Tutorial: Identifying Hardware Issues

Intel® VTune™ Amplifier for Windows® OS
C++ Sample Application Code

Legal Information
Contents

Legal Information.............................................................. 3
Overview................................................................................. 4

Chapter 1: Navigation Quick Start

Chapter 2: Identifying Hardware Issues
  Visual Studio* IDE: Choose Project and Build Application............. 10
  Build Application........................................................................ 14
  Configure and Run Analysis....................................................... 18
  Interpret Results......................................................................... 20
  Analyze Code........................................................................... 22
  Resolve Issue............................................................................. 24
  Resolve Next Issue................................................................. 25

Chapter 3: Summary

Chapter 4: Key Terms
**Legal Information**

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors which may cause deviations from published specifications. Current characterized errata are available on request.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: Learn About Intel® Processor Numbers

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Cilk, Intel, the Intel logo, Intel Atom, Intel Core, Intel Inside, Intel NetBurst, Intel SpeedStep, Intel vPro, Intel Xeon Phi, Intel XScale, Itanium, MMX, Pentium, Thunderbolt, Ultrabook, VTune and Xeon 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.

© 2016, Intel Corporation.
Overview

Discover how to use General Exploration analysis of the Intel® VTune™ Amplifier to identify hardware-related issues in your application such as data sharing, cache misses, branch misprediction, and others.

About This Tutorial
This tutorial uses the sample matrix application and guides you through basic steps required to analyze the code for general hardware issues on the Intel® microarchitecture code name Haswell.

Estimated Duration
10-15 minutes.

Learning Objectives
After you complete this tutorial, you should be able to:
- Choose an analysis target.
- Run the General Exploration analysis.
- Understand the event-based performance metrics.
- Identify the most critical hardware issues for the application as a whole.
- Identify the modules/functions that caused the most critical hardware issues.
- Analyze the source code to locate the most critical code lines.
- Identify the next steps in performance analysis.

More Resources

Start Here
Navigation Quick Start

Intel® VTune™ Amplifier provides information on code performance for users developing serial and multithreaded applications on Windows®, Linux®, Android, and OS X® operating systems. VTune Amplifier 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 in the Visual Studio® IDE: From the Windows® Start menu, choose Intel Parallel Studio XE version > Parallel Studio XE version with [VS2012 | VS2013 | VS2015].

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

- From the Windows® Start menu, choose Intel Parallel Studio XE version > Intel VTune Amplifier XE version.
- From the Windows® Start menu, choose Intel Parallel Studio XE version > Command Prompt > Parallel Studio XE > IA-32 Visual Studio [2012 | 2013 | 2015] mode to set your environment, then type amplxe-gui. You can also set the environment manually by running the <install-dir>\amplxe-vars.bat file.

VTune Amplifier for Systems Access

To access the VTune Amplifier in the Visual Studio® IDE: From the Windows® Start menu, choose Intel Parallel Studio XE version > Parallel Studio XE version with [VS2012 | VS2013 | VS2015].

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

- From the Windows® Start menu, choose Intel VTune Amplifier <version> for Systems
- From Eclipse®, click the Launch Intel VTune Amplifier button from the toolbar or select the Intel System Studio > VTune Amplifier > Launch VTune Amplifier menu option
VTune Amplifier/Visual Studio* IDE Integration

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

VTune Amplifier results *.amplxe show up in the Solution Explorer under the 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 version menu from the pop-up menu. To manage previously collected results, right-click the result (for example, r000hs.amplxe) and select the required command from the pop-up menu.

Click the (change) link 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 buttons on navigation toolbars to change window views and toggle window panes on and off.

In the Timeline pane, analyze the thread activity and transitions presented for the user-mode sampling and tracing analysis results (for example, Basic 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).

Use the Call Stack pane to view call paths for a function selected in the grid.
Use the filter toolbar to filter out the result data according to the selected categories.

**Standalone VTune Amplifier GUI**

Configure and manage projects and results, and launch new analyses from the primary toolbar. Click the **Configure Project** button on this toolbar and use the **Analysis Target** tab to manage result file locations. Newly completed and opened analysis results along with result comparisons appear in the results tab for easy navigation.

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

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**.

Click the **(change)** link 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.

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

Use the **Grouping** drop-down menu to choose a granularity level for grouping data in the grid.
Use the filter toolbar to filter out the result data according to the selected categories.

See Also
Click here for more Getting Started Tutorials
Identifying Hardware Issues

Intel® VTune™ Amplifier embodies powerful event-based sampling methods that enable you to identify hardware issues that have a significant impact on the performance of your application. This tutorial guides you through workflow steps running General Exploration analysis type on a sample application, *matrix*.

### Step 1: Prepare for analysis
Configure the development environment and build application in the Release mode.

### Step 2: Find hardware issues
- Configure a VTune Amplifier project and run General Exploration analysis.
- Explore event-based hardware metrics, identify a performance baseline, and interpret the result data.
- View and analyze code of the performance-critical function.

### Step 3: Resolve detected issues
- Modify the code to resolve the detected performance issues and rebuild the code enabling the vectorization option of the Intel compiler.
- Use more advanced algorithms to optimize the performance and verify optimization.

### 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-
Optimization Notice

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

Visual Studio* IDE: Choose Project and Build Application

Before you start analyzing hardware issues affecting the performance of your application, 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.

NOTE:
- The steps below are provided for Microsoft Visual Studio* 2013. Steps for other versions of Visual Studio IDE or for the standalone version of the Intel® VTune™ Amplifier may slightly differ.
- 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 matrix sample application:

- VTune Amplifier, including sample applications
- zip file extraction utility
- Supported compiler (see Release Notes for more information). Intel C++ compiler is recommended.

Acquire Intel VTune Amplifier

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

Install and Set Up VTune Amplifier Sample Applications

1. Copy the matrix_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 [Program Files]\IntelSWTools\VTune Amplifier XE 2016\.
2. Extract the sample from the .zip file.

NOTE:
- 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 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 where you extracted the matrix_vtune_amp_xe.zip file and select the matrix.sln file.

   The solution is added to Visual Studio and shows up in the Solution Explorer. VTune Amplifier automatically inherits Visual Studio settings and uses the currently opened project as a target project for performance analysis.

   When you choose a project in Visual Studio IDE, the VTune Amplifier automatically creates the config.amplxeproj project file and sets the matrix 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 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.
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 can open the source code.

1. Select the **matrix** project and go to **Project > Properties**.
2. From the **matrix Property Pages** dialog box, select **Configuration Properties > General** and make sure the selected **Configuration** (top of the dialog) is **Active(Release)**.
3. From the **matrix Property Pages** dialog box, select **C/C++ > General** pane and specify the **Debug Information Format** as **Program Database (/Zi)**.
4. From the **matrix Property Pages** dialog box, select **Linker > Debugging** and set the **Generate Debug Info** option to **Yes (/DEBUG)**.

5. Click **OK**.

**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 matrix**.

The **matrix.exe** application is built.

**Key Terms**

**Target**
Next Step
Run General Exploration 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

Build Application

Am

Before you start analyzing hardware issues affecting the performance of your application, do the following:

1. Get software tools.
2. If you work in Visual Studio® IDE, open the project.
3. 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.

Get Software Tools

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

- VTune Amplifier, including sample applications
- zip file extraction utility
- Supported compiler (see Release Notes for more information). For this tutorial, Intel® C++ compiler is used to build the application.
- Text editor

Acquire Intel VTune Amplifier

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

NOTE:
This document focuses on using the VTune™ Amplifier in Intel® Parallel Studio XE Professional Edition. You may see minor differences if you installed a different Intel product. For more information on product capabilities in your installed product, see the product-specific supplemental documentation in <install-dir>/<Intel_product>/documentation/.

Install and Set Up VTune Amplifier Sample Applications
1. Copy the `matrix_vtune_amp_xe.zip` file from the `<install_dir>\samples<locale>C++` directory to a writable directory or share on your system.

**NOTE:**
The default installation path for the VTune Amplifier XE is `[Program Files]\IntelSWTools\VTune Amplifier XE <version>`. For the VTune Amplifier for Systems, the default `<install_dir>` is `[Program Files]\IntelSWTools\system_studio_<version>\VTune Amplifier for Systems`.

2. Extract the sample from the `.zip` file.

**NOTE:**
- 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 features; they do not represent best practices for creating code.

### Open a Project in Visual Studio* IDE

**NOTE:**
The steps below are provided for Microsoft Visual Studio* 2013. Steps for other versions of Visual Studio IDE may differ slightly.

If you work in the Microsoft Visual Studio* IDE, open the sample project 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 where you extracted the `matrix_vtune_amp_xe.zip` file and select the `matrix.sln` file.
The solution is added to Visual Studio and shows up in the Solution Explorer. VTune Amplifier automatically inherits Visual Studio settings and uses the currently opened project as a target project for performance analysis.

When you choose a project in Visual Studio IDE, the VTune Amplifier automatically creates the config.amplxe.proj project file and sets the matrix 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 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.

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 can open the source code.

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

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

5. Click Ok.

Build the Target in the Release Mode

Build the target in the Release mode with full optimizations, which is recommended for the performance analysis.
1. From the Visual Studio menu, select **Project > Intel Compiler > Use Intel C++**.
2. Go to the **Build > Configuration Manager** dialog box and select the **Release** mode for your target project.
3. Click **Close**.
4. From the Visual Studio menu, select **Build > Build matrix**.
   The matrix.exe application is built.

**Key Terms**

**Target**

**Next Step**

**Configure and Run Analysis**

<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>Notice revision #20110804</td>
</tr>
</tbody>
</table>

**Configure and Run Analysis**

After building the sample application, configure the Intel VTune Amplifier for the General Exploration analysis as follows:

1. For standalone GUI:
   - **Create a VTune Amplifier project**, which is a container for an analysis target and analysis type configuration and data collection results.
   - **Specify your target application for analysis**.
2. **Select and run the General Exploration analysis** to discover what parts of the application code are being most used and see what hardware resources are being used there.

**Create a Project (Standalone GUI only)**

1. Execute the amplxe-vars script, located in the product installation directory, to set up the environment.
2. From the **Start** launch **Intel VTune Amplifier version**.
3. Click the menu button and select **New > Project**... to create a new project.
   The **Create a Project** dialog box opens.
4. Specify the project name **matrix** that will be used as the project directory name and click the **Create Project** button.
   By default, the VTune Amplifier creates a project directory under the %USERPROFILE%\My Documents \Amplifier XE\Projects (for VTune Amplifier XE) or %USERPROFILE%\My Documents\Amplifier for Systems\Projects (for VTune Amplifier for Systems) directory and opens the **New Amplifier Result** tab with the **Analysis Target** sub-tab active.
Specify Analysis Target

To specify your application as analysis target in the standalone VTune Amplifier GUI, configure the **Analysis Target** tab:

1. From the left pane, select the **local** target system from the **Accessible Targets** group.
2. From the right pane, select the **Launch Application** type of the analysis target drop-down menu.
3. In the **Application** field, click the **Browse** button and navigate to the *matrix.exe* application.
4. Click the **Choose Analysis** button on the right to switch to the **Analysis Type** tab.

Configure and Run Analysis Type

1. In the **Analysis Type** tab, select the **Microarchitecture Analysis > General Exploration** analysis type from the analysis tree on the left.
2. Click the **Start** button on the right to run the analysis.

General Exploration analysis collects event and IP information over a wide net of hardware issues that might affect application performance.

VTune Amplifier launches the *matrix* application that calculates matrix transformations before exiting. When the application exits or after a predefined interval, depending on how the collection run was configured, collection is completed and the VTune Amplifier enters the **finalization** process, where data are coalesced, symbols are reconnected to their addresses, and certain data are cached to speed the display of results.
NOTE:
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.

**Key Terms**
- Finalization
- Viewpoint

**Next Step**
Interpret Results

## Interpret Results

When the application exits, the Intel® VTune™ Amplifier finalizes the results and opens the Hardware Issues viewpoint. To interpret the collected data and understand where you should focus your tuning efforts for the specific hardware, do the following:

1. Understand the event-based metrics
2. Identify the hardware issues that affect the performance of your application

**NOTE:**
- The screenshots and execution time data provided in this tutorial are created on a system with 4 CPU cores, based on the Intel microarchitecture code name Haswell. Your data may vary depending on the number and type of CPU cores on your system.
- The screenshots and execution time data provided in this tutorial are created for a sample code compiled with Intel® C++ Compiler. Your data may vary depending on the compiler you use.

### Understand the Event-based Metrics

Click the **Summary** tab to explore the data provided in the **Summary** window for the whole application performance:

**Elapsed Time**\(^\text{①}\): **82.997s**
- **Clockticks**: 520,119,300,000
- **Instructions Retired**: 65,722,900,000
- **CPI Rate**\(^\text{②}\): 7.914
- **MUX Reliability**\(^\text{③}\): 0.999
- **Front-End Bound**\(^\text{④}\): 2.9%
- **Bad Speculation**\(^\text{⑤}\): 0.8%
- **Back-End Bound**\(^\text{⑥}\): 93.3%
- **Receiving**\(^\text{⑦}\): 3.0%
- **Total Thread Count**: 5
- **Paused Time**\(^\text{⑧}\): 0s
The **Elapsed time** metric shows the wall time from the beginning to the end of the collection. Treat this metric as your basic performance **baseline** against which you will compare subsequent runs of the application. The goal of your optimization is to reduce the value of this metric. All other metrics in this section are hardware event ratios provided by Intel architects. Mouse over the icon to see the metric description and formula used for the metric calculation. VTune Amplifier highlights metrics values that exceed the threshold set for the corresponding metric. Such a value highlighted in pink signifies an application-level hardware issue. The text below a metric with the detected hardware issue describes the issue, potential cause and recommendations on the next steps, and displays a threshold formula used for calculation. Mouse over the truncated text to read a full description.

Quick look at the summary results discovers that the **matrix** application has the following issues:

- CPI (Clockticks per Instructions Retired) Rate
- Back-End Bound

**Identify the Hardware Issues**

Click the **Bottom-up** tab to open the **Bottom-up** window and see how each program unit performs against the event-based metrics. Each row represents a program unit and percentage of the CPU cycles used by this unit. Program units that take more than 5% of the CPU time are considered hotspots. This means that by resolving a hardware issue that, for example, took about 20% of the CPU cycles, you can obtain 20% optimization for the hotspot.

By default, the VTune Amplifier sorts data in the descending order by Clockticks and provides the hotspots at the top of the list. The metric values for event ratios show up as numbers and/or bars. To change the data format, right-click a column and select **Show Data As > format**.

You see that the **multiply1** function is the most obvious hotspot in the **matrix** application. It has the highest event count (Clockticks and Instructions Retired events) and most of the hardware issues were also detected during execution of this function.

**NOTE:**

Mouse over a column header with an event-based metric name to see the metric description. Mouse over a highlighted cell to read the description of the hardware issue detected for the program unit.

For the **multiply1** function, the VTune Amplifier highlights the same issues that were detected as the issues affecting the performance of the whole application:
• **CPI Rate** is high (>1). Potential causes are memory stalls, instruction starvation, branch misprediction, or long-latency instruction. To define the cause for your code, explore other metrics in the Bottom-up window.

• The **Back-End Bound** metric describes a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

**Key Terms**

- Baseline
- Elapsed time
- Event-based metrics
- Viewpoint

**Next Step**

Analyze Code

**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

**Analyze Code**

You identified a hotspot function with a number of hardware issues. Double-click the `multiply1` function in the **Bottom-up** window to open the source code:
The table below explains some of the features available in the Source pane when viewing the event-based sampling analysis data.

1. **Assembly** button to toggle in the Assembly pane that displays assembly instructions for the selected function.

2. Hotspot navigation buttons to switch between code lines that took a long time to execute.

3. Source file editor button to open and edit your code in the default editor.

4. **Source** pane displaying the source code of the application, which is available if the function symbol information is available. The hottest code line is highlighted. The source code in the Source pane is not editable.

5. Values per hardware event attributed to a particular code line. By default, the data is sorted by the Clockticks event count. Focus on the events that constitute the metrics identified as performance-critical in the Bottom-up window. To identify these events, mouse over the metric column header in the Bottom-up window. Drag-and-drop the columns to organize the view for your convenience. VTune Amplifier remembers your settings and restores them each time you open the viewpoint.

When you drill-down from the grid to the source view, the VTune Amplifier automatically highlights the code line that has the highest event count. In the Source pane for the `multiply1` function, you see that line 51 took the most of the Clockticks event samples during execution. This code section multiplies matrices in the loop but ineffectively accesses the memory. Focus on this section and try to reduce the memory issues.

**Key Terms**

Event skid

**Next Step**

Resolve Issue
Resolve Issue

In the Source pane, you identified that in the multiply1 function the code line 51 resulted in the highest values for the Clockticks event. To solve this issue, do the following:

1. Change the multiplication algorithm.
2. Re-run the analysis to verify optimization.

Change Algorithm

NOTE:
The proposed solution is one of the multiple ways to optimize the memory access and is used for demonstration purposes only.

1. Open the multiply.h file from the Source Files of the matrix project.

For this sample, the multiply.h file is used to define the functions used in the multiply.c file.

2. In line 36, replace the multiply1 function name with the multiply2 function.

This new function uses the loop interchange mechanism that optimizes the memory access in the code.

Intel compiler helps vectorize the data, which means that it uses SIMD instructions that can work with several data elements simultaneously. If only one source file is used, the Intel compiler enables vectorization automatically. The current sample uses several source files, that is why the multiply2 function uses #pragma ivdep to instruct the compiler to ignore assumed vector dependencies. This information lets the compiler enable the Supplemental Streaming SIMD Extensions (SSSE).

3. Save files and rebuild the project using the compiler of your choice.
From Visual Studio menu, select **Build > Rebuild matrix**.

**Verify Optimization**

1. Re-run the General Exploration analysis:

   - From Visual Studio IDE: From the VTune Amplifier toolbar, click the down arrow next to the **New Analysis** button and select **General Exploration** from the drop-down menu.
   - From the Standalone GUI: From the **File** menu, select **New > General Exploration**.

   VTune Amplifier reruns the General Exploration analysis for the updated matrix target and creates a new result, r001ge, that opens automatically.

2. In the r001ge result, click the **Summary** tab to see the Elapsed time value for the optimized code:

   ![Elapsed Time](image)

   You see that the Elapsed time has reduced from 82.997 seconds to 7.776 seconds. CPI Rate and Back-End Bound is still an issue though has reduced significantly.

**Key Terms**

- Elapsed time
- Event-based metrics

**Next Step**

Resolve Next Issue

---

**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

---

**Resolve Next Issue**
You got a significant performance boost by optimizing the memory access for the multiply1 function. According to the data provided in the Summary window for your updated result, r001ge, you still have high CPI rate and issues. You can try to optimize your code further following the steps below:

1. Analyze results after optimization
2. Use more advanced algorithms
3. Verify optimization

**Analyze Results after Optimization**

To get more details on the issues that still affect the performance of the matrix application, switch to the Bottom-up window in the Hardware Issues viewpoint:

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>Clockticks</th>
<th>Instructions Retired</th>
<th>CPI Rate</th>
<th>Back-End Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply2</td>
<td>42,580,900,000</td>
<td>24,884,300,000</td>
<td>1.711</td>
<td></td>
</tr>
<tr>
<td>KiInterruptSubDispatchNoLockNoEtw</td>
<td>66,500,000</td>
<td>1,900,000</td>
<td>35.000</td>
<td>100.0%</td>
</tr>
<tr>
<td>KeRetireDpcList</td>
<td>64,600,000</td>
<td>7,600,000</td>
<td>8.500</td>
<td>70.6%</td>
</tr>
<tr>
<td>KeClockInterruptNotify</td>
<td>51,300,000</td>
<td>5,700,000</td>
<td>9.000</td>
<td>81.5%</td>
</tr>
<tr>
<td>SwapContext_PatchLdMxCsr</td>
<td>32,300,000</td>
<td>0</td>
<td></td>
<td>70.6%</td>
</tr>
</tbody>
</table>

You see that the multiply2 function (in fact, updated multiply1 function) is still a hotspot. Double-click this function to view the source code and click both the Source and Assembly buttons on the toolbar to enable the Source and Assembly panes.
In the Source pane, the VTune Amplifier highlights line 66 that took the highest number of Clockticks samples. This is again the section where matrices are multiplied. The Assembly pane is automatically synchronized with the Source pane. It highlights the basic blocks corresponding to the code line highlighted in the Source pane. If you compiled the application with the Intel® Compiler, you can see that highlighted block 20 includes vectorization instructions added after your previous optimization. All vectorization instructions have the p (packed) postfix (for example, mulpd). You may use the /Qvec-report3 option of the Intel compiler to generate the compiler optimization report and see which cycles were not vectorized and why. For more details, see the Intel compiler documentation.

Use More Advanced Algorithms

1. Open the multiply.h file from the Source Files of the matrix project.
2. In line 36, replace the multiply2 function name with the multiply3 function.

This function enables uploading the matrix data by blocks.
3. Save the files and rebuild the project.

Verify Optimization

1. Re-run the General Exploration analysis:
   - For Visual Studio IDE: From the VTune Amplifier toolbar, click the down arrow next to the **New Analysis** button and select **General Exploration**.
   - For Standalone UI: From the product menu, select **New > General Exploration Analysis**.

VTune Amplifier reruns the General Exploration analysis for the updated `matrix` target and creates a new result, `r002ge`, that opens automatically.

2. In the `r002ge` result, click the **Summary** tab to see the Elapsed time value for the optimized code:
You see that the Elapsed time has reduced significantly: from 7.776 seconds to 4.302 seconds but one of hardware issues identified in the previous run, Back-End Bound, stayed. This means that there is more room for improvement and you can try other, more effective, mechanisms of matrix multiplication.

### Key Terms
- Elapsed time
- Event-based metrics

### 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

### See Also
**Related information**

**Summary**
Summary

You have completed the Identifying Hardware Issues tutorial. Here are some important things to remember when using the Intel® VTune™ Amplifier to analyze your code for hardware issues:

<table>
<thead>
<tr>
<th>Step</th>
<th>Tutorial Recap</th>
<th>Key Tutorial Take-aways</th>
</tr>
</thead>
</table>
| 1. Prepare for analysis.                                            | If you used the Visual Studio* IDE: You selected the matrix project as an analysis target, set up your environment to enable generating symbol information for system libraries and your binary files, and built the target in the Release mode.  
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, and created the VTune Amplifier 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.  
• Use the Analysis Target tab to choose and configure your analysis target. For Visual Studio* projects, the analysis target settings are inherited automatically. |
| 2. Find hardware issues                                             | You ran the General Exploration analysis that monitors how your application performs against a set of event-based hardware metrics as follows:  
• Analyzed the data provided in the Hardware Issues viewpoint, explored the event-based metrics, identified the areas where your sample application had hardware issues, and found the exact function with poor performance per metrics that could be a good candidate for further analysis.  
• Analyzed the code for the hotspot function identified in the Bottom-up window and located the hotspot line that generated a high number of CPU Clockticks. | • Use the Analysis Type tab to choose, configure, and run the analysis. You may choose between a predefined analysis type like the General Exploration type used in this tutorial, or create a new custom analysis type and add events of your choice. For more details on the custom collection, see the Creating a New Analysis Type topic in the product online help.  
• See the Details section of the General Exploration configuration pane to get the list of processor events used for this analysis type. |
<p>| 3. Resolve detected issues                                          | You solved the memory access issue for the sample application by interchanging the loops and sped up the execution time. You                                                                                                                                               | • Start analyzing the performance of your application from the Summary window to explore the event-based performance metrics for the whole application. Mouse                                                                                   |</p>
<table>
<thead>
<tr>
<th>Step</th>
<th>Tutorial Recap</th>
<th>Key Tutorial Take-aways</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>also considered using the Intel compiler to enable instruction vectorization. You also tried optimizing the mechanism of matrix multiplication. Both tuning scenarios gave you 78.695 seconds of optimization in the application execution time.</td>
<td>over the help icons to read the metric descriptions. Use the Elapsed time value as your performance baseline.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Move to the <strong>Bottom-up</strong> window and analyze the performance per function. Focus on the <em>hotspots</em> - functions that took the highest Clockticks event count. By default, they are located at the top of the table. Analyze the hardware issues detected for the hotspot functions. Hardware issues are highlighted in pink. Mouse over a highlighted value to read the issues description and see the threshold formula.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Double-click the hotspot function in the <strong>Bottom-up</strong> pane to open its source code and identify the code line that took the highest Clockticks event count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Consider using Intel Compiler to vectorize instructions. Explore the compiler documentation for more details.</td>
</tr>
</tbody>
</table>

**Next step:** Prepare your own application(s) for analysis. Then use the VTune Amplifier to find hardware issues and fix them.

**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

**See Also**

[Click here for more Getting Started Tutorials](#)
**Key Terms**

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

**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.

**event-based metrics**: Event ratios with their own threshold values. VTune Amplifier collects event data, calculates the ratios, and provides the results in the corresponding columns of the Bottom-up/Top-down Tree windows and in the Summary window. As soon as the performance of a program unit per metric exceeds the threshold, the VTune Amplifier marks this value as a performance issue (in pink) and provides recommendations how to fix it. For the full list of metrics used by the VTune Amplifier, see the Reference > CPU Metrics topic in the online help.

**event skid**: An event detected not exactly on the code line that caused the event. Event skids may even result in a caller function event being recorded in the callee function. See the online help for more details.

**finalization**: A process during which the Intel® VTune™ Amplifier 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.

**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 shows in the windows/panes of the result tab. To select the required viewpoint, click the (change) link at the top of the result tab and use the drop-down menu to select a required viewpoint.

**See Also**

Click here for more Getting Started Tutorials