Tutorial: Identifying Hardware Issues

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

Legal Information

Important

This document was last updated for the Intel VTune Amplifier 2017 product release. If you are using this tutorial with a newer version of VTune Amplifier, you may see differences in analysis type names and user interface design.
Contents

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

Chapter 1: Navigation Quick Start

Chapter 2: Identifying Hardware Issues
  Build Application...................................................................................................... 8
  Configure and Run Analysis..................................................................................... 9
  Interpret Results........................................................................................................ 11
  Analyze Code........................................................................................................... 14
  Resolve Issue........................................................................................................... 15
  Resolve Next Issue................................................................................................... 18

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, the Intel logo, Intel Atom, Intel Core, Intel Xeon Phi, 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.

Java is a registered trademark of Oracle and/or its affiliates.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

Copyright 2011-2018 Intel Corporation.

This software and the related documents are Intel copyrighted materials, and your use of them is governed by the express license under which they were provided to you (License). Unless the License provides otherwise, you may not use, modify, copy, publish, distribute, disclose or transmit this software or the related documents without Intel's prior written permission.

This software and the related documents are provided as is, with no express or implied warranties, other than those that are expressly stated in the License.
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.

<table>
<thead>
<tr>
<th>About This Tutorial</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated Duration</td>
<td>10-15 minutes.</td>
</tr>
<tr>
<td>Learning Objectives</td>
<td>After you complete this tutorial, you should be able to:</td>
</tr>
<tr>
<td></td>
<td>• Choose an analysis target.</td>
</tr>
<tr>
<td></td>
<td>• Run the General Exploration analysis.</td>
</tr>
<tr>
<td></td>
<td>• Understand the event-based performance metrics.</td>
</tr>
<tr>
<td></td>
<td>• Identify the most critical hardware issues for the application as a whole.</td>
</tr>
<tr>
<td></td>
<td>• Identify the modules/functions that caused the most critical hardware issues.</td>
</tr>
<tr>
<td></td>
<td>• Analyze the source code to locate the most critical code lines.</td>
</tr>
<tr>
<td></td>
<td>• Identify the next steps in performance analysis.</td>
</tr>
</tbody>
</table>


Start Here
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**

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

1. From the installation directory, type `source amplxe-vars.sh`.
   
   This script sets the PATH environment variable that specifies locations of the product graphical user interface utility and command line utility.

   **NOTE:**

   The default installation directory is:

   - For root users: `/opt/intel/vtune_amplifier_xe_version`
   - For non-root users: `$HOME/intel/vtune_amplifier_xe_version`

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

**VTune Amplifier for Systems Access**

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

1. From the installation directory, type `source amplxe-vars.sh` for bourne or korn or bash shells, or `source amplxe-vars.csh` if you are using the C shell.
   
   This script sets the PATH environment variable that specifies locations of the product graphical user interface utility and command line utility.

   The default installation directory is `/opt/intel/system_studio_version/vtune_amplifier_version_for_systems`

2. Type `amplxe-gui` to launch the product graphical interface.
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
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**
Build an application with full optimizations.

**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-
Build Application

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

1. Get software tools.
2. Build application with full optimizations.

Get Software Tools

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

- VTune Amplifier, including sample applications
- `tar` 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.tgz` 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 `/opt/intel/vtune Amplifier_xe_version`. For the VTune Amplifier for Systems, the default `<install_dir>` is:

- For super-users: `/opt/intel/system_studio_version/vtune Amplifier_for_systems`
- For ordinary users: `$HOME/intel/system_studio_version/vtune Amplifier_for_systems`

2. Extract the sample from the `.tgz` 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.

Build the Target

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

1. Browse to the directory where you extracted the sample code (for example, /home/sample/matrix/linux). Make sure this directory contains Makefile.
2. Build your target in the release mode using the make icc command.

   The matrix application is automatically built with the Intel C++ compiler (as matrix.icc) and stored in the matrix/linux directory.

Key Terms

Target

Next Step

Configure and Run 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

Configure and Run Analysis

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

1. Create a VTune Amplifier project, which is a container for an analysis target and analysis type configuration and data collection results.
2. Specify your target application for analysis.
3. 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

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

   $ export EDITOR=gedit
2. Execute the `amplxe-vars` script, located in the product installation directory, to set up the environment.

3. Run the `amplxe-gui` script launching VTune Amplifier GUI.

4. Click the menu button and select **New > Project**... to create a new project. The **Create a Project** dialog box opens.

5. 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 `$HOME/intel/amplxe/projects` (for VTune Amplifier XE) or `$HOME/intel/amplsys/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, 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.icc` application.
4. Click the **Choose Analysis** button on the right to switch to the **Analysis Type** tab.

#### Choose Target and Analysis Type

<table>
<thead>
<tr>
<th>Analysis Target</th>
<th>Analysis Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accessible Targets</strong></td>
<td></td>
</tr>
<tr>
<td>local</td>
<td></td>
</tr>
<tr>
<td>remote Linux</td>
<td></td>
</tr>
<tr>
<td>Intel Xeon Phi</td>
<td></td>
</tr>
<tr>
<td>Intel Xeon Phi</td>
<td></td>
</tr>
<tr>
<td>Arbitrary Targets</td>
<td></td>
</tr>
<tr>
<td>local</td>
<td></td>
</tr>
<tr>
<td>Intel Xeon Phi</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th><strong>Application</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>ix/linux/matrix.icc</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th><strong>Use application directory as working directory</strong></th>
</tr>
</thead>
</table>

#### 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 Merom. 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:

<table>
<thead>
<tr>
<th>Elapsed Time 2: 140.218s</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clockticks</strong>: 1,690,065,600,000</td>
</tr>
<tr>
<td><strong>Instructions Retired</strong>: 70,851,200,000</td>
</tr>
<tr>
<td><strong>CPI Rate</strong>: 23.854</td>
</tr>
<tr>
<td><strong>LLC Miss</strong>: 0.816</td>
</tr>
<tr>
<td><strong>Memory Bus Transactions</strong>: 46,780,000,000</td>
</tr>
<tr>
<td><strong>Instruction Starvation</strong>: 0.005</td>
</tr>
<tr>
<td><strong>Branch Mispredict</strong>: 0.0%</td>
</tr>
<tr>
<td><strong>Total Thread Count</strong>: 9</td>
</tr>
<tr>
<td><strong>Paused Time</strong>: 0s</td>
</tr>
</tbody>
</table>

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
• LLC Miss

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 `multiply` 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 `multiply` 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.
- **LLC miss** metric shows that about 81.6% (0.816) of CPU cycles were spent waiting for LLC load misses to be serviced. Possible optimizations are to reduce data working set size, improve data access locality, blocking and consuming data in chunks that fit in the LLC, or better exploit hardware prefetchers. Consider using software prefetchers but beware that they can increase latency by interfering with normal loads and can increase pressure on the memory system.

**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
Optimization Notice

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:

```c
void multiply1(int msize, int tidx, int numt, 
{
    int i,j,k;

    // Naive implementation
    for(i=tidx; i<msize; i=i+numt) {
        for(j=0; j<msize; j++) {
            for(k=0; k<msize; k++) {
                c[i][j] = c[i][j] + a[i][k]
            }
        }
    }
}
```

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.
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 yours 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 sample code directory (for example, /home/sample/matrix/src).
   For this sample, the multiply.h file is used to define the functions used in the multiply.c file.

   ```c
   // Select which multiply kernel to use via the following
   // macro so that the
   // kernel being used can be reported when the test is run.
   #define MULTIPLY multiply1
   ```

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.

```c
#include <vector>

void multiply2(const std::vector<std::vector<double>>& a, const std::vector<std::vector<double>>& b, std::vector<std::vector<double>>& c)
{
    for (int i = 0; i < msize; i++)
        for (int j = 0; j < msize; j++)
            for (int k = 0; k < msize; k++)
                c[i][j] += a[i][k] * b[k][j];
}
```

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.

Run Intel C++ Compiler from the code sample directory (for example: `/home/sample/matrix/linux`) as follows:

```
make icc
```

The matrix application is automatically built with the Intel compiler (as `matrix.icc`) and stored in the `matrix/linux` directory.

### Verify Optimization

1. From the VTune Amplifier toolbar, click the **Configure Project...** button.

The **Choose Target and Analysis Type** window opens with the **Analysis Target** tab active. Make sure the **local** target system and **Launch Application** target type are selected.

2. In the **Application** field, click the **Browse...** button and navigate to the updated `matrix` application.
3. Click OK to close the dialog box.

4. Re-run the General Exploration analysis: From the Analysis Type tab, select Microarchitecture Analysis > General Exploration.

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

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

Elapse Time: 10.032s

- Clockticks: 115,876,800,000
- Instructions Retired: 20,827,200,000
- CPI Rate: 5.564
- LLC Miss: 0.248
- Memory Bus Transactions: 5,442,400,000
- Instruction Starvation: 0.023
- Branch Mispredict: 0.1%
- Total Thread Count: 9
- Paused Time: 0s

You see that the Elapsed time has reduced from 140.218 seconds to 10.032 seconds. CPI Rate and LLC Miss 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.
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

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 LLC MissBack-End Bound 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:

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, \texttt{mulp dx}). 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 \texttt{multiply2} function name with the \texttt{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: 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:

   ![Elapsed Time](image)

   **Elapsed Time**: 1.783s
   - **Clockticks**: 20,217,600,000
   - **Instructions Retired**: 24,120,000,000
   - **CPI Rate**: 0.838
   - **LLC Miss**: 0.059
   - **Memory Bus Transactions**: 203,200,000
   - **Instruction Starvation**: 0.010
   - **Branch Mispredict**: 0.0%
   - **Total Thread Count**: 9
   - **Paused Time**: 0s
You see that the Elapsed time has reduced significantly: from 10.032 seconds to 1.783 seconds but one of hardware issues identified in the previous run, LLC Miss, 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
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>
<tbody>
<tr>
<td>1. Prepare for analysis.</td>
<td>You built the target in the Release mode and created the VTune Amplifier project for your analysis target.</td>
<td>• Create a VTune Amplifier project and use the Analysis Target tab to choose and configure your analysis target.</td>
</tr>
<tr>
<td>2. Find hardware issues</td>
<td>You ran the General Exploration analysis that monitors how your application performs against a set of event-based hardware metrics as follows:</td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td>• See the Details section of the General Exploration configuration pane to get the list of processor events used for this analysis type.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
</tr>
<tr>
<td>3. Resolve detected issues</td>
<td>You solved the memory access issue for the sample application by interchanging the loops and sped up the execution time. You also considered using the Intel compiler to enable instruction vectorization. You also tried optimizing the mechanism of matrix multiplication. Both tuning scenarios gave you 138.435 seconds of optimization in the application execution time.</td>
<td>• Start analyzing the performance of your application from the Summary window to explore the event-based performance metrics for the whole application. Mouse 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 Bottom-up window and analyze the performance per function. Focus on the hotspots - 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>
</tbody>
</table>
Step | Tutorial Recap | Key Tutorial Take-aways
--- | --- | ---

- Double-click the hotspot function in the **Bottom-up** pane to open its source code and identify the code line that took the highest Clockticks event count.
- Consider using Intel Compiler to vectorize instructions. Explore the compiler documentation for more details.

**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: \( \text{Wall clock time at end of application} - \text{Wall clock time at start of application} \).

**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