Intel® Software Development Products

Getting Started Tutorial: Finding Hotspots

Intel® VTune™ Amplifier XE 2013 for Windows* OS

C++ Sample Application Code

Document Number: 326704-002

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.

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

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/

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

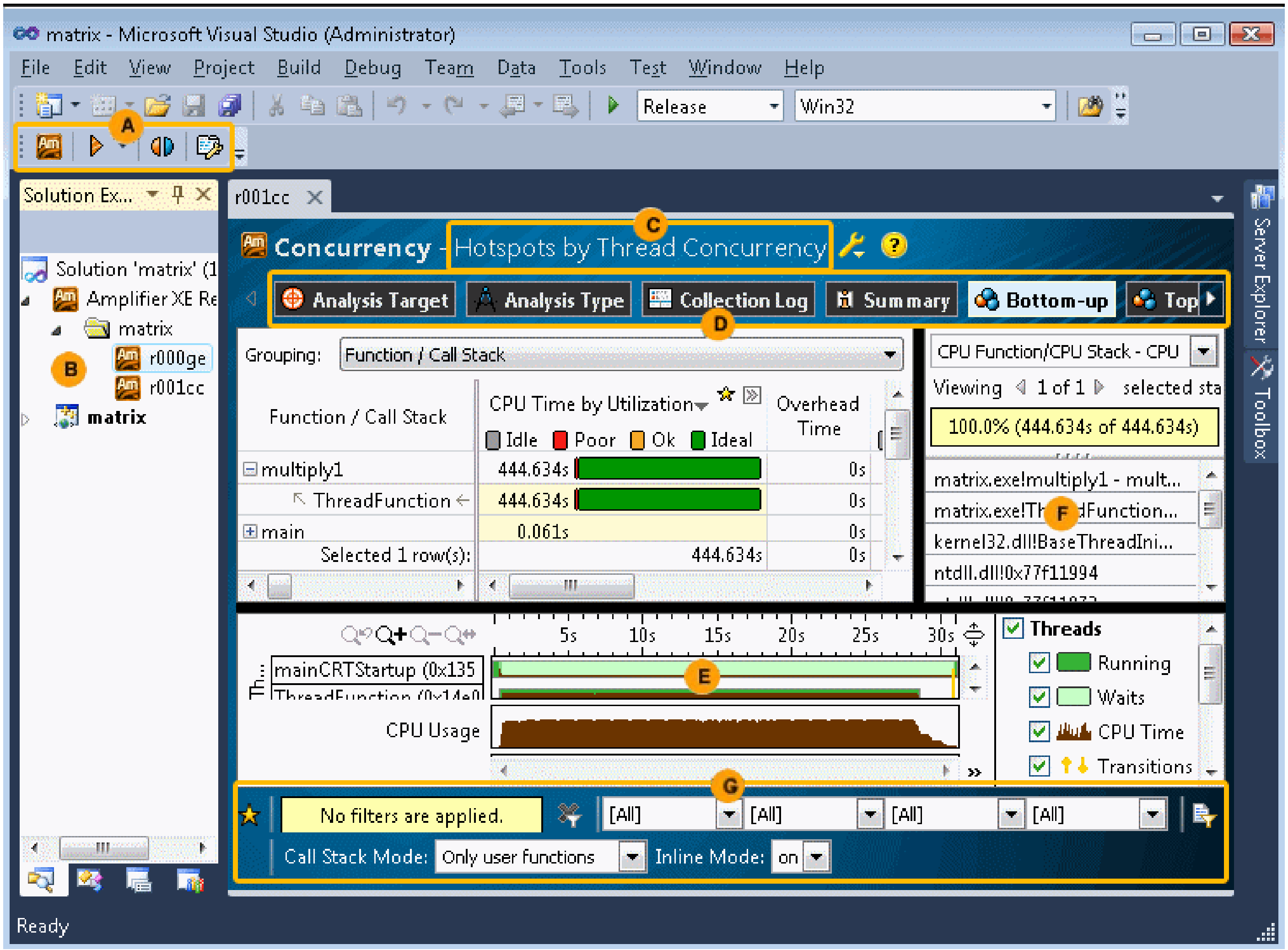
VTune Amplifier XE Access

To access the VTune Amplifier XE in the Visual Studio* IDE: From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2013** > **Parallel Studio XE 2013 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 2013** > **Intel VTune Amplifier XE 2013**.
- From the Windows* **Start** menu, choose **Intel Parallel Studio XE 2013** > **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



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 2013** menu from the pop-up menu. To manage previously collected results, right-click the result (for example, `r000ge.amplxe`) and select the required command from the pop-up menu.

C From the drop-down menu, 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.

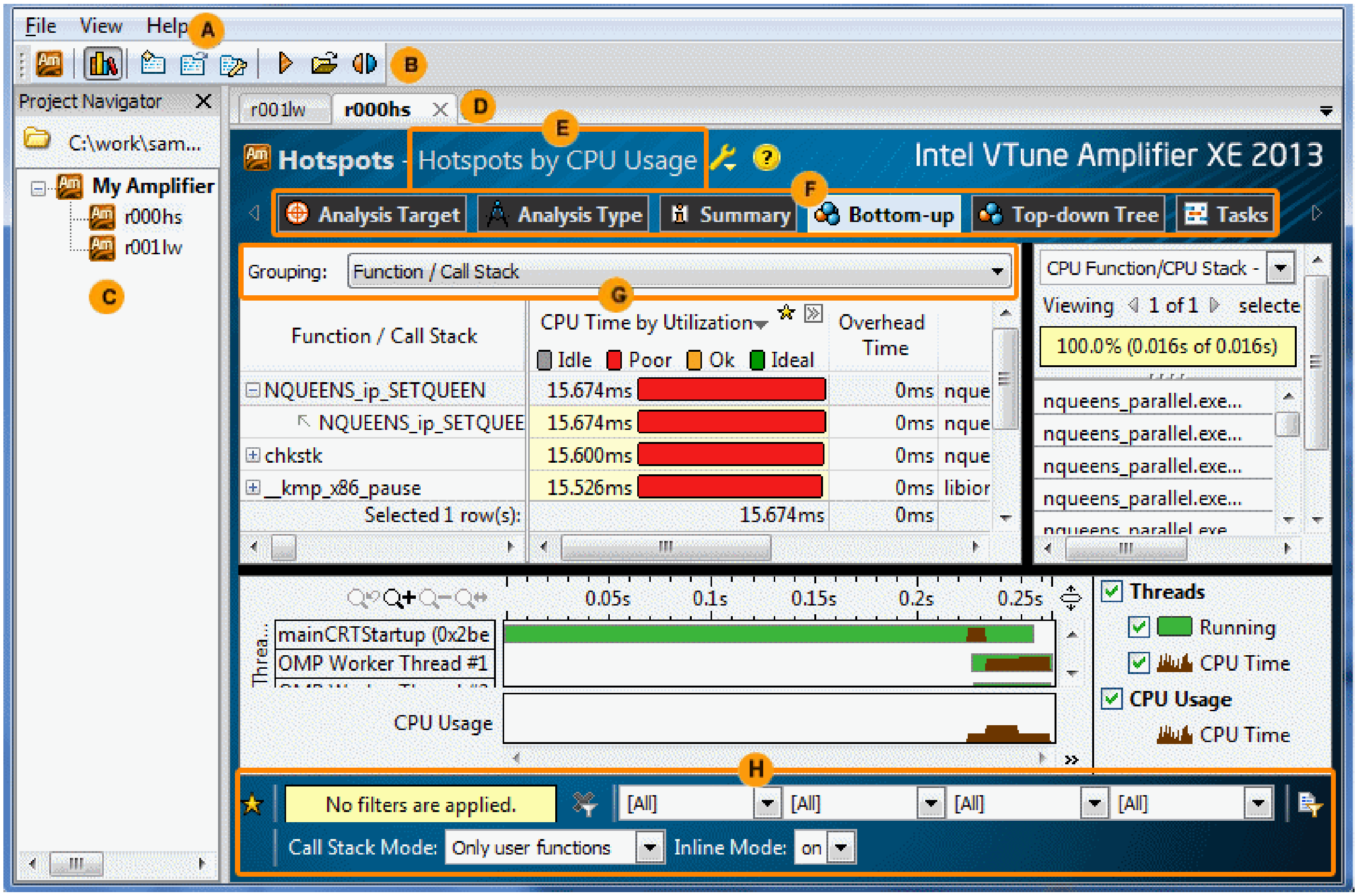
D Click the buttons on navigation toolbars to change window views and toggle window panes on and off.

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: Function / Call Stack

Viewing 1 of 1 selecte

100.0% (0.016s of 0.016s)

Function / Call Stack	CPU Time by Utilization	Overhead Time	
NQUEENS_ip_SETQUEEN	15.674ms	0ms	nque
NQUEENS_ip_SETQUEE	15.674ms	0ms	nque
chkstk	15.600ms	0ms	nque
__kmp_x86_pause	15.526ms	0ms	libior
Selected 1 row(s):	15.674ms	0ms	

No filters are applied.

Call Stack Mode: Only user functions

Inline Mode: on

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.

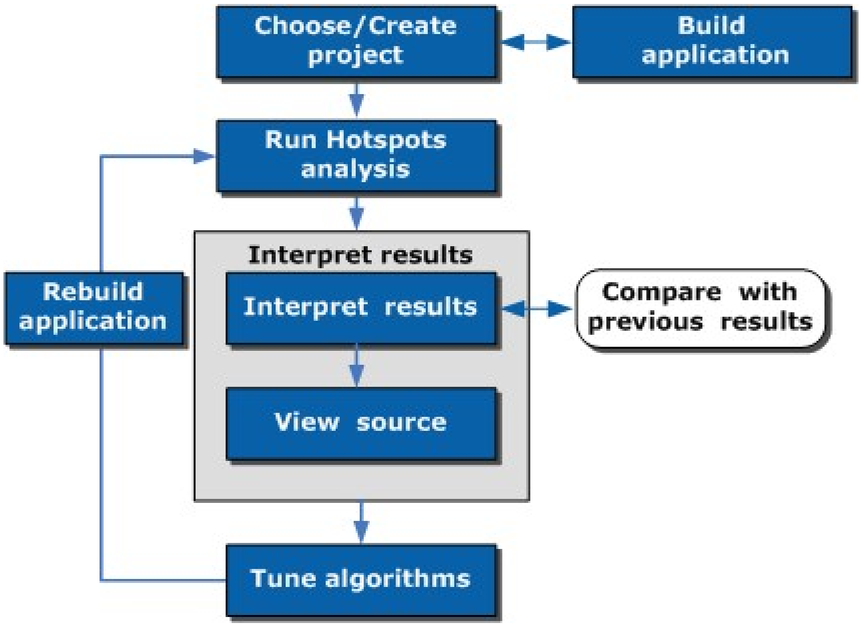
G Use the **Grouping** drop-down menu to choose a granularity level for grouping data in the grid.

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	Do one of the following: • In the Visual Studio* IDE: Choose a project, verify settings, build application, and create a baseline • In the Standalone Intel VTune Amplifier XE GUI: Build an application to analyze for hotspots, create a baseline, 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.

Visual Studio* IDE: Choose Project and Build Application

Before you start analyzing your application target for hotspots, 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. They may slightly differ for other versions of Visual Studio.
- 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); 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.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 (x86)\Intel\VTune Amplifier XE 2013\` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`).
2. Extract the sample from the `.zip` file.

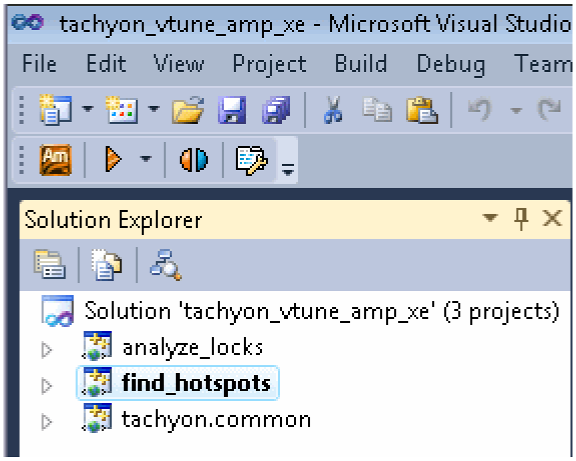
- Samples are non-deterministic. Your screens may vary from the screen captures shown throughout this tutorial.
- Samples are designed only to illustrate the VTune Amplifier XE features; they do not represent best practices for creating code.

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 extract 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 IDE and shows up in the Solution Explorer.

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

find_hotspots appears in bold in the Solution Explorer.

When you choose a project in Visual Studio IDE, the VTune Amplifier XE automatically creates the `config.amplxeproj` project file and sets the `find_hotspots` application as an analysis target in the project properties.

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...**.

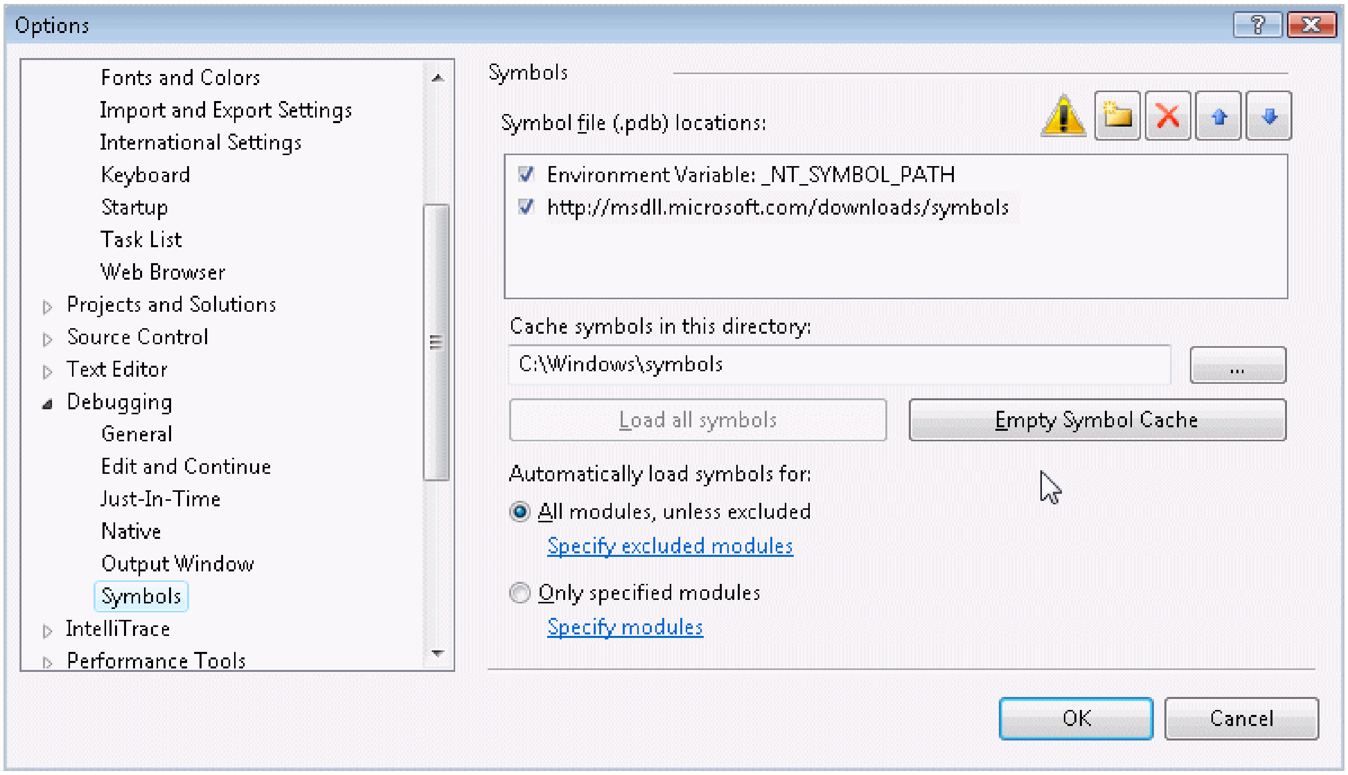
The **Options** dialog box opens.

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.



Options

Symbols

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

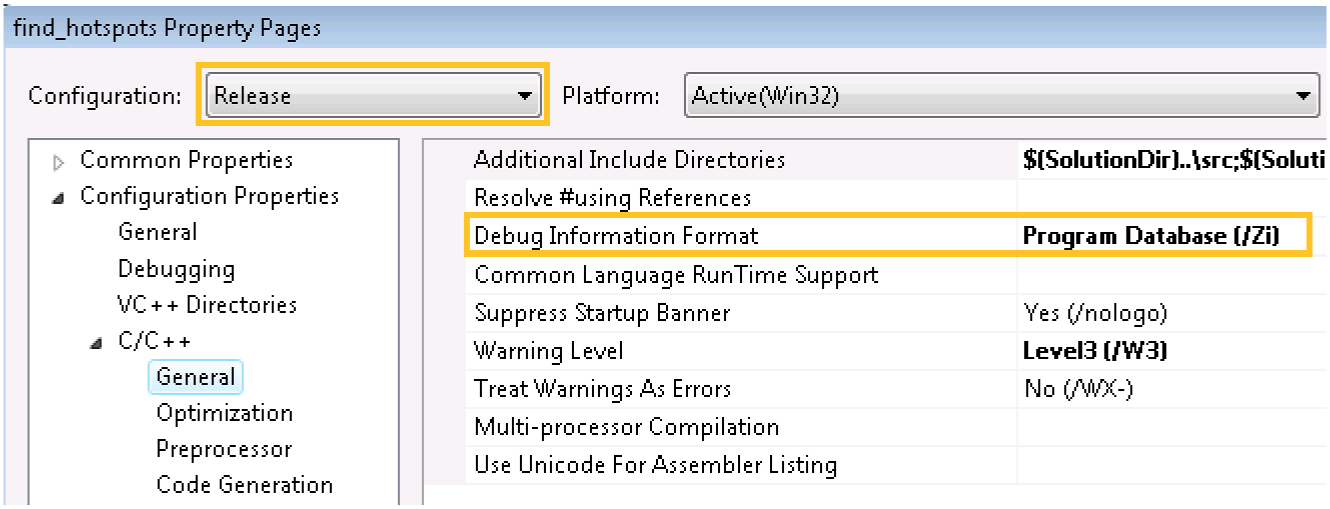
6. Click **OK**.

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 **find_hotspots** project and go to **Project > Properties**.

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

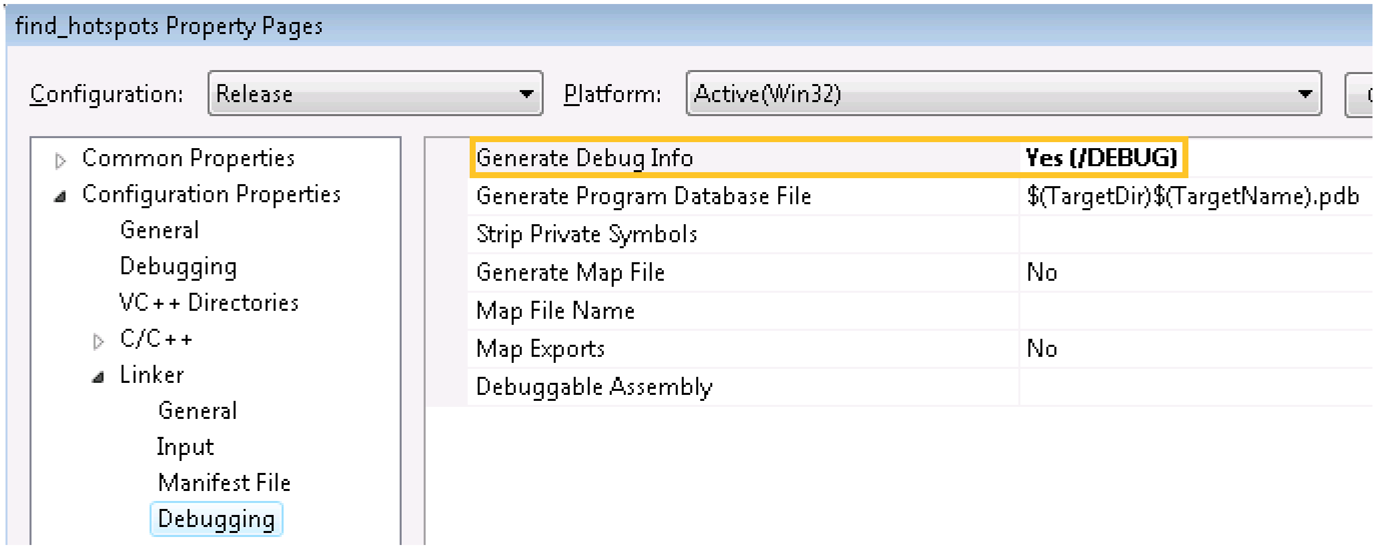


find_hotspots Property Pages

Configuration: Release   Platform: Active(Win32)

Additional Include Directories	$(SolutionDir)..\src;$(Soluti
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 **find_hotspots Property Pages** dialog box, select **Linker > Debugging** and set the **Generate Debug Info** option to **Yes (/DEBUG)**.



find_hotspots 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 find_hotspots**.

   The `find_hotspots.exe` application is built.

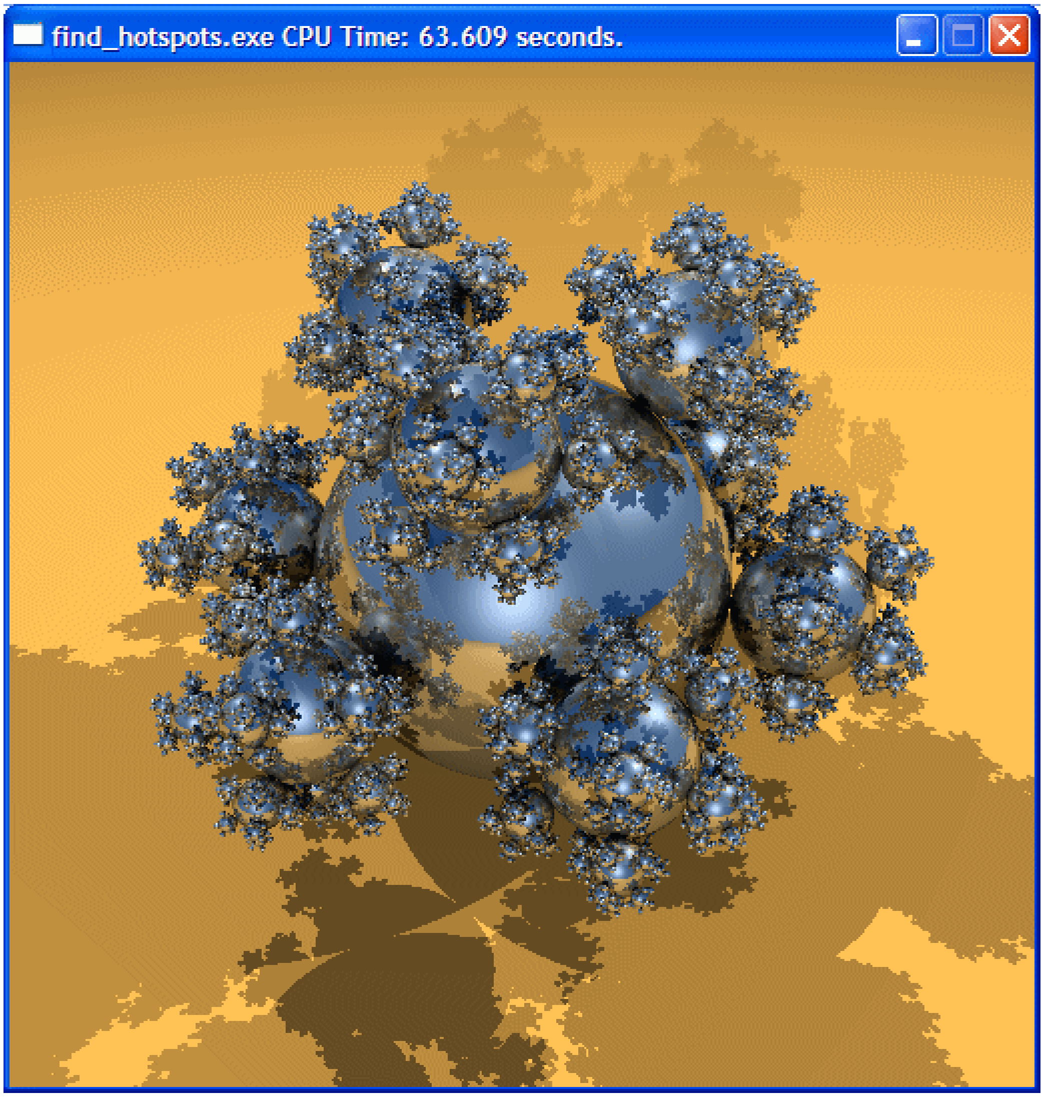
NOTE The build configuration for `tachyon` may initially be set to Debug, which is typically used for development. When analyzing performance issues with the VTune Amplifier XE, you are recommended to use the Release build with normal optimizations. In this way, the VTune Amplifier XE is able to analyze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

Create a Performance Baseline

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

   The `find_hotspots.exe` application starts running.

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



find_hotspots.exe CPU Time: 63.609 seconds.

2. Note the execution time displayed in the window caption. For the `find_hotspots.exe` executable in the figure above, the execution time is 63.609 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

- Baseline
- Hotspots analysis

- 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

Standalone GUI: 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.

   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. 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
- `zip` 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.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 2013\` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`).
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. They may differ slightly for other versions of Visual Studio.

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

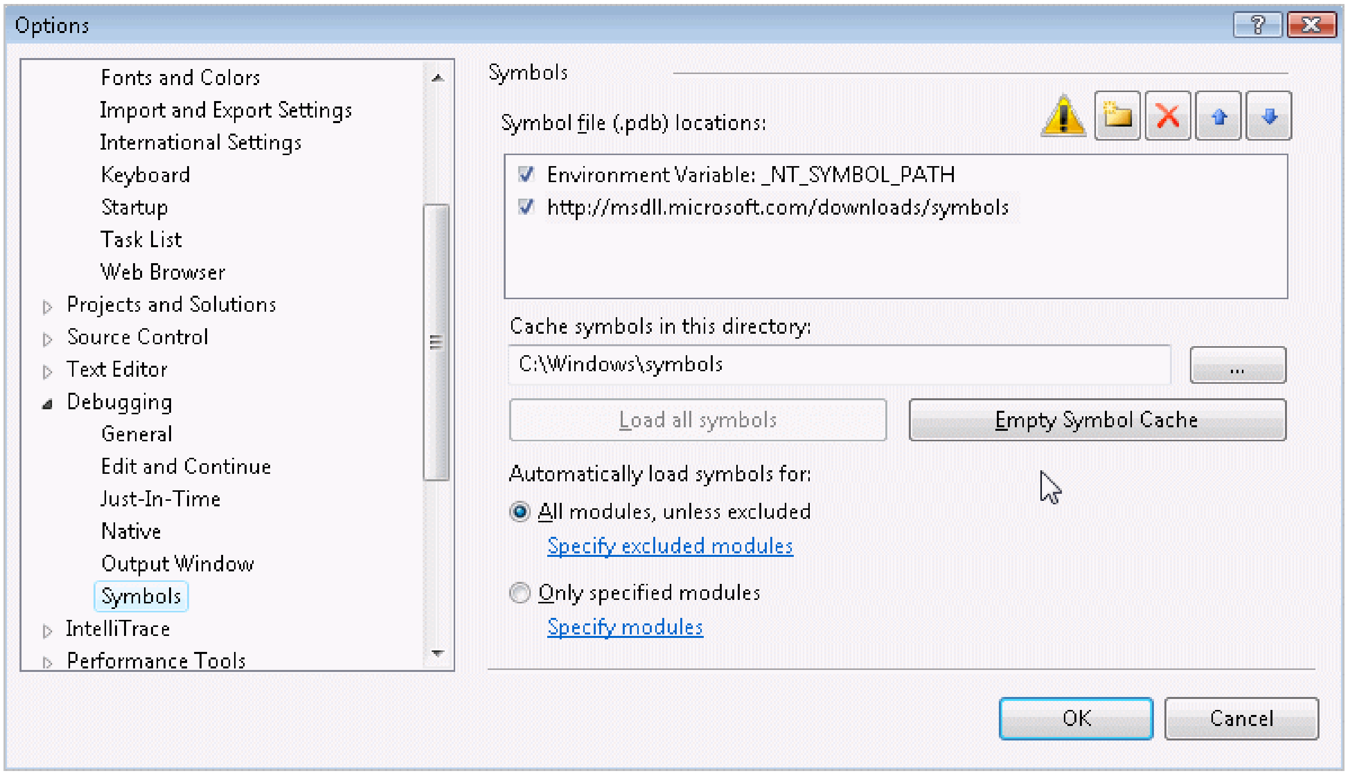
The **Options** dialog box opens.

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

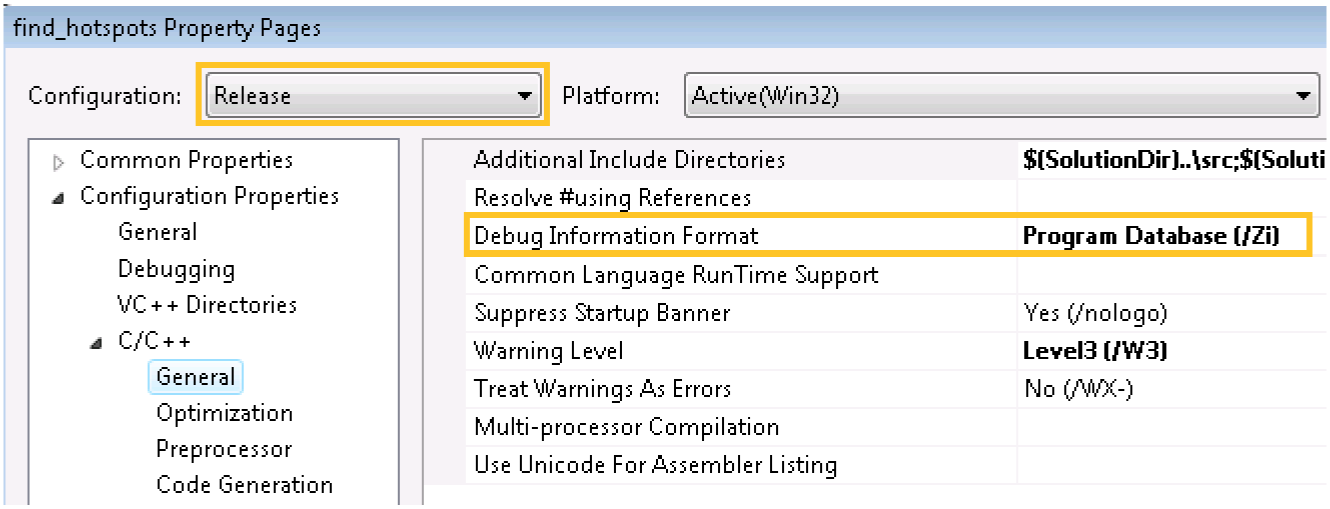
6. Click **OK**.

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 **find_hotspots** project and go to **Project > Properties**.

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



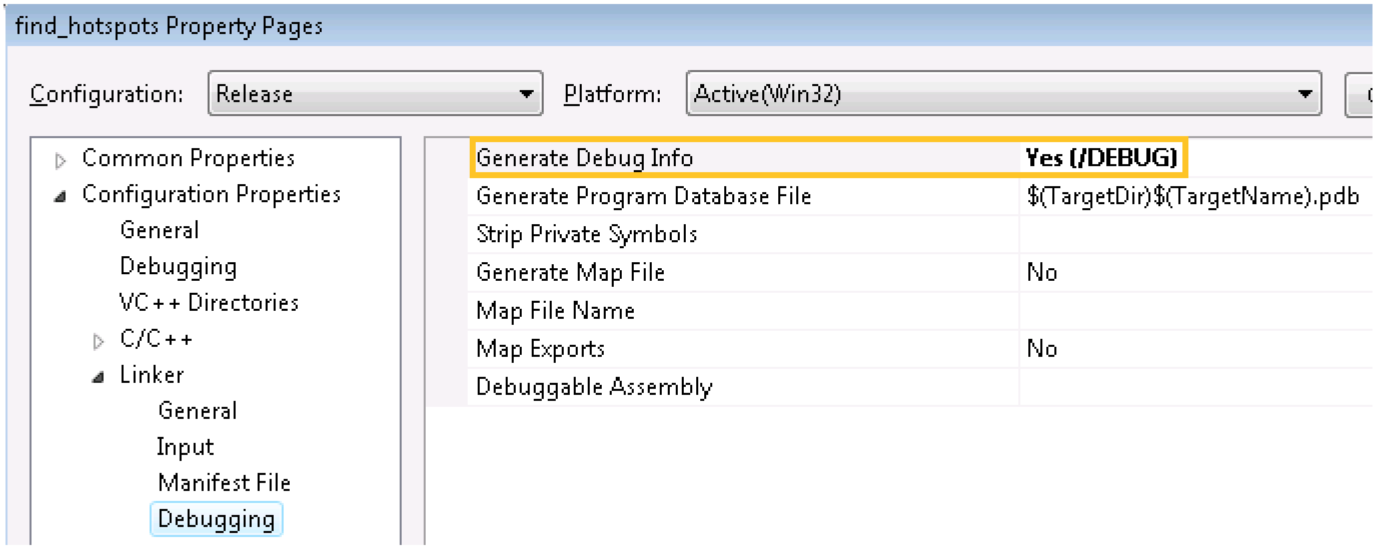
find_hotspots Property Pages

Configuration: Release Platform: Active(Win32)

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation

Additional Include Directories	$(SolutionDir)..\src;$(Soluti
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 **find_hotspots Property Pages** dialog box, select **Linker > Debugging** and set the **Generate Debug Info** option to **Yes (/DEBUG)**.



find_hotspots Property Pages

Configuration: Release Platform: Active(Win32)

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
    - General
    - Input
    - Manifest File
    - Debugging

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 find_hotspots**.

   The `find_hotspots.exe` application is built.

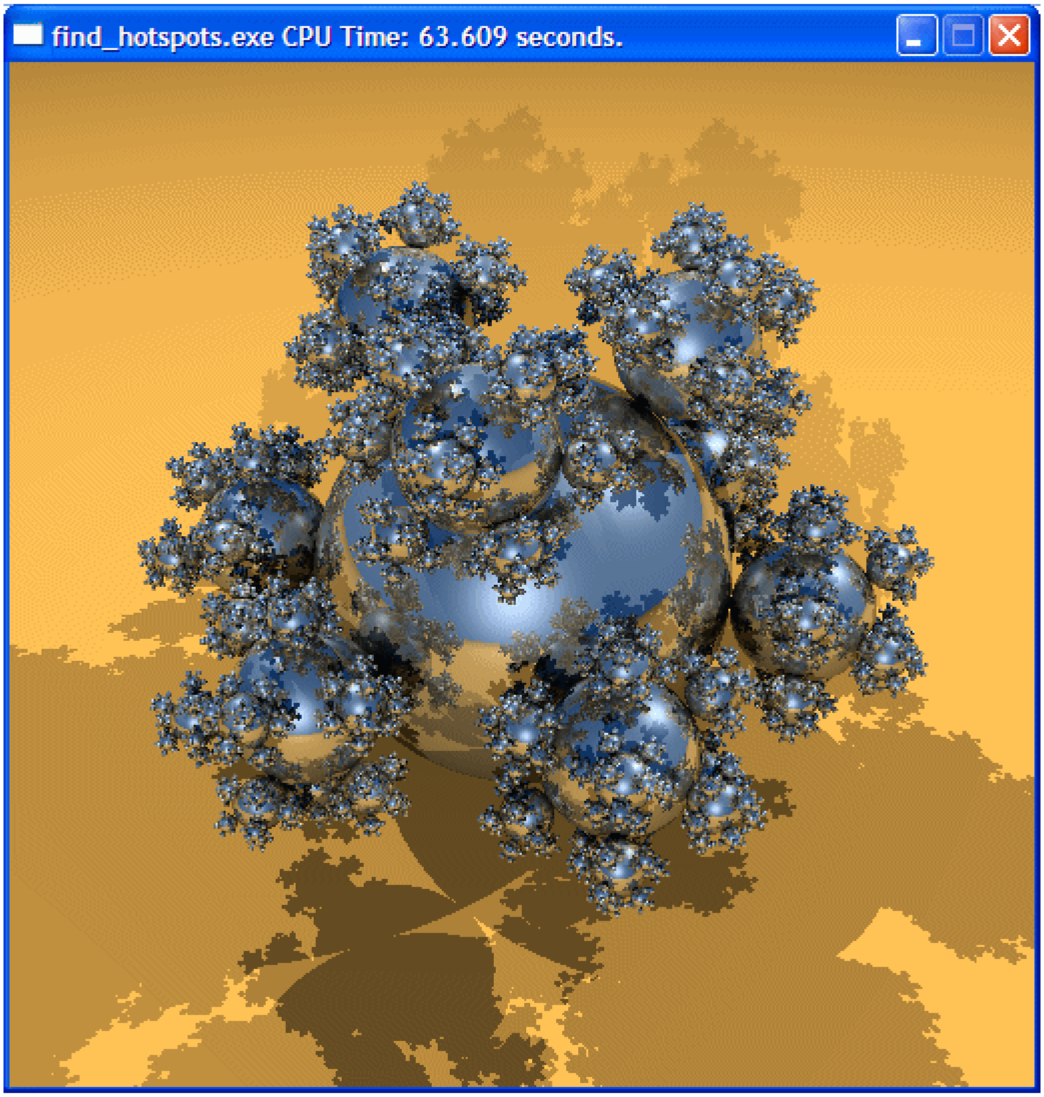
NOTE The build configuration for `tachyon` may initially be set to Debug, which is typically used for development. When analyzing performance issues with the VTune Amplifier XE, you are recommended to use the Release build with normal optimizations. In this way, the VTune Amplifier XE is able to analyze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

Create a Performance Baseline

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

   The `find_hotspots.exe` application starts running.

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



find_hotspots.exe CPU Time: 63.609 seconds.

2. Note the execution time displayed in the window caption. For the `find_hotspots.exe` executable in the figure above, the execution time is 63.609 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. From the **Start** menu select **Intel Parallel Studio XE 2013** > **Intel VTune Amplifier XE 2013** 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 the `tachyon` project directory under the `%USERPROFILE%\My Documents \Amplifier\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 `<`*`sample_code_dir`*`>\find_hotspots.exe`, for example: `C:\samples\tachyon\vc9\find_hotspots_Win32_Release\find_hotspots.exe`.
5. 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

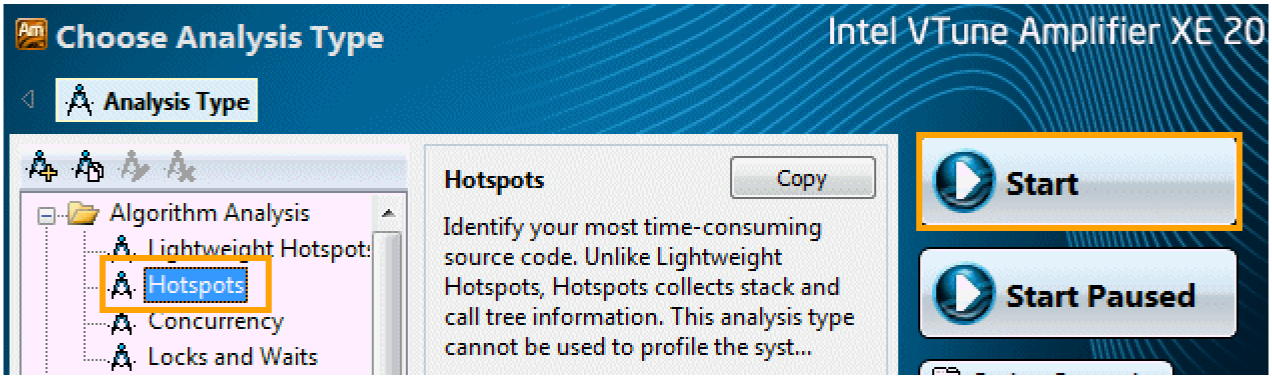
In this tutorial, you run the Hotspots analysis to identify the hotspots that took much time to execute.

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 the syst...

VTune Amplifier XE launches the `find_hotspotstachyon_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

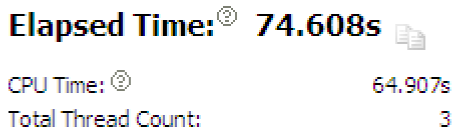
When the sample application exits, the Intel® VTune™ Amplifier XE finalizes the results and opens the **Hotspots** viewpoint where each window or pane is configured to display code regions that consumed a lot of CPU time. To interpret the data on the sample code performance, do the following:

- 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.

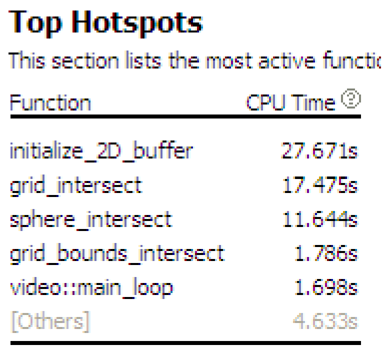


Elapsed Time: 74.608s

CPU Time:	64.907s
Total Thread Count:	3

Note that **CPU Time** for the sample application is equal to 64.907 seconds. It is the sum of CPU time for all application threads. **Total Thread Count** is 3, so the sample application is multi-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 functic

Function	CPU Time
initialize_2D_buffer	27.671s
grid_intersect	17.475s
sphere_intersect	11.644s
grid_bounds_intersect	1.786s
video::main_loop	1.698s
[Others]	4.633s

For the sample application, the `initialize_2D_buffer` function, which took 27.671 seconds to execute, shows up at the top of the list as the hottest function.

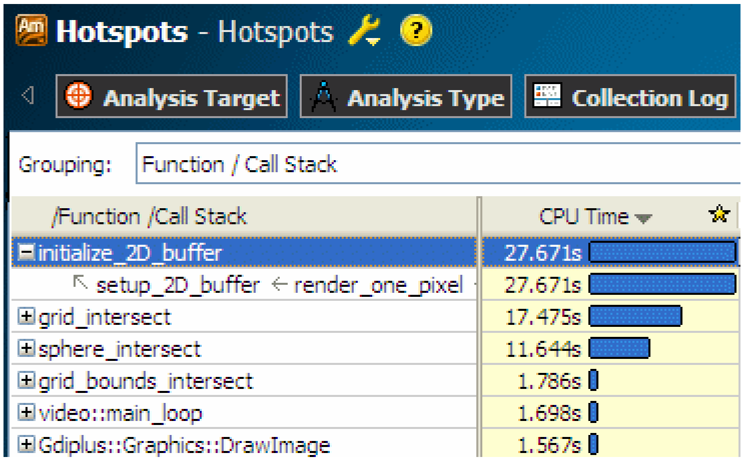
The `[Others]` entry at the bottom shows the sum of CPU time for all functions not listed in the table.

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 27.671 seconds to execute. Click the plus 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.



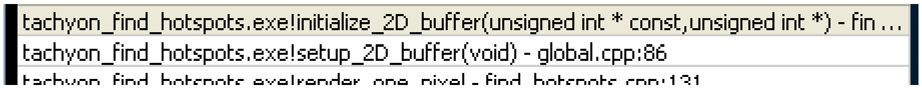
Grouping: Function / Call Stack

Function / Call Stack	CPU Time ★
⊟initialize_2D_buffer	27.671s
↖ setup_2D_buffer ← render_one_pixel	27.671s
⊞grid_intersect	17.475s
⊞sphere_intersect	11.644s
⊞grid_bounds_intersect	1.786s
⊞video::main_loop	1.698s
⊞Gdiplus::Graphics::DrawImage	1.567s

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:

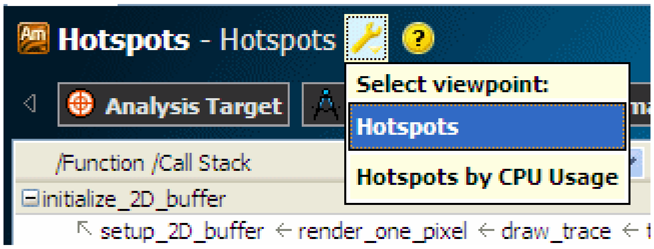
<module>!<function> - <file>:<line number>, where the line number corresponds to the line calling the next function in the stack.



```
tachyon_find_hotspots.exe!initialize_2D_buffer(unsigned int * const,unsigned int *) - fin ...
tachyon_find_hotspots.exe!setup_2D_buffer(void) - global.cpp:86
tachyon_find_hotspots.exe!render_one_pixel - find_hotspots.cpp:131
```

For the sample application, the hottest function `initialize_2D_buffer` is called at line 86 of the `setup_2D_buffer` function in the `global.cpp` file.

Analyze CPU Usage per Function

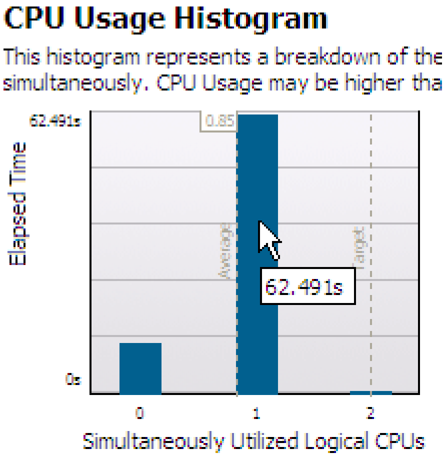


VTune Amplifier XE enables you to analyze the collected data from different perspectives by using multiple viewpoints.

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. Ideally, the highest bar of your chart should match the Target level.

The `find_hotspots` application ran mostly on one logical CPU. If you hover over the highest bar, you see that it spent 62.491 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.

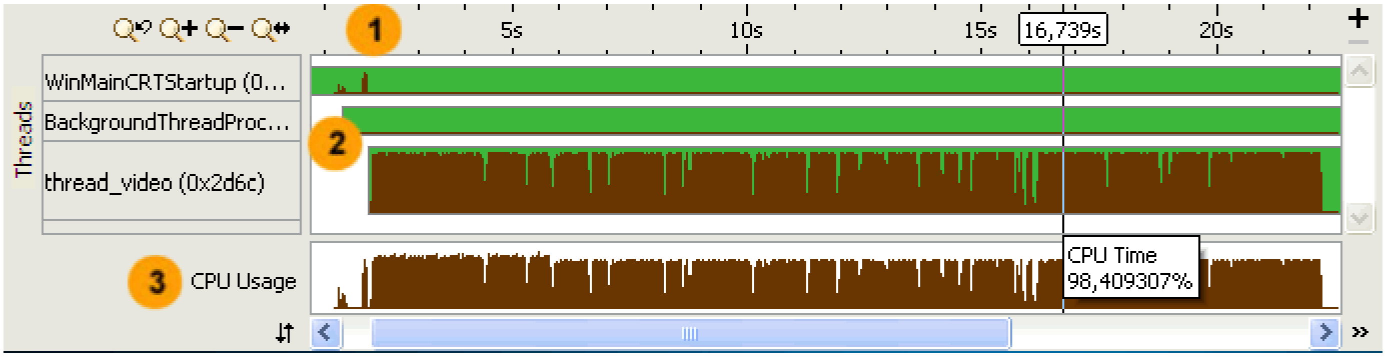

CPU Usage Histogram

This histogram represents a breakdown of the simultaneously. CPU Usage may be higher tha

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.

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 rarely exceed 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 `find_hotspots` application.

Key Terms

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

Next Step

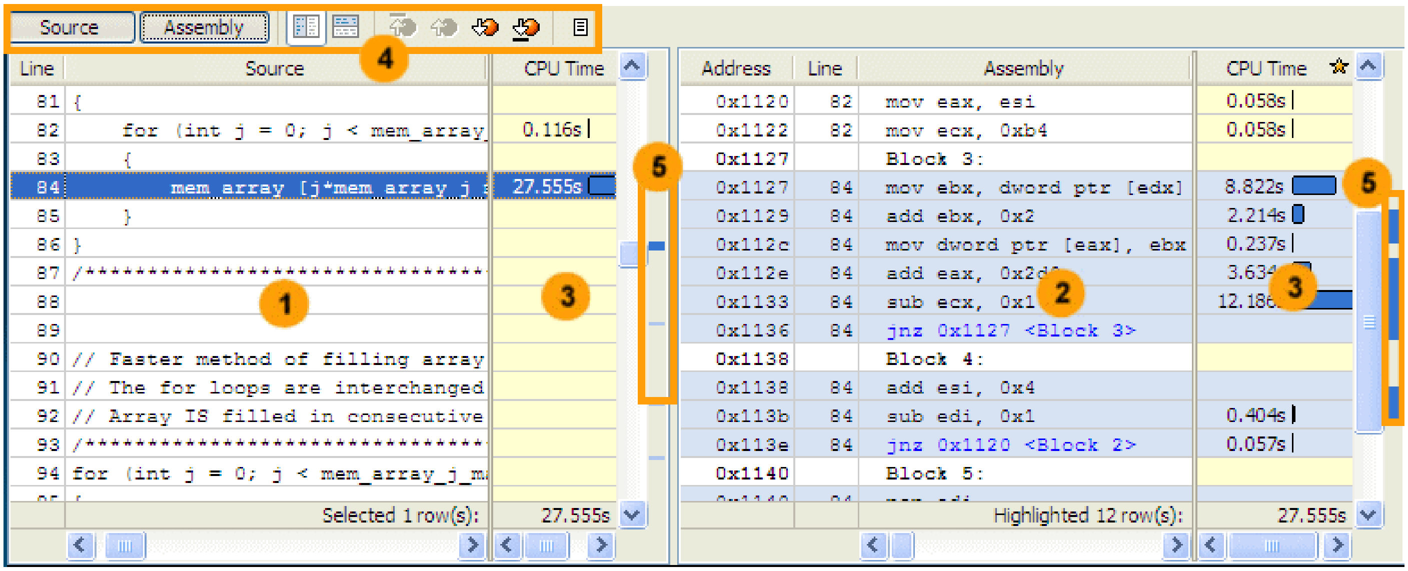
Analyze Code

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
81	{	
82	for (int j = 0; j < mem_array_	0.116s
83	{	
84	mem_array [j*mem_array_j_	27.555s
85	}	
86	}	
87	/***********************************	
88		
89		
90	// Faster method of filling array	
91	// The for loops are interchanged	
92	// Array IS filled in consecutive	
93	/***********************************	
94	for (int j = 0; j < mem_array_j_ma	
	Selected 1 row(s):	27.555s

Address	Line	Assembly	CPU Time
0x1120	82	mov eax, esi	0.058s
0x1122	82	mov ecx, 0xb4	0.058s
0x1127		Block 3:	
0x1127	84	mov ebx, dword ptr [edx]	8.822s
0x1129	84	add ebx, 0x2	2.214s
0x112c	84	mov dword ptr [eax], ebx	0.237s
0x112e	84	add eax, 0x2d	3.634
0x1133	84	sub ecx, 0x1	12.186
0x1136	84	jnz 0x1127 <Block 3>	
0x1138		Block 4:	
0x1138	84	add esi, 0x4	
0x113b	84	sub edi, 0x1	0.404s
0x113e	84	jnz 0x1120 <Block 2>	0.057s
0x1140		Block 5:	
		Highlighted 12 row(s):	27.555s

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 84. 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

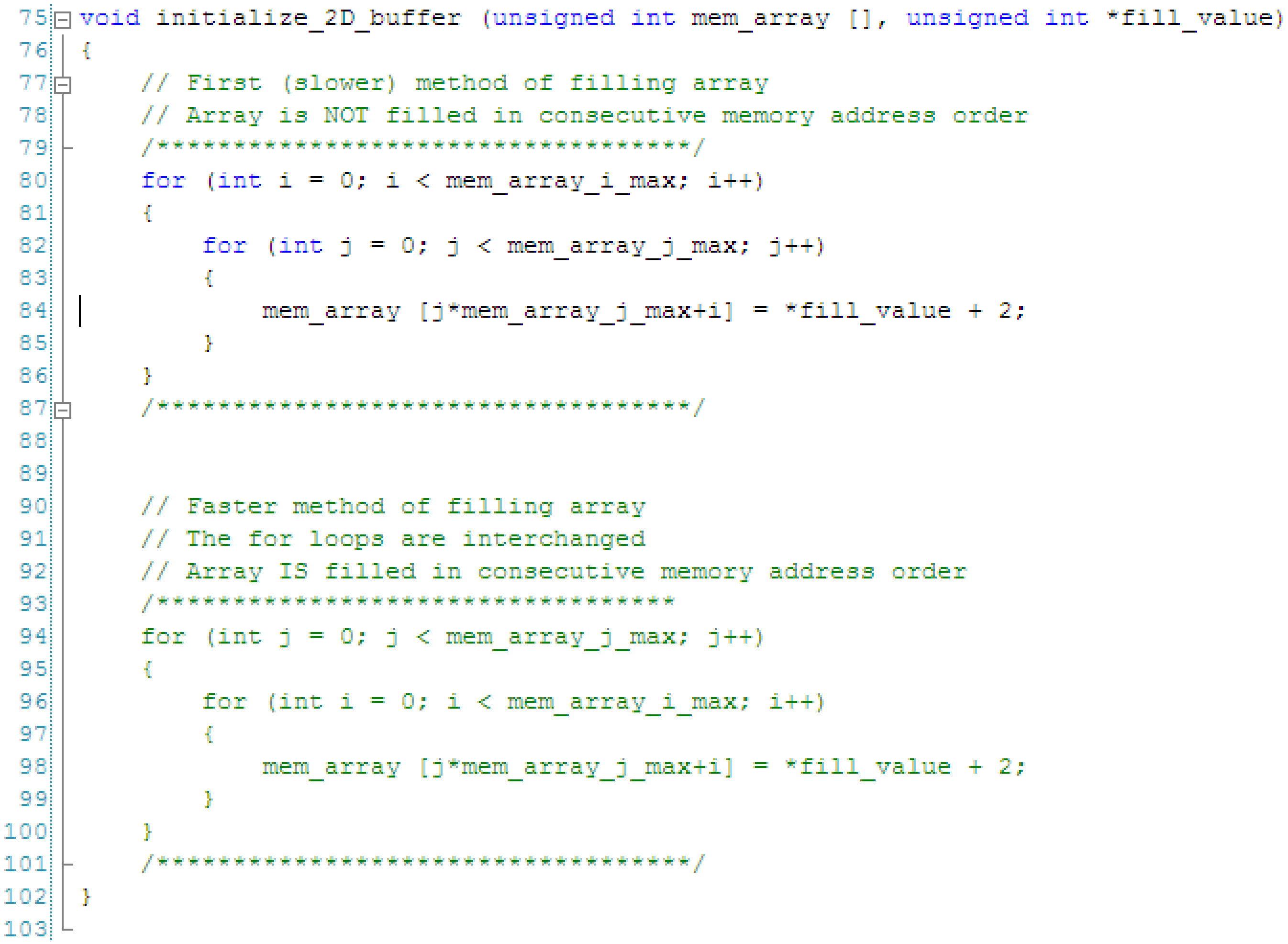
Tune Algorithms

In the **Source** window, you identified that in the `initialize_2D_buffer` hotspot function the code line 84 took the most CPU time. Focus on this line and do the following:

1. Open the code editor.
2. Resolve the performance problem using any of these options:
   - Optimize the algorithm used in this code section.
   - Recompile the code with the Intel® Compiler.

Open the Code Editor

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



```
75  void initialize_2D_buffer (unsigned int mem_array [], unsigned int *fill_value)
76  {
77      // First (slower) method of filling array
78      // Array is NOT filled in consecutive memory address order
79      /***********************************/
80      for (int i = 0; i < mem_array_i_max; i++)
81      {
82          for (int j = 0; j < mem_array_j_max; j++)
83          {
84              mem_array [j*mem_array_j_max+i] = *fill_value + 2;
85          }
86      }
87      /***********************************/
88
89
90      // Faster method of filling array
91      // The for loops are interchanged
92      // Array IS filled in consecutive memory address order
93      /**********************************
94      for (int j = 0; j < mem_array_j_max; j++)
95      {
96          for (int i = 0; i < mem_array_i_max; i++)
97          {
98              mem_array [j*mem_array_j_max+i] = *fill_value + 2;
99          }
100     }
101     /***********************************/
102 }
103
```

Hotspot line 84 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, use one of the following methods:

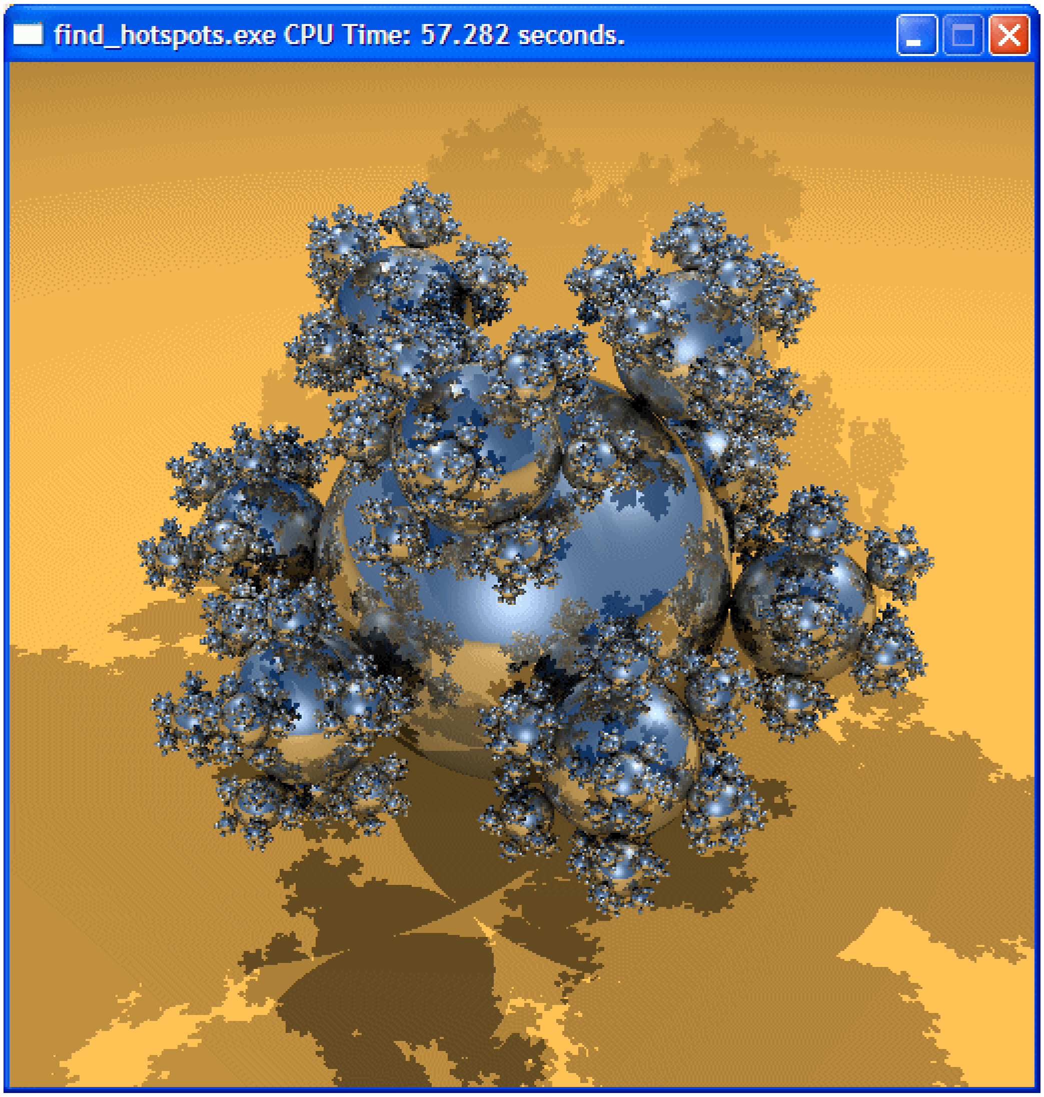
Option 1: Optimize your algorithm

1. Edit line 79 to comment out code lines 82-88 marked as a "First (slower) method".
2. Edit line 95 to uncomment code lines 98-104 marked as a "Faster method".

   In this step, you interchange the `for` loops to initialize the code in sequential memory locations.

3. From the Visual Studio menu, select **Build > Rebuild find_hotspots**.

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



Visual Studio runs the `find_hotspots.exe`. Note that execution time has reduced from 63.609 seconds to 57.282 seconds.

Option 2: Recompile the code with Intel® Compiler

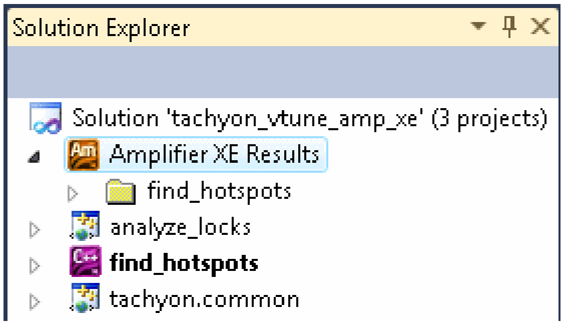
This option assumes that you have Intel® Composer XE installed. Intel Composer XE is part of Intel® Parallel Studio XE. By default, the Intel® Compiler, one of the Intel Composer components, uses powerful optimization switches, which typically provides some gain in performance. For more details on the Intel compiler, see the Intel Composer documentation.

As an alternative, you may consider running the default Microsoft Visual Studio compiler applying more aggressive optimization switches.

To recompile the code with the Intel compiler:

1. From Visual Studio **Project** menu, select **Intel Composer XE <*version*> > Use Intel C++...**.
2. In the **Confirmation** window, click **OK** to confirm your choice.

   The project in Solution Explorer appears with the Intel Composer XE icon:



Solution Explorer

Solution 'tachyon_vtune_amp_xe' (3 projects)

3. From the Visual Studio menu, select **Build > Rebuild find_hotspots**.

   The project is rebuilt with the Intel compiler.

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

   Visual Studio runs the `find_hotspots.exe`. Note that the execution time has reduced.

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

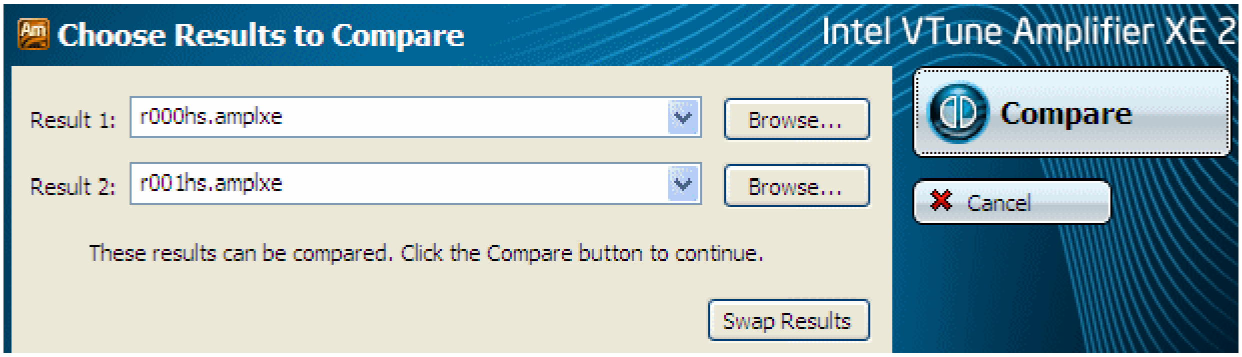
You optimized your code to apply a loop interchange mechanism that gave you 6 seconds of improvement in the application execution time. To understand whether you got rid of the hotspot and what kind of optimization you got per function, re-run the Hotspots analysis on the optimized code and compare results:

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:

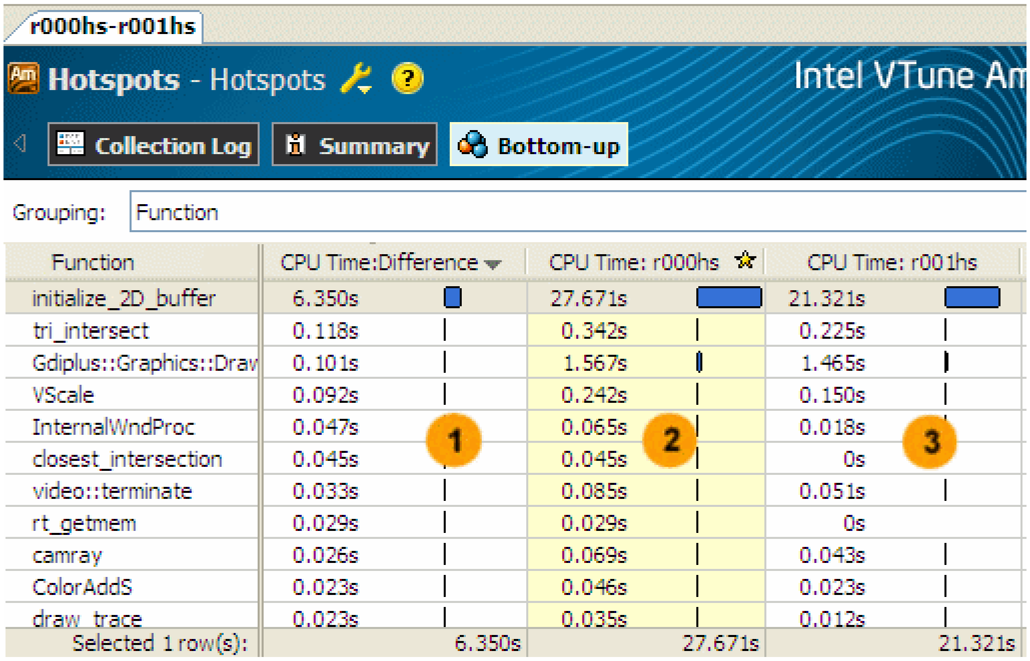


Result 1: r000hs.amplxe

Result 2: r001hs.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: r000hs	CPU Time: r001hs
initialize_2D_buffer	6.350s	27.671s	21.321s
tri_intersect	0.118s	0.342s	0.225s
Gdiplus::Graphics::Draw	0.101s	1.567s	1.465s
VScale	0.092s	0.242s	0.150s
InternalWndProc	0.047s	0.065s	0.018s
closest_intersection	0.045s	0.045s	0s
video::terminate	0.033s	0.085s	0.051s
rt_getmem	0.029s	0.029s	0s
camray	0.026s	0.069s	0.043s
ColorAddS	0.023s	0.046s	0.023s
draw_trace	0.023s	0.035s	0.012s
Selected 1 row(s):	6.350s	27.671s	21.321s

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

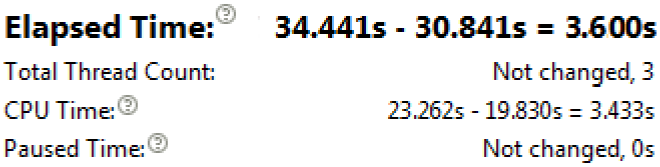
2 CPU time for the initial version of the `find_hotspots.exe` application.

3 CPU time for the optimized version of the `find_hotspots.exe`.

Identify the Performance Gain

Explore the **Bottom-up** pane to compare CPU time data for the first hotspot: CPU Time:r000hs - CPU Time:r001hs = CPU Time: Difference. 27.671s - 21.321s = 6.350s, which means that you got the optimization of ~6 seconds for the `initialize_2D_buffer` function.

If you switch to the **Summary** window, you see that the Elapsed time also shows 3.6 seconds of optimization for the whole application execution:



Elapsed Time:	**34.441s - 30.841s = 3.600s**
Total Thread Count:	Not changed, 3
CPU Time:	23.262s - 19.830s = 3.433s
Paused Time:	Not changed, 0s

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	If you used the Visual Studio* IDE: You chose the target for the Hotspots analysis, set up your environment to enable generating symbol information for system libraries and your binary files, built the target in the Release mode, and created the performance baseline. 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.	• 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. • 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 6 seconds. You also considered an alternative optimization technique using the Intel C++ compiler.	Consider using Intel® Compiler, part of the Intel® Composer XE, to optimize your tuning algorithms. Explore the compiler documentation for more details.

Step	Tutorial Recap	Key Tutorial Take-aways
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 `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.	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.

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

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.

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.