



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

Intel® VTune™ Amplifier 2013 for Linux\* OS

C++ Sample Application Code

Document Number: 327971-006

[Legal Information](#)

# Contents

<b>Legal Information.....</b>	<b>3</b>
<b>Overview.....</b>	<b>4</b>
<b>Chapter 1: Navigation Quick Start</b>	
<b>Chapter 2: Finding Hotspots</b>	
Build Application .....	8
Create Project and Configure Target.....	10
Run Hotspots Analysis.....	12
Interpret Results.....	12
Analyze Code.....	14
Tune Algorithms.....	15
Compare with Previous Result.....	17
<b>Chapter 3: Summary</b>	
<b>Chapter 4: Key Terms</b>	

# Legal Information

---

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, Flexpipe, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel CoFluent, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel Xeon Phi, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Pentium, Pentium Inside, Puma, skool, the skool logo, SMARTi, Sound Mark, Stay With It, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2009-2013, Intel Corporation. All rights reserved.

# 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.

**About This Tutorial**

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).

**Estimated Duration**

10-15 minutes.

**Learning Objectives**

After you complete this tutorial, you should be able to:

- Create a VTune Amplifier project.
- Use an `sshscript` to specify an analysis target and run it on a card.
- Configure and run Hotspots analysis for the Intel Xeon Phi coprocessor.
- Identify the modules/functions that consumed a lot of CPU time.
- Analyze the source code to locate the most critical code lines.

**More Resources**

- Intel Parallel Studio XE tutorials (HTML, PDF): <http://software.intel.com/en-us/articles/intel-software-product-tutorials/>
- Intel Parallel Studio XE support page: <http://software.intel.com/en-us/articles/intel-parallel-studio-xe/>
- Intel VTune Amplifier support page: <http://software.intel.com/en-us/intel-vtune-amplifier-xe>

# Navigation Quick Start



Intel® VTune™ Amplifier provides information on code performance for users developing serial and multithreaded applications on Windows\* and Linux\* operating systems. VTune Amplifier helps you analyze algorithm choices and identify where and how your application can benefit from available hardware resources.

## VTune Amplifier 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.

The default installation directory is `/opt/intel/vtune_amplifier_xe_201n`.

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

## VTune Amplifier GUI

The screenshot displays the Intel VTune Amplifier GUI. The main window is titled "Locks and Waits" and shows a detailed analysis of synchronization objects. The table below represents the data shown in the "Sync Object / Function / Call Stack" view.

Sync Object / Function / Call Stack	Wait Time by Utilization	Wait Count
Socket 0x3d5dfe4	0.033ms	1
Stream 0x2a2a73c3	8.609ms	7
GetTexture	0.092ms	2
GetObject ← read	0.092ms	2
Selected 1 row(s):		8.609ms 7

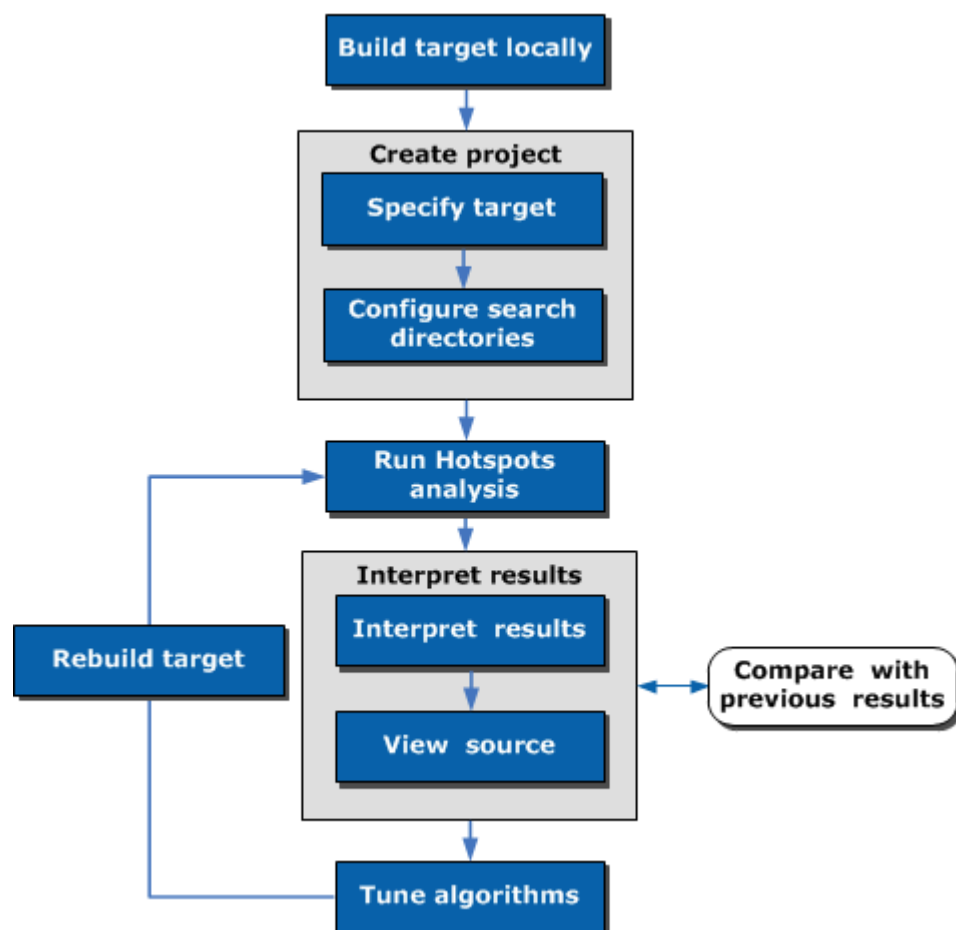
The interface also features a "Thread Con..." visualization at the bottom, showing thread execution over time. The status bar at the bottom indicates "No filters are applied" and "Call Stack Mode: Only user functions".

- A** Configure and manage projects and results, and launch new analyses from the primary toolbar. Click the **Project Properties** button on this toolbar to manage result file locations. Newly completed and opened analysis results along with result comparisons appear in the results tab for easy navigation.
- B** Use the VTune Amplifier menu to control result collection, define and view project properties, and set various options.
- C** The **Project Navigator** provides an iconic representation of your projects and analysis results. Click the **Project Navigator** button on the toolbar to enable/disable the **Project Navigator**.
- D** 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.
- E** Switch between window tabs to explore the analysis type configuration options and collected data provided by the selected viewpoint.
- F** Use the **Grouping** drop-down menu to choose a granularity level for grouping data in the grid.
- G** Use the filter toolbar to filter out the result data according to the selected categories.

# 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 Hotspots analysis type on a sample application, `matrix`.



<b>Step 1: Prepare for analysis</b>	<ul style="list-style-type: none"> <li>Build an application, copy it to the card, and set a performance baseline.</li> <li>Create a VTune Amplifier project, specify a launch script as a target application and path to the application on the card as application parameters, and configure search directories</li> </ul>
<b>Step 2: Find hotspots</b>	<ul style="list-style-type: none"> <li>Choose and run Hotspots analysis.</li> <li>Explore performance metrics and identify a hotspot.</li> <li>View and analyze code of the performance-critical function.</li> </ul>
<b>Step 3: Eliminate hotspots</b>	<ul style="list-style-type: none"> <li>Modify the code to resolve the detected performance issues and rebuild the code enabling the vectorization option of the Intel compiler.</li> </ul>

**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.

- Intel Manycore Platform Software Stack. See Release Notes for more information.
- `tar` file extraction utility
- Intel C++ compiler installed on the host. See Release Notes for more information.

### 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.tgz` file from the `<install_dir>/samples/<locale>/C++` directory to a writable directory or share on your system. The default installation path is `opt/intel/vtune_amplifier_xe_<version>`.
2. Extract the sample from the `.tgz` file.

- Samples are non-deterministic. Your screens may vary from the screen captures shown throughout this tutorial.



- Samples are designed only to illustrate the VTune Amplifier 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.

## 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 (MPSS) 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:

In this tutorial's scenario, the command running the `matrix` application is added to the `runmatrix.bat`. Edit this `bat` file to add a path to the private key file on your system.

```
[vtune@nnlpatlr98-71 bin64]$ ssh mic0 /tmp/matrix.mic
vtune@mic0's password:
Addr of buf1 = 0x0x7fbf8d96a010
Offs of buf1 = 0x0x7fbf8d96a180
Addr of buf2 = 0x0x7fbf868e9010
Offs of buf2 = 0x0x7fbf868e91c0
Addr of buf3 = 0x0x7fbf7f868010
Offs of buf3 = 0x0x7fbf7f868100
Matrix size: 3840
Using multiply kernel: multiply1
Threads #: 240, Execution time = 72.240 seconds
```

3. Note the execution time displayed at the bottom. For the `matrix.mic` executable in the figure above, the execution time is 72.240 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.

- 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 *Choosing a Target on the Intel® Xeon Phi™ Coprocessor* online help topic.

### Key Terms

- Target

### Next Step

Create Project and Configure Target

#### 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

## Create Project and Configure Target



To start performance analysis on the Intel® Xeon Phi™ coprocessor, do the following:

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.

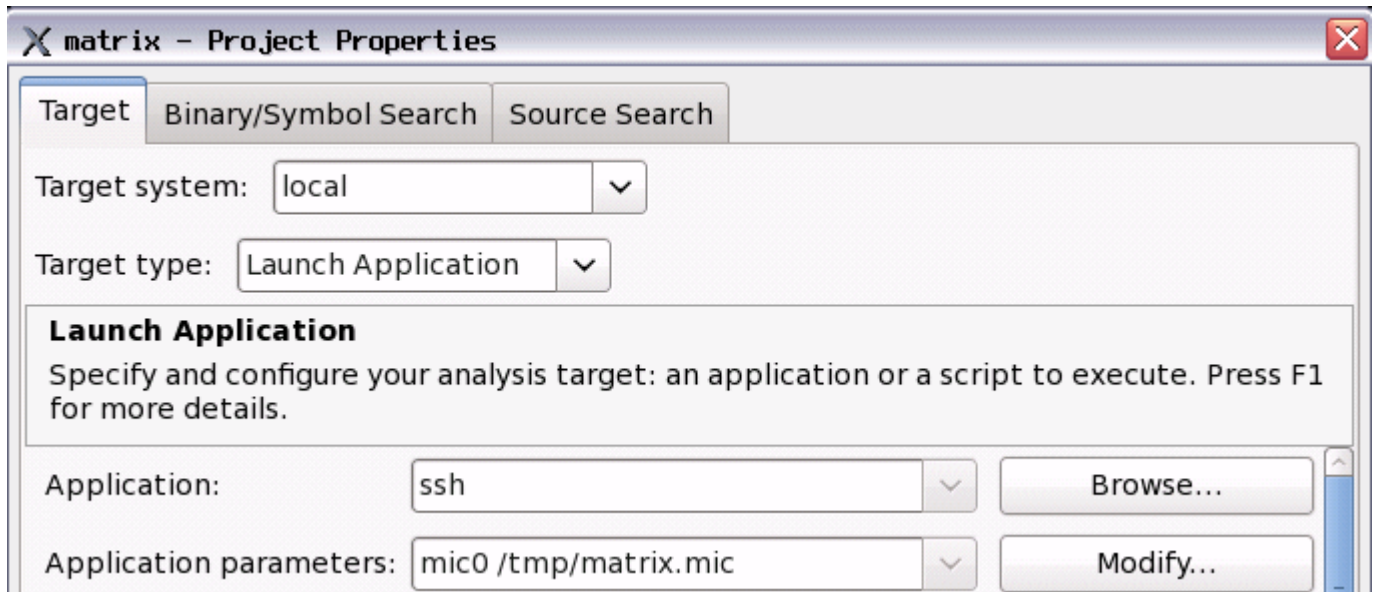
3. Specify the project name `matrix` that will be used as the project directory name and click the **Create Project** button.

By default, the VTune Amplifier creates a project directory under the `$HOME/intel/ampl/projects` directory and opens the **Project Properties: Target** dialog box.

### Specify an Analysis Target

1. In the **Target** tab of the **Project Properties** dialog box, select the **Application to Launch** target type.

- In the **Application** field, specify `ssh` as the application to launch. In the **Application parameters** field, specify the path to the copied application on the Intel Xeon Phi coprocessor card.



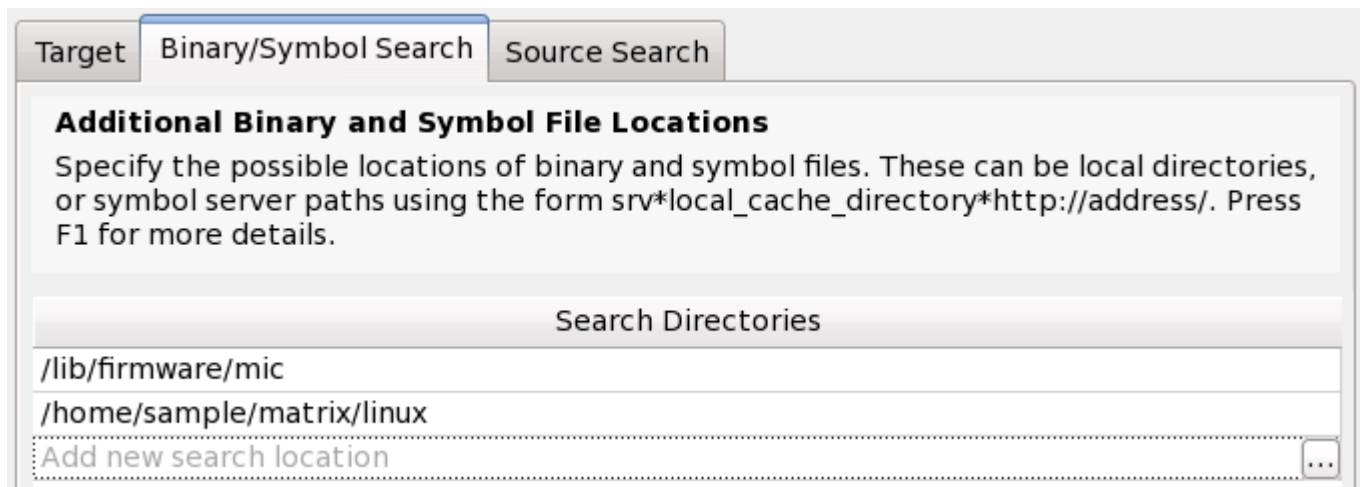
If you need to use `sudo` to run the application on the coprocessor, filled the following way:

- **Application:** `sudo`
- **Application parameters:** `ssh mic0 /tmp/matrix.mic`

### 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 search paths for the application and for Intel Xeon Phi coprocessor modules on the host before the collection:

- In the **Project Properties** dialog box, switch to the **Binary/Symbol Search** tab.
- In the **Search Directories** field, click the **Browse** button to add the application directory and the directory for local Intel Xeon Phi coprocessor modules: `lib/firmware/mic`.
- Click **OK** to save your changes.



### Key Terms

- Target

- [Project](#)

### Next Step

[Run Hotspots Analysis](#)

## Run 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. 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.

### To run the analysis:

1. From the VTune Amplifier toolbar, click the **New Analysis** button.  
The **New Amplifier Result** tab opens with the **Analysis Type** configuration window active.
2. From the analysis tree on the left, select the **Knights Corner Platform Analysis > Hotspots** analysis type.  
The Hotspots predefined configuration opens on the right. You may specify which cards to use for collection using the **List of Intel Xeon Phi coprocessor cards** option. By default, the data is collected on card 0.
3. Click the **Start** button on the right to run the analysis.

VTune Amplifier starts the `ssh` script that runs the `matrix.mic` application on the Intel Xeon Phi coprocessor card. 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.

<b>Elapsed Time:</b> ⓘ	<b>79.174s</b> ⓘ
<b>CPU Time:</b> ⓘ	16853.340s
<b>Instructions Retired:</b>	744,720,000,000
<b>CPI Rate:</b> ⓘ	22.630

- **Elapsed time** for the `matrix` application is 79.174 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 16853.340 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 in pink 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.

Function / Call Stack	CPU Time*	Instructions Retired	CPI Rate
multiply1	16720.170s	736,220,000,000	22.711
__do_softirq	90.260s	3,630,000,000	24.865
default_send_IPI_mask	7.660s	540,000,000	14.185
__raw_spin_unlock_irq	6.160s	650,000,000	9.477
weighted_cpuload	2.420s	140,000,000	17.286
source_load	2.240s	110,000,000	20.364
Selected 1 row(s):	16720.170s	736,220,000,000	22.711

### 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	Assembly	Source	CPU Time*	Instructions Re...
36		// Naive implementation		
37		for(i=tidx; i<msize; i=i+numt) {		
38		for(j=0; j<msize; j++) {	0.550s	80,000,000
39		for(k=0; k<msize; k++) {	2.400s	130,000,000
40		c[i][j] = c[i][j] + a[i][k] * b[k][j];	16717.220s	736,010,000,000
41		}		
42		}		
43		}		
44		}		
45		void multiply2(int msize, int tidx, int numt, TYPE a[][NUM		

The table below explains some of the features available in the **Source** window when viewing Hotspots analysis data.

- 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 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 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 CPU 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 `matrix.c` file from the sample code directory (for example, `/home/sample/matrix/src`). For this sample, the `matrix.c` file is used to initialize the functions used in the `multiply.c` file.

```

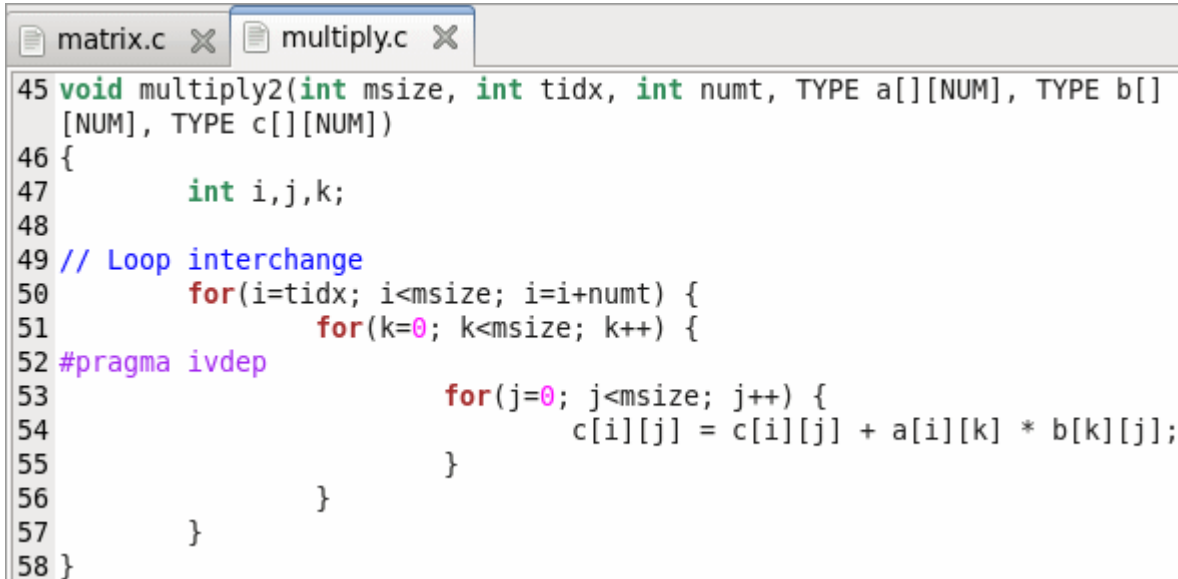
matrix.c x multiply.c x
34 // Select which multiply kernel to use via the following macro so that the
35 // kernel being used can be reported when the test is run.
36 #define MULTIPLY multiply1
37
38 int NTHREADS = MAXTHREADS;
39
40 //static TYPE a[NUM][NUM], b[NUM][NUM], c[NUM][NUM];
41 typedef TYPE array[NUM];
42 typedef unsigned long long UINT64;

```



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.



```

45 void multiply2(int msize, int tidx, int numt, TYPE a[][NUM], TYPE b[]
    [NUM], TYPE c[][NUM])
46 {
47     int i,j,k;
48
49 // Loop interchange
50     for(i=tidx; i<msize; i=i+numt) {
51         for(k=0; k<msize; k++) {
52 #pragma ivdep
53             for(j=0; j<msize; j++) {
54                 c[i][j] = c[i][j] + a[i][k] * b[k][j];
55             }
56         }
57     }
58 }

```

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).

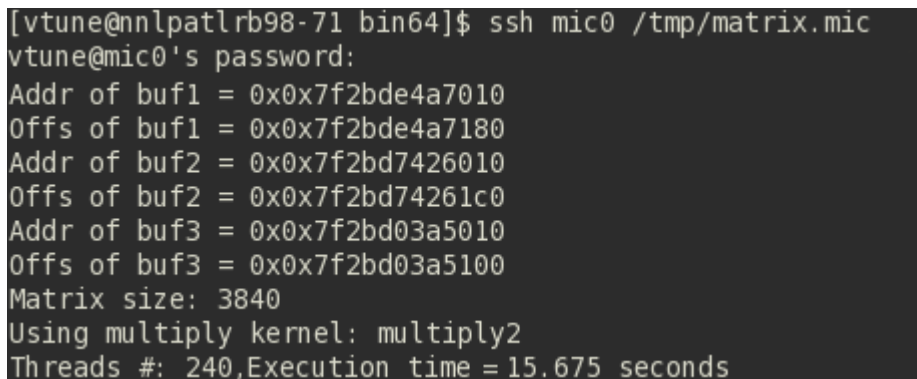
3. Save files and rebuild the application:

```
$ make mic
```

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

### Verify Optimization

Re-run the application via `ssh` script:



```

[vtune@nnlpatlrb98-71 bin64]$ ssh mic0 /tmp/matrix.mic
vtune@mic0's password:
Addr of buf1 = 0x0x7f2bde4a7010
Offs of buf1 = 0x0x7f2bde4a7180
Addr of buf2 = 0x0x7f2bd7426010
Offs of buf2 = 0x0x7f2bd74261c0
Addr of buf3 = 0x0x7f2bd03a5010
Offs of buf3 = 0x0x7f2bd03a5100
Matrix size: 3840
Using multiply kernel: multiply2
Threads #: 240,Execution time = 15.675 seconds

```

You see that the Execution time has reduced significantly and you got about 56.5 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

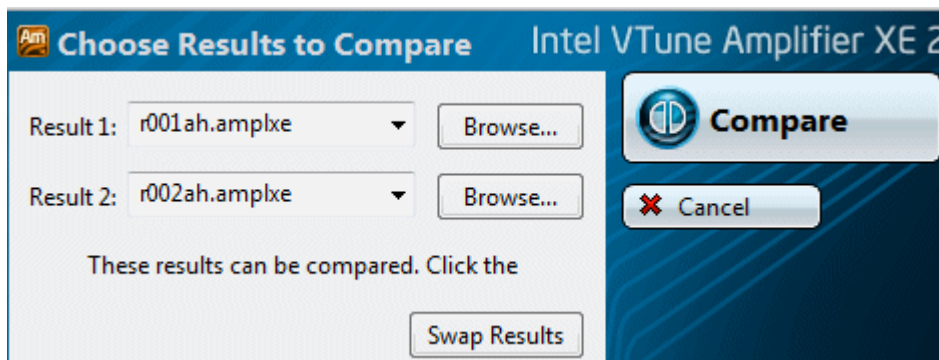


You optimized your code to apply a loop interchange mechanism that gave you 56.5 seconds of improvement in the application execution time. To understand whether you got rid of the hotspot and what kind of optimization you got per function, re-run the Hotspots analysis on the optimized code and compare results:

1. [Compare results before and after optimization.](#)
2. [Identify the performance gain.](#)

### Compare Results Before and After Optimization

1. From the **File** menu select **New > Knights Corner Platform - Hotspots Analysis**.  
VTune Amplifier reruns 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 Hotspots analysis results you want to compare and click the **Compare Results** button.



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: **<Result 1 metric> - <Result 2 metric> = <metric Difference>**.

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

<b>Elapsed Time:</b>	<b>79.174s - 15.362s = 63.813s</b>
<b>CPU Time:</b>	16853.340s - 514.740s = 16338.600s
<b>Instructions Retired:</b>	744,720,000,000 - 129,170,000,000 = 615,550,000,000
<b>CPI Rate:</b>	22.630 - 3.985 = 18.645
<b>Paused Time:</b>	Not changed, 0s

Switch to the **Bottom-up** window to view the CPU time usage per function for each result and their differences side by side.

Function / Call Stack	CPU Time:Difference	CPU Time:r001lh	CPU Time:r002lh	Module
multiply1	16720.170s	16720.170s		matrix.mic
_do_softirq	85.920s	90.260s	4.340s	vmlinux
default_send_IPI_ma	3.720s	7.660s	3.940s	vmlinux
weighted_cpuload	2.360s	2.420s	0.060s	vmlinux
run_timer_softirq	1.070s	1.170s	0.100s	vmlinux

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:r002ah** column to sort the data in the grid by this column.

Function / Call Stack	CPU Time:Difference	CPU Time:r001lh	CPU Time:r002lh	Module
multiply2	-338.060s		338.060s	matrix.mic
_ticket_spin_lock	-139.920s	0.290s	140.210s	vmlinux
_raw_spin_unlock_ir	-0.860s	6.160s	7.020s	vmlinux
_do_softirq	85.920s	90.260s	4.340s	vmlinux
default_send_IPI_ma	3.720s	7.660s	3.940s	vmlinux

The `multiply2` function shows up on top as the biggest CPU Time hotspot for the result `r002ah`, though it performs much 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

# 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:

Step	Tutorial Recap	Key Tutorial Take-aways
<b>1. Prepare for analysis</b>	<ul style="list-style-type: none"> <li>You made the build script to copy the <code>matrix</code> application to the card after each recompilation.</li> <li>You built the target application with the Intel C++ compiler, ran it on the card via <code>ssh</code>, and recorded a performance baseline.</li> <li>You created a VTune Amplifier project, specified the <code>ssh</code> script as an application to launch and specified the path to the application on the card as application parameters.</li> <li>You configured search directories to resolve symbol information for Intel Xeon Phi coprocessor-based modules and application modules.</li> </ul>	<ul style="list-style-type: none"> <li>Create a VTune Amplifier project and use the <b>Project Properties: Target</b> tab to choose and configure your analysis target.</li> <li>VTune Amplifier starts target applications from the host. It is not able to start an application directly on Intel Xeon Phi coprocessor architecture cards.</li> <li>To copy the target 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.</li> <li>To run applications on the Intel Xeon Phi coprocessor card, use <code>ssh</code> tools. See the <i>Choosing a Target on Intel® Xeon Phi™ Coprocessor</i> online help topic for other options.</li> <li>Use the <b>Analysis Type</b> configuration window to choose, configure, and run the analysis. You may choose between a predefined analysis type like the 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 <i>Creating a New Analysis Type</i> topic in the product online help.</li> <li>See the <b>Details</b> section of an analysis configuration pane to get the list of processor events used for this analysis type.</li> </ul>
<b>2. Find hotspots</b>	<p>You launched Hotspots analysis that analyzes CPU time spent in each program unit of your application and identified the following hotspots:</p> <ul style="list-style-type: none"> <li>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.</li> <li>Identified the code section that took the most CPU time to execute.</li> </ul>	<ul style="list-style-type: none"> <li>Start analyzing the performance of your application from the <b>Summary</b> 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.</li> <li>Move to the <b>Bottom-up</b> window and analyze the performance per function. Focus on the <i>hotspots</i> - 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</li> </ul>

Step	Tutorial Recap	Key Tutorial Take-aways
<b>3. Eliminate hotspots</b>	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.	<p>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.</p> <ul style="list-style-type: none"> <li>• Double-click the hotspot function in the <b>Bottom-up</b> pane to open its source code and identify the code line that took the most CPU Time and accumulated events.</li> </ul>
<b>4. Check your work</b>	You ran Hotspots analysis on the optimized code and compared the results before and after optimization	<p>Consider using the Intel compiler to vectorize instructions. Explore the compiler documentation for more details.</p> <p>Perform regular regression testing by comparing analysis results before and after optimization. From GUI, click the  <b>Compare Results</b> button on the VTune Amplifier toolbar. From command line, use the <code>amplxe-cl</code> command.</p>

**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

# 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 *Hardware Event-based 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 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.