Tutorial: Finding Hotspots on the Intel® Xeon Phi™ Coprocessor

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

Legal Information
Contents

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

Chapter 1: Navigation Quick Start

Chapter 2: Finding Hotspots
   Build Application ........................................................................................................ 8
   Create Project and Configure Target ........................................................................ 11
   Run Advanced Hotspots Analysis .............................................................................. 13
   Interpret Results .......................................................................................................... 14
   Analyze Code .............................................................................................................. 16
   Tune Algorithms .......................................................................................................... 17
   Compare with Previous Result ..................................................................................... 19

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 analyze a native application running on an Intel® Xeon Phi™ coprocessor card with the Intel® VTune™ Amplifier and identify the most performance-critical code.

<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 hotspots on the Intel Xeon Phi coprocessor (code name: Knights Corner) based on Intel Many Integrated Core architecture (Intel MIC architecture).</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>• Create a VTune Amplifier project.</td>
</tr>
<tr>
<td></td>
<td>• Configure an analysis target for Intel Xeon Phi coprocessor (native) target system type and run it on a card.</td>
</tr>
<tr>
<td></td>
<td>• Configure and run Advanced Hotspots analysis for the Intel Xeon Phi coprocessor.</td>
</tr>
<tr>
<td></td>
<td>• Identify the modules/functions that consumed a lot of CPU time.</td>
</tr>
<tr>
<td></td>
<td>• Analyze the source code to locate the most critical code lines.</td>
</tr>
</tbody>
</table>

More Resources

• Intel VTune Amplifier support page: https://software.intel.com/en-us/intel-vtune-amplifier-xe-support/
Intel® VTune™ Amplifier provides information on code performance for users developing serial and multithreaded applications on Windows®, Linux®, 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: `~/.vhome/~/intel/vtune_amplifier_xe_version`

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
Finding Hotspots

You can use the Intel® VTune™ Amplifier to identify the most performance-critical functions in an application running on the Intel® Xeon Phi™ coprocessor (code name: Knights Corner). This tutorial guides you through workflow steps running Advanced Hotspots analysis type on a sample application, matrix.

Step 1: Prepare for analysis
- Build an application, copy it to the card, and set a performance baseline.
- Create a VTune Amplifier project, specify the Intel Xeon Phi (native) target system, provide a full path to the target application on the card, and configure search directories.

Step 2: Find hotspots
- Choose and run Advanced Hotspots analysis.
- Explore performance metrics and identify a hotspot.
- View and analyze code of the performance-critical function.

Step 3: Eliminate hotspots
- Modify the code to resolve the detected performance issues and rebuild the code enabling the vectorization option of the Intel compiler.
Step 4: Check your work

- Re-run the analysis and compare results before and after 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-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

Before you start identifying hotspots in your native Intel® Xeon Phi™ coprocessor application, do the following:

1. **Get software tools.**
2. **Build application with full optimizations on the host.**
3. **Create a performance baseline.**

### Get Software Tools

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

- Intel® VTune™ Amplifier, including sample applications
- Sampling driver, set up during the VTune Amplifier installation

**NOTE:**

If, for some reason, the VTune Amplifier was not able to install the driver, you will not be able to run the analysis and will see a warning message. See online help for additional instructions how to install the driver manually.

- tar file extraction utility
- Intel® C++ Compiler installed on the host. See Release Notes for more information.

### Acquire Intel VTune Amplifier


### 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.
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 on the host with full optimizations, which is recommended for performance analysis.

1. Browse to the linux directory within where you extracted the sample code (for this example assume that location is /home/sample/matrix/linux). Make sure this directory contains Makefile.

2. Set up the environment for Intel C++ Compiler:
   
   source <path_to_compiler_bin>/compilervars.sh intel64

3. Build the code using the make command:
   
   $ make mic

   The matrix application is built as matrix.mic and stored in the matrix/linux directory.

NOTE:
This application uses OpenMP* library for compilation. To run the sample on the Intel Xeon Phi coprocessor, make sure to copy the OpenMP library to the card and set up the default path.

**Create a Performance Baseline**

To communicate with the Intel Xeon Phi coprocessor cards, you may use any of the following mechanisms:

- Mount an NFS share. See the NFS Mounting a Host Export topic in the Intel Manycore Programming Software Stack help for details.
- Use existing SSH tools.

1. Ensure that the binary to analyze is copied to the Intel Xeon Phi coprocessor. You can do this by using scp, for example:

   scp matrix.mic mic0:/tmp
NOTE:
You may add this command to build scripts to automate a copy action after the binary recompilation. In this tutorial's scenario, `scp` command is added to the `Makefile`. So, the `matrix` application is built and automatically copied to the Intel Xeon Phi coprocessor.

2. Run the application on the coprocessor using `ssh` and record the results to establish a performance baseline:

```
[vtune@nntvtune235 bin64]$ ssh mic0 /tmp/matrix.mic
Addr of buf1 = 0x7fd5149ee010
Offs of buf1 = 0x7fd5149ee180
Addr of buf2 = 0x7fd50d96d010
Offs of buf2 = 0x7fd50d96d1c0
Addr of buf3 = 0x7fd5068ec010
Offs of buf3 = 0x7fd5068ec100
Addr of buf4 = 0x7fd4ff86b010
Offs of buf4 = 0x7fd4ff86b140
Threads #: 192 Pthreads
Matrix size: 3840
Using multiply kernel: multiply1
Execution time = 30.466 seconds
Freq = 1.100000 GHz
```

3. Note the execution time displayed at the bottom. For the `matrix.mic` executable in the figure above, the execution time is 30.466 seconds. Use this metric as a baseline against which you will compare subsequent runs of the application.

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

NOTE:
• If you experience a problem with permissions to run the commands, use `sudo` or root access.
• Alternatively, you may create an `ssh` script to copy and launch your application on a card or use the `micnativeloadex` utility. For details, see the *Preparing an Intel® Xeon Phi™ Coprocessor System for Analysis* online help topic.

Key Terms
• Target

Next Step
Create Project and Configure Target
Create Project and Configure Target

1. Create a VTune Amplifier project.
2. Specify an analysis target.
3. Configure search directories for symbol resolution.

Create a VTune Amplifier Project

1. Type `amplxe-gui` to launch the VTune Amplifier GUI.
2. Create a new project via New > Project... menu.
   
   The Create a Project dialog box opens.
3. 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 Choose Target and Analysis Type window with the Analysis Target tab active.

Specify an Analysis Target

NOTE:

This tutorial focuses on analyzing a native Intel Xeon Phi coprocessor application that is copied and run on the card. To configure an analysis for an offload application target, make sure to select the Intel Xeon Phi coprocessor (host launch) type of the target system and specify the application name with the full path on the host in the Application field in the Analysis Target tab. See VTune Amplifier Help for more details.

1. In the Analysis Target tab, select the Intel Xeon Phi coprocessor (native) type from the left pane.
   
   The configuration pane on the right is updated with the options applicable to the selected target system type.
2. Make sure the Launch Application target type is selected in the drop-down menu on the right.
3. Specify the Intel Xeon Phi coprocessor card where your application is located. By default, 0(mic0) card is pre-selected.
4. In the Application field, specify a full path to the matrix application on the card: `/tmp/matrix.mic` as an application to launch.
Configure Search Directories

VTune Amplifier resolves symbols for Intel Xeon Phi coprocessor-based modules on the host during collection post-processing. For proper symbol resolution, you need to specify host search paths for the application and for Intel Xeon Phi coprocessor modules before the collection:

1. In the Choose Target and Analysis Type window, click the Binary/Symbol Search button on the right.

   The Binary/Symbol Search dialog box opens.

2. In the Search Directories field, click the Browse button to add the application directory on the host and OpenMP* directory.

3. Click OK to save your changes.

   **Additional Binary and Symbol File Locations**

   Specify the possible locations of binary and symbol files. These can be local directories, or symbol server paths using the form srv*local_cache_directory*http://address/. Press F1 for more details.

<table>
<thead>
<tr>
<th>Search Directories</th>
</tr>
</thead>
<tbody>
<tr>
<td>/opt/mpss/3.3/sysroots/k1om-mpss-linux/boot</td>
</tr>
<tr>
<td>/home/sample/matrix/Linux</td>
</tr>
</tbody>
</table>

   Add new search location

   **Key Terms**

   - Target
   - Project
### Next Step

Run Hotspots Analysis

## Run Advanced Hotspots Analysis

After building the sample application and collecting baseline performance data for it, rerun it under the scrutiny of Intel® VTune™ Amplifier to discover what parts of the code are being most used. Advanced Hotspots analysis collects event and IP (Instruction Pointer) information to reveal evidence of a basic set of hardware issues induced by the application code that may be affecting its performance.

**Prerequisites:** You created a project and specified your sample application as an Intel Xeon Phi coprocessor (native) target in the Analysis Target tab.

**To run the analysis:**

1. Click the Choose Analysis button on the right, or switch to the Analysis Type tab.

   **NOTE:**

   VTune Amplifier automatically detects your target system configuration and displays analysis types applicable to the Intel® Xeon Phi™ coprocessor.

2. From the analysis tree on the left, select the Algorithm Analysis > Advanced Hotspots analysis type.

   The Advanced Hotspots predefined configuration opens on the right.

3. Click the Start button on the right to run the analysis.

   VTune Amplifier starts the matrix.mic application on the Intel Xeon Phi coprocessor card via SSH connection. The application calculates a large matrix multiply 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 its 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 sample application exits, the Intel® VTune™ Amplifier finalizes the results and opens the Hotspots viewpoint where each window or pane is configured to easily identify code regions that consumed a lot of CPU time. To interpret the data on the sample code performance, do the following:

- Analyze basic performance statistics.
- Identify most performance-critical functions

NOTE:
The screenshots and execution time data provided in this tutorial are created on a system with more than 240 logical cores. Your data may vary depending on the number and type of CPU cores on your system.

Analyze Basic Performance Statistics

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:** 33.245s

**CPU Time:** 5521.700s

**Effective Time:** 5521.685s

- **Spin Time:** 0.015s
- **Overhead Time:** 0s

**Instructions Retired:** 684,310,000,000

**CPI Rate:** 8.876

**CPU Frequency Ratio:** 1.000

**Total Thread Count:** 448

**Paused Time:** 0s
• **Elapsed time** for the `matrix` application is 33.245 seconds. This is wall clock time from the beginning to the end of the collection, including data allocation and calculation. Note that Elapsed time metric provided in the **Summary** window is different from the Execution time provided in the application output, which includes calculation only.

• **CPU Time** is equal to 5521.685 seconds. It is the sum of CPU time for all application threads.

• **Instructions Retired** metric displays an estimated number of instructions executed during the collection (captured in the INSTRUCTIONS_EXECUTED hardware event), which is an essential component of the ratio given below.

• **Clockticks per Instructions Retired (CPI Rate)** is an event ratio, also known as Cycles per Instructions, which is one of the basic performance metrics. High CPI Rate is marked with a red flag and signals a possible performance issue. Potential causes are memory stalls, instruction starvation, branch misprediction, or long-latency instruction.

For more detailed analysis, switch to the **Bottom-up** window to identify code sections responsible for detected problems.

**Identify Most Performance-Critical Functions**

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. You may hover over a column name to see the formula used for the calculation of this metric. The **CPU Time** column is marked with a yellow star as the Data of Interest column, which means that the VTune Amplifier 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 `multiply1` function is an obvious hotspot that took the most CPU Time and the biggest count for the Instructions Retired event. Its CPI rate is also considered as high and is marked by the VTune Amplifier as a performance issue. You may hover over the pink cell to read a description of the issue and proposed solution.

**Key Terms**

- Elapsed time
- Event-based metrics

**Next Step**

Analyze Code
Analyze Code

You identified `multiply1` as the hottest function. In the Bottom-up window, double-click this function to open the Source window and view the source code and event distribution:

![Source window screenshot]

The table below explains some of the features available in the Source window when viewing Advanced 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. 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.

3. 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 Advanced 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.

4. 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 lines it identifies.

Click the hotspot navigation button to go to the code line that has the highest Effective CPU Time. In the Source pane for the `multiply1` function, you see that VTune Amplifier highlights the code section that multiplies matrix elements in the loop but ineffectively accesses the memory. Focus on this section and try to reduce memory issues.

**Key Terms**

- Hotspot
Next Step
Tune Algorithms

Tune Algorithms

In the Source pane, you identified the code line in the multiply1 function that accumulated the highest Effective Time values. To solve this issue, do the following:

1. Change the multiplication algorithm and enable vectorization.
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 initialize the functions used in the multiply.c file.

```c
31 typedef double TYPE;
32 typedef TYPE array[NUM];
33
34 // Select which multiply kernel to use via the following macro so that the
35 // kernel being used can be reported when the test is run.
36 #define MULTIPLY multiply1
37```

2. Replace the #define MULTIPLY multiply1 with the #define MULTIPLY multiply2.
   The new multiply2 function swaps the indices of the innermost two loops, in a method called loop interchange. Note in the innermost loops that three of the array references use j as the second index and the fourth reference does not use j at all. In C that last index addresses locations adjacent in memory sequence, taking advantage of cache locality to use adjacent data all in one pass, and that optimizes the memory access in the code by minimizing cache line thrash. Moreover, arranging successive computations in array order this way makes them more likely to be recognized by the compiler for vectorization.
When you build the code with the Intel® C++ Compiler, it vectorizes the computation, which means that it uses SIMD (Single Instruction Multiple Data) style 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, which is why the multiply2 function uses #pragma ivdep to instruct the compiler to ignore assumed vector dependencies. This information lets the compiler employ the Supplemental Streaming SIMD Extensions (SSSE).

3. Save files and rebuild the application:

$ make mic

When the matrix.mic application is built and stored in the matrix/linux directory, copy it to the coprocessor.

Verify Optimization

Re-run the application via ssh script:

[vtune@ntvtune235 linux]$ ssh mic0 /tmp/matrix.mic
Addr of buf1 = 0x7f3581934010
Offs of buf1 = 0x7f3581934180
Addr of buf2 = 0x7f357a8b3010
Offs of buf2 = 0x7f357a8b31c0
Addr of buf3 = 0x7f3573832010
Offs of buf3 = 0x7f3573832100
Addr of buf4 = 0x7f356c7b1010
Offs of buf4 = 0x7f356c7b1140
Threads #: 192 Pthreads
Matrix size: 3840
Using multiply kernel: multiply2
Execution time = 3.572 seconds
Freq = 1.100000 GHz

You see that the Execution time has reduced significantly and you got about 27 seconds of optimization.
Key Terms

- Elapsed time
- Event-based metrics

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 27 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 Advanced 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. From the product menu select **New > Advanced Hotspots Analysis**.
   
   VTune Amplifier reruns Advanced Hotspots analysis for the updated matrix target and creates a new result (for example, r002ah) that opens automatically.

2. Click the **Compare Results** button on the Intel® VTune™ Amplifier toolbar.
   
   The Compare Results window opens.

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

![Choose Results to Compare](Image)
The Summary window opens displaying application-level performance statistics for both results and their difference values.

**Identify the Performance Gain**

The Result Summary section of the Summary window shows difference information as follows: \(<\text{Result 1 metric} > - \text{<Result 2 metric>} = \text{<metric Difference>}\).

You see that after optimization all metrics values have reduced significantly, though CPI Rate is still an issue (>1).

<table>
<thead>
<tr>
<th>Elapsed Time:</th>
<th>33.245s – 5.387s = 27.859s</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Time:</td>
<td>5521.700s – 570.634s = 4951.066s</td>
</tr>
<tr>
<td>Effective Time:</td>
<td>5521.685s – 570.622s = 4951.063s</td>
</tr>
<tr>
<td>Spin Time:</td>
<td>0.015s - 0.012s = 0.003s</td>
</tr>
<tr>
<td>Overhead Time:</td>
<td>Not changed, 0s</td>
</tr>
<tr>
<td>Instructions Retired:</td>
<td>684,310,000,000 - 72,389,900,000 = 611,920,100,000</td>
</tr>
<tr>
<td>CPI Rate:</td>
<td>8.876 - 8.671 = 0.205</td>
</tr>
<tr>
<td>CPU Frequency Ratio:</td>
<td>Not changed, 1.000</td>
</tr>
<tr>
<td>Total Thread Count:</td>
<td>448 - 286 = 162</td>
</tr>
<tr>
<td>Paused Time:</td>
<td>Not changed, 0s</td>
</tr>
<tr>
<td>Frame Count:</td>
<td>Not changed, [Unknown]</td>
</tr>
</tbody>
</table>

Switch to the Bottom-up window to view the CPU time usage per function for each result and their differences side by side.
Since for the second run you removed the `multiply1` function, its time shows up in the Difference column as a performance gain.

Click the CPU Time: r001ah column to sort the data in the grid by this column.

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time: Difference</th>
<th>CPU Time: r000ah</th>
<th>CPU Time: r001ah</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Effective Time</td>
<td>Spin Time</td>
<td>Ov.. Time</td>
</tr>
<tr>
<td><code>multiply2</code></td>
<td>-556.431s</td>
<td>0s</td>
<td>0s</td>
</tr>
<tr>
<td><code>tasklet_hi_action</code></td>
<td>18.282s</td>
<td>0s</td>
<td>0s</td>
</tr>
<tr>
<td><code>dynamic_irq_clean</code></td>
<td>18.403s</td>
<td>0s</td>
<td>0s</td>
</tr>
<tr>
<td><code>vsnprintf</code></td>
<td>-0.096s</td>
<td>0s</td>
<td>0s</td>
</tr>
<tr>
<td><code>update_cpu_load</code></td>
<td>0.790s</td>
<td>0s</td>
<td>0s</td>
</tr>
</tbody>
</table>

The `multiply2` function shows up on top as the biggest CPU Time hotspot for the result r001ah, though it performs better than `multiply1`. You may try to optimize the code further using more advanced algorithms, for example, block-structuring access to matrix data to maximize cache reuse.

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

**See Also**
Summary
You have completed the Finding Hotspots on the Intel® Xeon Phi™ Coprocessor tutorial. Here are some important things to remember when using the Intel® VTune™ Amplifier to analyze your code for hotspots:

<table>
<thead>
<tr>
<th>Step</th>
<th>Tutorial Recap</th>
<th>Key Tutorial Take-aways</th>
</tr>
</thead>
</table>
| 1. Prepare for analysis | • You built and copied the matrix application to the card after each recompilation.  
• You built the target application with the Intel® C++ Compiler, ran it on the card via ssh, and recorded a performance baseline.  
• You created a VTune Amplifier project, specified the ssh script as an application to launch and specified the path to the application on the card as application parameters.  
• You configured search directories to resolve symbol information for Intel Xeon Phi coprocessor-based modules and application modules. | • Create a VTune Amplifier project and use the Analysis Target tab to specify the target system and an application to analyze.  
• VTune Amplifier starts target applications. It is not able to start an application directly on Intel Xeon Phi coprocessor architecture cards.  
• To copy the target native application to the card, you may either add the copy action to the build script or mount the host directory so that the binary is visible on the Intel MIC architecture target.  
• Use the Analysis Type tab to choose, configure, and run the analysis. You may choose between a predefined analysis type like the Advanced Hotspots 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 Custom Analysis topic in the product online help.  
• See the Details section of an analysis configuration pane to get the list of processor events used for this analysis type. |
| 2. Find hotspots | You launched Advanced Hotspots analysis that analyzes CPU time spent in each program unit of your application and identified the following hotspots:  
• Identified a function that took the most CPU time, the highest event count and CPI Rate. This function is 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 event-based performance metrics for the whole application. Mouse over help icons to read metric descriptions. Use the Elapsed time value as your performance baseline.  
• Move to the Bottom-up window and analyze the performance per function. Focus on the hotspots - functions that consumed the most CPU Time. In the initial sort, they are located at the top of the table. Use the CPU Rate metric to understand the efficiency of your code. If the metric value exceeds a threshold, the VTune Amplifier highlights it in pink as a performance issue. Mouse over a highlighted value to read the issue description and see the threshold formula. |
<table>
<thead>
<tr>
<th>Step</th>
<th>Tutorial Recap</th>
<th>Key Tutorial Take-aways</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Eliminate hotspots</td>
<td>You solved the memory access issue for the sample application by interchanging the loops and sped up the execution time. You also used the Intel compiler to enable instruction vectorization.</td>
<td>Consider using the Intel compiler to vectorize instructions. Explore the compiler documentation for more details.</td>
</tr>
<tr>
<td>4. Check your work</td>
<td>You ran Advanced Hotspots analysis on the optimized code and compared the results before and after optimization.</td>
<td>Perform regular regression testing by comparing analysis results before and after optimization. From GUI, click the <strong>Compare Results</strong> button on the VTune Amplifier toolbar. From command line, use the <code>amplxe-cl</code> command.</td>
</tr>
</tbody>
</table>

**Next step:** Prepare your own application(s) for analysis. Then use the VTune Amplifier 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

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

**Effective time**: CPU time spent in the user code. This metric does not include Spin and Overhead time.

**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 for Performance Metrics chapter 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 the processors spend a lot of time executing. Some hotspots may indicate bottlenecks and can be removed, while other hotspots inevitably take a long time to execute due to their nature.

**project**: A container for an analysis target configuration and data collection results.

**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 and use the drop-down menu at the top of the result tab.

See Also

Click here for more Getting Started Tutorials