Getting Started Tutorial: Analyzing Locks and Waits

Intel® VTune™ Amplifier XE 2011 for Windows* OS

C++ Sample Application Code

Document Number: 326706-001

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 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 XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, 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.

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Overview

Discover how to use the 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.

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

About This Tutorial	This tutorial uses the sample `tachyon_find_hotspots` application 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/

Navigation Quick Start

1

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 the algorithm choices and identify where and how your application can benefit from available hardware resources.

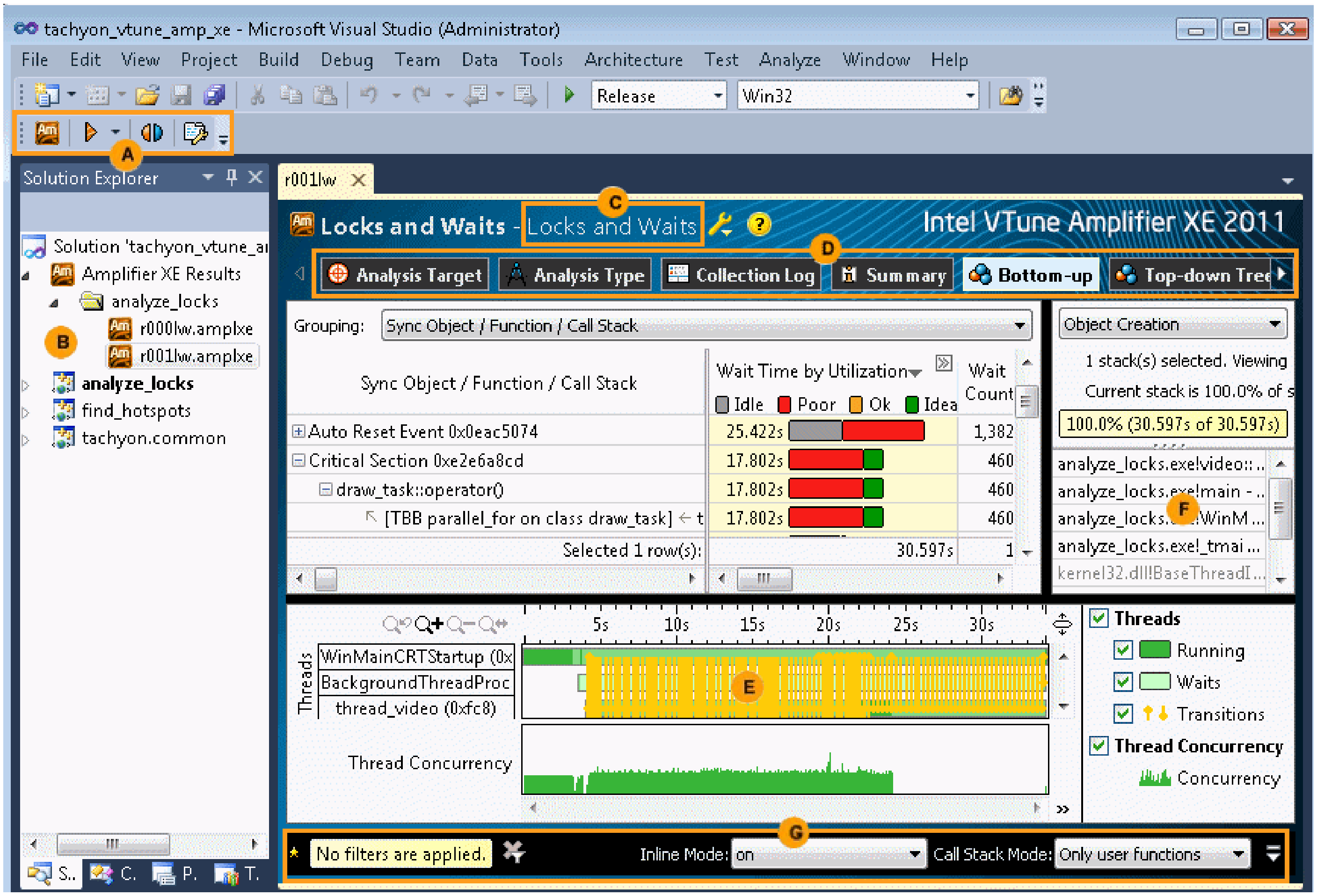
VTune Amplifier XE Access

To access the VTune Amplifier XE in the Visual Studio* IDE: From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Parallel Studio XE 2011 with [VS2008 | VS2010]**.

To access the Standalone VTune Amplifier XE GUI, do one of the following:

- From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Intel VTune Amplifier XE 2011**.
- From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2011** > **Command Prompt** > **Parallel Studio XE** > **IA-32 Visual Studio [2008 | 2010] mode** to set your environment, then type `amplxe-gui`.

VTune Amplifier XE/Visual Studio* IDE Integration



tachyon_vtune_amp_xe - Microsoft Visual Studio (Administrator)

Release | Win32

Grouping: Sync Object / Function / Call Stack

Sync Object / Function / Call Stack	Wait Time by Utilization ▨ Idle 🟥 Poor 🟧 Ok 🟩 Idea	Wait Count
⊞ Auto Reset Event 0x0eac5074	25.422s	1,382
⊟ Critical Section 0xe2e6a8cd	17.802s	460
⊟ draw_task::operator()	17.802s	460
[TBB parallel_for on class draw_task] ← t	17.802s	460
Selected 1 row(s):	30.597s	1

Object Creation

1 stack(s) selected. Viewing

Current stack is 100.0% of s

100.0% (30.597s of 30.597s)

analyze_locks.exe!video::...
analyze_locks.exe!main - ...
analyze_locks.exe!WinM...
analyze_locks.exe!_tmai...
kernel32.dll!BaseThreadI...

WinMainCRTStartup (0x
BackgroundThreadProc
thread_video (0xfc8)

No filters are applied.

Inline Mode: on

Call Stack Mode: Only user functions

A Use the VTune Amplifier XE toolbar to configure and control result collection.

B VTune Amplifier XE results `*.amplxe` show up in the **Solution Explorer** under the **My Amplifier XE Results** folder. To configure and control result collection, right-click the project in the **Solution Explorer** and select the **Intel VTune Amplifier XE 2011** menu from the pop-up menu. To manage previously collected results, right-click the result (for example, `r002hs.amplxe`) and select the required command from the pop-up menu.

C 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 preset configurations to focus on particular performance metrics.

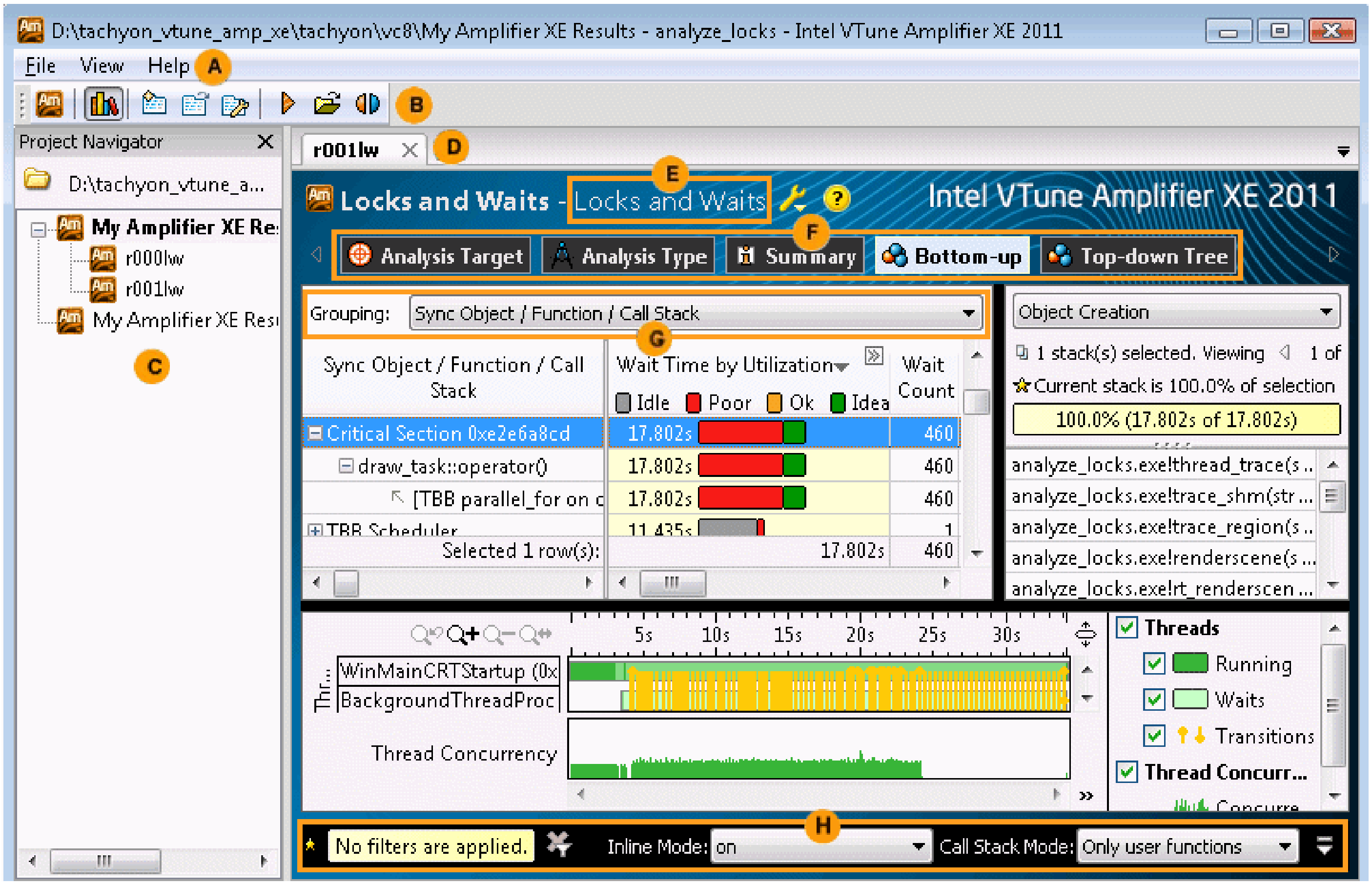
D Click the buttons on navigation toolbars to change window views and toggle window panes on and off.

E In the Timeline pane, analyze the thread activity and transitions presented for the user-mode sampling and tracing analysis results (for example, Hotspots, Concurrency, Locks and Waits) or analyze the distribution of the application performance per metric over time for the event-based sampling analysis results (for example, Memory Access, Bandwidth Breakdown).

F Use the Call Stack pane to view call paths for a function selected in the grid.

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

Standalone VTune Amplifier XE GUI



Grouping: Sync Object / Function / Call Stack

Sync Object / Function / Call Stack	Wait Time by Utilization ▨ Idle 🟥 Poor 🟧 Ok 🟩 Idea	Wait Count
⊟ Critical Section 0xe2e6a8cd	17.802s	460
⊟ draw_task::operator()	17.802s	460
[TBB parallel_for on c	17.802s	460
⊞ TBB Scheduler	11.435s	1
Selected 1 row(s):	17.802s	460

analyze_locks.exe!thread_trace(s...
analyze_locks.exe!trace_shm(str...
analyze_locks.exe!trace_region(s...
analyze_locks.exe!renderscene(s...
analyze_locks.exe!rt_renderscen...

No filters are applied. Inline Mode: on 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 Use the VTune Amplifier XE toolbar to configure and control result collection.

C Use the Project Navigator to manage your VTune Amplifier XE projects and collected analysis results. Click the **Project Navigator** button on the toolbar to enable/disable the Project Navigator.

D Use the VTune Amplifier XE result tabs to manage result data. You can view or change the result file location from the **Project Properties** dialog box.

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 preset configurations 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.

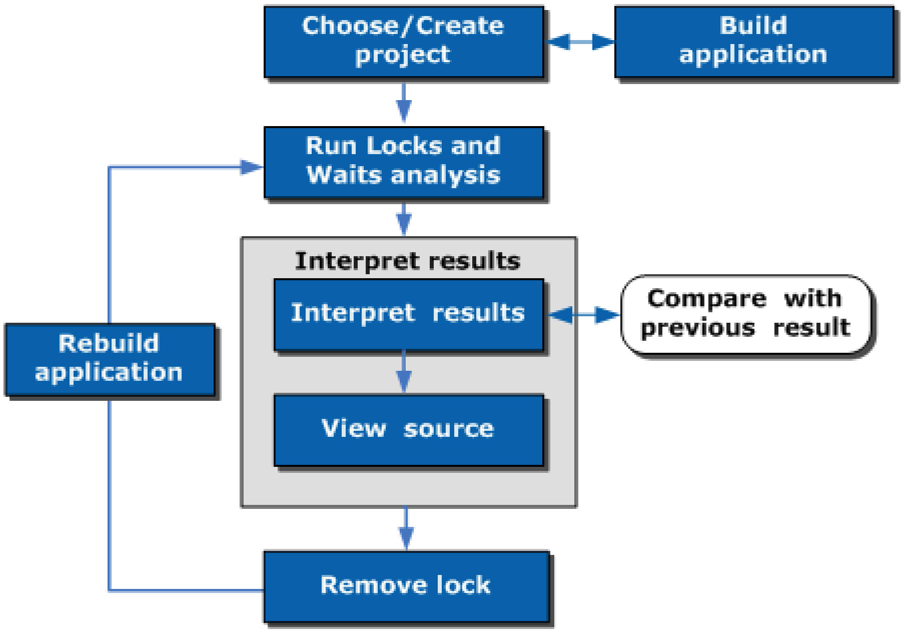
G Use the **Grouping** drop-down menu to choose a granularity level for grouping data in the grid.

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

Analyzing Locks and Waits

2

You can use the Intel® VTune™ Amplifier XE to understand the cause of the ineffective processor utilization 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	Do one of the following: • Visual Studio* IDE: Choose a project, verify settings, and build application. • Standalone GUI: Build an application to analyze for locks and waits and create a new VTune Amplifier XE project.
Step 2. Find lock	• Choose and run the Locks and Waits analysis. • Interpret the result data. • View and analyze code of the performance-critical function.
Step 3. Remove lock	Modify the code to remove the lock.
Step 4. Check your work	Re-build the target, re-run the Locks and Waits analysis, and compare the result data before and after optimization.

Visual Studio* IDE: Choose Project and Build Application

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

1. Get software tools.
2. Choose a project.
3. Configure the Microsoft* symbol server.

4. Verify optimal compiler/linker options.
5. Build the target in the release mode.
6. Create a performance baseline.

- The steps below are provided for Microsoft Visual Studio* 2010. Steps for other versions of Visual Studio IDE may slightly differ. See online help for details.
- Steps provided by this tutorial are generic and applicable to any application. You may choose to follow the proposed workflow using your own application.

Get Software Tools

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

- VTune Amplifier XE, including sample applications
- `.zip` file extraction utility
- Supported compiler (see Release Notes for more information)

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.zip` file from the `<install-dir>\samples\<locale>\C++\` directory to a writable directory or share on your system. The default installation path is `C:\Program Files\Intel\VTune Amplifier XE 2011\C:\Program Files\Intel\VTune Amplifier XE 2011\` (on certain systems, instead of `Program Files`, the directory name is `Program Files (x86)`).
2. Extract the sample from the `.zip` file.

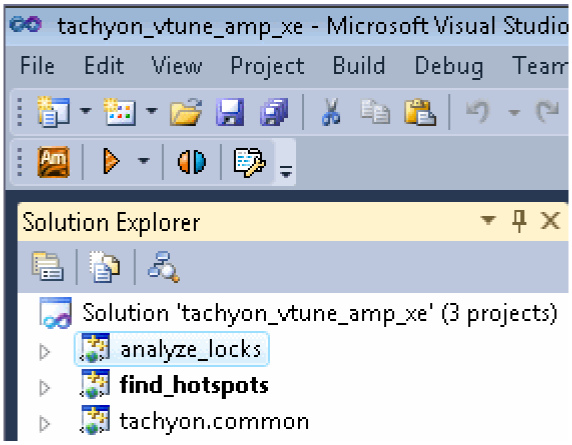
- Samples are non-deterministic. Your screens may vary from the screen captures shown throughout this tutorial.
- Samples are designed only to illustrate the VTune Amplifier XE features; they do not represent best practices for creating code.

Choose a Project

Choose a project with the analysis target in the Visual Studio IDE as follows:

1. From the Visual Studio menu, select **File > Open > Project/Solution...**.

   The **Open Project** dialog box opens.
2. In the **Open Project** dialog box, browse to the location you used to unzip the `tachyon_vtune_amp_xe.zip` file and select the `tachyon_vtune_amp_xe.sln` file.


tachyon_vtune_amp_xe - Microsoft Visual Studio
File
Edit
View
Project
Build
Debug
Team
Solution Explorer
Solution 'tachyon_vtune_amp_xe' (3 projects)
analyze_locks
find_hotspots
tachyon.common

The solution is added to Visual Studio and shows up in the Solution Explorer.

3. In Solution Explorer, right-click the **analyze_locks** project and select **Project > Set as StartUp Project**.

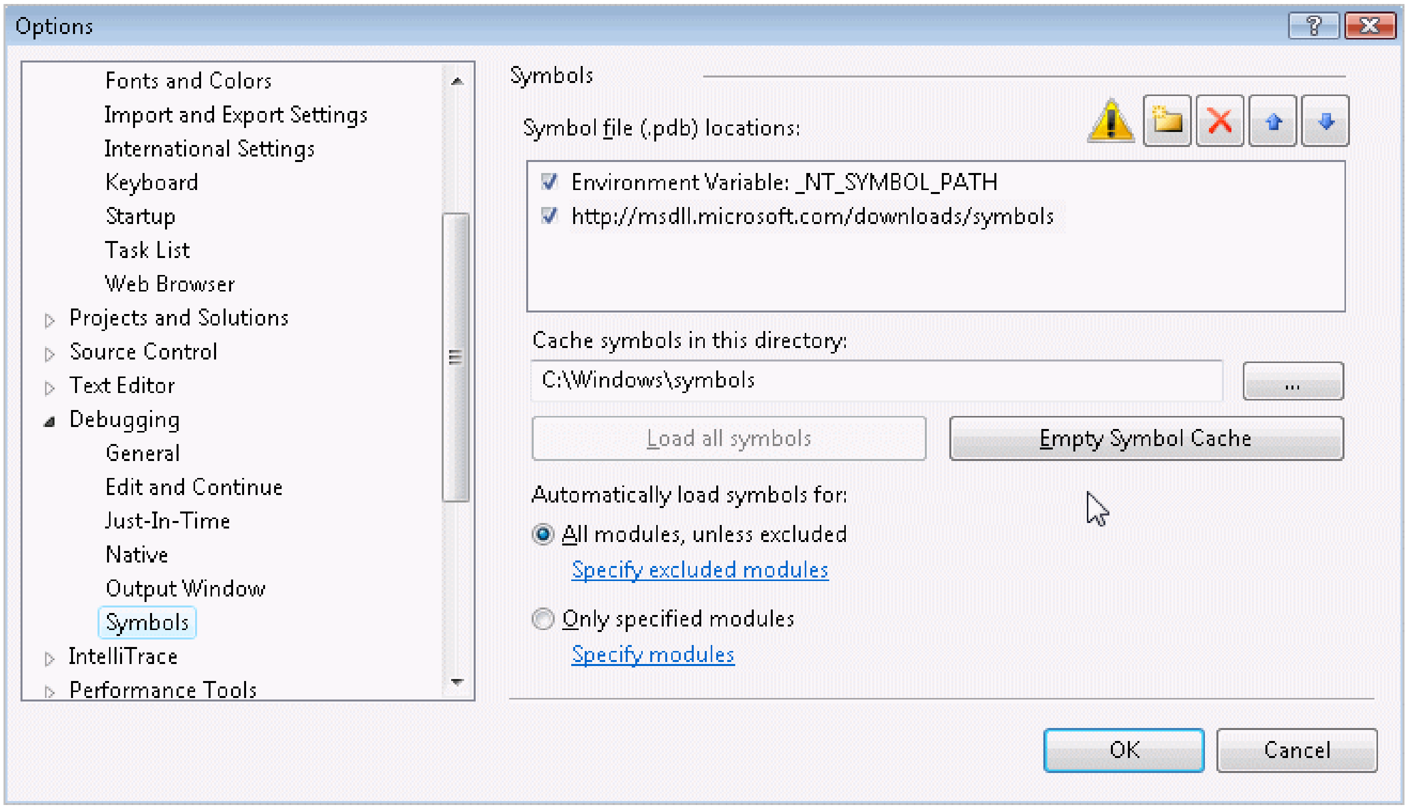
   analyze_locks appears in bold in Solution Explorer.

Configure the Microsoft* Symbol Server

Configure the Visual Studio environment to download the debug information for system libraries so that the VTune Amplifier XE can properly identify system functions and classify/attribute functions.

1. Go to **Tools > Options...**.

   The **Options** dialog box opens.
2. From the left pane, select **Debugging > Symbols**.
3. In the **Symbol file (.pdb) locations** field, click the 📁 button and specify the following address: http://msdl.microsoft.com/download/symbols.
4. Make sure the added address is checked.
5. In the **Cache symbols in this directory** field, specify a directory where the downloaded symbol files will be stored.



Symbol file (.pdb) locations:

☑ Environment Variable: _NT_SYMBOL_PATH

☑ http://msdll.microsoft.com/downloads/symbols

Cache symbols in this directory:

C:\Windows\symbols

Automatically load symbols for:

● All modules, unless excluded

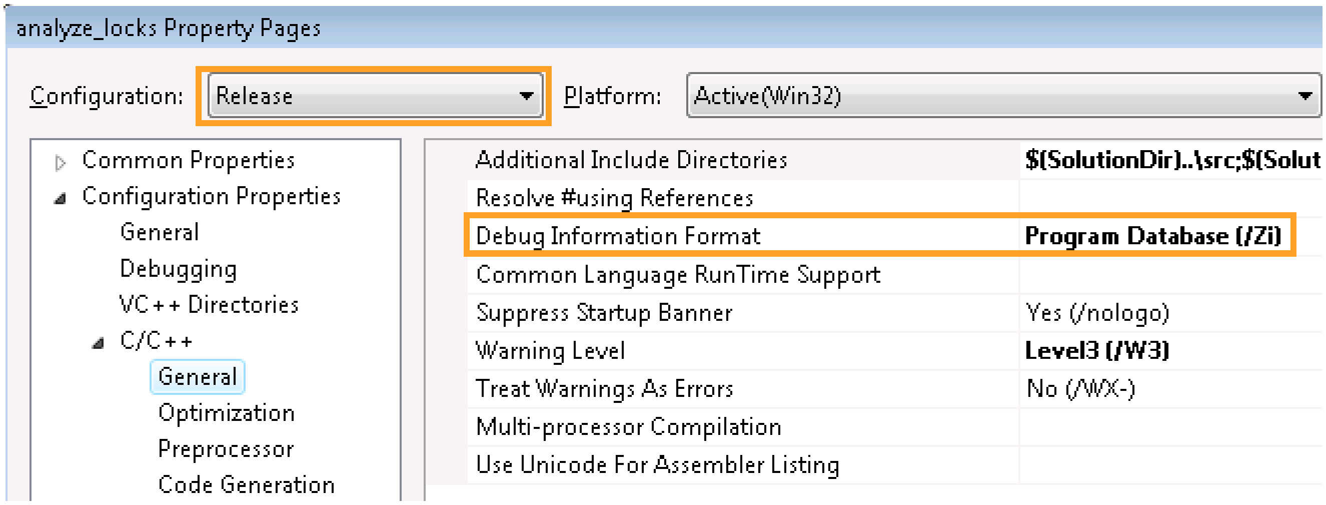
○ Only specified modules

6. Click **Ok**.

Verify Optimal Compiler/Linker Options

Configure Visual Studio project properties to generate the debug information for your application so that the VTune Amplifier XE can open the source code.

1. Select the **analyze_locks** project and go to **Project > Properties**.
2. From the **analyze_locks Property Pages** dialog box, select **Configuration Properties > General** and make sure the selected **Configuration** (top of the dialog) is **Release**.
3. From the **analyze_locks Property Pages** dialog box, select **C/C++ > General** pane and specify the **Debug Information Format** as **Program Database (/Zi)**.



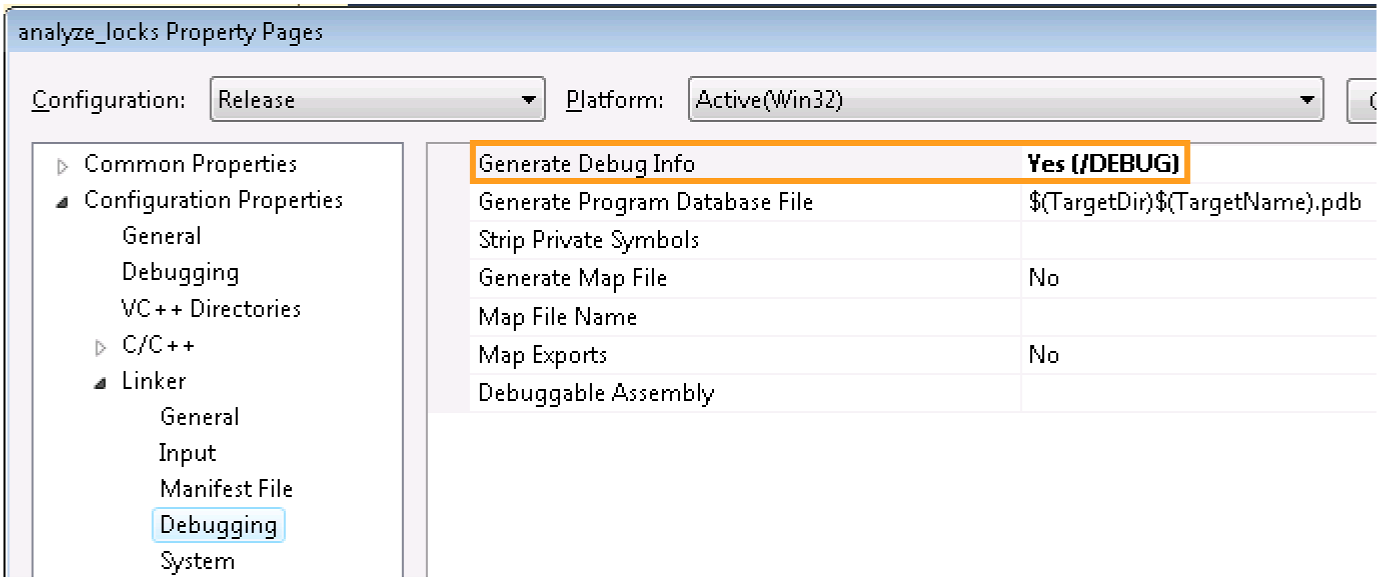
analyze_locks Property Pages

Configuration: Release

Platform: Active(Win32)

Additional Include Directories	$(SolutionDir)..\src;$(Solut
Resolve #using References	
Debug Information Format	Program Database (/Zi)
Common Language RunTime Support	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Multi-processor Compilation	
Use Unicode For Assembler Listing	

4. From the **analyze_locks Property Pages** dialog box, select **Linker > Debugging** and set the **Generate Debug Info** option to **Yes (/DEBUG)**.



analyze_locks Property Pages

Configuration: Release

Platform: Active(Win32)

Generate Debug Info	Yes (/DEBUG)
Generate Program Database File	$(TargetDir)$(TargetName).pdb
Strip Private Symbols	
Generate Map File	No
Map File Name	
Map Exports	No
Debuggable Assembly	

Build the Target in the Release Mode

Configure Visual Studio project properties to generate the debug information for your application so that the VTune Amplifier XE can open the source code.

1. Go to the **Build > Configuration Manager...** dialog box and select the **Release** mode for your target project.
2. From the Visual Studio menu, select **Build > Build analyze_locks**.

   The `tachyon_analyze_locks` application is built.

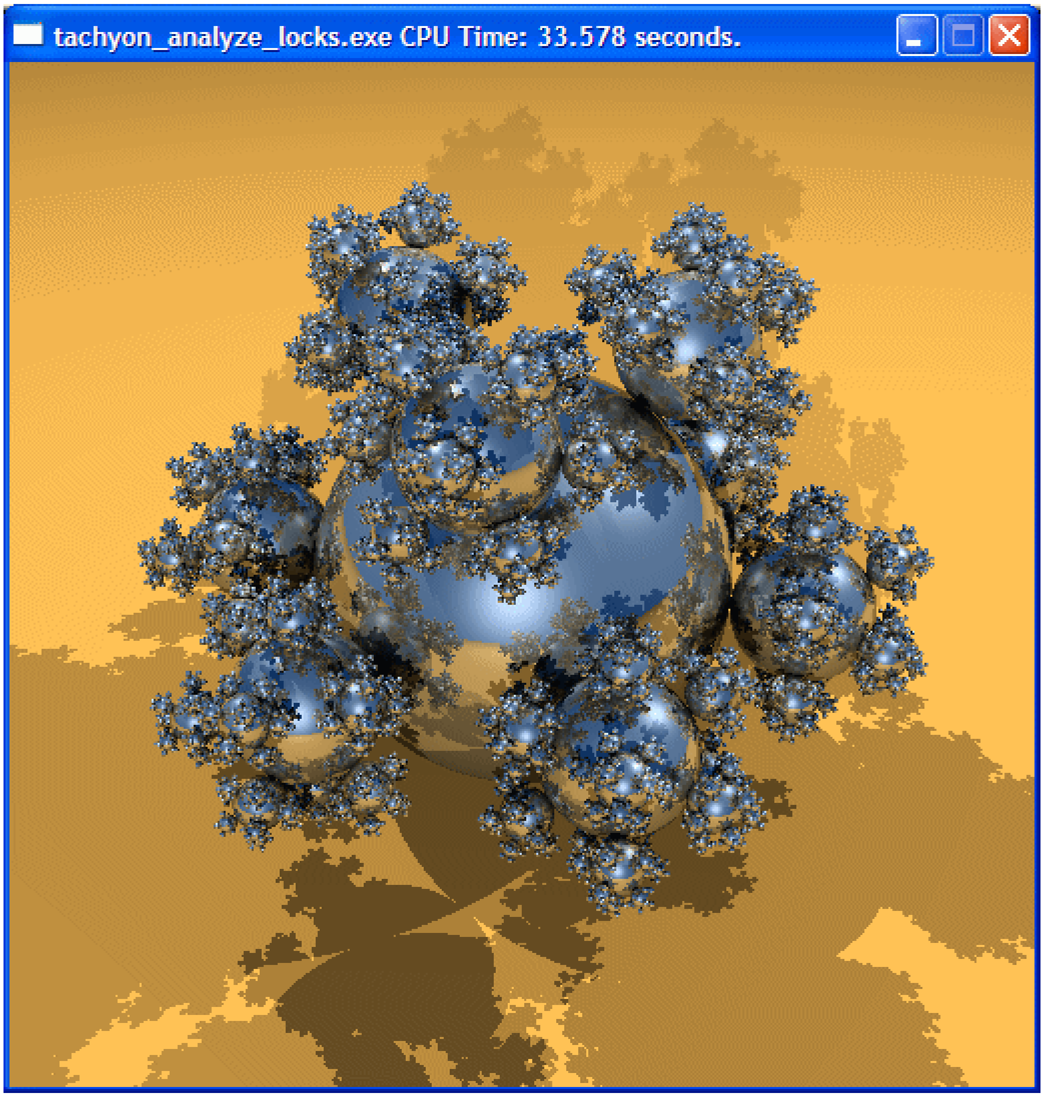
NOTE The build configuration for `tachyon` may initially be set to Debug, which is typically used for development. When analyzing performance issues with the VTune Amplifier XE, you are recommended to use the Release build with normal optimizations. In this way, the VTune Amplifier XE is able to analyze the realistic performance of your application.

Create a Performance Baseline

1. From the Visual Studio menu, select **Debug > Start Without Debugging**.

The `tachyon_analyze_locks` application runs in multiple sections (depending on the number of CPUs in your system).

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



tachyon_analyze_locks.exe CPU Time: 33.578 seconds.

2. Note the execution time displayed in the window caption. For the `tachyon_analyze_locks` executable in the figure above, the execution time is 33.578 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.

Key Terms

Target

Next Step

Run Locks and Waits Analysis

Standalone GUI: Build Application and Create New Project

Before you start analyzing your application for locks and waits, do the following:

1. Get software tools.
2. Build application.

   If you build the code in Visual Studio*, make sure to:

   - Configure the Microsoft* symbol server.
   - Verify optimal compiler/linker options.
   - Build the target in the release mode.
3. 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
- `.zip` file extraction utility
- Supported compiler (see Release Notes for more information)

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.zip` file from the `<install-dir>\samples\<locale>\C++\` directory to a writable directory or share on your system. The default installation path is `C:\Program Files\Intel\VTune Amplifier XE 2011\` (on certain systems, instead of `Program Files`, the directory name is `Program Files (x86)`).
2. Extract the sample from the `.zip` 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.

Configure the Microsoft* Symbol Server

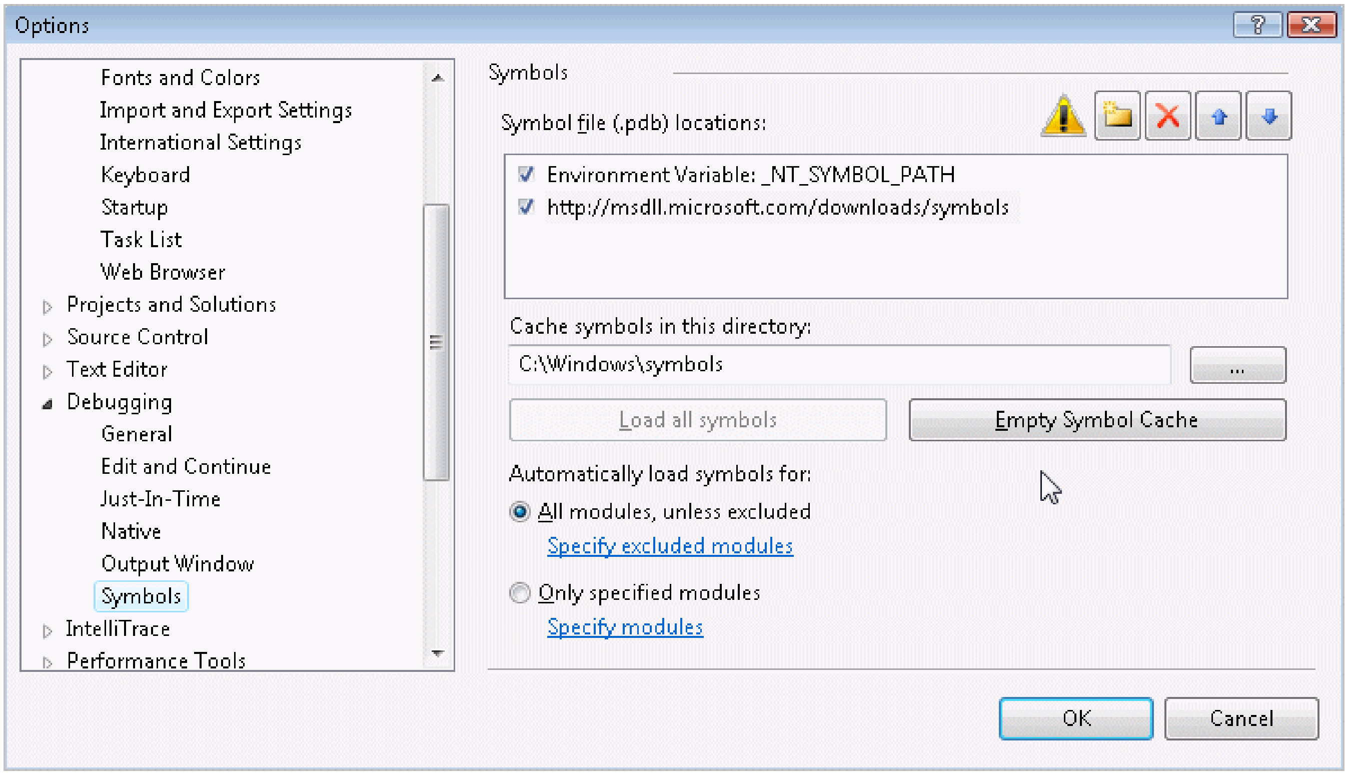
Configure the Visual Studio environment to download the debug information for system libraries so that the VTune Amplifier XE can properly identify system functions and classify/attribute functions.

NOTE The steps below are provided for Microsoft Visual Studio* 2010. Steps for other versions of Visual Studio IDE may differ slightly.

1. Go to **Tools > Options...**.

   The **Options** dialog box opens.
2. From the left pane, select **Debugging > Symbols**.
3. In the **Symbol file (.pdb) locations** field, click the 📁 button and specify the following address: http://msdl.microsoft.com/download/symbols.
4. Make sure the added address is checked.

5. In the **Cache symbols in this directory** field, specify a directory where the downloaded symbol files will be stored.



Symbol file (.pdb) locations:

☑ Environment Variable: _NT_SYMBOL_PATH
☑ http://msdll.microsoft.com/downloads/symbols

Cache symbols in this directory: C:\Windows\symbols

Automatically load symbols for:

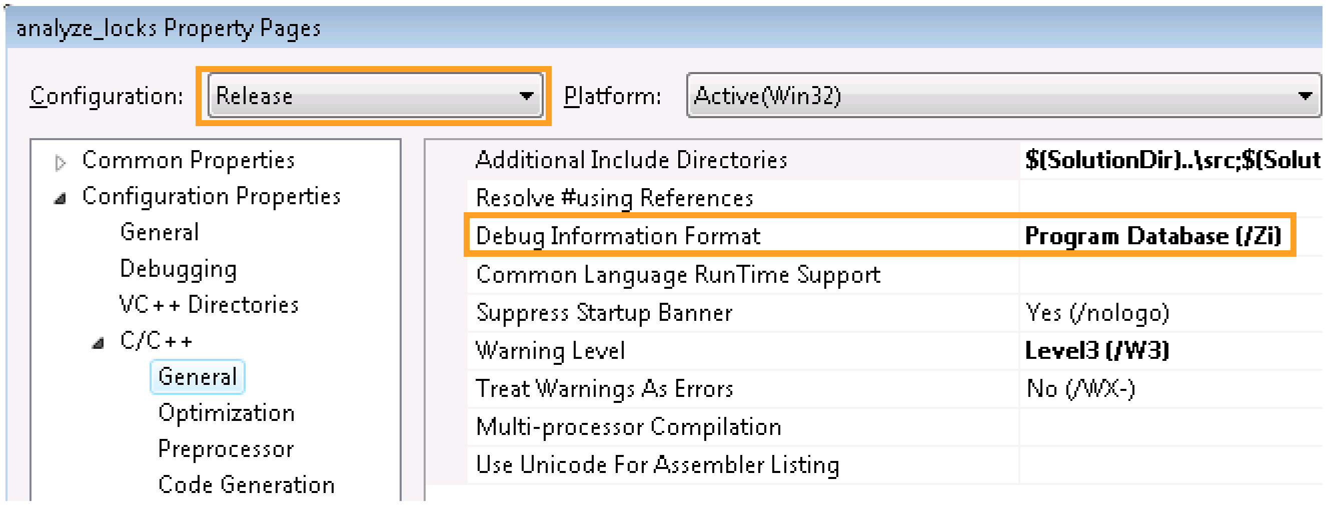
● All modules, unless excluded
○ Only specified modules

6. Click **Ok**.

Verify Optimal Compiler/Linker Options

Configure Visual Studio project properties to generate the debug information for your application so that the VTune Amplifier XE can open the source code.

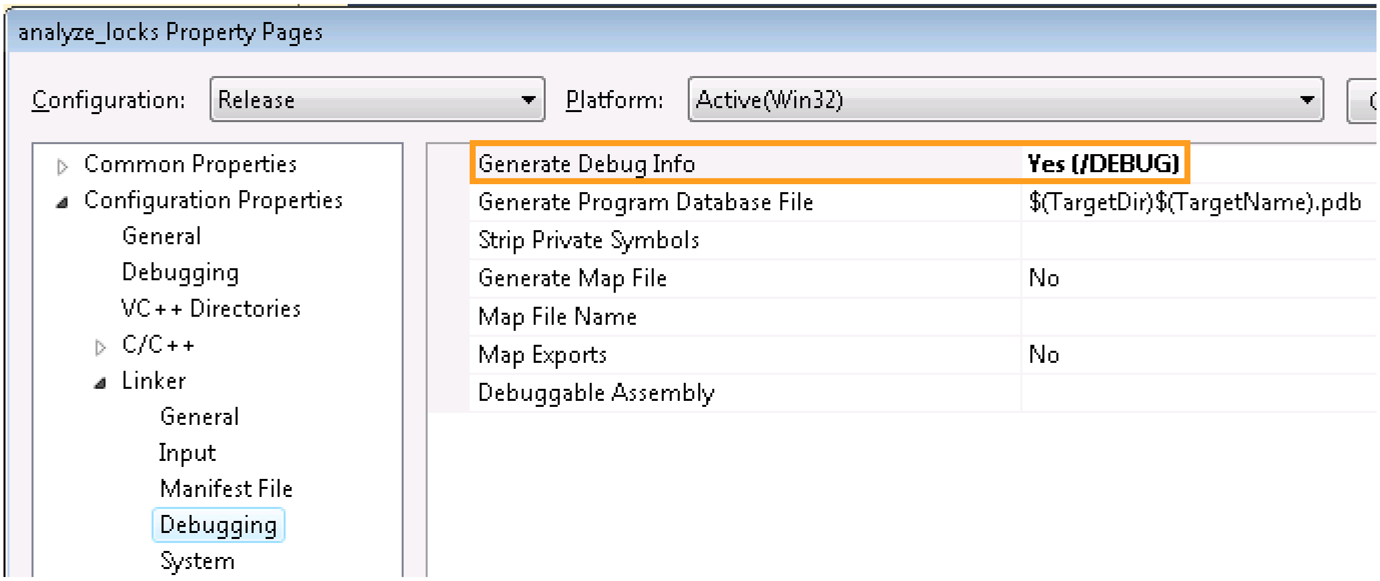
1. Select the **analyze_locks** project and go to **Project > Properties**.
2. From the **analyze_locks Property Pages** dialog box, select **Configuration Properties > General** and make sure the selected **Configuration** (top of the dialog) is **Release**.
3. From the **analyze_locks Property Pages** dialog box, select **C/C++ > General** pane and specify the **Debug Information Format** as **Program Database (/Zi)**.



Configuration: Release
Platform: Active(Win32)

Additional Include Directories	$(SolutionDir)..\src;$(Solut
Resolve #using References	
Debug Information Format	Program Database (/Zi)
Common Language RunTime Support	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Multi-processor Compilation	
Use Unicode For Assembler Listing	

4. From the **analyze_locks Property Pages** dialog box, select **Linker > Debugging** and set the **Generate Debug Info** option to **Yes (/DEBUG)**.


analyze_locks Property Pages

Configuration: Release

Platform: Active(Win32)

Generate Debug Info	Yes (/DEBUG)
Generate Program Database File	$(TargetDir)$(TargetName).pdb
Strip Private Symbols	
Generate Map File	No
Map File Name	
Map Exports	No
Debuggable Assembly	

Build the Target in the Release Mode

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

1. Go to the **Build > Configuration Manager...** dialog box and select the **Release** mode for your target project.
2. From the Visual Studio menu, select **Build > Build analyze_locks**.

   The `tachyon_analyze_locks` application is built.

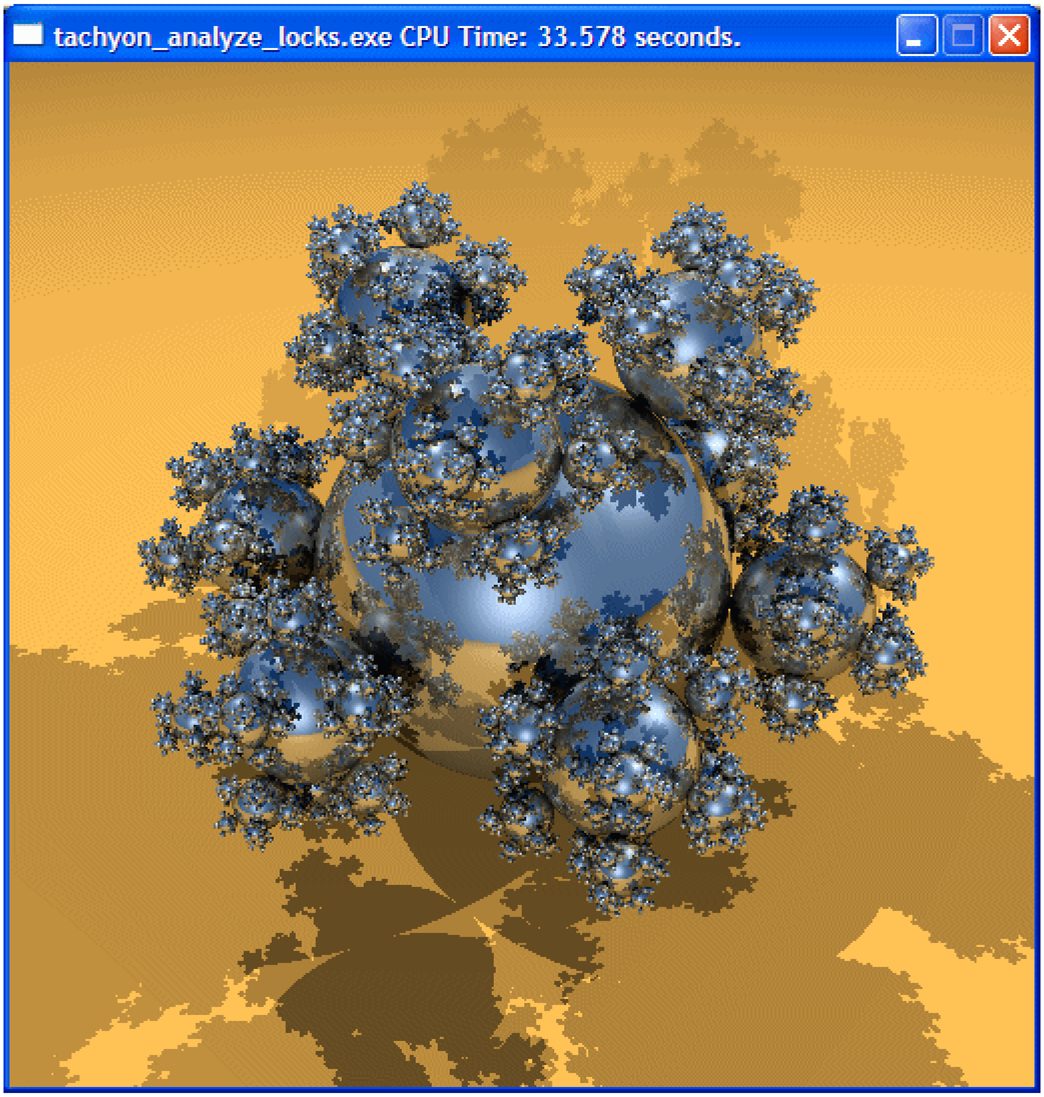
NOTE The build configuration for `tachyon` may initially be set to Debug, which is typically used for development. When analyzing performance issues with the VTune Amplifier XE, you are recommended to use the Release build with normal optimizations. In this way, the VTune Amplifier XE is able to analyze the realistic performance of your application.

Create a Performance Baseline

1. From the Visual Studio menu, select **Debug > Start Without Debugging**.

   The `tachyon_analyze_locks` application runs in multiple sections (depending on the number of CPUs in your system).

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



2. Note the execution time displayed in the window caption. For the `tachyon_analyze_locks` executable in the figure above, the execution time is 33.578 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

1. From the **Start** menu select **Intel Parallel Studio XE 2011 > Intel VTune Amplifier XE 2011** to launch the VTune Amplifier XE standalone GUI.
2. Create a new project via **File > New > Project...**.

   The **Create a Project** dialog box opens.
3. Specify the project name `tachyon` that will be used as the project directory name.

   VTune Amplifier XE creates a project directory under the `%USERPROFILE%\My Documents\My Amplifier XE Projects` directory and opens the **Project Properties: Target** dialog box.
4. In the **Application to Launch** pane of the **Target** tab, specify and configure your target as follows:

   - For the **Application** field, browse to: `<`*`tachyon_dir`*`>\analyze_locks.exe`.
5. Click **OK** to apply the settings and exit the **Project Properties** dialog box.

Key Terms

- Baseline
- Target

Next Step

Run Locks and Waits Analysis

Run Locks and Waits Analysis

Before running an analysis, choose a configuration level to define the Intel® VTune™ Amplifier XE analysis scope and running time. In this tutorial, you run the Locks and Waits analysis to identify synchronization objects that caused contention and fix the problem in the source.

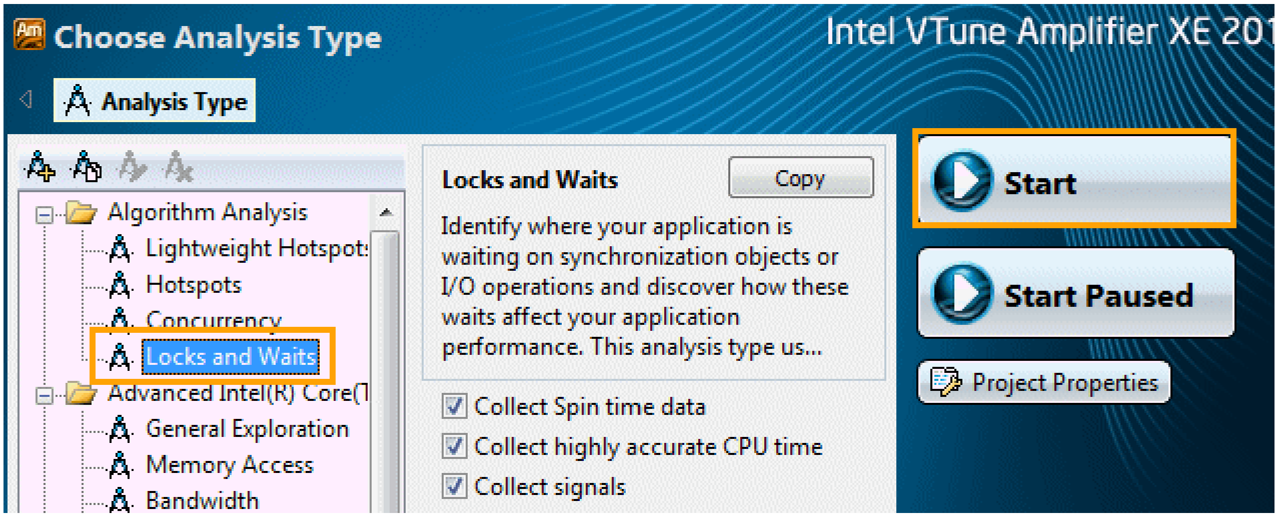
To run an analysis:

1. From the VTune Amplifier XE toolbar, analysis type from the drop-down menuclick the **New Analysis** button.

   The VTune Amplifier XE result tab opens with the **Analysis Type** window active.

2. From the analysis tree on the left, select **Algorithm Analysis > Locks and Waits**.

   The right pane is updated with the default options for the Locks and Waits analysis.
3. Click the **Start** button on the right command bar.



Locks and Waits

Identify where your application is waiting on synchronization objects or I/O operations and discover how these waits affect your application performance. This analysis type us...

☑ Collect Spin time data

☑ Collect highly accurate CPU time

☑ Collect signals

The VTune Amplifier XE launches the `tachyon_analyze_locks` executable that renders `balls.dat` as an input file, calculates the execution time, and exits. The VTune Amplifier XE finalizes the collected data and opens the results in the Locks and Waits 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.
- 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

- Finalization

- 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 Locks and Waits viewpoint that consists of the Summary window, Bottom-up pane, Top-down Tree pane, Call Stack pane, and Timeline pane. To interpret the data on the sample code performance, do the following:

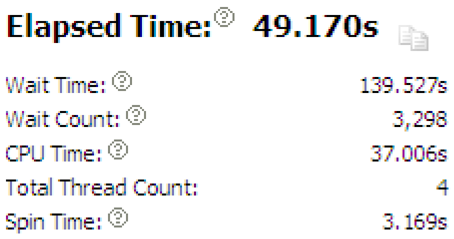
1. Analyze the basic performance metrics provided by the Locks and Waits analysis.
2. Identify locks.

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.

Analyze the Basic Locks and Waits Metrics

Start with exploring the data provided in the Summary window for the whole application performance. To interpret the data, hover over the question mark icons ? to read the pop-up help and better understand what each performance metric means.

The **Result Summary** section provides data on the overall application performance per the following metrics:



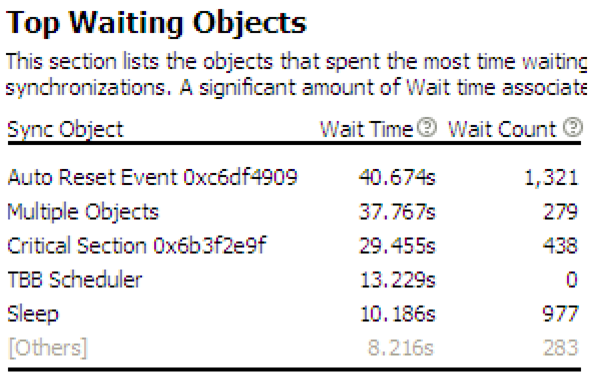
Elapsed Time: ? **49.170s**

Wait Time: ?	139.527s
Wait Count: ?	3,298
CPU Time: ?	37.006s
Total Thread Count:	4
Spin Time: ?	3.169s

1) **Elapsed Time** is the total time for each core when it was either waiting or not utilized by the application; 2)**Total Thread Count** is the number of threads in the application; 3)**Wait Time** is the amount of time the application threads waited for some event to occur, such as synchronization waits and I/O waits; 4) **Wait Count** is the overall number of times the system wait API was called for the analyzed application; 5) **CPU Time** is the sum of CPU time for all threads; 6) **Spin Time** is the time a thread is active in a synchronization construct.

For the `tachyon_analyze_locks` application, the Wait time is high. To identify the cause, you need to understand how this Wait time was distributed per synchronization objects.

The **Top Waiting Objects** section provides the list of five synchronization objects with the highest Wait Time and Wait Count, sorted by the Wait Time metric.



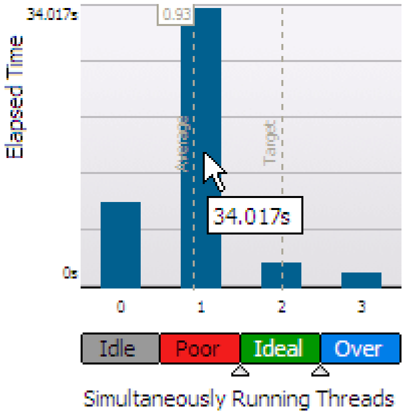
Top Waiting Objects

This section lists the objects that spent the most time waitinc synchronizations. A significant amount of Wait time associate

Sync Object	Wait Time ?	Wait Count ?
Auto Reset Event 0xc6df4909	40.674s	1,321
Multiple Objects	37.767s	279
Critical Section 0x6b3f2e9f	29.455s	438
TBB Scheduler	13.229s	0
Sleep	10.186s	977
[Others]	8.216s	283

For the `tachyon_analyze_locks` application, focus on the first three objects and explore the Bottom-up pane data for more details.

The **Thread Concurrency Histogram** represents the Elapsed time and concurrency level for the specified number of running threads. Ideally, the highest bar of your chart should be within the Ok or Ideal utilization range.



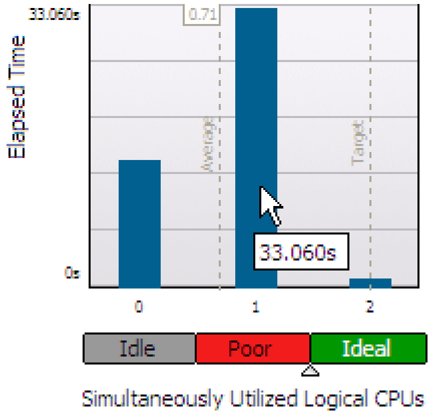
Note the **Target** value. By default, this number is equal to the number of physical cores. Consider this number as your optimization goal.

The **Average** metric is calculated as CPU time / Elapsed time. Use this number as a baseline for your performance measurements. The closer this number to the number of cores, the better.

For the sample code, the chart shows that `tachyon_analyze_locks` is a multithreaded application running four threads on a machine with four cores. But it is not using available cores effectively. The Average CPU Usage on the chart is about 0.7 while your target should be making it as closer to 4 as possible (for the system with four cores).

Hover over the second bar to understand how long the application ran serially. The tooltip shows that the application ran one thread for almost 15 seconds, which is classified as Poor concurrency.

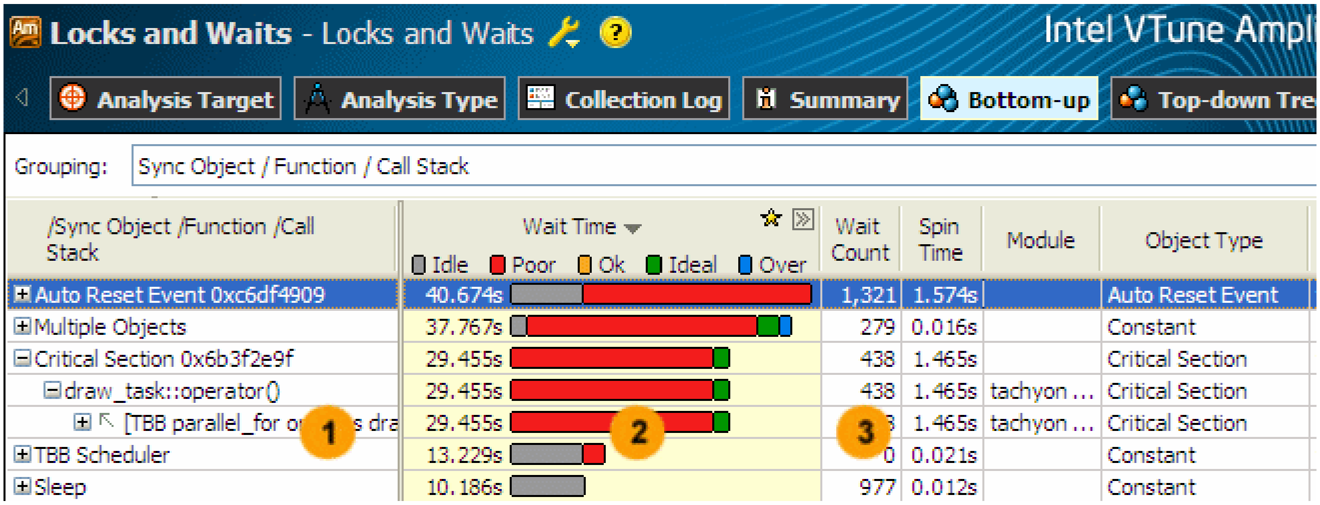
The **CPU Usage Histogram** represents the Elapsed time and usage level for the logical CPUs. Ideally, the highest bar of your chart should be within the Ok or Ideal utilization range.



The `tachyon_analyze_locks` application ran mostly on one logical CPU. If you hover over the second bar, you see that it spent 16.603 seconds using one core only, which is classified by the VTune Amplifier XE as a Poor utilization. To understand what prevented the application from using all available logical CPUs effectively, explore the Bottom-up pane.

Identify Locks

Click the **Bottom-up** tab to open the Bottom-up pane.



Grouping: Sync Object / Function / Call Stack

/Sync Object /Function /Call Stack	Wait Time Idle · Poor · Ok · Ideal · Over	Wait Count	Spin Time	Module	Object Type
⊞ Auto Reset Event 0xc6df4909	40.674s	1,321	1.574s		Auto Reset Event
⊞ Multiple Objects	37.767s	279	0.016s		Constant
⊟ Critical Section 0x6b3f2e9f	29.455s	438	1.465s		Critical Section
⊟ draw_task::operator()	29.455s	438	1.465s	tachyon ...	Critical Section
⊞ ↸ [TBB parallel_for o s dra	29.455s	3	1.465s	tachyon ...	Critical Section
⊞ TBB Scheduler	13.229s	0	0.021s		Constant
⊞ Sleep	10.186s	977	0.012s		Constant

1. Synchronization objects that control threads in the application. The hash (unique number) appended to some names of the objects identify the stack creating this synchronization object.

   For Intel® Threading Building Blocks (Intel® TBB), VTune Amplifier XE is able to recognize all types of Intel TBB objects. To display an overhead introduced by Intel TBB library internals, the VTune Amplifier XE creates a pseudo synchronization object **TBB scheduler** that includes all waits from the Intel TBB runtime libraries.

2. The utilization of the processor time when a given thread waited for some event to occur. By default, the synchronization objects are sorted by **Poor** processor utilization type. Bars showing OK or Ideal utilization (orange and green) are utilizing the processors well. You should focus your optimization efforts on functions with the longest poor CPU utilization (red 🟥 bars if the bar format is selected). Next, search for the longest over-utilized time (blue 🟦 bars).

   This is the Data of Interest column for the Locks and Waits analysis results that is used for different types of calculations, for example: call stack contribution, percentage value on the filter toolbar.

3. Number of times the corresponding system wait API was called. For a lock, it is the number of times the lock was contended and caused a wait. Usually you are recommended to focus your tuning efforts on the waits with both high Wait Time and Wait Count values, especially if they have poor utilization.

4. Wait time, during which the CPU is busy. This often occurs when a synchronization API causes the CPU to poll while the software thread is waiting. Some Spin time may be preferable to the alternative of the increased thread context switches. However, too much Spin time can reflect lost opportunity for productive work.

For the analyzed sample code, you see that the top three synchronization objects caused the longest Wait time. The red bars in the **Wait Time** column indicate that most of the time for these objects processor cores were underutilized.

From the code knowledge, you may understand that the Manual and Auto Reset Event objects are most likely related to the join where the main program is waiting for the worker threads to finish. This should not be a problem. Consider the third item in the Bottom-up pane that is more interesting. It is a Critical Section that shows much serial time and is causing a wait. Click the plus sign ⊞ at the object name to expand the node and see the `draw_task` wait function that contains this critical section and call stack. Double-click the Critical Section to see the source code for the wait function.

Key Terms

- CPU usage

- Elapsed time
- Wait time

Next Step

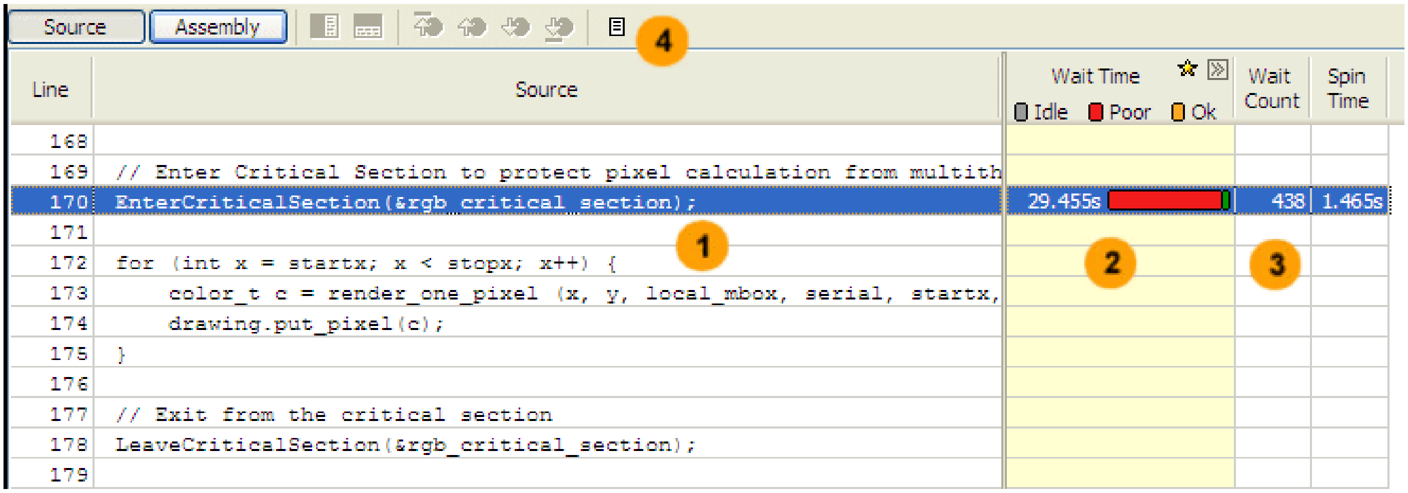
Analyze Code

Analyze Code

You identified the critical section that caused significant Wait time and poor processor utilization. Double-click this critical section in the Bottom-up pane to view the source. The Intel® VTune™ Amplifier XE opens source and disassembly code. Focus on the Source pane and analyze the source code:

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

Understand Basic Source View Options



Line	Source	Wait Time ■ Idle ■ Poor ■ Ok	Wait Count	Spin Time
168				
169	`// Enter Critical Section to protect pixel calculation from multith`			
170	`EnterCriticalSection(&rgb_critical_section);`	29.455s 🟥	438	1.465s
171				
172	`for (int x = startx; x < stopx; x++) {`			
173	`    color_t c = render_one_pixel (x, y, local_mbox, serial, startx,`			
174	`    drawing.put_pixel(c);`			
175	`}`			
176				
177	`// Exit from the critical section`			
178	`LeaveCriticalSection(&rgb_critical_section);`			
179				

The table below explains some of the features available in the Source panefor the Locks and Waits viewpoint.

1 Source code of the application displayed if the function symbol information is available. When you go to the source by double-clicking the synchronization object in the Bottom-up pane, the VTune Amplifier XE opens the wait function containing this object and highlights the code line that took the most Wait time. 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 wait function. To view the source code in the Source pane, make sure to build the target properly.

2 Processor time and utilization bar attributed to a particular code line. The colored bar represents the distribution of the Wait time according to the utilization levels (Idle, Poor, Ok, Ideal, and Over) defined by the VTune Amplifier XE. The longer the bar, the higher the value. Ok utilization level is not available for systems with a small number of cores.

This is the Data of Interest column for the Locks and Waits analysis.

3 Number of times the corresponding system wait API was called while this code line was executing. For a lock, it is the number of times the lock was contended and caused a wait.

4 Source window toolbar. Use 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 Locks and Waits analysis, this is Wait Time. Use the source file editor button to open and edit your code in your default editor.

Identify the Hottest Code Lines

The VTune Amplifier XE highlights line 170 entering the critical section `rgb_critical_section` in the `draw_task` function. The `draw_task` function was waiting for almost 27 seconds while this code line was executing and most of the time the processor was underutilized. During this time, the critical section was contended 438 times.

The `rgb_critical section` is the place where the application is serializing. Each thread has to wait for the critical section to be available before it can proceed. Only one thread can be in the critical section at a time.

You need to optimize the code to make it more concurrent. Click the 🗎 **Source Editor** button on the Source window toolbar to open the code editor and optimize the code.

Key Terms

- CPU usage
- Wait time

Next Step

Remove Lock

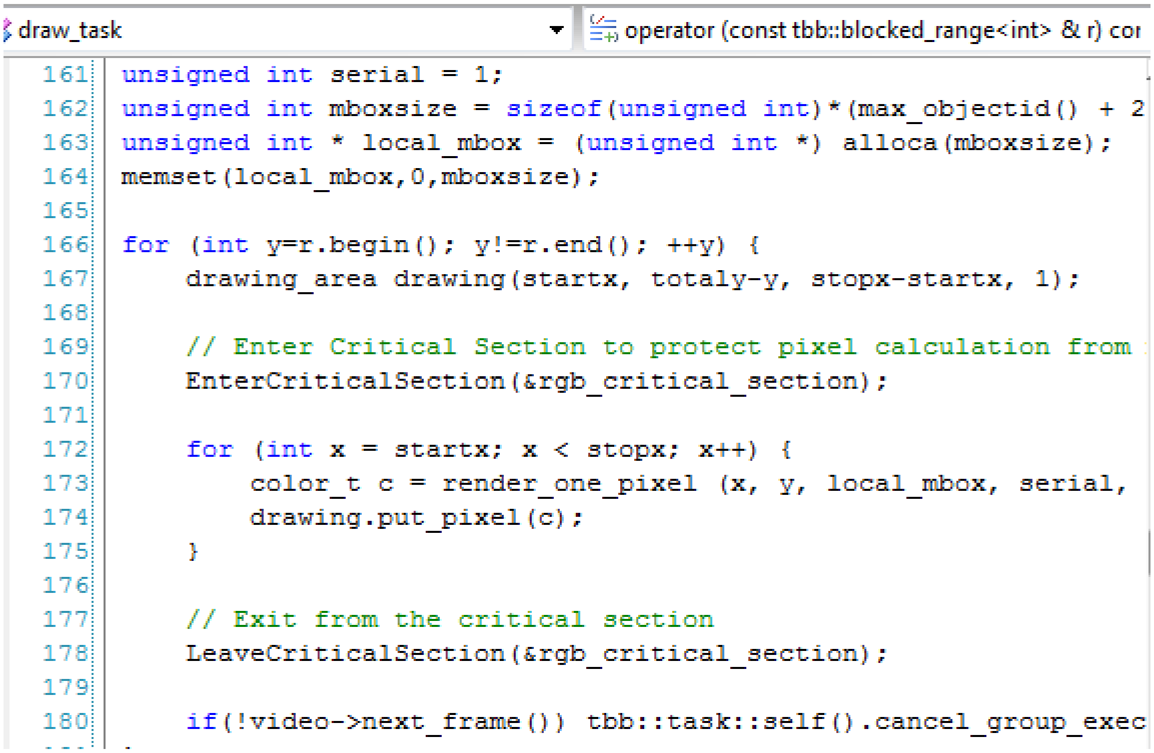
Remove Lock

In the Source window, you located the critical section that caused a significant wait while the processor cores were underutilized and generated multiple wait count. Focus on this line and do the following:

1. Open the code editor.
2. Modify the code to remove the lock.

Open the Code Editor

Click the 🗎 **Source Editor** button to open the `analyze_locks.cpp` file in your default editor at the hotspot code line:



```
161 unsigned int serial = 1;
162 unsigned int mboxsize = sizeof(unsigned int)*(max_objectid() + 2
163 unsigned int * local_mbox = (unsigned int *) alloca(mboxsize);
164 memset(local_mbox,0,mboxsize);
165
166 for (int y=r.begin(); y!=r.end(); ++y) {
167     drawing_area drawing(startx, totaly-y, stopx-startx, 1);
168
169     // Enter Critical Section to protect pixel calculation from
170     EnterCriticalSection(&rgb_critical_section);
171
172     for (int x = startx; x < stopx; x++) {
173         color_t c = render_one_pixel (x, y, local_mbox, serial,
174         drawing.put_pixel(c);
175     }
176
177     // Exit from the critical section
178     LeaveCriticalSection(&rgb_critical_section);
179
180     if(!video->next_frame()) tbb::task::self().cancel_group_exec
```

Remove the Lock

The `rgb_critical_section` was introduced to protect calculation from multithreaded access. The brief analysis shows that the code is thread safe and the critical section is not really needed.

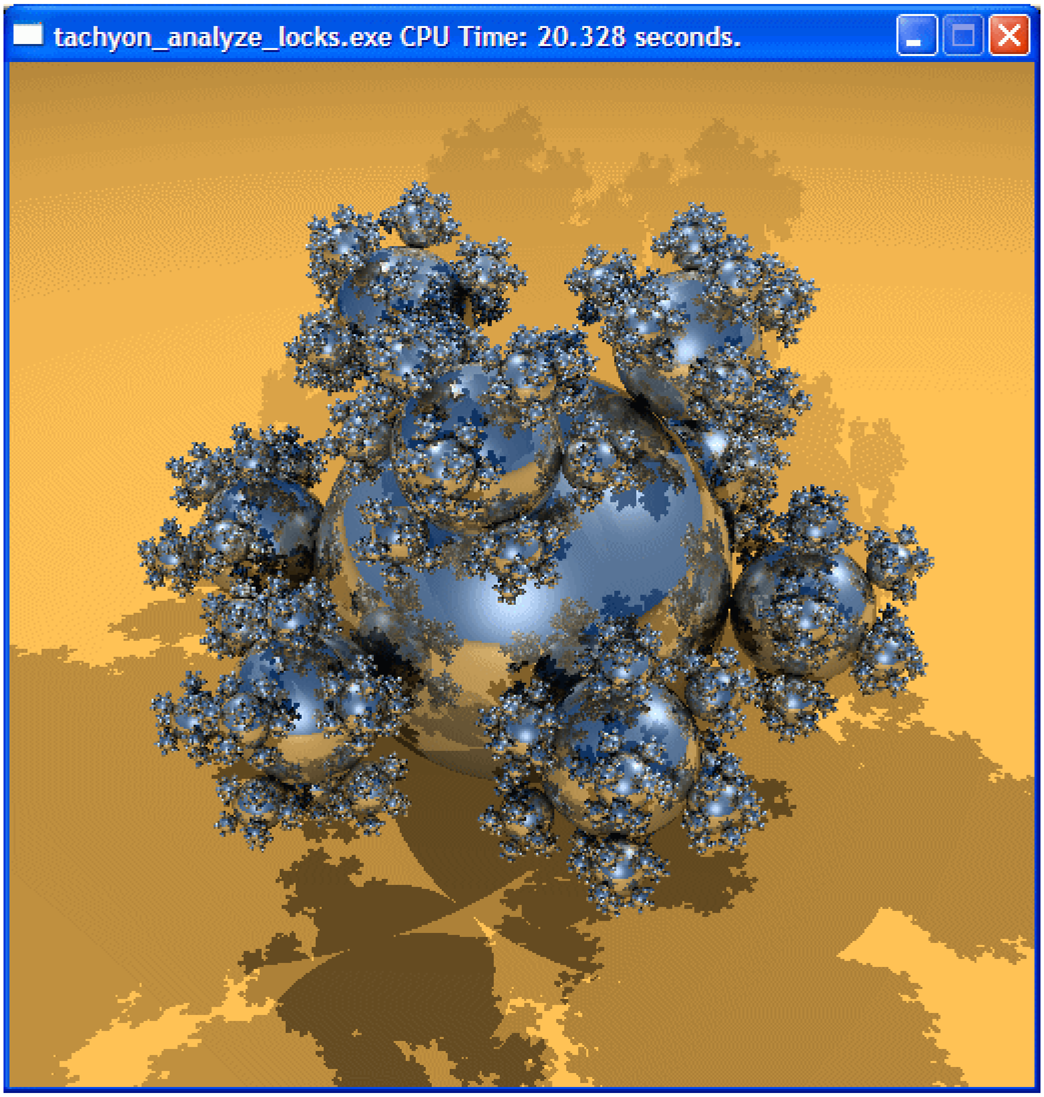
To resolve this issue:

NOTE The steps below are provided for Microsoft Visual Studio* 2010. Steps for other versions of Visual Studio IDE may slightly differ.

1. Comment out code lines 170 and 178 to disable the critical section.
2. From Solution Explorer, select the **analyze_locks** project.
3. From Visual Studio menu, select **Build > Rebuild analyze_locks**.

   The project is rebuilt.
4. From Visual Studio menu, select **Debug > Start Without Debugging** to run the application.

   Visual Studio runs the `tachyon_analyze_locks.exe`. Note that execution time reduced from 33.578 seconds to 20.328 seconds.



tachyon_analyze_locks.exe CPU Time: 20.328 seconds.

Key Terms

Hotspot

Next Step

Compare with Previous Result

Compare with Previous Result

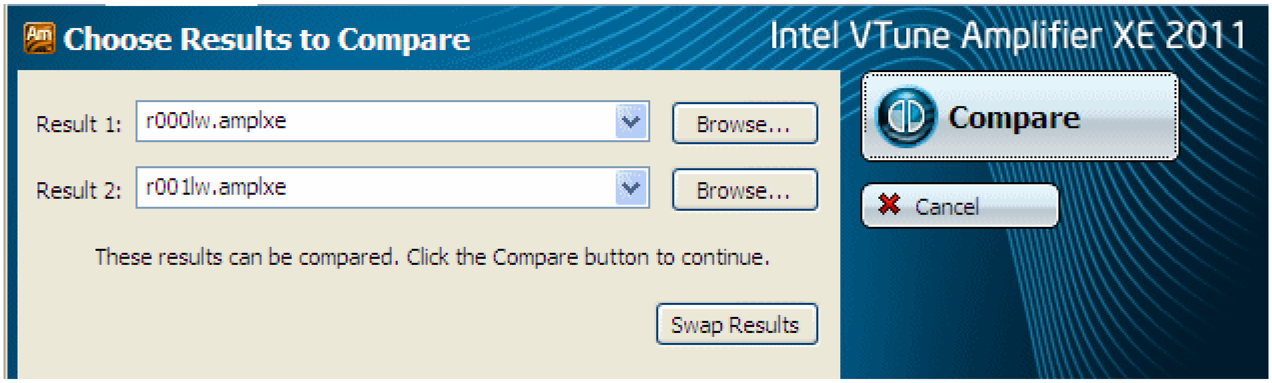
You made sure that removing the critical section gave you 13 seconds of optimization in the application execution time. To understand the impact of your changes and how the CPU utilization has changed, re-run the Locks and Waits analysis on the optimized code and compare results:

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

Compare Results Before and After Optimization

1. Run the Locks and Waits 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 Locks and Waits analysis results you want to compare:



Choose Results to Compare

Intel VTune Amplifier XE 2011

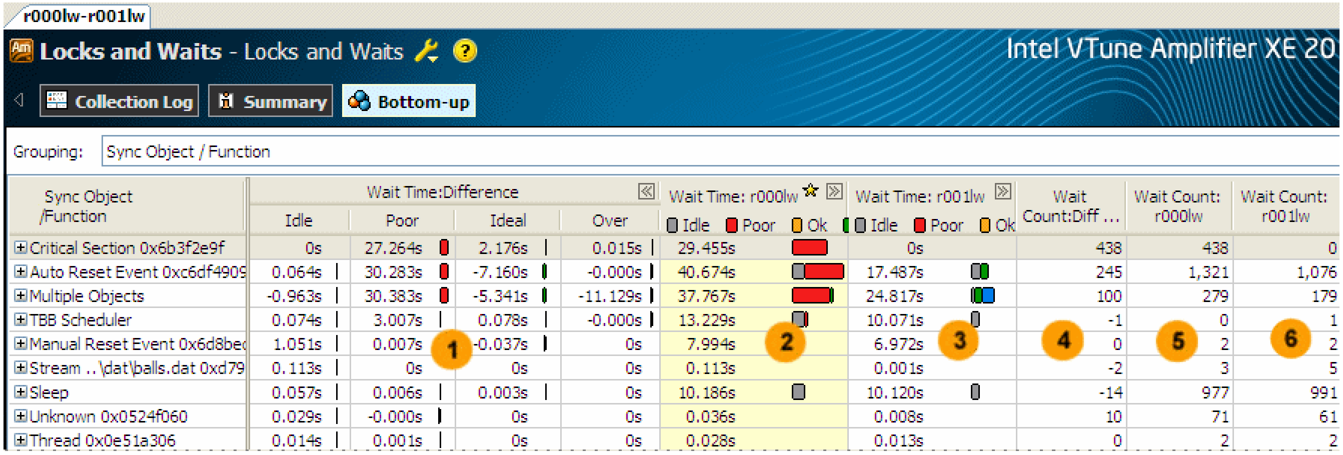
Result 1: r000lw.amplxe [Browse...]

Result 2: r001lw.amplxe [Browse...]

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

The Summary window opens providing the statistics for the difference between collected results.

Click the **Bottom-up** tab to see the list of synchronization objects used in the code, Wait time utilization across the two results, and the differences side by side:



r000lw-r001lw
Locks and Waits - Locks and Waits
Intel VTune Amplifier XE 20
Collection Log
Summary
Bottom-up
1
2
3
4
5
6

Grouping: Sync Object / Function

Sync Object /Function	Wait Time:Difference				Wait Time: r000lw	Wait Time: r001lw	Wait Count:Diff ...	Wait Count: r000lw	Wait Count: r001lw
	Idle	Poor	Ideal	Over	Idle Poor Ok	Idle Poor Ok			
⊞Critical Section 0x6b3f2e9f	0s	27.264s	2.176s	0.015s	29.455s	0s	438	438	0
⊞Auto Reset Event 0xc6df4909	0.064s	30.283s	-7.160s	-0.000s	40.674s	17.487s	245	1,321	1,076
⊞Multiple Objects	-0.963s	30.383s	-5.341s	-11.129s	37.767s	24.817s	100	279	179
⊞TBB Scheduler	0.074s	3.007s	0.078s	-0.000s	13.229s	10.071s	-1	0	1
⊞Manual Reset Event 0x6d8bec	1.051s	0.007s	-0.037s	0s	7.994s	6.972s	0	2	2
⊞Stream ..\dat\balls.dat 0xd79	0.113s	0s	0s	0s	0.113s	0.001s	-2	3	5
⊞Sleep	0.057s	0.006s	0.003s	0s	10.186s	10.120s	-14	977	991
⊞Unknown 0x0524f060	0.029s	-0.000s	0s	0s	0.036s	0.008s	10	71	61
⊞Thread 0x0e51a306	0.014s	0.001s	0s	0s	0.028s	0.013s	0	2	2

1 Difference in Wait time per utilization level between the two results in the following format: <Difference Wait Time> = <Result 1 Wait Time> – <Result 2 Wait Time>. By default, the Difference column is expanded to display comparison data per utilization level. You may collapse the column to see the total difference data per Wait time.

2 Wait time and CPU utilization for the initial version of the code.

3 Wait time and CPU utilization for the optimized version of the code.

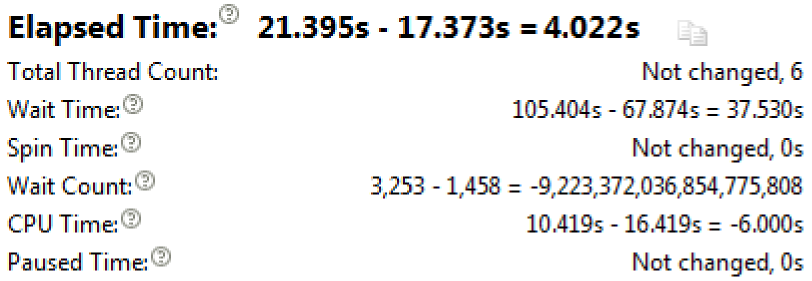
4 Difference in Wait count between the two results in the following format: <Difference Wait Count> = <Results 1 Wait Count> - <Result 2 Wait Count>.

5 Wait count for the initial version of the code.

6 Wait count for the optimized version of the code.

Identify the Performance Gain

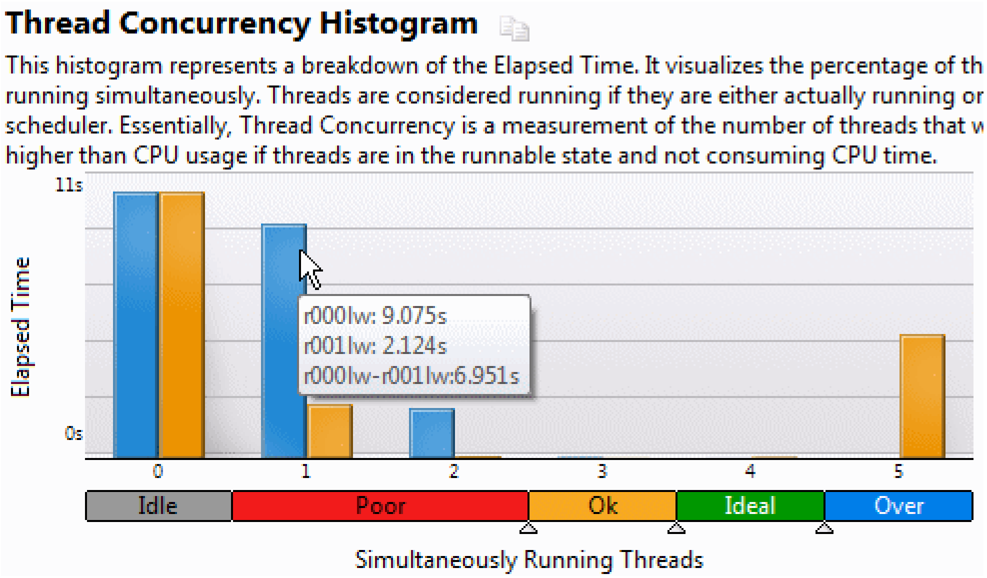
The Elapsed time data in the Summary window shows the optimization of 4 seconds for the whole application execution and Wait time decreased by 37.5 seconds.



Elapsed Time: 21.395s - 17.373s = 4.022s

Total Thread Count:	Not changed, 6
Wait Time:	105.404s - 67.874s = 37.530s
Spin Time:	Not changed, 0s
Wait Count:	3,253 - 1,458 = -9,223,372,036,854,775,808
CPU Time:	10.419s - 16.419s = -6.000s
Paused Time:	Not changed, 0s

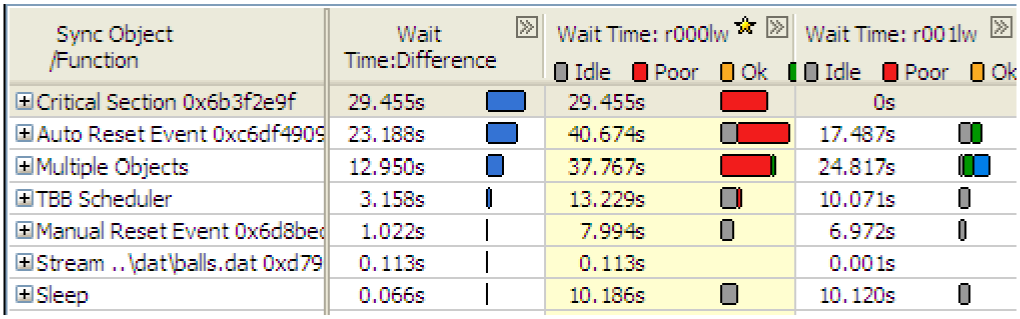
According to the **Thread Concurrency** histogram, before optimization (blue bar) the application ran serially for 9 seconds poorly utilizing available processor cores but after optimization (orange bar) it ran serially only for 2 seconds. After optimization the application ran 5 threads simultaneously overutilizing the cores for almost 5 seconds. Further, you may consider this direction as an additional area for improvement.



Thread Concurrency Histogram

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of th running simultaneously. Threads are considered running if they are either actually running or scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that w higher than CPU usage if threads are in the runnable state and not consuming CPU time.

In the Bottom-up pane, locate the Critical Section you identified as a bottleneck in your code. Since you removed it during optimization, the optimized result r001lw does not show any performance data for this synchronization object. If you collapse the **Wait Time:Difference** column by clicking the ⧏ button, you see that with the optimized result you got almost 29 seconds of optimization in Wait time.



Sync Object /Function	Wait Time:Difference	Wait Time: r000lw	Wait Time: r001lw
		■ Idle ■ Poor ■ Ok	■ Idle ■ Poor ■ Ok
⊞Critical Section 0x6b3f2e9f	29.455s	29.455s	0s
⊞Auto Reset Event 0xc6df4909	23.188s	40.674s	17.487s
⊞Multiple Objects	12.950s	37.767s	24.817s
⊞TBB Scheduler	3.158s	13.229s	10.071s
⊞Manual Reset Event 0x6d8bec	1.022s	7.994s	6.972s
⊞Stream ..\dat\balls.dat 0xd79	0.113s	0.113s	0.001s
⊞Sleep	0.066s	10.186s	10.120s

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 `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.

Key Terms

- CPU usage
- Elapsed time
- Hotspot
- Wait time

Summary

3

You have completed the Analyzing Locks and Waits tutorial. Here are some important things to remember when using the Intel® VTune™ Amplifier XE to analyze your code for locks and waits:

Step	Tutorial Recap	Key Tutorial Take-aways
1. Prepare for analysis	If you used the Visual Studio* IDE: You selected the **analyze_locks** project as the target for the Locks and Waits analysis. If you used the standalone GUI: You set up your environment to enable generating symbol information for system libraries and your binary files, built the target in the Release mode, created the performance baseline, and created the VTune Amplifier XE project for your analysis target. Your application is ready for analysis.	• Configure the Microsoft* symbol server and your project properties to get the most accurate results for system and user binaries and to analyze the performance of your application at the code line level. • Create a performance baseline to compare the application versions before and after optimization. Make sure to use the same workload for each application run. • Use the **Project Properties: Target** tab to choose and configure your analysis target. For Visual Studio* projects, the analysis target settings are inherited automatically.
2. Find lock	You ran the Locks and Waits data collection and identified the following hotspots: • You identified a synchronization object with the high Wait Time and Wait Count values and poor CPU utilization that could be a lock affecting application parallelism. Your next step is to analyze the code of this function. • Identified the code section that caused a significant wait and during which the processor was poorly utilized.	• Use the **Analysis Type** configuration window to choose, configure, and run the analysis. For example, you may limit the data collection to a predefined amount of data or enable the VTune Amplifier XE to collect more accurate CPU time data. You can also run the analysis from command line using the `amplxe-cl` command. • Start analyzing the performance of your application with the Summary pane to explore the performance metrics for the whole application. Then, move to the Bottom-up window to analyze the synchronization objects. Focus on the synchronization objects that under- or over-utilized the available logical CPUs and have the highest Wait time and Wait Count values. By default, the objects with the highest Wait time values show up at the top of the window.
3. Remove lock	You optimized the application execution time by removing the unnecessary critical section that caused a lot of Wait time.	Expand the most time-critical synchronization object in the Bottom-up pane and double-click the wait function it belongs to. This opens the source code for this wait function at the code line with the highest Wait time value.

Step	Tutorial Recap	Key Tutorial Take-aways
4. Check your work	You ran the Locks and Waits analysis on the optimized code and compared the results before and after optimization using the Compare mode of the VTune Amplifier XE. The comparison shows that, with the optimized version of the `tachyon_analyze_locks` application (r001lw result), you managed to remove the lock preventing application parallelism and significantly reduce the application execution time.	• 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. • Expand each data column by clicking the » button to identify the performance gain per CPU utilization level.

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

Key Terms

4

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.

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

thread concurrency: A performance metric that helps identify how an application utilizes the processors in the system by comparing the application concurrency level (the number of active threads) and target concurrency level (by default, equal to the number of physical cores). Thread concurrency may be higher than CPU usage if threads are in the runnable state and not consuming CPU time.

Utilization Type	Default color	Description
Idle	⬜	All threads in the program are waiting - no threads are running. There can be only one node in the **Summary** chart indicating idle utilization.
Poor	🟥	Poor utilization. By default, poor utilization is when the number of threads is up to 50% of the target concurrency.
OK	🟧	Acceptable (OK) utilization. By default, OK utilization is when the number of threads is between 51-85% of the target concurrency.

Utilization Type	Default color	Description
Ideal	🟩	Ideal utilization. By default, ideal utilization is when the number of threads is between 86-115% of the target concurrency.
Over	🟦	Over-utilization. By default, over-utilization is when the number of threads is more than 115% of the target concurrency.

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.

Wait time: The amount of time that a given thread waited for some event to occur, such as: synchronization waits and I/O waits.