixes to the Dependencies problem(s) for the original function and the cloned function. For example, this approach might allow you to privatize data in either the original function or the cloned function, which was not possible originally.

As with any program, after you modify the code within a parallel site or the annotations, you should run and analyze your program again using the Dependencies and Suitability tools.

See Also

[Memory That is Accessed Through a Pointer](#)

[Data Sharing Problem Types](#)

Memory That is Accessed Through a Pointer

In the topic Pointer Dereferences, we saw that there are techniques for dealing with incidental sharing of pointer-accessed storage in particular cases.

In general, to deal with sharing problems at indirect references you have to really understand what your program is doing. You cannot just do a text search for all the uses of a shared location and apply some transformation mechanically.

Although you may not know which memory location is being accessed by an indirect reference, you may be able to tell that a set of indirect references using the same pointer value implement an independent update pattern. The process for synchronizing independent updates of indirect references is the same as for variables. The only special concern is that you need to use the same lock for all data accesses that might be accessing the same memory locations. Using the same lock in more places means that your tasks will spend more time waiting for it.

Finally, your design may have tasks working on separate parts of a larger data structure. If you find sharing problems, it may be that the parts are not as independent as designed. In that case, you are likely to get the best results by disentangling the data structures to resolve the sharing problems.

See Also

[Pointer Dereferences](#)

Notational Conventions

The following conventions may be used in this document.

Convention	Explanation	Example
<i>Italic</i>	Used for introducing new terms, denotation of terms, placeholders, or titles of manuals.	The filename consists of the <i>basename</i> and the <i>extension</i> . For more information, see the <i>Intel® Advisor User Guide</i> .
Bold	Denotes GUI elements	Click Cancel .
>	Indicates a menu item inside a menu.	File > Close indicates to select Close from the File menu.
Monospace	Indicates directory paths and filenames, or text that can be part of source code.	<code>ippsapi.h</code> <code>\alt\include</code> Use the <code>okCreateObjs()</code> function to... <code>printf("hello, world\n");</code>

Convention	Explanation	Example
*	An asterisk at the end of a word or name indicates it is a third-party product trademark.	Adobe Acrobat*
Windows* OS, Windows operating system	These terms refer to all supported Windows* operating systems.	This table contains a summary of Windows* OS Linking Behavior.
Linux* OS, Linux operating system	These terms refer to all supported Linux* operating systems.	This table contains a summary of Linux* OS Linking Behavior.

Key Concepts

This group of topics introduces you to the key concepts and terms needed to add parallelism to a program. A list of key terms is also provided.

Over the last few years, processor technology found in personal laptops, desktops, and enterprise servers has shifted from making single-core processors faster to having multiple cores in each processor.

In a parallel program, portions of the program (*tasks*) may execute at the same time. On multi-core systems, this can provide better performance.

To parallelize your application, you need to identify the potential parallel tasks, modify your code to run correctly when these tasks execute in parallel, and add code to execute them in parallel. Intel® Advisor combines a methodology with a set of tools to help you add this parallelism to your program. You work on the sequential version of your program and the tools *model* how it would behave if it was parallelized in the ways you specify. As an add-in to Microsoft Visual Studio*, Intel Advisor fits right into a Windows* OS development environment.

NOTE Support for Microsoft* Visual Studio* 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

Your final step will be to express the parallelism in your program using a high-level parallel framework (threading model) like Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP*, or low-level threading APIs.

For native C/C++ or Fortran code, Intel recommends using the high-level oneTBB or OpenMP frameworks, which are included with several Intel® oneAPI Toolkits. Intel Advisor's documentation shows you how to introduce parallelism into your program using these frameworks. Intel Advisor provides multiple C/C++ samples and several Fortran samples.

For managed C# code on Windows* OS, use the Microsoft Task Parallel Library* (TPL). Intel Advisor provides a C# `nqueens` sample.

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

This is an interactive process, where you repeat these basic steps as you identify more sites for adding parallelism.

The related group of topics provides an introduction to parallelism, and to parallel framework implementations.

See Also
[Parallelism](#)
[Glossary](#)

Glossary

Amdahl's law: A theoretical formula for predicting the maximum performance benefits of parallelizing application programs. Amdahl's law states that run-time execution time speedup is limited by the part of the program that is not parallelized (executes serially). To achieve results close to this potential, overhead must be minimized and all cores need to be fully utilized. See also *Use Amdahl's Law and Measuring the Program*.

annotation: A method of conveying information about proposed parallel execution. In the Intel® Advisor, you create annotations by adding macros or function calls. These annotations are used by Intel Advisor tools to predict parallel execution. For example, the C/C++ `ANNOTATE_SITE_BEGIN(sitename)` macro identifies where a *parallel site* begins. Later, to allow this code to execute in parallel, you replace the annotations with code needed to use a *parallel framework*. See also *parallel framework* and *Annotation Types Summary*.

atomic operation: An operation performed by a thread on a memory location(s) that is guaranteed not to be interfered with by other threads. See also *synchronization*.

chunking: The ability of a parallel framework to aggregate multiple instances of a task into groups for more efficient parallel processing. For tasks that do small amounts of computation and many iterations, task chunking can minimize task overhead. You can also restructure a single loop into an inner and outer loop (strip-mining). See also *task* and *Enable Task Chunking*.

code region: A subtree of loops/functions in a call tree. Synonym *whole Loopnest*.

critical section: A *synchronization* construct that allows only one thread to enter its associated code region at a time. Critical sections enforce *mutual exclusion* on enclosed regions of code. With Intel Advisor, mark critical sections by using `ANNOTATE_LOCK_ACQUIRE()` and `ANNOTATE_LOCK_RELEASE()` annotations.

data race: When multiple threads share (read/write) a memory location, if the program does not implement controls to manage the sequence of concurrent memory accesses, one thread can inadvertently overwrite data written by another thread, or otherwise read or write stale data. This can produce execution errors that are difficult to detect and reproduce, such as obtaining different calculated results when the same executable is run on different systems. To prevent data races, you can add data synchronization constructs that restrict shared memory access to one thread at a time, or you might eliminate the sharing.

data parallelism: Occurs when a single portion of code is paired with multiple portions of data, and each pairing executes as a task. For example, tasks are made by pairing a loop body with each element of an array iterated by the loop, and the tasks execute in parallel. See also *Task Patterns*. Contrast *task parallelism*.

data set: A set of data to be used as input or with an interactive application the way you interact with the application to cause a portion of the application to be executed. Because the Dependencies tool watches each memory access in a parallel site in great detail, the parallel site's code takes much longer to run than usual. To limit the time needed to run Dependencies analysis, reduce the data (such as the number of loop iterations) and when using an interactive program, create a very small test case. See also *Choose a Small, Representable Data Set for the Dependencies Tool*.

deadlock: A situation where a set of threads have each acquired some locks and are waiting for other locks to be released. All threads in the set are waiting for a lock held by a different thread, and since none can proceed and release their lock(s), they all remain waiting.

dynamic extent: All code that may possibly be executed by a *parallel site* or *task*. For example, a dynamic extent might include a loop, all functions called from the loop, all functions the called functions may in turn call, and so on. Contrast *static extent*. See also *Task Organization and Annotations*.

false positive: When viewing the Dependencies Report, a problem reported by the Dependencies tool that is not an actual problem.

framework: See *parallel framework*

head: A loop or function at the top of a subtree, which contains one or more child loops/functions.

hotspot: A small code region that consumes much of the program's run time. Hotspots can be identified by a profiler, such as the Intel Advisor Survey tool. See also *Use Amdahl's Law and Measuring the Program*.

Intel® oneAPI Threading Building Blocks (oneTBB) : A C++ template library for writing programs that take advantage of multiple cores. You can use this library to write scalable programs that specify tasks rather than threads, emphasize data parallel programming, and take advantage of concurrent collections and

parallel algorithms. This is provided as an Intel® software product - *Intel® oneAPI Threading Building Blocks (oneTBB)* - as well as open source. Intel® oneAPI Threading Building Blocks (oneTBB) is one of several *parallel frameworks*. Abbreviation oneTBB .

load balancing: The equal division of work among cores. If the load is balanced, the cores are busy most of the time.

lock: A *synchronization* mechanism that allows one thread to wait until another thread allows it to continue. A lock can be used to synchronize threads accessing a specific memory location. See also *synchronization* and *nested lock*.

multi-core: A processor that combines two or more independent cores. Although each core shares interconnection to the rest of the system, it executes instructions independently by using its dedicated CPU, architectural state, and interrupt controllers, as well as private and/or shared cache. Most multi-core systems use identical cores. The number of cores used determines whether it is called dual-core (2), quad-core (4), or many-core system.

multithreaded processing: See *parallel processing*

mutual exclusion: A type of locking typically used to prevent actions occurring at the same time. Abbreviation *mutex*. See also *synchronization*

nested lock: A type of *lock* that can be locked again by a task when the task already owns the lock. Nested locks are convenient when several inter-related functions use the same lock. See also *synchronization* and *lock*

node: A loop or function.

oneTBB : See *Intel® oneAPI Threading Building Blocks (oneTBB)*

OpenMP*: A high-level parallel framework and language extension designed to support shared-memory parallel programming that consists of compiler directives (C/C++ pragmas and Fortran directives), library functions, and environment variables. The OpenMP specification was developed by multiple hardware and software vendors to provide a scalable, portable interface for parallel programming on a variety of platforms. OpenMP is one of several parallel frameworks. See also <http://openmp.org>.

parallel framework: A combination of libraries, language features, or other software techniques that enable code for a program to execute in parallel. Examples include *OpenMP*, *Intel® oneAPI Threading Building Blocks (oneTBB)* , Message Passing Interface (MPI), Intel® Concurrent Collections for C/C++ , Microsoft Task Parallel Library* (TPL), and low-level, basic threading APIs, like POSIX* threads (Pthreads). Some parallel frameworks support shared-memory parallel processing, while others like MPI support non-shared-memory parallel processing. See also *Intel® oneAPI Threading Building Blocks (oneTBB)* and *Parallel Frameworks Overview*.

parallel processing: The use of multiple threads during execution of a program. Intel Advisor focuses on parallel processing for *shared-memory systems*. There are other types of parallel processing, such as for clusters or grids and vector processing. Shortened version is *parallelism*. See also *hotspot* and *thread*.

parallel region: *Offload Modeling term*. A code region that starts with a specific parallel framework construction. Intel® oneAPI Threading Building Blocks (oneTBB), Intel® oneAPI Data Analytics Library (oneDAL), OpenMP*, SYCL parallel frameworks are supported.

parallel site: A region of code that contains tasks that can execute in parallel. See also *annotation* and *Task Organization and Annotations*

pipeline: An approach to organizing task computations that uses both data parallelism and task parallelism, and organizes the computation into stages that run in a predetermined order.

self time: In the Survey Report window, how much time was spent in a particular function or loop.

site: See *parallel site*

shared-memory parallelism: See *parallel processing*

static extent: The code between a site's or a task's `_BEGIN` and `_END` annotations. A static extent might not be lexically paired; for example, a parallel site may have one `_BEGIN` point, but may require multiple independent `_END` exit points. Contrast with *dynamic extent*. See also *annotation*, *parallel site*, and *Task Organization and Annotations*.

synchronization: Coordinating the execution of multiple threads. In some cases, you can provide synchronization within a task by using a private memory location instead of a shared memory location. In other cases, a *lock* or *mutex* can be used to restrict access to a shared data. See also Data Sharing Problem Types.

task: A portion of code and its data that can be given to a thread to execute. See also Task Organization and Annotations, Choosing the Tasks, and *chunking*.

task parallelism: Occurs when two different portions of the code are made into tasks and execute in parallel. For example, a task is made by pairing a display algorithm with the state to display, another task by pairing a compute-next-state algorithm with the same state, and the two tasks execute in parallel. See also Task Patterns. Contrast *data parallelism*

thread: A thread executes instructions within a process. Each process has one or more threads active at a time. Threads share the address space of the process, but have their own stack, program counters, and other registers.

total time: In the Survey Report window, how much time was spent in a particular function or loop, plus the time spent by anything that entity calls.

vector processing: A form of parallel processing where multiple data items are packed together in vector registers to allow vector instructions to operate on the packed data with a single instruction. Reducing the number of instructions needed to process the packed vector data minimizes memory use and latency, and provides good locality of reference and data cache utilization. Vector instructions are Single Instruction Multiple Data (SIMD) instructions. Some SIMD vector instructions support large register sizes to accommodate more packed data, such as Intel® Advanced Vector Extensions (Intel® AVX).

Parallelism

The following topics describe some key terms related to multithreaded parallel processing (parallelism), an overview of multithreaded parallelism, and common issues when adding multithreaded parallelism to your program:

- If you are just learning about adding multithreaded parallel processing to application programs, please read these topics carefully.
- If you have advanced knowledge about multithreaded parallel processing and are familiar with the concepts, quickly read (scan) these topics so you are familiar with the terms used.

Parallel Processing Terminology

A serial (non-parallel) program uses a single thread, so you do not need to control the side-effects that can occur when multiple threads interact with shared resources.

A program takes time to run to completion. A serial program only uses a single core, so its run time will not decrease by running it on a system with multiple cores. However, if you add parallel processing (parallelism) to parts of the program, it can use more cores, so it finishes sooner.

Threads and Tasks

An operating system *process* has an address space, open files, and other resources. A *thread* executes instructions within a process. Each process has one or more threads active at a time. Threads share the address space of the process, but have their own stack, program counter, and other registers. A program that uses multiple threads is called a *multithreaded* or *parallel* program.

A *task* is a portion of a program that can be run in parallel with other portions of the program and other instances of that task. Each task instance is run by a thread, and the operating system assigns threads to cores.

Hotspots - Find Where a Program Spends Its Time

A *hotspot* is a small code region that consumes much of the program's run time. You can use profiling tools such as the Survey tool provided with Intel Advisor to identify where your program spends its time. To improve your program's performance when you add parallelism:

- Find the hotspots and hot parts of the call tree, such as hot loops or hot routines. The Intel Advisor Survey tool's report provides an extended top-down call tree that identifies the top hot loops.
- Examine all the functions in the call tree from `main()` to each hot routine or loop. You want to distribute frequently executed instructions to different tasks that can run at the same time.

Data and Task Parallelism

If the hot part of the call tree is caused by executing the same region of code many times, it may be possible to divide its execution by running multiple instances of its code, each on a separate core. This is called *data parallelism* because each execution is processing different parts of the same composite data item. Compute-intensive loops over arrays are often good candidates for data parallelism. For example, the line `process(a[i]);` below is a possible task:

```
for (int i = 0; i != n; ++i) {  
    process(a[i]);  
}
```

If two or more hotspots are close to each other in the serial execution, and do not share data, it may be possible to execute the hotspots as tasks. This is *task parallelism*. For example:

```
initialize(data);  
while (!done) {  
    old_data = data;  
    display_on_screen(old_data);  
    update(data);  
}
```

Making effective use of multiple cores may require both data-level parallelism to process large amounts of data, and task-parallelism to overlap the execution of unrelated portions of the program.

Add Parallelism

The best performance improvements from adding parallel execution (parallelism) to a program occur when many cores are busy most of the time doing useful work. Achieving this requires a lot of analysis, knowledge, and testing.

Because your serial program was not designed to allow parallel execution, as you convert parts of it to use parallel execution, you may encounter unexpected errors that occur only during parallel execution. Instead of wasting effort on portions of a program that use almost no CPU time, you must focus on the hotspots, and the functions between the main entry point and each hotspot.

If you naively add parallel execution to a program without proper preparation, unpredictable crashes, program hangs, and wrong answers can result from incorrect parallel task interactions. For example, you may need to add *synchronization* to avoid incorrect parallel task interactions, but this must be done carefully because locking overhead and serial synchronization can reduce the benefits of the parallel execution.

Intel Advisor helps you:

- Find the possible code regions where you could add parallel execution.
- Choose the code regions best-suited for parallel execution. This includes measuring approximate parallel performance so you can experiment with different possible parallel code regions.
- Find and eliminate potential data sharing problems before parallel execution is introduced.

See Also

[Common Issues When Adding Parallelism](#)

Common Issues When Adding Parallelism

The types of problems encountered by parallel programs include shared memory data conflicts and incorrect locking.

Shared Memory Problems

Introducing parallelism can result in unexpected problems when parallel tasks access the same memory location. Such problems are known as *data races*. For example, in the `Primes` sample, the following line calls the function `Tick()`:

```
if (IsPrime(p)) Tick();
```

The called function `Tick()` increments the global variable `primes`:

```
void Tick() { primes++; }
```

Consider the following scenario, where the value of `primes` is incremented only once instead of twice:

Time	Thread 0	Thread 1
T1	Enters function <code>Tick()</code>	
T2		Enters function <code>Tick()</code>
T3		Load value of <code>primes</code>
T4	Load value of <code>primes</code>	
T5		Increment loaded value
T6		Store value of <code>primes</code>
T7	Increment loaded value	
T8	Store value of <code>primes</code>	
T9	Return	
T10		Return

If you run this as a serial program, this problem does not occur. However, when you run it with multiple threads, the tasks may run in parallel and `primes` may not be incremented enough.

Such problems are non-deterministic, difficult to detect, and at first glance might seem to occur at random. The results can vary based on multiple factors, including the workload on the system, the data being processed, the number of cores, and the number of threads.

It is possible to use *locks* to restrict access to a shared memory location to one task at a time. However, all implementations of locks add overhead. It is more efficient to avoid the sharing by replicating the storage. This is possible if data values are not being communicated between the tasks, even though the memory locations are being reused.

Lock Problems

One thread (thread A) may have to wait for another thread (thread B) to release a lock before it can proceed. The core executing thread A is not performing useful work. This is a case of lock contention. In addition, thread B may be waiting for thread A to release a different lock before it can proceed. Such a condition is called a *deadlock*.

Like a data race, a deadlock can occur in a non-deterministic manner. It might occur only when certain factors exist, such as the workload on the system, the data being processed, or the number of threads.

Ensuring the Parallel Portions of a Program are Thread Safe

Intel® Advisor can detect many problems related to parallelism. Because it only analyzes the serial execution of your program, Intel Advisor cannot detect all possible errors. When you have finished using Intel Advisor to introduce parallelism into your program, you should use the Intel® Inspector and other Intel software suite products. These tools and using a debugger can detect parallelism problems that normal testing will not detect, and can also identify times when the cores are idle.

See Also

[Parallel Programming Implementations](#)

[Using Intel® Inspector and Intel® VTune™ Profiler](#)

[Debugging Parallel Programs](#)

[Data Sharing Problem Types](#)

Parallel Programming Implementations

There are two popular approaches for adding parallelism to programs. You can use either:

- A high-level parallel framework like Intel® oneAPI Threading Building Blocks (oneTBB) or OpenMP*. Of these parallel frameworks for native code, oneTBB supports C++ programs and OpenMP supports C, C++, or Fortran programs. For managed code on Windows* OS such as C#, use the Microsoft Task Parallel Library* (TPL).

NOTE C# and .NET support is deprecated starting Intel® Advisor 2021.1.

- A low-level threading API like Windows* threads or POSIX* threads. In this case, you directly create and control threads at a low level. These implementations may not be as portable as high-level frameworks.

There are several reasons that Intel recommends using a high-level parallel framework:

- **Simplicity:** You do not have to code all the detailed operations required by the threading APIs. For example, the OpenMP* `#pragma omp parallel for` (or Fortran `!$OMP PARALLEL DO`) and the oneTBB `parallel_for()` are designed to make it easy to parallelize a loop (see Reinders Ch. 3). With frameworks, you reason about tasks and the work to be done; with threads, you also need to decide how each thread will do its work.
- **Scalability:** The frameworks select the best number of threads to use for the available cores, and efficiently assign the tasks to the threads. This makes use of all the cores available on the current system.
- **Loop Scalability:** oneTBB and OpenMP assign contiguous *chunks* of loop iterations to existing threads, amortizing the threading overhead across multiple iterations (see oneTBB `grain_size`: Reinders Ch. 3).
- **Automatic Load Balancing:** oneTBB and OpenMP have features for automatically adjusting the grain size to spread work amongst the cores. In addition, when the loop iterations or parallel tasks do uneven amounts of work, the oneTBB scheduler will dynamically reschedule the work to avoid idle cores.

To implement parallelism, you can use any parallel framework you are familiar with.

The high-level parallel frameworks available for each programming language include:

Language	Available High-Level Parallel Frameworks
C	OpenMP
C++	Intel® oneAPI Threading Building Blocks (oneTBB) OpenMP
C#	Microsoft Task Parallel Library* (Windows* OS only)
Fortran	OpenMP

See Also

[Parallel Frameworks](#)

Other Parallel Frameworks

Related Information

A variety of resources provide additional information on a number of topics.

Intel Analyzers

Explore more profiling and optimization opportunities with Intel performance analysis tools:

- [Intel® Advisor](#) to design your code performance on Intel hardware with the roofline methodology and explore potential for vectorization, threading, and offload optimizations.
- [Intel® Inspector](#) to analyze your code for threading, memory, and persistent memory errors.
- [Intel® VTune™ Profiler](#) to analyze your algorithm choices and identify where and how your application can benefit from available hardware resources.
- [Intel® Graphics Performance Analyzers](#) to analyze performance of your game applications (system, frame, and trace analysis).

More Resources

To View	Access
Intel® oneAPI DPC++/C++ Compiler	See the Intel® oneAPI DPC++/C++ Compiler documentation at https://www.intel.com/content/www/us/en/develop/tools/oneapi/components/dpc-compiler.html .
Intel® Fortran Compiler Classic documentation	See the Intel® Fortran Compiler Classic documentation at https://www.intel.com/content/www/us/en/develop/tools/oneapi/components/fortran-compiler.html .
Intel® oneAPI Threading Building Blocks (oneTBB) documentation	See the documentation for oneTBB at https://www.intel.com/content/www/us/en/develop/tools/oneapi/components/onetbb.html .
Intel® MPI Library documentation and resources	Intel MPI Library documentation at: https://www.intel.com/content/www/us/en/develop/tools/mpi-library/get-started.html Articles about using Intel MPI Library , such as <i>Hybrid applications: Intel MPI Library and OpenMP</i> on the Intel® Developer Zone at: https://www.intel.com/content/www/us/en/develop/articles/hybrid-applications-intelmpi-openmp.html
Intel® oneAPI Programming Guide	https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top.html
Description of Intel® microarchitectures and their instruction sets	http://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures

For Intel® software product documentation, see <https://www.intel.com/content/www/us/en/develop/documentation.html>

For additional technical product information, including white papers about Intel products, see the Intel® Developer Zone at <https://www.intel.com/content/www/us/en/develop/home.html>