Getting Started Tutorial: Finding Hotspots

Intel® VTune™ Amplifier XE 2013 for Linux* OS

C++ Sample Application Code

Document Number: 326705-002

Legal Information

Contents

Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, Flexpipe, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel CoFluent, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel Xeon Phi, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Pentium, Pentium Inside, Puma, skoool, the skoool logo, SMARTi, Sound Mark, Stay With It, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Overview

Discover how to use Hotspots analysis of the Intel® VTune™ Amplifier XE to understand where your application is spending time, identify *hotspots* - the most time-consuming program units, and detect how they were called.

Hotspots analysis is useful to analyze the performance of both serial and parallel applications.

About This Tutorial	This tutorial uses the sample `tachyon` and guides you through basic steps required to analyze the code for hotspots.
Estimated Duration	10-15 minutes.
Learning Objectives	After you complete this tutorial, you should be able to: • Choose an analysis target. • Choose the Hotspots analysis type. • Run the Hotspots analysis to locate most time-consuming functions in an application. • Analyze the function call flow and threads. • Analyze the source code to locate the most time-critical code lines. • Compare results before and after optimization.
More Resources	• Intel® Parallel Studio XE tutorials (HTML, PDF): http://software.intel.com/en-us/articles/intel-software-product-tutorials/ • Intel® Parallel Studio XE support page: http://software.intel.com/en-us/articles/intel-parallel-studio-xe/

1 Navigation Quick Start

Intel® VTune™ Amplifier XE, an Intel® Parallel Studio XE tool, provides information on code performance for users developing serial and multithreaded applications on Windows* and Linux* operating systems. VTune Amplifier XE helps you analyze algorithm choices and identify where and how your application can benefit from available hardware resources.

VTune Amplifier XE Access

VTune Amplifier XE installation includes shell scripts that you can run in your terminal window to set up environment variables:

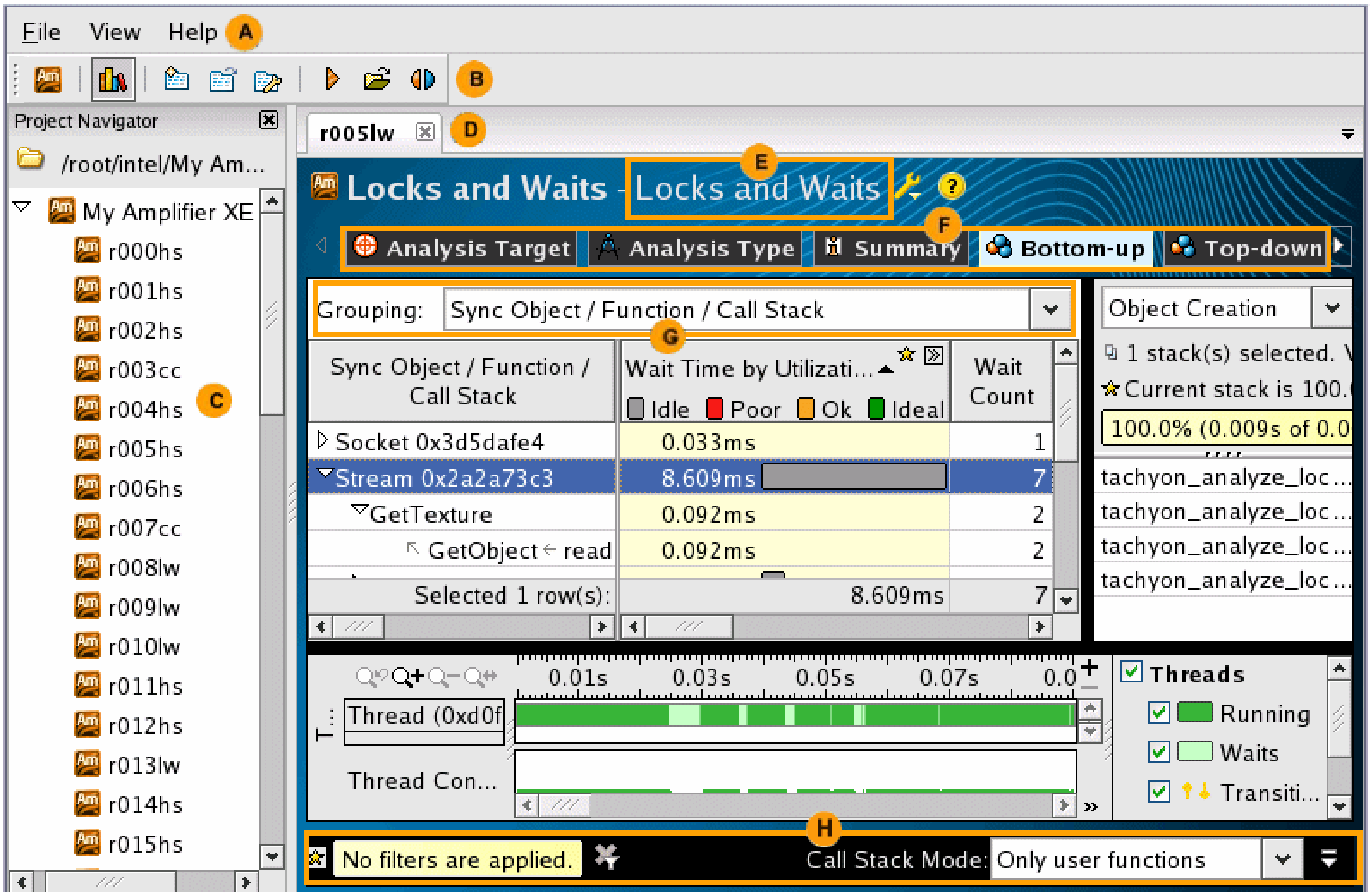
1. From the installation directory, type `source amplxe-vars.sh`.

   This script sets the PATH environment variable that specifies locations of the product graphical user interface utility and command line utility.

> **NOTE** The default installation directory is `/opt/intel/vtune_amplifier_xe_2013`.

2. Type `amplxe-gui` to launch the product graphical interface.

VTune Amplifier XE GUI



Grouping: Sync Object / Function / Call Stack

Sync Object / Function / Call Stack	Wait Time by Utilizati... ▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal	Wait Count
▷ Socket 0x3d5dafe4	0.033ms	1
▽ Stream 0x2a2a73c3	8.609ms	7
▽ GetTexture	0.092ms	2
↖ GetObject ← read	0.092ms	2
Selected 1 row(s):	8.609ms	7

Object Creation

1 stack(s) selected. V

Current stack is 100.

100.0% (0.009s of 0.0

tachyon_analyze_loc...
tachyon_analyze_loc...
tachyon_analyze_loc...
tachyon_analyze_loc...

No filters are applied.

Call Stack Mode: Only user functions

A Use the VTune Amplifier XE menu to control result collection, define and view project properties, and set various options.

B Configure and manage projects and results, and launch new analyses from the primary toolbar. Click the **Project Properties** button on this toolbar to manage result file locations.

C The **Project Navigator** provides an iconic representation of your projects and analysis results. Click the **Project Navigator** button on the toolbar to enable/disable the **Project Navigator**.

D Newly completed and opened analysis results along with result comparisons appear in the results tab for easy navigation.

E Use the drop-down menu to select a *viewpoint*, a preset configuration of windows/panes for an analysis result. For each analysis type, you can switch among several viewpoints to focus on particular performance metrics. Click the yellow question mark icon to read the viewpoint description.

F Switch between window tabs to explore the analysis type configuration options and collected data provided by the selected viewpoint.

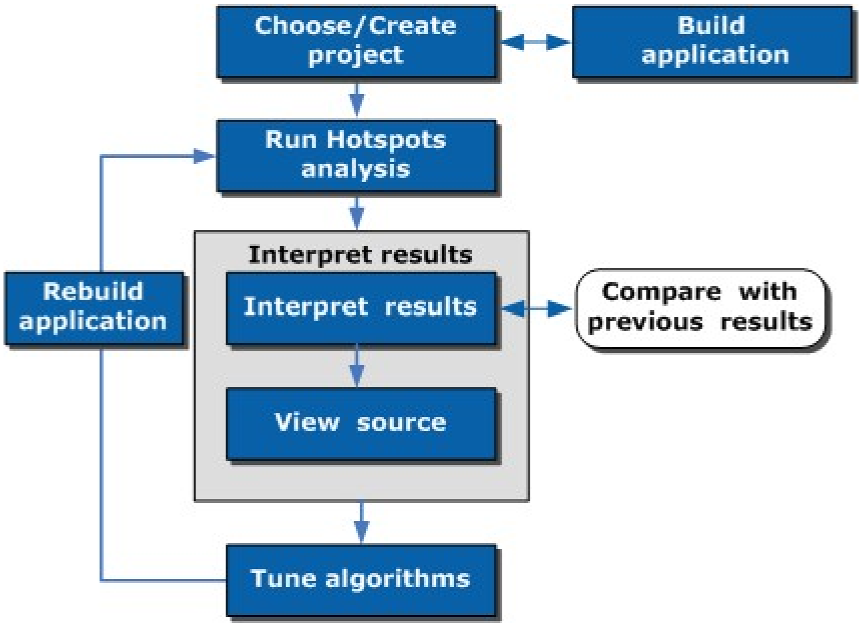
G Use the **Grouping** drop-down menu to choose a granularity level for grouping data in the grid.

H Use the filter toolbar to filter out the result data according to the selected categories.

Finding Hotspots

2

You can use the Intel® VTune™ Amplifier XE to identify and analyze hotspot functions in your serial or parallel application by performing a series of steps in a workflow. This tutorial guides you through these workflow steps while using a sample ray-tracer application named `tachyon`.



Step 1: Prepare for analysis	Build an application to analyze for hotspots and create a new VTune Amplifier XE project
Step 2: Find hotspots	• Choose and run the Hotspots analysis. • Interpret the result data. • View and analyze code of the performance-critical function.
Step 3: Eliminate hotspots	• Modify the code to tune the algorithms or rebuild the code with Intel® Compiler.
Step 4: Check your work	Re-build the target, re-run the Hotspots analysis, and compare the result data before and after optimization.

Build Application and Create New Project

Before you start analyzing your application target for hotspots, do the following:

1. Get software tools.
2. Build application with full optimizations.
3. Run the application without debugging to create a performance baseline.
4. Create a VTune Amplifier XE project.

Get Software Tools

You need the following tools to try tutorial steps yourself using the tachyon sample application:

- VTune Amplifier XE, including sample applications
- tar file extraction utility
- Supported compiler (see Release Notes for more information); optionally, Intel® C++ compiler

Acquire Intel VTune Amplifier XE

If you do not already have access to the VTune Amplifier XE, you can download an evaluation copy from http://software.intel.com/en-us/articles/intel-software-evaluation-center/.

Install and Set Up VTune Amplifier XE Sample Applications

1. Copy the tachyon_vtune_amp_xe.tgz file from the <install-dir>/samples/<locale>/C++/ directory to a writable directory or share on your system. The default installation path is opt/intel/vtune_amplifier_xe_2013.
2. Extract the sample from the tar file.

- Samples are non-deterministic. Your screens may vary from the screen captures shown throughout this tutorial.
- Samples are designed only to illustrate the VTune Amplifier XE features; they do not represent best practices for creating code.

Build the Target

Build the target with full optimizations, which is recommended for performance analysis.

1. Browse to the directory where you extracted the sample code (for example, /home/myuser/samples/tachyon). Make sure this directory contains Makefile.
2. Build your target:

   ```
   $ make
   ```

   The tachyon_find_hotspots application is built.

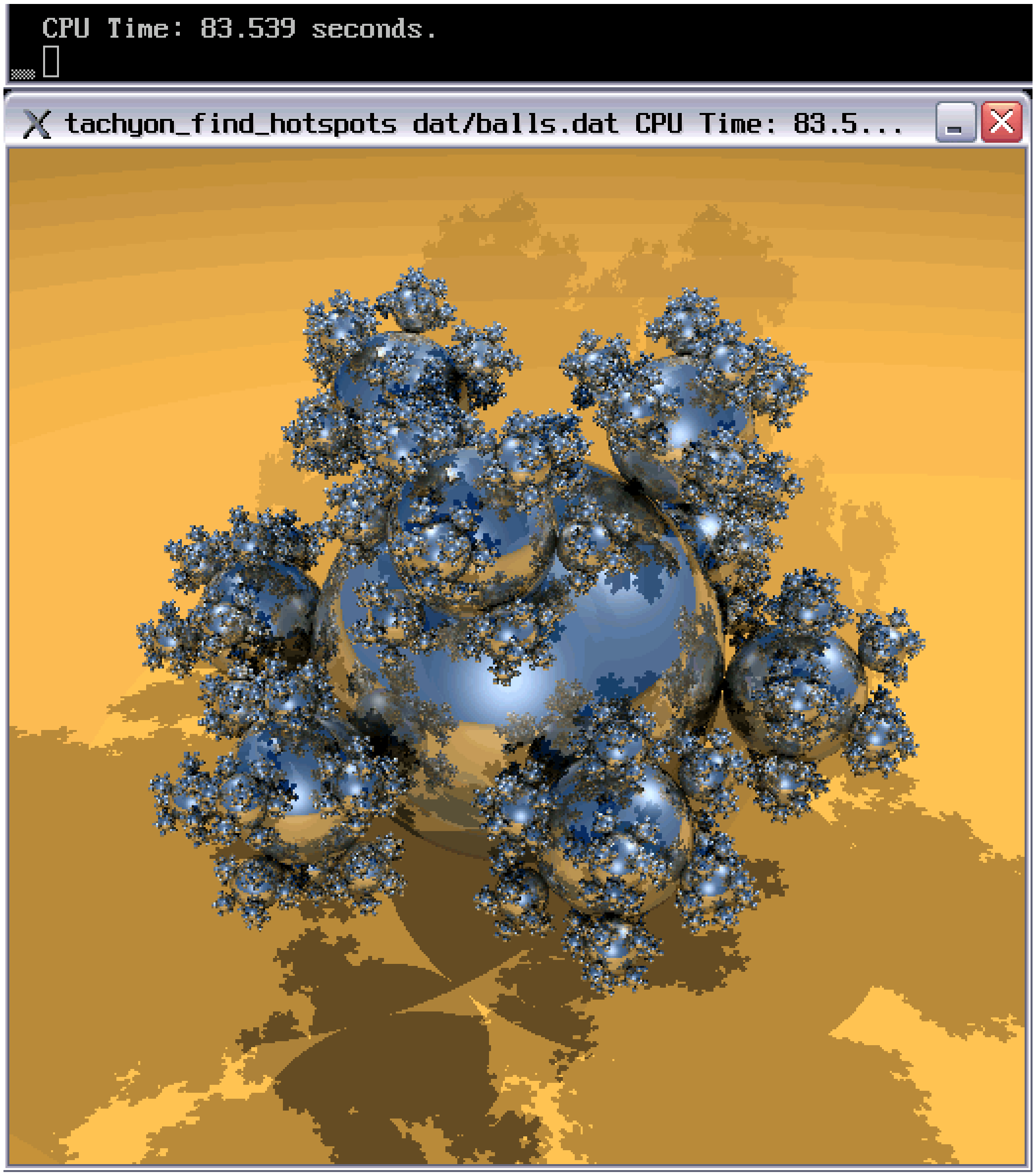
Create a Performance Baseline

1. Run tachyon with dat/balls.dat as an input parameter. For example:

   ```
   $ ./tachyon_find_hotspots dat/balls.dat
   ```

   The tachyon_find_hotspots application starts running.

NOTE Before you start the application, minimize the amount of other software running on your computer to get more accurate results.



```
CPU Time: 83.539 seconds.
```

2. Note the execution time displayed in the window caption or in the shell window. For the `tachyon_find_hotspots` executable in the figure above, the execution time is 83.539 seconds. The total execution time is the baseline against which you will compare subsequent runs of the application.

NOTE Run the application several times, note the execution time for each run, and use the average number. This helps to minimize skewed results due to transient system activity.

Create a Project

To analyze your target the VTune Amplifier XE, you need to create a project, which is a container for an analysis target configuration and data collection results.

1. Set the `EDITOR` or `VISUAL` environment variable to associate your source files with the code editor (like emacs, vi, vim, gedit, and so on). For example:

```
$ export EDITOR=gedit
```

2. Run the `amplxe-gui` script launching the VTune Amplifier XE GUI.
3. Create a new project via **File > New > Project...**.

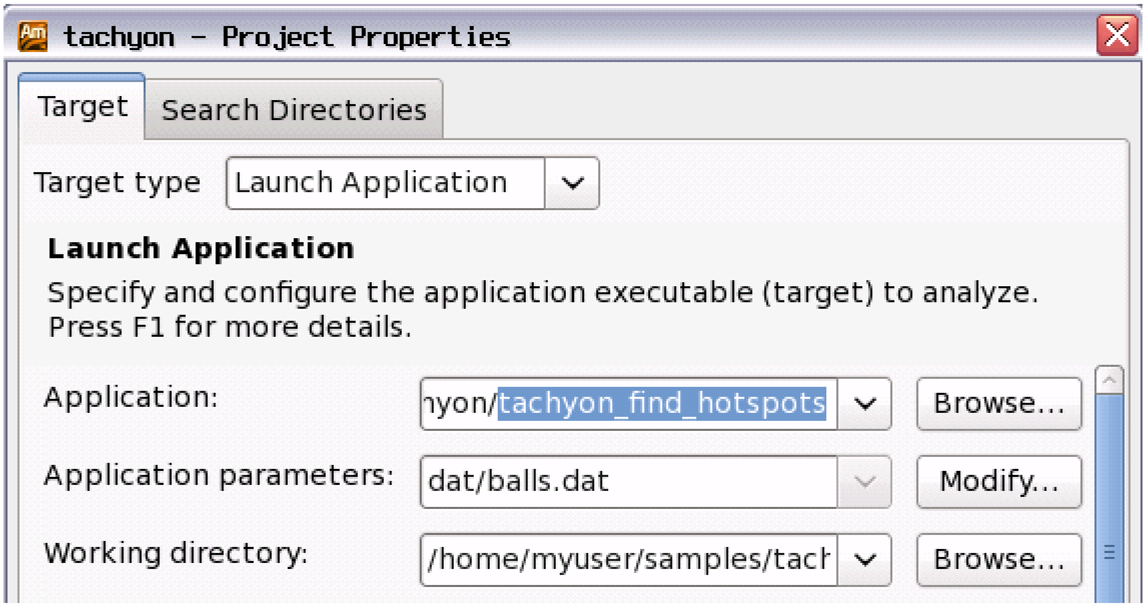
The **Create a Project** dialog box opens.

4. Specify the project name `tachyon` that will be used as the project directory name.

   VTune Amplifier XE creates the `tachyon` project directory under the `$HOME/intel/amplxe/projects` directory and opens the **Project Properties: Target** dialog box.

5. In the **Application to Launch** pane of the **Target** tab, specify and configure your target as follows:

   - For the **Application** field, browse to `<`*`sample_code_dir`*`>/tachyon_find_hotspots`, for example: `/home/myuser/samples/tachyon/tachyon_find_hotspots`.
   - For the **Application parameters** field, enter `dat/balls.dat`.



tachyon - Project Properties

Target | Search Directories

Target type: Launch Application

Launch Application

Specify and configure the application executable (target) to analyze. Press F1 for more details.

Application: hyon/tachyon_find_hotspots

Application parameters: dat/balls.dat

Working directory: /home/myuser/samples/tach

6. Click **OK** to apply the settings and exit the **Project Properties** dialog box.

Key Terms

- Baseline
- Target

Next Step

Run Hotspots Analysis

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Run Hotspots Analysis

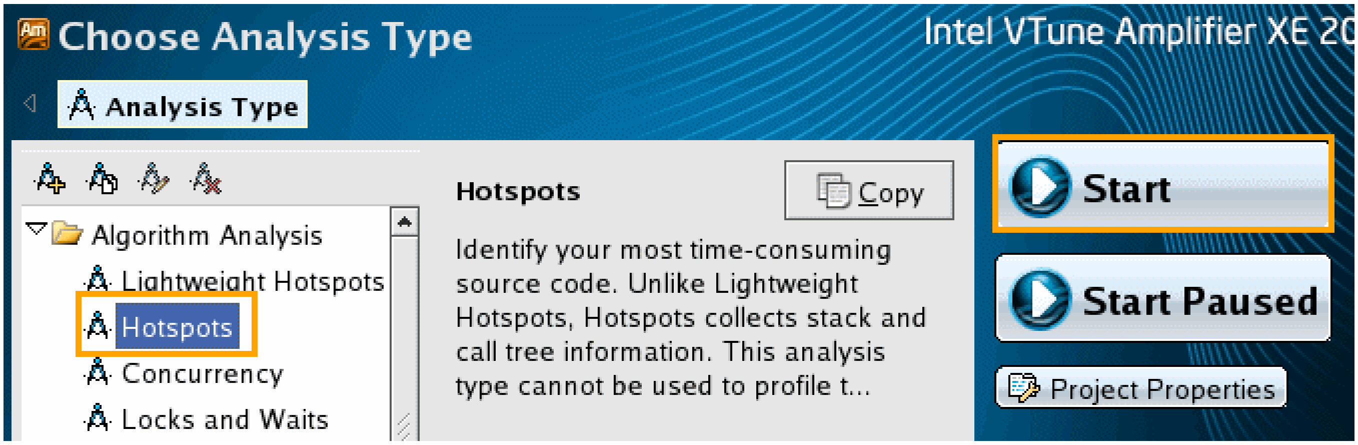
Before running an analysis, choose a configuration level to influence Intel® VTune Amplifier XE analysis scope and running time. In this tutorial, you run the Hotspots analysis to identify the hotspots that took much time to execute.

To run an analysis:

1. From the VTune Amplifier XE toolbar, click the **New Analysis** button.

   VTune Amplifier XE Result tab opens with the **Analysis Type** window active.
2. On the left pane of the **Analysis Type** window, locate the analysis tree and select **Algorithm Analysis > Hotspots**.

   The right pane is updated with the default options for the Hotspots analysis.
3. Click the **Start** button on the right command bar to run the analysis.



Identify your most time-consuming source code. Unlike Lightweight Hotspots, Hotspots collects stack and call tree information. This analysis type cannot be used to profile t...

VTune Amplifier XE launches the `tachyon_find_hotspots` application that takes the `balls.dat` as input and renders an image displaying the execution time before exiting. VTune Amplifier XE finalizes the collected results and opens the **Hotspots** viewpoint.

To make sure the performance of the application is repeatable, go through the entire tuning process on the same system with a minimal amount of other software executing.

NOTE This tutorial explains how to run an analysis from the VTune Amplifier XE graphical user interface (GUI). You can also use the VTune Amplifier XE command-line interface (`amplxe-cl` command) to run an analysis. For more details, check the *Command-line Interface Support* section of the VTune Amplifier XE Help.

Key Terms

- Elapsed time
- Finalization
- Hotspot
- Hotspots analysis
- Viewpoint

Next Step

Interpret Result Data

Interpret Result Data

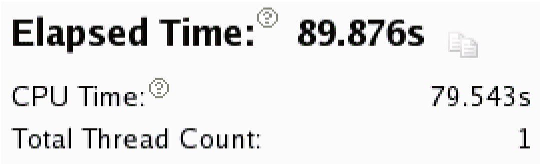
When the sample application exits, the Intel® VTune™ Amplifier XE finalizes the results and opens the **Hotspots** viewpoint where each window or pane is configured to display code regions that consumed a lot of CPU time. To interpret the data on the sample code performance, do the following:

- Understand the basic performance metrics provided by the Hotspots analysis.
- Analyze the most time-consuming functions.
- Analyze CPU usage per function.

NOTE The screenshots and execution time data provided in this tutorial are created on a system with four CPU cores. Your data may vary depending on the number and type of CPU cores on your system.

Understand the Basic Hotspots Metrics

Start analysis with the **Summary** window. To interpret the data, hover over the question mark icons ? to read the pop-up help and better understand what each performance metric means.

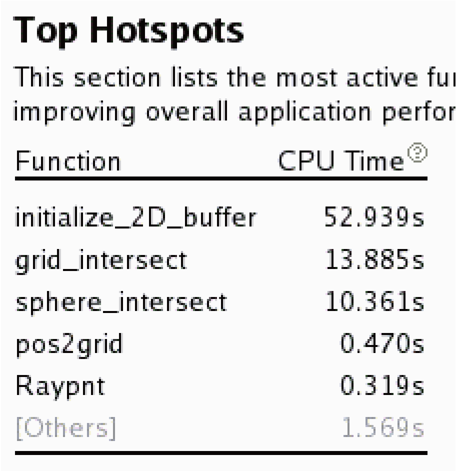


Elapsed Time: ? **89.876s**

CPU Time: ?	79.543s
Total Thread Count:	1

Note that **CPU Time** for the sample application is equal to 89.876 seconds. It is the sum of CPU time for all application threads. **Total Thread Count** is 1, so the sample application is single-threaded.

The **Top Hotspots** section provides data on the most time-consuming functions (*hotspot functions*) sorted by CPU time spent on their execution.



Top Hotspots

This section lists the most active fu
improving overall application perfor

Function	CPU Time ?
initialize_2D_buffer	52.939s
grid_intersect	13.885s
sphere_intersect	10.361s
pos2grid	0.470s
Raypnt	0.319s
[Others]	1.569s

For the sample application, the `initialize_2D_buffer` function, which took 52.939 seconds to execute, shows up at the top of the list as the hottest function.

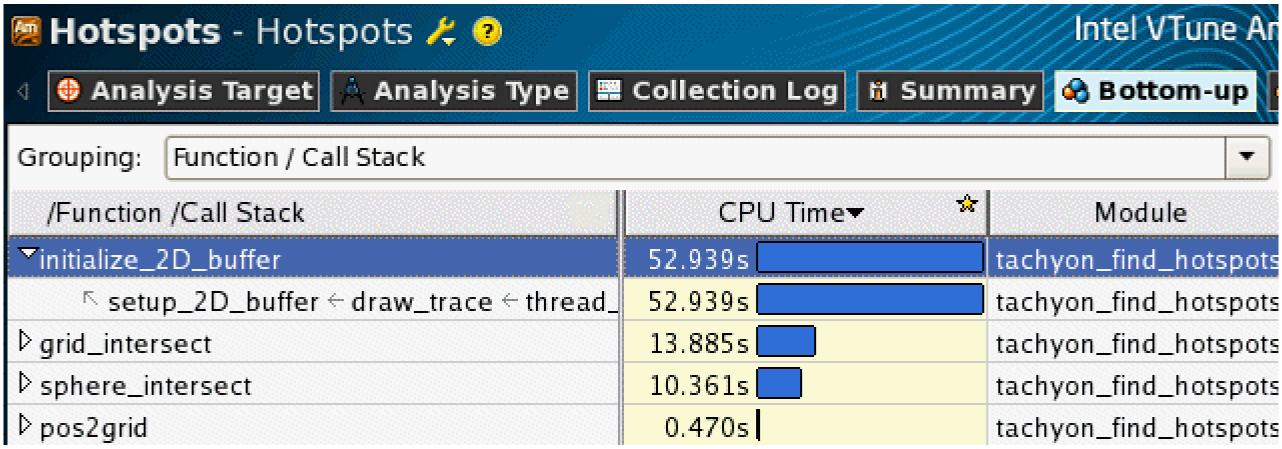
The `[Others]` entry at the bottom shows the sum of CPU time for all functions not listed in the table.

Analyze the Most Time-consuming Functions

Click the **Bottom-up** tab to explore the **Bottom-up** pane. By default, the data in the grid is sorted by Function. You may change the grouping level using the **Grouping** drop-down menu at the top of the grid.

Analyze the **CPU Time** column values. This column is marked with a yellow star as the Data of Interest column. It means that the VTune Amplifier XE uses this type of data for some calculations (for example, filtering, stack contribution, and others). Functions that took most CPU time to execute are listed on top.

The `initialize_2D_buffer` function took 52.939 seconds to execute. Click the arrow sign ▷ at the `initialize_2D_buffer` function to expand the stacks calling this function. You see that it was called only by the `setup_2D_buffer` function.



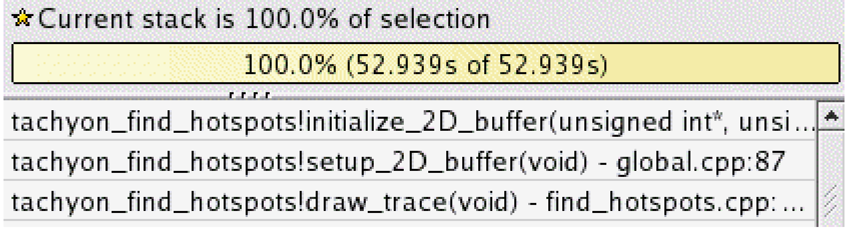
Grouping: Function / Call Stack

/Function /Call Stack	CPU Time▼	Module
▽initialize_2D_buffer	52.939s	tachyon_find_hotspots
↖ setup_2D_buffer ← draw_trace ← thread_	52.939s	tachyon_find_hotspots
▷ grid_intersect	13.885s	tachyon_find_hotspots
▷ sphere_intersect	10.361s	tachyon_find_hotspots
▷ pos2grid	0.470s	tachyon_find_hotspots

Select the `initialize_2D_buffer` function in the grid and explore the data provided in the **Call Stack** pane on the right.

The **Call Stack** pane displays full stack data for each hotspot function, enables you to navigate between function call stacks and understand the impact of each stack to the function CPU time. The stack functions in the **Call Stack** pane are represented in the following format:

<module>!<function> - <file>:<line number>, where the line number corresponds to the line calling the next function in the stack.



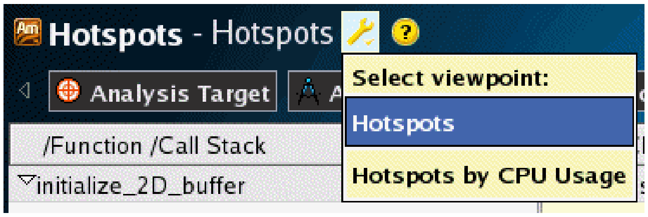
Current stack is 100.0% of selection

100.0% (52.939s of 52.939s)

```
tachyon_find_hotspots!initialize_2D_buffer(unsigned int*, unsi...
tachyon_find_hotspots!setup_2D_buffer(void) - global.cpp:87
tachyon_find_hotspots!draw_trace(void) - find_hotspots.cpp: ...
```

For the sample application, the hottest function is called at line 87 of the `setup_2D_buffer` function in the `global.cpp` file.

Analyze CPU Usage per Function

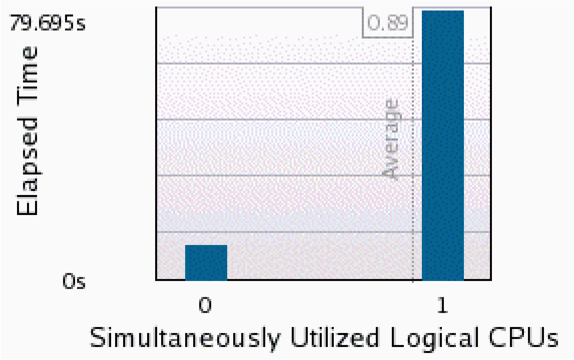


VTune Amplifier XE enables you to analyze the collected data from different perspectives by using multiple viewpoints.

For the Hotspots analysis result, you may switch to the **Hotspots by CPU Usage** viewpoint to understand how your hotspot function performs in terms of the CPU usage. Explore this viewpoint to determine how your application utilized available cores and identify the most serial code.

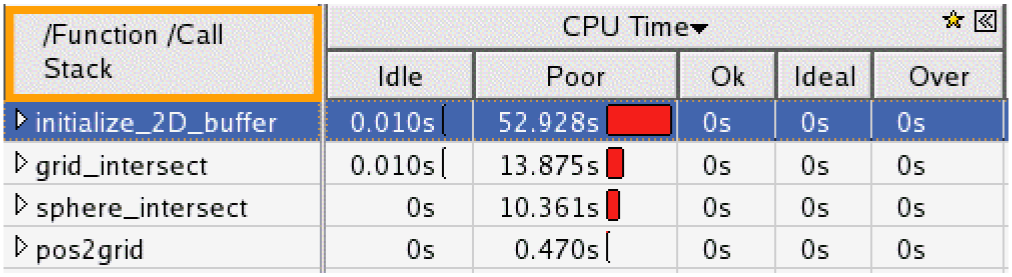
If you go back to the **Summary** window, you can see the **CPU Usage Histogram** that represents the Elapsed time and usage level for the available logical processors.

The `tachyon_find_hotspots` application ran mostly on one logical CPU. If you hover over the highest bar, you see that it spent 79.695 seconds using one core only, which is classified by the VTune Amplifier XE as a Poor utilization for a dual-core system. To understand what prevented the application from using all available logical CPUs effectively, explore the **Bottom-up** pane.



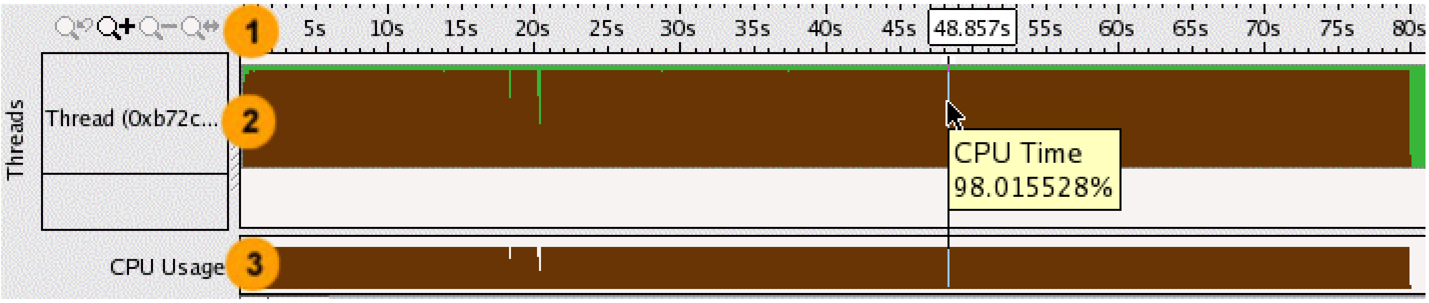
To get the detailed CPU usage information per function, use the ⧉ button in the **Bottom-up** window to expand the **CPU Time** column.

Note that `initialize_2D_buffer` is the function with the longest poor CPU utilization (red 🟥 bars). This means that the processor cores were underutilized most of the time spent on executing this function.



/Function /Call Stack	CPU Time				
	Idle	Poor	Ok	Ideal	Over
initialize_2D_buffer	0.010s	52.928s 🟥	0s	0s	0s
grid_intersect	0.010s	13.875s 🟥	0s	0s	0s
sphere_intersect	0s	10.361s 🟥	0s	0s	0s
pos2grid	0s	0.470s	0s	0s	0s

If you change the grouping level (highlighted in the figure above) in the Bottom-up pane from **Function/ Call Stack** to **Thread/Function/Call Stack**, you see that the `initialize_2D_buffer` function belongs to the `thread_video` thread. This thread is also identified as a hotspot and shows up at the top in the **Bottom-up** pane. To get detailed information on the hotspot thread performance, explore the **Timeline** pane.



1 **Timeline** area. When you hover over the graph element, the timeline tooltip displays the time passed since the application has been launched.

2 **Threads** area that shows the distribution of CPU time utilization per thread. Hover over a bar to see the CPU time utilization in percent for this thread at each moment of time. Green zones show the time threads are active.

3 **CPU Usage** area that shows the distribution of CPU time utilization for the whole application. Hover over a bar to see the application-level CPU time utilization in percent at each moment of time.

VTune Amplifier XE calculates the overall **CPU Usage** metric as the sum of CPU time per each thread of the **Threads** area. Maximum **CPU Usage** value is equal to `[number of processor cores] x 100%`.

The Timeline analysis also identifies the `thread_video` thread as the most active. The tooltip shows that CPU time values are about 100% whereas the maximum CPU time value for dual-core systems is 200%. This means that the processor cores were half-utilized for most of the time spent on executing the `tachyon_find_hotspots` application.

Key Terms

- CPU time
- CPU usage
- Elapsed time
- Hotspots analysis
- Viewpoint

Next Step

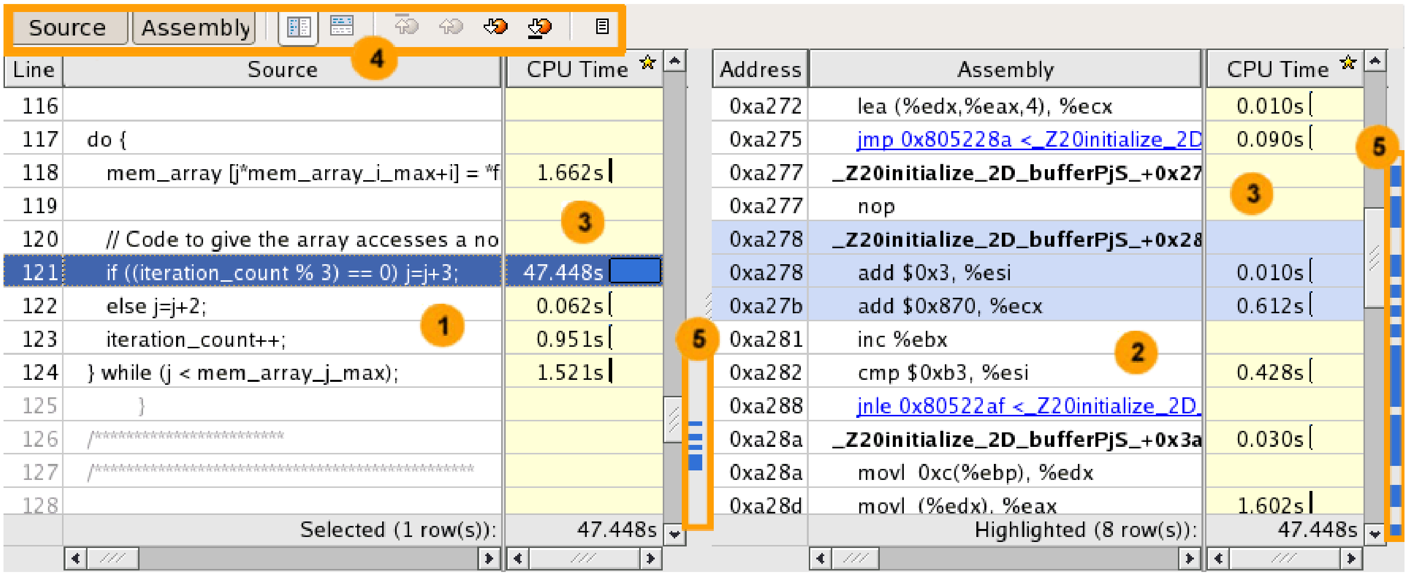
Analyze Code

Analyze Code

You identified `initialize_2D_buffer` as the hottest function. In the **Bottom-up** pane, double-click this function to open the **Source** window and analyze the source code:

1. Understand basic options provided in the **Source** window.
2. Identify the hottest code lines.

Understand Basic Source Window Options



Line	Source	CPU Time
116		
117	do {	
118	mem_array [j*mem_array_i_max+i] = *f	1.662s
119		
120	// Code to give the array accesses a no	
121	if ((iteration_count % 3) == 0) j=j+3;	47.448s
122	else j=j+2;	0.062s
123	iteration_count++;	0.951s
124	} while (j < mem_array_j_max);	1.521s
125	}	
126	/**************************	
127	/**	
128		
	Selected (1 row(s)):	47.448s

Address	Assembly	CPU Time
0xa272	lea (%edx,%eax,4), %ecx	0.010s
0xa275	jmp 0x805228a <_Z20initialize_2D	0.090s
0xa277	**_Z20initialize_2D_bufferPjS_+0x27**	
0xa277	nop	
0xa278	**_Z20initialize_2D_bufferPjS_+0x28**	
0xa278	add $0x3, %esi	0.010s
0xa27b	add $0x870, %ecx	0.612s
0xa281	inc %ebx	
0xa282	cmp $0xb3, %esi	0.428s
0xa288	jnle 0x80522af <_Z20initialize_2D_	
0xa28a	**_Z20initialize_2D_bufferPjS_+0x3a**	0.030s
0xa28a	movl 0xc(%ebp), %edx	
0xa28d	movl (%edx), %eax	1.602s
	Highlighted (8 row(s)):	47.448s

The table below explains some of the features available in the **Source** window when viewing the Hotspots analysis data.

1 **Source** pane displaying the source code of the application if the function symbol information is available. The beginning of the function is highlighted. The source code in the **Source** pane is not editable.

If the function symbol information is not available, the **Assembly** pane opens displaying assembler instructions for the selected hotspot function. To enable the **Source** pane, make sure to build the target properly.

2 Assembly pane displaying the assembler instructions for the selected hotspot function. Assembler instructions are grouped by basic blocks. The assembler instructions for the selected hotspot function are highlighted. To get help on an assembler instruction, right-click the instruction and select **Instruction Reference**.

NOTE To get the help on a particular instruction, make sure to have the Adobe* Acrobat Reader* 9 (or later) installed. If an earlier version of the Adobe Acrobat Reader is installed, the Instruction Reference opens but you need to locate the help on each instruction manually.

3 Processor time attributed to a particular code line. If the hotspot is a system function, its time, by default, is attributed to the user function that called this system function.

4 Source window toolbar. Use the hotspot navigation buttons to switch between most performance-critical code lines. Hotspot navigation is based on the metric column selected as a Data of Interest. For the Hotspots analysis, this is **CPU Time**. Use the **Source**/**Assembly** buttons to toggle the **Source**/**Assembly** panes (if both of them are available) on/off.

5 Heat map markers to quickly identify performance-critical code lines (hotspots). The bright blue markers indicate hot lines for the function you selected for analysis. Light blue markers indicate hot lines for other functions. Scroll to a marker to locate the hot code line it identifies.

Identify the Hottest Code Lines

When you identify a hotspot in the serial code, you can make some changes in the code to tune the algorithms and speed up that hotspot. Another option is to parallelize the sample code by adding threads to the application so that it performs well on multi-core processors. This tutorial focuses on algorithm tuning.

By default, when you double-click the hotspot in the **Bottom-up** pane, VTune Amplifier XE opens the source file positioning at the beginning of this function. Click the hotspot navigation button to go to the code line that took the most CPU time. For the `initialize_2D_buffer` function, the hottest code line is 121. This code is used to initialize a memory array using non-sequential memory locations. Click the **Source Editor** button on the **Source** window toolbar to open the default code editor and work on optimizing the code.

Key Terms

- CPU time
- Hotspot
- Hotspots analysis

Next Step

Tune Algorithms

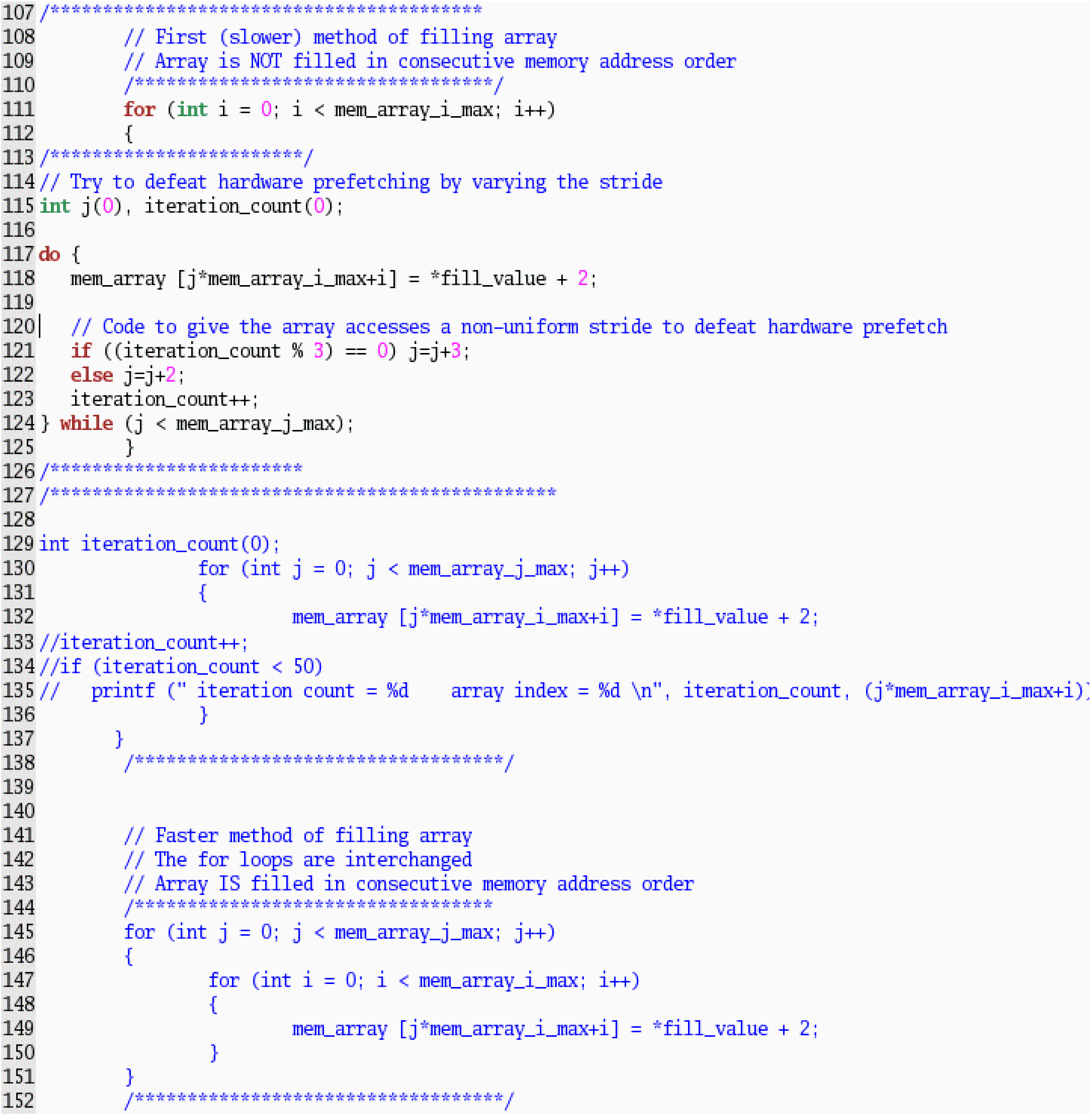
Tune Algorithms

In the **Source** window, you identified that in the `initialize_2D_buffer` hotspot function the code line 121 took the most CPU time. Focus on this line and do the following:

1. Open the code editor.
2. Optimize the algorithm used in this code section.

Open the Code Editor

In the **Source** window, click the ▤ **Source Editor** button to open the `initbuffer.cpp` file in the default code editor at the hotspot line:



```
/******************************************
        // First (slower) method of filling array
        // Array is NOT filled in consecutive memory address order
        /**********************************/
        for (int i = 0; i < mem_array_i_max; i++)
        {
/************************/
// Try to defeat hardware prefetching by varying the stride
int j(0), iteration_count(0);

do {
   mem_array [j*mem_array_i_max+i] = *fill_value + 2;

   // Code to give the array accesses a non-uniform stride to defeat hardware prefetch
   if ((iteration_count % 3) == 0) j=j+3;
   else j=j+2;
   iteration_count++;
} while (j < mem_array_j_max);
        }
/************************
/************************************************

int iteration_count(0);
               for (int j = 0; j < mem_array_j_max; j++)
               {
                        mem_array [j*mem_array_i_max+i] = *fill_value + 2;
//iteration_count++;
//if (iteration_count < 50)
//   printf (" iteration count = %d    array index = %d \n", iteration_count, (j*mem_array_i_max+i)
               }
       }
        /**********************************/


        // Faster method of filling array
        // The for loops are interchanged
        // Array IS filled in consecutive memory address order
        /**********************************
        for (int j = 0; j < mem_array_j_max; j++)
        {
                for (int i = 0; i < mem_array_i_max; i++)
                {
                        mem_array [j*mem_array_i_max+i] = *fill_value + 2;
                }
        }
        /**********************************/
```

Hotspot line is used to initialize a memory array using non-sequential memory locations. For demonstration purposes, the code lines are commented as a slower method of filling the array.

Resolve the Problem

To resolve this issue, optimize your algorithm as follows:

1. Edit lines 110 and 113 to comment out code lines 111-125 marked as a "First (slower) method".
2. Edit line 144 to uncomment code lines 145-151 marked as a "Faster method".

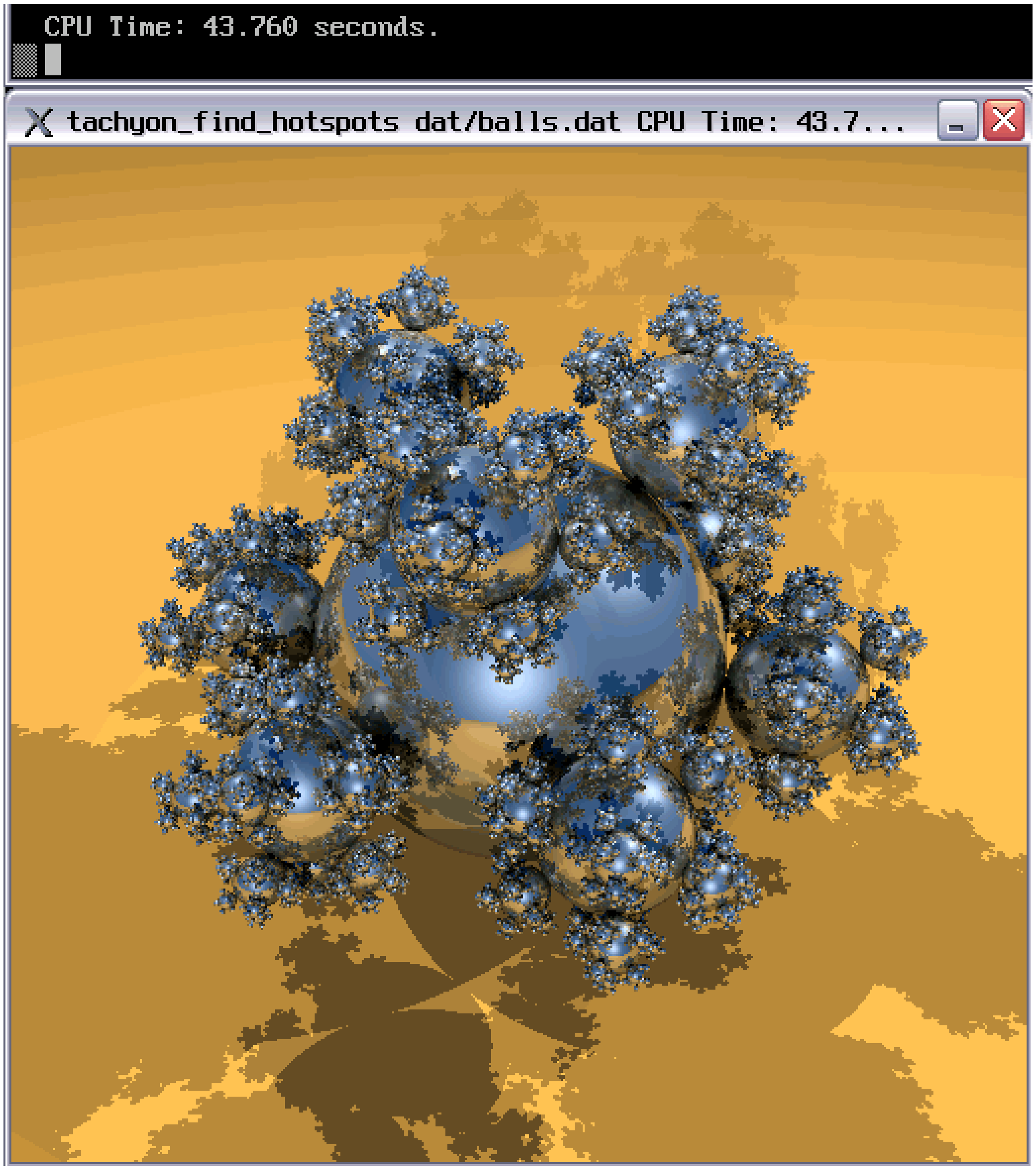
   In this step, you interchange the `for` loops to initialize the code in sequential memory locations.
3. Save the changes made in the source file.
4. Browse to the directory you extracted the sample code (for example, `/home/myuser/samples/tachyon`).
5. Rebuild your target using the `make` command as follows:

   ```
   $ make clean
   ```

   ```
   $ make
   ```

   The `tachyon_find_hotspots` application is rebuilt and stored in the `tachyon` directory.
6. Run `tachyon_find_hotspots` as follows:

   ```
   $ ./tachyon_find_hotspots dat/balls.dat
   ```



```
CPU Time: 43.760 seconds.
```

System runs the `tachyon_find_hotspots` application. Note that execution time reduced from 83.539 seconds to 43.760 seconds.

Key Terms

Hotspot

Next Step

Compare with Previous Result

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Compare with Previous Result

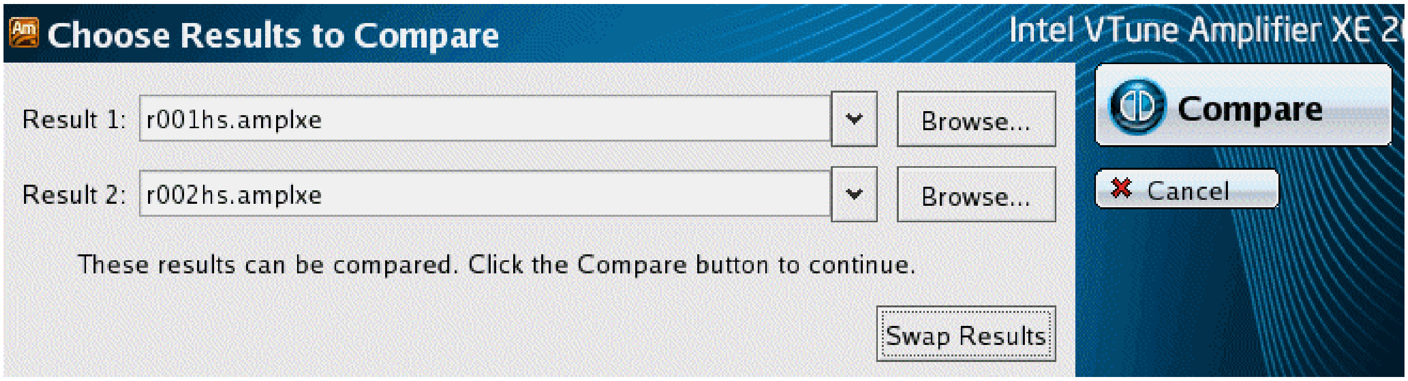
You optimized your code to apply a loop interchange mechanism that gave you 40 seconds of improvement in the application execution time. To understand whether you got rid of the hotspot and what kind of optimization you got per function, re-run the Hotspots analysis on the optimized code and compare results:

1. Compare results before and after optimization.
2. Identify the performance gain.

Compare Results Before and After Optimization

1. Run the Hotspots analysis on the modified code.
2. Click the **Compare Results** button on the Intel® VTune™ Amplifier XE toolbar.

   The **Compare Results** window opens.
3. Specify the Hotspots analysis results you want to compare and click the **Compare Results** button.

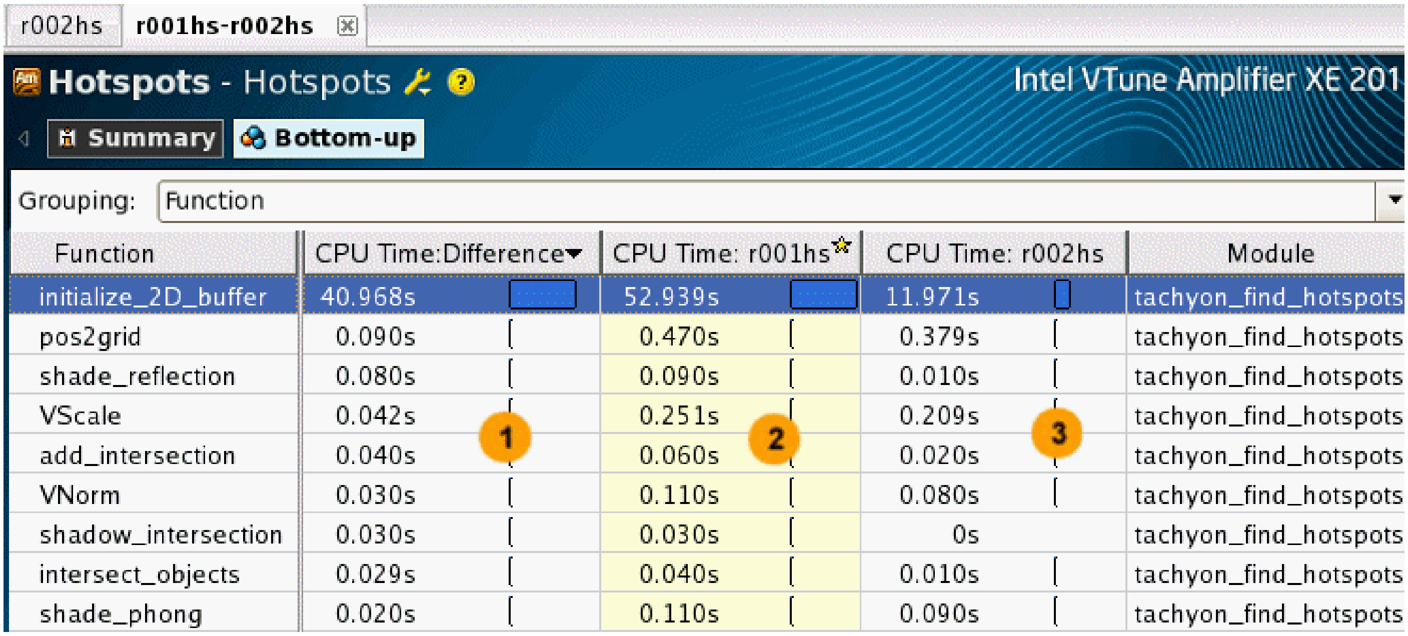


Result 1: r001hs.amplxe

Result 2: r002hs.amplxe

These results can be compared. Click the Compare button to continue.

The Hotspots **Bottom-up** window opens, showing the CPU time usage across the two results and the differences side by side.



Grouping: Function

Function	CPU Time:Difference	CPU Time: r001hs	CPU Time: r002hs	Module
initialize_2D_buffer	40.968s	52.939s	11.971s	tachyon_find_hotspots
pos2grid	0.090s	0.470s	0.379s	tachyon_find_hotspots
shade_reflection	0.080s	0.090s	0.010s	tachyon_find_hotspots
VScale	0.042s	0.251s	0.209s	tachyon_find_hotspots
add_intersection	0.040s	0.060s	0.020s	tachyon_find_hotspots
VNorm	0.030s	0.110s	0.080s	tachyon_find_hotspots
shadow_intersection	0.030s	0.030s	0s	tachyon_find_hotspots
intersect_objects	0.029s	0.040s	0.010s	tachyon_find_hotspots
shade_phong	0.020s	0.110s	0.090s	tachyon_find_hotspots

1 Difference in CPU time between the two results in the following format: <Difference CPU Time> = <Result 1 CPU Time> − <Result 2 CPU Time>.

2 CPU time for the initial version of the `tachyon_find_hotspots` application.

3 CPU time for the optimized version of the `tachyon_find_hotspots`.

Identify the Performance Gain

Explore the **Bottom-up** pane to compare CPU time data for the first hotspot: CPU Time:r001hs - CPU Time:r002hs = CPU Time: Difference. 52.939s - 11.971s = 40.968s, which means that you got the optimization of ~41 seconds for the `initialize_2D_buffer` function.

Key Terms

- CPU time
- Elapsed time
- Hotspot
- Hotspots analysis

Summary 3

You have completed the Finding Hotspots tutorial. Here are some important things to remember when using the Intel® VTune™ Amplifier XE to analyze your code for hotspots:

Step	Tutorial Recap	Key Tutorial Take-aways
1. Prepare for analysis	You built the target in the Release mode, created the performance baseline, and created the VTune Amplifier XE project for your analysis target.	• Create a performance baseline to compare the application versions before and after optimization. Make sure to use the same workload for each application run. • Create a VTune Amplifier XE project and u se the **Project Properties: Target** tab to choose and configure your analysis target. • Use the **Analysis Type** configuration window to choose, configure, and run the analysis. You can also run the analysis from command line using the `amplxe-cl` command.
2. Find hotspots	You launched the Hotspots data collection that analyzes function calls and CPU time spent in each program unit of your application and identified the following hotspots: • Identified a function that took the most CPU time and could be a good candidate for algorithm tuning. • Identified the code section that took the most CPU time to execute.	• Start analyzing the performance of your application from the **Summary** window to explore the performance metrics for the whole application. Then, move to the **Bottom-up** window to analyze the performance per function. Focus on the *hotspots* - functions that took the most CPU time. By default, they are located at the top of the table. • Double-click the hotspot function in the **Bottom-up** pane or **Call Stack** pane to open its source code and identify the code line that took the most CPU time.
3. Eliminate hotspots	You interchanged the loops in the hotspot function, rebuilt the application, and got performance gain of 40 seconds.	Use the **Source Editor** button to open the code editor from the VTune Amplifier XE directly at the hotspot line.
4. Check your work	You ran the Hotspots analysis on the optimized code and compared the results before and after optimization using the Compare mode of the VTune Amplifier XE. Compare analysis results regularly to look for regressions and to track how incremental changes to the code affect its performance. You may also want to use the VTune Amplifier XE command-line interface and run the	Perform regular regression testing by comparing analysis results before and after optimization. From GUI, click the **Compare Results** button on the VTune Amplifier XE toolbar. From command line, use the `amplxe-cl` command.

Step	Tutorial Recap	Key Tutorial Take-aways
	`amplxe-cl` command to test your code for regressions. For more details, see the *Command-line Interface Support* section in the VTune Amplifier XE online help.	

Next step: Prepare your own application(s) for analysis. Then use the VTune Amplifier XE to find and eliminate hotspots.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

4

Key Terms

baseline: A performance metric used as a basis for comparison of the application versions before and after optimization. Baseline should be measurable and reproducible.

CPU time: The amount of time a thread spends executing on a logical processor. For multiple threads, the CPU time of the threads is summed. The application CPU time is the sum of the CPU time of all the threads that run the application.

CPU usage: A performance metric when the VTune Amplifier XE identifies a processor utilization scale, calculates the target CPU usage, and defines default utilization ranges depending on the number of processor cores.

Utilization Type	Default color	Description
Idle	▨	All CPUs are waiting - no threads are running.
Poor	🟥	Poor usage. By default, poor usage is when the number of simultaneously running CPUs is less than or equal to 50% of the target CPU usage.
OK	🟧	Acceptable (OK) usage. By default, OK usage is when the number of simultaneously running CPUs is between 51-85% of the target CPU usage.
Ideal	🟩	Ideal usage. By default, Ideal usage is when the number of simultaneously running CPUs is between 86-100% of the target CPU usage.

Elapsed time:The total time your target ran, calculated as follows: **Wall clock time at end of application – Wall clock time at start of application**.

finalization: A process during which the Intel® VTune™ Amplifier XE converts the collected data to a database, resolves symbol information, and pre-computes data to make further analysis more efficient and responsive.

hotspot: A section of code that took a long time to execute. Some hotspots may indicate bottlenecks and can be removed, while other hotspots inevitably take a long time to execute due to their nature.

hotspots analysis: An analysis type used to understand the application flow and identify hotspots. VTune Amplifier XE creates a list of functions in your application ordered by the amount of time spent in a function. It also detects the call stacks for each of these functions so you can see how the hot functions are called. VTune Amplifier XE uses a low overhead (about 5%) user-mode sampling and tracing collection that gets you the information you need without slowing down the application execution significantly.

target: A *target* is an executable file you analyze using the Intel® VTune™ Amplifier XE.

viewpoint: A preset result tab configuration that filters out the data collected during a performance analysis and enables you to focus on specific performance problems. When you select a viewpoint, you select a set of performance metrics the VTune Amplifier XE shows in the windows/panes of the result tab. To select the required viewpoint, click the 🔧 button and use the drop-down menu at the top of the result tab.