

# Intel® Performance Tuning Utility 4.0

## User Guide

---

Copyright © 2006-2009 Intel Corporation

All Rights Reserved

Document Number: 315953-035US

Revision: 3.1.5

World Wide Web: <http://www.intel.com>



## Disclaimer and 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. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

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 by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See [http://www.intel.com/products/processor\\_number](http://www.intel.com/products/processor_number) for details.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

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

Copyright (C) 2006–2009, Intel Corporation. All rights reserved.

## Revision History

Document Number	Revision Number	Description	Revision Date
315953-003US	3.0	Initial release.	November 2007
315953-031US	3.1	<p>IPTU 3.1 release</p> <ul style="list-style-type: none"><li>• Updated information on Data Access View</li><li>• Updated information on Sampling View</li><li>• Updated information on Hotspot Analysis Configuration settings</li></ul>	February 2008
315953-032US	3.1.1	<p>IPTU 3.1 update 2</p> <p>Added information on Importing Data.</p>	April 2008
315953-033US	3.1.2	<p>IPTU 3.1 update 3</p> <p>Updated information on Data Access View.</p>	June 2008



315953-034US	3.1.3	IPTU 3.1 update 4 Added information on event ratios.	July 2008
315953-034US	3.1.4	IPTU 3.2 update 1 Added information on Advanced Profile view, Registers view, call sites, and improved experiment naming scheme.	December 2008
315953-035US	3.1.5	IPTU 4.0 Added information on Overtime view, Hotspot view with Loop Granularity, basic block execution counters, updated Advanced Profile view and information on modifying ratios.	September 2009



# Contents

---

1	About the Intel® Performance Tuning Utility .....	6
2	Data Collection .....	8
2.1	Sampling .....	8
2.2	Statistical Call Graph .....	9
2.3	Exact Call Graph .....	10
2.4	Heap Profiling.....	10
2.5	Data Access Profiling .....	11
3	Profiling Experiment .....	13
4	File Resolution .....	14
4.1	Automatic File Resolution .....	14
4.2	Search Directories .....	15
4.3	Predefined Search Directories .....	15
4.4	File Search Order.....	16
5	Event Ratios .....	17
5.1	Modifying / Creating Ratios .....	17
6	Working with the Command-line Interface .....	20
6.1	Sampling Hotspot Analysis .....	20
6.1.1	Collecting Sampling Data .....	20
6.1.2	Viewing Sampling Data .....	23
6.1.3	Comparing Two Sampling Experiments .....	23
6.2	Statistical Call Graph Analysis.....	25
6.2.1	Collecting Statistical Call Graph Data .....	25
6.2.2	Collecting Data for Specific Code Regions .....	27
6.2.3	Viewing Statistical Call Graph Data.....	28
6.2.4	Identifying Loops .....	29
6.3	Exact Call Graph Analysis.....	29
6.3.1	Collecting Call Graph Data .....	29
6.3.2	Viewing Call Graph Data .....	30
6.4	Memory Analysis .....	31
6.4.1	Collecting Heap Profile Data .....	31
6.4.2	Viewing Heap Profile Data .....	32
6.4.3	Collecting Data Access Results.....	35
6.4.4	Viewing Data Access Results.....	35
6.5	Specifying Search Directories .....	36
7	Working with the Graphical Interface.....	38
7.1	Profiling Concepts .....	38
7.1.1	Project .....	39
7.1.2	Profiling Configuration .....	40
7.2	Configuration Settings .....	41
7.2.1	Hotspot Analysis Configuration Settings .....	42



7.2.2	Call Count Analysis Configuration Settings.....	44
7.2.3	Heap Analysis Configuration Settings.....	45
7.2.4	Configuration-specific Project Properties.....	46
7.3	Workspace and Tuning Navigator .....	48
7.4	Data Views .....	48
7.4.1	Filtering Controls .....	49
7.4.2	Sampling View .....	50
7.4.3	Overtime View .....	57
7.4.4	Statistical Call Graph View .....	66
7.4.5	Call Count View .....	69
7.4.6	Heap Profiling View .....	69
7.4.7	Data Access View.....	69
7.4.8	Source View.....	75
7.4.9	Hotspot Difference Views .....	82
7.5	Specifying Search Directories .....	85
7.6	Importing Data.....	86
8	Collecting and Analyzing Data on Different Systems .....	87
8.1	Collecting Data .....	87
8.2	Converting Data .....	87
8.3	Copying Data to the Analysis Machine.....	88
9	Troubleshooting .....	89
9.1	Troubleshooting the Hotspot View .....	89
Appendix A: Command-line Reference.....		91
Appendix C: Glossary .....		99



# 1 About the Intel® Performance Tuning Utility

---

The Intel® Performance Tuning Utility (Intel® PTU) is a cross-platform performance analysis tool set succeeding the VTune™ Performance Analyzer. Along with such traditional features of the VTune analyzer as identifying the hottest modules and functions of the application, tracking call sequences, identifying performance-critical source code, the Intel Performance Tuning Utility has new, more powerful capabilities of data collection, analysis, and visualization. Most of these capabilities are developed upon requests of the users experienced in performance tuning but at the same time they are easy to use for novices.

In addition to the standard VTune analyzer functionality, this version of the Intel Performance Tuning Utility introduces the following new features:

Capability	Description	Target Audience
<b>Statistical Call Graph</b>	<ul style="list-style-type: none"><li>Analyzes and reconstructs a call flow of your application</li><li>No source code or binary instrumentation required</li><li>Low collection overhead</li><li>No driver dependency on Linux</li></ul>	Novice to Expert
<b>Predefined Profile Configurations</b>	Ensures to collect the most important and informative events for target system	Novice to Expert
<b>Automatic identification of potential performance issues</b>	Uses event ratios to automatically identify potential performance problems and bring them to the user attention for any predefined or user-created event profiles. Tuning advice can also be viewed for these issues.	Novice to Expert
<b>Call counts per call source</b>	Detects the number of times each function was called by each of its callers.	Novice to Expert
<b>Call argument statistics</b>	Provides minimum, maximum, average, and RMS values of call arguments on Intel® 64 architecture systems.	Expert
<b>Loop trip count for counted loops</b>	Provides minimum, maximum, average, and RMS values of counted loop trip counts.	Expert
<b>Advanced event-based sampling</b>	<ul style="list-style-type: none"><li>Event Multiplexing</li><li>Data Profiling</li></ul>	Expert



<b>Loop profiler</b>	Identifies loop(s) in a function and its caller function.	Intermediate
<b>Basic Block analysis</b>	<ul style="list-style-type: none"> <li>Displays disassembly code structured in basic blocks.</li> <li>Annotates each basic block with the number of event occurred in this block</li> <li>Provides Control Flow Graph for a function.</li> </ul>	Intermediate / Expert
<b>Result difference</b>	Analyzes and reports difference of two sampling experiments.	Expert
<b>Events over IP view (histogram)</b>	<ul style="list-style-type: none"> <li>Plots the number of events collected per IP range (or single) address in the selected module.</li> <li>Supports drill-down from selected IP address to its source code or disassembly if no source is available.</li> </ul>	Intermediate / Expert
<b>Heap profiling</b>	Identifies unnecessary memory allocations and memory leaks.	Intermediate / Expert
<b>Aligning Linux and Windows profiling capabilities</b>	Makes tools easier to learn and use	Novice to Expert

You can work with the Intel Performance Tuning Utility using the command-line utilities and graphical components integrated to the Eclipse\* development environment both on the Microsoft\* Windows\* and Linux\* operating systems.



## 2 Data Collection

---

The Intel Performance Tuning Utility enables collecting performance data in the following modes:

- Sampling
- Statistical call graph
- Exact call graph
- Heap profiling
- Data access profiling

There are two types of the **data collection mechanism**: **time-based** and **event-based**. The collection mechanism you choose depends on the type of interrupt you wish to use for data collection: the Operating System (OS) **timer or processor event(s)**. If you use the OS timer, the Intel Performance Tuning Utility interrupts the processor and **collects** samples **after** the specified **time interval**. If you use the processor event, the Intel Performance Tuning Utility programs the processor to interrupt execution and collects a sample **after the specified number of occurrences** of the selected processor **event(s)**. The sampling collector interrupts the processor after a specified interval (time or event-based), collects the **samples of active instruction addresses** for each processor on your system, and **writes them to a file**.

This version of the product supports the following:

- Event-based sampling collection on Windows and Linux
- Time-based statistical call graph collection on Windows and Linux
- Event-based statistical call graph collection on Windows. Limited to one PMU event per one collection run.

### 2.1 Sampling

The sampling collector profiles the whole system by interrupting active software based on the OS timer (**Time Based Sampling, TBS**) or Performance Monitoring Unit (PMU) counter overflow (**Event Based Sampling, EBS**) and captures the **IP** of interrupted process at the time of the interrupt. Statistically collected IPs of active processes enable the **viewer** to **show statistically important code regions** that affect software performance.

Sampling collection with the Intel Performance Tuning Utility is based on SEP (**Sampling Enabling Product**) and has the same average overhead – about 2% on 1ms





sampling interval. Sampling collection with the Intel Performance Tuning Utility is very similar to the sampling collector in the VTune analyzer but it has a number of enhancements in collection configuration and application management.

The number of **hardware events** that can be collected simultaneously is limited by CPU capabilities. Usually, it is **no more than 4 events**. To overcome this limitation, the Intel Performance Tuning Utility **splits** the event list **into several event** groups. Each group consists of events that can be collected simultaneously. The Intel Performance Tuning Utility uses one of the following techniques:

- Run an application several times collecting one event group during each run. This is a classical approach, well-known to the users of the VTune analyzer.
- Run an application only once and multiplex the event groups in a round robin fashion during the run. This is a new approach that is more transparent and convenient. But beware that it may not work on some OS/hardware combinations.

**See Also:**

[Sampling Hotspot Analysis](#) (CLI)

[Hotspot Analysis Configuration Settings](#) (GUI)

[Sampling View](#) (GUI)

## 2.2 Statistical Call Graph

The **statistical call graph collector** profiles your application using the OS timer (**TBS**) or PMU event (**EBS**) depending on the selected sampling mechanism. **The collector interrupts a process, collects samples of all active instruction addresses, and restores a call sequence upon each sample.** Sampled instruction pointer along with a call sequence (stack) is stored in data collection files and then can be analyzed by the Intel Performance Tuning Utility viewer. Statistically **collected IP samples** with call sequences enable the viewer to display a **call graph or/and the most time-consuming** paths. Use this data to understand the control flow for statistically important code sections.

On Linux, the statistical call graph collector embeds an agent library into the profiled application **using the LD\_PRELOAD variable**. This variable affects dynamic loader behavior and sets up the **OS timer for each thread** in the application. Upon timer expiration, **the application receives the SIGPROF** signal that is handled by the collector. The collector identifies the application stack on each sample **using the information the compiler stores inside the application.**

On Windows, the collector attaches to the process using OS debugging mechanism, and the sampling driver generates a single step on each interrupt from PMU counter overflow. The collector catches that single step debug exception and unwinds stack for currently active thread.



Average overhead of the statistical call graph collector is about 5% when sampling using default interval – 10ms.

**See Also:**

[Statistical Call Graph Analysis](#) (CLI)

[Hotspot Analysis Configuration Settings](#) (GUI)

[Statistical Call Graph View](#) (GUI)

## 2.3 Exact Call Graph

The exact call graph collector profiles your application by instrumenting function entry and exit points using the PIN dynamic instrumentation framework. For each function call, the Intel Performance Tuning Utility records a caller-callee pair together with timing and call count information. It enables you to identify how much time the application spent in each function. You may use this information to identify the code sections that could be optimized.

To analyze the resulting call graph, use the Intel Performance Tuning Utility command-line viewer that provides the following information:

- List of functions with their parents, children, and timing information
- Flat data: list of functions with timing and call information with no relationships
- Per-thread view for multithreaded applications

Collection overhead may vary depending on the overall number of loaded libraries and length of initialization stage. For computationally-intensive console applications, call graph overhead typically does not exceed 2x-8x. Call count overhead does not exceed 1.1-x1.5x.

**See Also:**

[Exact Call Graph Analysis](#) (CLI)

[Call Count Analysis Configuration Settings](#) (GUI)

[Call Count View](#) (GUI)

## 2.4 Heap Profiling

The heap profiler enables you to identify when and where dynamic memory was allocated in your application. Unnecessary allocations can decrease the application locality, resulting in performance degradation. The heap profiler helps you identify unnecessary memory allocations, as well as logical memory leaks – references to blocks of memory that are not de-allocated promptly when they are no longer used by the application.



Since the heap profiler collection requires instrumentation of your application, your application may execute slower. Collection overhead may vary depending on the overall number of loaded libraries and length of initialization stage and typically does not exceed 3x-8x for computationally-intensive console applications in exact mode and 1.1-2.0x in fast mode.

**See Also:**

[Memory Analysis](#) (CLI)

[Heap Analysis Configuration Settings](#) (GUI)

[Heap Profiling View](#) (GUI)

## 2.5 Data Access Profiling

The data access profiling enables the **analysis of data access and layout-related performance problems** in your application. The Intel Performance Tuning Utility performs the data profiling via Event-Based Sampling data collection that uses the **precise events** related to data memory operations such as loads and stores. The **events can be configured to collect extended information**, the values of all the **registers** evaluated at the IP of the interrupt, on IA-32 and Intel® 64 architecture systems. The **register values and the disassembly allows the reconstruction of the linear address of the memory operation that caused the event**. Intel PTU does this automatically. This will work with the predefined event profiles enabled for data profiling, or with user-created profiles when they are similarly enabled through the check box in the profile configuration interface. Check the event description to make sure the events you use are precise. To see the event description:

1. Open your configuration in the **Profile** dialog box.
2. From the **Hotspot Analysis** tab, click the **Add** button.  
The **Select events to add** dialog box opens.
3. Select an event from the list and see the short event description below.  
Precise events are marked as follows: INST\_RETIRED.ANY\_P (**precise**).

The following precise events are used:

Processors	Events
Intel® Core™ i7 processor	MEM_INST_RETIRED.LATENCY_ABOVE_THRESHOLD_0  <b>NOTE:</b> You may use any other MEM_INST_RETIRED.LATENCY_ABOVE_THRESHOLD_* events. MEM_INST_RETIRED.LOADS MEM_INST_RETIRED.STORES MEM_LOAD_RETIRED.DTLB_MISS MEM_LOAD_RETIRED.HIT_LFB MEM_LOAD_RETIRED.L1D_HIT



	MEM_LOAD_RETIRE.L2_HIT MEM_LOAD_RETIRE.LLC_MISS MEM_LOAD_RETIRE.LLC_UNSHARED_HIT MEM_LOAD_RETIRE.OTHER_CORE_L2_HIT_HITM MEM_STORE_RETIRE.DTLB_MISS MEM_UNCORE_RETIRE.LOCAL_DRAM MEM_UNCORE_RETIRE.OTHER_CORE_L2_HITM MEM_UNCORE_RETIRE.REMOTE_CACHE_LOCAL_HOME_HIT MEM_UNCORE_RETIRE.REMOTE_DRAM
Intel® Core™ 2 Duo processors	INST_RETIRE.ANY_P MEM_LOAD_RETIRE.L1D_MISS MEM_LOAD_RETIRE.L2_MISS MEM_LOAD_RETIRE.L1D_LINE_MISS MEM_LOAD_RETIRE.L2_LINE_MISS MEM_LOAD_RETIRE.DTLB_MISS
Intel® Pentium® 4 processors	Loads Retired 1st Level Cache Load Misses Retired 2nd Level Cache Load Misses Retired
Intel® Itanium® 2 processors	DEAR_LATENCY_ANY <b>NOTE:</b> You may use any other DEAR_LATENCY_* events.

**NOTE:** It is not necessary to collect all listed events. For example, when collecting data for Intel Core 2 Duo processors, you may use only MEM\_LOAD\_RETIRE.L1D\_MISS and MEM\_LOAD\_RETIRE.L2\_MISS events. This results in less accuracy but may significantly decrease the size of collection results and, consequently, processing time.

Due to the statistical nature of sampling, data access profiling results are also only statistically correct.

**See Also:**

[Collecting Data Access Results](#) (CLI)  
[Viewing Data Access Results](#) (CLI)  
[Hotspot Analysis Configuration Settings](#) (GUI)  
[Data Access View](#) (GUI)



## 3 Profiling Experiment

---

*Profiling experiment (or experiment)* is a basic concept of the performance analysis with the Intel Performance Tuning Utility.

Treat your profiling experiment as a physical experiment. Thus, in the physical experiment, an event under research is *sensitive to the research tools and environment conditions*. The same is with profiling when you measure something about your application or system. Once application, or system, or measurement tools change (or you just suspect that one of them might have changed), this becomes another experiment. For example, once you *recompile* the code of your application, or give *different data for the input*, or invoke another *data collector* – all these cases become *separate experiments*.

It is dangerous to mix data you collected during different experiments since it could lead to wrong conclusions. To ensure the experiment data accuracy, the Intel Performance Tuning Utility relies on you when you place the data related to a specific experiment under the *experiment directory*. This is the reason why the *command-line tools* of the Intel Performance Tuning Utility accept the *experiment directory as the first (position)* parameter. GUI version of the product facilitates this procedure by creating experiment directories automatically and locating the data collected during one profiling run under a specific experiment directory.

**See Also:**

[Profiling Concepts](#) (GUI)



## 4 File Resolution

---

Viewing and analyzing data collection results requires data from user files: modules (binaries), symbol files, and source files. Sometimes it could be problematic for the tool to find correct binaries and/or symbols, define boundaries of functions where hotspots occurred, and locate the source files. **Consider the following examples:**

- Collection is done for binaries with no debug information and no source code provided.
- Analysis is done several days after collection when some binaries and source files have been updated and do not correspond to the collection state.
- Collection is done on one machine while analysis is performed on another machine.

This section describes basic concepts of file resolution mechanism used in the Intel Performance Tuning Utility.

### 4.1 Automatic File Resolution

In many cases the Intel Performance Tuning Utility is able to resolve user files automatically. **Automatic file resolution** includes the following steps:

- Look-up by the absolute name
- Heuristic resolution of broken or incomplete file names and paths
- Search in system directories (binary and symbol files only)
- Search around corresponding binaries (symbol and source files only)

System directories are specific to the target operating system and include the following:

#### **Windows:**

- %SYSTEMROOT%, %SYSTEMROOT%/System32, %SystemRoot%/System32/drivers (binary files only)
- Directories specified in %PATH% (binary files only)
- Directories specified in %\_NT\_SYMBOL\_PATH% (symbol files only). Only local directories from this path are currently taken into account. References to the Microsoft\* Symbol Server\* are ignored.

#### **Linux:**

All the directories below are used for locating binary files only.



- `/bin`, `/usr/bin`, `/usr/local/bin`
- `/lib`, `/usr/lib`, `/usr/local/lib`
- `/lib/modules/<current kernel version>/kernel` and its subdirectories
- Directories specified in `$PATH` and `$LD_LIBRARY_PATH`
- Directories specified in the `/etc/ld.so.conf` file

## 4.2 Search Directories

Search directories provide additional **information for locating user files**. Search directories can be specific to the user file category (binary/symbol/source files) and common to all files. During resolution of a user file category (binary/symbol/source files), the Intel Performance Tuning Utility searches the directories specific to this category and, then, common search directories.

There is a set of [predefined search directories](#) used by the Intel Performance Tuning Utility. But you can specify additional search directories using both [graphical user interface](#) and [command-line interface](#).

You can set the high priority to a search directory. This means that search in this directory must be performed before automatic file resolution. High priority search directories are very important if you collect and view results on different machines. In this case, high priority search directories should point to the location of original binary/symbol/source files to prevent the Intel Performance Tuning Utility from picking up wrong versions of these files during automatic resolution.

## 4.3 Predefined Search Directories

While resolving user files for the specific experiment, the Intel Performance Tuning Utility automatically adds a set of **subfolders in the experiment directory to a search directories list**:

- `<experiment_dir>/binaries` for binary files
- `<experiment_dir>/sources` for source files
- `<experiment_dir>/symbols` for symbol files
- `<experiment_dir>/all` for all the above files

All of these search directories are recursive and of high priority. Their use makes the experiment self-contained and simplifies further work with it, for example transition to another machine.



## 4.4 File Search Order

The Intel Performance Tuning Utility uses the following order when resolving user files:

- Predefined search directories
- User-defined high priority search directories
- Automatic files resolution
- Other user-defined search directories





## 5 Event Ratios

---

Event ratios help identify potential performance problems for a code region automatically by using values of events collected during event-based sampling.

An event ratio is a dynamically calculated value based on the events that make up the formula.

A wide variety of commonly-used event ratios are predefined for your convenience. If you collect events that make up a predefined ratio, the data for the event ratio may be shown in the sampling data from the [command line](#) or GUI [Sampling Hotspot view](#).

By default, the Intel Performance Tuning Utility calculates predefined event ratios. But you can also modify them or create new ones. However, to do that, you are expected to have an expertise.

Some ratios define a threshold so that if the ratio value is greater than this threshold, it signals of a potential performance issue. By default, the Intel Performance Tuning Utility is configured to highlight cells in GUI Sampling Hotspot view where corresponding ratios exceed the threshold. This enables automatic identification of potential performance issues.

The Sampling Hotspot view also uses the ratios to define the displayed column order. The predefined ratios are all approximately normalized so that the value indicates what fraction of the execution time can be attributed to the issue the ratio identifies. The event data columns are ordered (left to right) to first show some fixed events if they are present (cycles, instructions, stalled cycles). These are then followed by the events identified in the ratio definitions, in the order of the ratio values. For the Sampling Hotspot view, the evaluation is done on the basis of all the collected events. In the source and assembly views, the order is defined by just the events in the function. This technique brings the most critical issues into immediate view.

**See Also:**

[Advanced Profile View. Automatic Identification of Performance Issues](#) (GUI)

### 5.1 Modifying / Creating Ratios

Event ratios are defined in a text file with the `.vtr` extension located in the `<install_dir>/bin` folder. There is a separate `.vtr` file for each processor supported by the product. For example, ratios for Intel® Core™ 2 processor family built using 65-nm processor technology are defined in the `pmm.vtr` file.



To modify an existing ratio or create a new one, you simply need to edit the corresponding `.vtr` file. The format for ratio definition is the following:

```
<ratio_name>[<number_format>]=<ratio_formula>
<list of optional attributes>
<ratio_description>
---
```

Where:

- `<ratio_name>` is any string that defines a ratio name;
- `<number_format>` is a ratio display format (for example, `'%'`, `'0.000'`, etc);
- `<ratio_formula>` is a formula to calculate the ratio. It can consist of events, operators, and constants. If the formula consists of events, it has the following format: `[event generator:EventName]` where `Event Generator` is the processor that collects the event data. If the formula consists of constants, the format is `[constant:Name]` where `Name` is the name of a constant;
- `<list of optional attributes>` is any of the following attributes:

Attributes	Description
<code>HID=&lt;val&gt;, HelpFileName=&lt;val&gt;</code>	Points to the ratio documentation in the product help system. Currently not used.
<code>Threshold=&lt;val&gt;</code>	Defines a threshold value. Exceeding this value indicates a potential performance issue.
<code>ThresholdEvent=&lt;event name&gt;</code>	Defines the <i>key event in the ratio</i> that you should pay attention to if the ratio value exceeds the <code>Threshold</code> value. Used together with the <code>Threshold</code> attribute.
<code>CyclesRatio=&lt;yes/no&gt;</code>	Identifies whether the ratio value means relative number of CPU clockticks.
<code>ShortDescription</code>	Provides a short ratio description to display it in a tooltip.
<code>Dependency="ratio name1, ratio name2,..."</code>	Specifies other ratios this ratio depends on. It means that if this ratio exceeds the threshold, it is not considered as an issue unless all ratios it depends on also exceed their threshold.

- `<ratio_description>` is a text describing the ratio;



- '---' is a three 'minus' symbol marking the end of the ratio definition.

The Intel PTU supports the following constants:

Constant	Description
CPU_REF_FREQ	Equals to the CPU reference frequency.
latency:data_source	Represents average access latency in CPU cycles when accessing a certain data source. Possible data sources are: <ul style="list-style-type: none"> <li>• Unknown</li> <li>• On Core</li> <li>• Local LLC</li> <li>• Remote LLC</li> <li>• Local DRAM</li> <li>• Remote DRAM</li> </ul>

Ratios are classified into groups of different priority. This affects the default column ordering in the Hotspot view. Threshold events for ratios from groups of higher priority appear before those from groups of lower priority. This enables you to focus on the most important performance issues.

To change the default grouping, modify the following settings in the .vtr file:

- SubGroup=Name – specify the name of the ratio group;
- SubGroupPriority=number – set the priority number for the ratio group. The smaller the number, the higher the group priority. The highest priority is 0.

**See Also:**

[Sampling View Preferences](#)



## 6 Working with the Command-line Interface

---

If you are interested only in the graphical interface of the product, you may skip this section and go directly to the [Working with the Graphical Interface](#) section. However, be aware that the following features are available via command line only:

- pause/resume for sampling and statistical call graph
- attach/detach for statistical call graph on Windows
- detailed loop analysis
- exact call graph data collection and analysis

### 6.1 Sampling Hotspot Analysis

#### 6.1.1 Collecting Sampling Data

To analyze your application performance using the sampling collector, configure the data collection session and launch the collector from the command line as follows:

```
vtsarun <experiment_dir> [options] [-- <application> [arguments]]
```

Where:

- <experiment\_dir> is the directory where the collection results are stored;
- [options] are the configuration settings for your experiment;
- <application> is the application to analyze;
- [arguments] are the arguments passed to the analyzed application.

Sampling collector in the Intel Performance Tuning Utility is based on SEP and supports exactly the same syntax as the SEP commands for collection configuration. It also is able to display information about event modifiers.

This version of the product supports event multiplexing. Traditionally, the number of events per sampling run is limited by the number of performance counters. Event



multiplexing is an enhancement to the traditional sampling model which removes this restriction by multiplexing the use of physical counters within a single sampling run. Event multiplexing removes the need for multiple runs of the application, thereby reducing the time needed to complete sampling collection. Event sample counts collected in the multiplexed mode are extrapolated to the total collection runtime.

Event multiplexing is also useful if the application does not have a long steady state or takes a long time to get to steady state. On the other hand, if application initialization is short and it gets to steady state quickly, then you can do multiple short runs and will not need to do event multiplexing. To enable event multiplexing, use the `-em` option. The event groups are interleaved (cycled) in round-robin fashion within the default data slice. If `dt_s` (default time slice) is not specified, the default is 50 milliseconds.

### Usage examples:

#### 1. Sampling an application:

```
$ vtsarun ./exp -- ./my_app
```

This example runs the data collection session for the application `./my_app` using default events and stores data collection files in the experiment directory `./exp`.

```
$ vtsarun ./exp -sd 20 -d 30 -ec "<event_1>:sa=10000" -- ./my_app
```

This example runs the data collection session for the application `./my_app` on event `<event_1>` with the Sample After value 10000. The collector launches the application, waits for 20 seconds, starts data collection for 30 seconds, and then stops data collection and terminates the application.

#### 2. Sampling a system:

```
$ vtsarun ./exp -s -ec "<event_1>:sa=10000" -d 10
```

This example runs a 10-second data collection session without running any application on event `<event_1>` with the Sample After value 10000. Data collection results are stored in the experiment directory `./exp`.

```
$ vtsarun ./exp -s -ec "<event_1>,<event_2>:sa=5000,<event_3>" -d 10
```

This example runs a 10-second data collection session without running any application on events `<event_1>`, `<event_2>`, and `<event_3>`. Data collection results are stored in the experiment directory `./exp`. `<event_1>` and `<event_3>` are collected with the default Sampling After value. Sampling After value for `<event_2>` is 5000.

```
$ vtsarun ./exp -s
```

Start system-wide sampling collection for default events forever until it is explicitly stopped with the `--stop` command.

#### 3. Pausing, resuming, and stopping the collection:

```
$ vtsarun ./exp --pause
```



This example pauses the collection specified by the experiment `./exp`. The pause command simply suspends generation of profiling data while application continues executing. You may use pause/resume commands to collect the data on interesting code regions only.

```
$ vtsarun ./exp --resume
```

This example resumes the sampling collection specified by the experiment `./exp`.

```
$ vtsarun ./exp --stop
```

This example stops the collection identified by the experiment `./exp`. If the experiment corresponds to the collection where application was launched, `--stop` command terminates the profiled application as well.

#### 4. Using built-in event information:

```
$ vtsarun -cl
```

This example displays the list of supported CPU names along with their aliases/shortcuts for convenience.

```
$ vtsarun -el -c I50
```

This example displays all events available for the Intel® Core 2 Duo processor (alias name is I50). If the `-c` option is omitted, `vtsarun` displays available events for the current processor.

**NOTE:** Enter `vtsarun -cl` to know the alias name for your processor. Alias names are provided in parentheses. For example: for Intel® Core™ i7 processor (I70), I70 is an alias name. In the CPU list, your current processor is marked with an asterisk in the beginning. If you choose to use a full processor name, make sure to use quotation marks.

```
$ vtsarun -ml RESOURCE_STALLS.ANY -c I50
```

This example displays descriptions of all event modifiers available for the `RESOURCE_STALLS.ANY` PMU event on the Intel Core 2 Duo processor (alias name is I50).

#### 5. Event multiplexing:

```
$ vtsarun ./exp -s -em -dts 100 -ec  
INST_RETIRED.ANY_P, MEM_LOAD_RETIRED.L1D_MISS, MEM_LOAD_RETIRED.L2_MISS -d  
10
```

This example samples `INST_RETIRED.ANY_P`, `MEM_LOAD_RETIRED.L1D_MISS`, and `MEM_LOAD_RETIRED.L2_MISS` events within one experiment run using the event multiplexing approach with the data slice time equal to 100ms. Without event multiplexing, this example requires three runs.

#### See Also:

[Profiling Configuration](#) (GUI)

[Hotspot Analysis Configuration Settings](#) (GUI)

[Configuration-specific Project Properties](#) (GUI)



## 6.1.2 Viewing Sampling Data

To view the collected data from the command line, run the `vtssaveview` command as follows:

```
vtssaveview <experiment_dir> [options]
```

Where:

- `<experiment_dir>` is the directory where the collection results were stored;
- `[options]` are the options for the data view.

When the `vtssaveview` is called for the first time for some `<experiment_dir>` it first performs conversion of all `tb5` files in the directory to a single database. This database will contain all information needed by sampling viewer including functions names and basic blocks. After the conversion is done the experiment could be easily copied to another machine and analyzed there.

### Usage examples:

```
$ vtssaveview ./exp --re-convert
```

This example converts raw data in the `./exp` directory to a single database.

```
$ vtssaveview ./exp -g f
```

This example displays functions hotspots with the profiling data in the directory `./exp`.

```
$ vtssaveview ./exp -g rva -f module,my_module
```

This example displays RVAs hotspots for the module with name `my_module` with the profiling data in the directory `./exp`.

```
$ vtssaveview ./exp -g f -er
```

This example displays functions hotspots with collected events and ratios calculated based on them.

### See Also:

[Sampling View](#) (GUI)

## 6.1.3 Comparing Two Sampling Experiments

Use the `vtssadiff` command to compare two sampling experiment results. This types of analysis could be useful if you measure the application performance, optimize the code, and run the experiment with the same workload to identify the performance difference against the benchmarks. The `vtssadiff` command displays the results of the selected experiments and provides difference in sample count for each event.

Run the `vtssadiff` command as follows:



```
vtسادiff [experiment 1 options] <experiment_dir1> [experiment 2 options]  
<experiment_dir2> [common options]
```

Where:

- [experiment options] list the [search directories](#) for modules; these options are experiment-specific.
- <experiment\_dir> is an experiment directory containing all \*.tb5 files generated for the specified experiment;
- [common options] are filtering options for the data view. There are primary and secondary view options. Secondary options can be used only when preceded with the corresponding primary option.

In the **vtسادiff** output, the Clockticks-like event values are represented as the Time value (in milliseconds). These values approximately correspond to the time spent for execution of the respective unit (module, function, basic block, and so on). The Time value is calculated as follows:

Time = Clockticks total value / (CPU frequency \* 1000), where CPU frequency is in MHz.

The **Time:Diff** column value displays the time difference between the first and second experiments:

**Time:Diff = Exp1 Time - Exp2 Time.**

You can modify the CPU frequency value and analyze how the code execution time depends on the CPU frequency by using the `--cpu-frequency` (or `-f`) option. Or, you may use this option to specify the CPU frequency for a particular experiment if it was not properly detected during the data collection. Specify the frequency value `-f <value>` before the experiment you change this value for. If you specify the same frequency for both experiments, specify the `-f <value>` option after both experiments.

### Usage examples:

```
$ vtسادiff ./exp1 ./exp2
```

This example displays difference at function level for two experiments residing in subdirectories `exp1` and `exp2` in the current directory.

```
$ vtسادiff /tmp/exp1 /tmp/exp2 --search-dir bin=/tmp/modules -g m
```

This example displays difference at module level for two experiments residing in the directories `/tmp/exp1` and `/tmp/exp2`. **vtسادiff** extends the search for binaries to the directory `/tmp/modules` for both experiments.

```
$ vtسادiff --search-dir bin=/tmp/modules1 ./exp1 --search-dir  
bin=/tmp/modules2 ./exp2 -g b --filter-module vtune
```





This example displays difference at basic block level. `vtسادiff` displays basic blocks that belong to the module with the substring `vtune` in its name. `vtسادiff` extends the search for binaries to the directory `/tmp/modules1` for the first experiment and to the directory `/tmp/modules2` for the second experiment.

```
$ vtسادiff ./exp1 ./exp2 -g rva --filter-module vtunedemo,uhci_hcd
```

This example displays difference at RVA level. `vtسادiff` displays addresses that belong to module(s) with the substring `vtunedemo` or `uhci_hcd` in the name.

```
$ vtسادiff ./exp1 -f 3000 ./exp2
```

This example displays difference at function level for two experiments residing in subdirectories `exp1` and `exp2` in the current directory. For the second experiment, the CPU frequency value is changed to 3000Mhz. The updated Time value is displayed in the **Time(msec)** column.

**See Also:**

[Hotspot Difference Views](#) (GUI)

[Modifying the CPU Frequency](#) (GUI)

## 6.2 Statistical Call Graph Analysis

### 6.2.1 Collecting Statistical Call Graph Data

To analyze your application performance using the statistical call graph collector, configure the data collection session and launch the collector from the command line as follows:

```
vtssrun <experiment_dir> [options] -- <application> [arguments]
```

Where:

- `<experiment_dir>` is the directory where the collection results (with \*.vtss extension) are stored;
- `[options]` are the configuration settings for your experiment;
- `<application>` is the application to analyze;
- `[arguments]` are the arguments of the analyzed application.

**NOTE:** You may provide a shell or Perl script as an application. In this case, the collector profiles the shell by itself and all children of the shell process.

To view the profiling results on any machine and speed up the result display, convert all the collected data to a database file (\*.db) as follows:

```
vtssview <experiment_dir> --convert
```

**NOTE:** You can copy/move database file from one platform to another.

**Usage examples:**

```
$ vtssrun ./exp -- /bin/ps
```

This example runs the data collection session for the application `/bin/ps` and stores data collection files in the `./exp` directory.

```
$ vtssrun ./exp -st shell:notrace:trace -st sh:notrace:trace --  
./my_script
```

This example runs an experiment with a script as an application and disables shell profiling.

```
$ vtssrun ./exp -nt -s gcc:trace,notrace -- make
```

This example profiles only processes with `gcc` name launched from `make`.

```
$ vtssrun ./exp -nt -st gcc:trace,trace -- make
```

This example profiles the `gcc` compiler and all the processes it launches.

```
$ vtssrun ./exp -st bash:notrace:trace -st perl:notrace:trace -- myscript
```

This example tells the collector to profile all processes except for 'bash' and 'perl' processes.

```
$ vtssrun ./exp -sd 20 -d 30 -i 1 -- ./my_app
```

This example runs the data collection session for the application `./my_app` with sampling interval equal to 1ms (NOTE: It is limited by OS timer granularity so actual interval may be higher). The collector launches the application, waits for 20 seconds, starts data collection for 30 seconds, and then it stops data collection and terminates the application.

**Windows specific usage examples:**

```
$ vtssrun ./exp -ec CPU_CLK_UNHALTED.CORE:sa=100000 -- ./my_app
```

This example runs statistical call graph collection for event-based sampling. Note you may specify only one hardware event. Syntax of `-ec` option is exactly the same as for `vtssrun` command.

```
$ vtssrun ./exp -a 213
```

This example attaches to process with PID 213 and runs time-based statistical call graph collection for it. To stop the collection, press `Ctrl+C` in the terminal window or use the `-d` command in other console.

```
$ vtssrun ./exp -d 213
```

This example stops collection for process with PID 213 and the collector detaches from that process. If no process with the given PID was attached to this experiment, error message is printed.

**See Also:**

[Profiling Configuration](#) (GUI)

[Hotspot Analysis Configuration Settings](#) (GUI)



## 6.2.2 Collecting Data for Specific Code Regions

Use the pause/resume options to focus the profiling on specific code regions. Consider using these options for long-running server applications.

For example, to profile the `myserver` application under the workload generated by the `myworkload` command, do the following:

1. Start the collector in a paused mode.

```
$ vtssrun ./exp -p all -- myserver
```

Server launches and stabilizes.

2. In another terminal window, resume the profiling of the experiment:

```
$ vtssrun ./exp -r all
```

The collector looks into the experiment directory, determines currently running processes, and resumes profiling of these processes.

3. Launch the `myworkload` command.

```
$ myworkload
```

4. Pause the collection again to ignore further collection.

```
$ vtssrun ./exp -p all
```

5. View the results.

```
$ vtssview ./exp -p
```

**NOTE:** You must specify value for `-p` command. To pause all processes in experiment directory, use **all** value. If the value is not specified, it is interpreted as syntax error.

You may pause/resume a profiling experiment any amount of times and see the results after each pause command.

You may also pause/resume each running process individually. To see what process and in which state is currently running:

1. Use the `--list (-l)` command or pass the experiment directory to the collector command.

```
$ vtssrun ./exp
```

2. Pause/resume an individual process by passing PIDs to `-p` and `-r` commands.

```
$ vtssrun ./exp -p 23567 23678
```

```
$ vtssrun ./exp -r 23567
```

**NOTE:** You may stop the collection by terminating all running applications, for example:

```
$ vtssrun ./exp -s
```



## 6.2.3 Viewing Statistical Call Graph Data

To view the collected data from the command line, run the `vtssview` command as follows:

```
vtssview <experiment_dir> <options>
```

Where:

- `<experiment_dir>` is the directory where the collection results (with \*.vtss extension) are stored;
- `<options>` are the view mode and view options. There are primary and secondary view options. Secondary options can be used only when preceded with the corresponding primary option.

The command-line viewer displays the collected data in the following levels of detail:

- **Flat profile view** (`--flat-profile` | `-p`) displays the most time-consuming functions (hotspot functions).
- **Call Graph view** (`--graph` | `-q`) lists functions, their callers and callees.
- **Hot Stack view** (`--hot-stack` | `-t`) displays a list of call sequences for each function.

### Usage examples:

```
$ vtssview ./exp -c
```

This example opens the directory with the experiment results and `converts` all data collection files to a single database. The viewer creates a .db file with converted results in the specified directory and adds loop detection information to the database.

```
$ vtssview ./exp -p
```

This example displays the `Flat Profile` view with the profiling data in the experiment directory. The viewer looks for the .db file in the experiment directory and uses it to show the view. If the viewer cannot find a database, it automatically converts all data collection files from the experiment directory to a database and uses it to show the view.

```
$ vtssview ./exp -q
```

This example displays the `Call Graph` view from a .db file in an experiment result directory. The viewer automatically converts data collection files to a database, if no database is found.

```
$ vtssview ./exp -t
```

This example displays the `Hot Stack` view from the experiment result directory. The viewer automatically converts data collection files to a database, if no database is found.

```
$ vtssview ./exp -t -m m
```



This example displays the Hot Stack view from the experiment result directory with mangled function names.

```
$ vtssview ./exp -p -n 5 -s module -F function,main
```

This example displays the Flat Profile view from the specified experiment directory, limits output to 5 functions, sorts the table by the Module column, and excludes functions that contain 'main' string in their name.

```
$ vtssview ./exp -t -p -q -width function,50,module,50
```

This example displays the Flat Profile, Hot Stack, and Call Graph views from the specified experiment directory and sets Function and Module column width to 50.

```
$ vtssview ./exp -t -q -f module,libc
```

This example displays the Hot Stack and Call Graph views from the specified experiment directory and shows information for modules with the 'libc' string in their name only.

**See Also:**

[Statistical Call Graph View](#) (GUI)

## 6.2.4 Identifying Loops

To view loop data from the command line, use the `-loops` (or `-l`) option. For example:

```
$ vtssview ./exp -p -l
```

This example opens the Flat Profile view from the specified experiment directory and shows the collected data with timing information for detected loops.

In the data view, the Intel Performance Tuning Utility displays Total and Self time for detected loops, where:

- **Loop Total time** is the time spent in a loop body, including time spent in nested loops and time spent in functions called from the loop and its nested loops.
- **Loop Self time** is the time spent in a loop body without including nested loops and loops located in functions called from this loop.

## 6.3 Exact Call Graph Analysis

### 6.3.1 Collecting Call Graph Data

To analyze your application performance using the exact call graph collector, configure the data collection session and launch the collector from the command line as follows:



```
vtcgrun <experiment_dir> [options] -- <application> [arguments]
```

Where:

- `<experiment_dir>` is an experiment directory where profiling results are stored;
- `[options]`, specify the configuration settings for your experiment;
- `<application>` is the application to analyze;
- `[arguments]` are the arguments passed to the application.

#### Usage examples:

```
$ vtcgrun ./data -- mozilla
```

This example runs the data collection session for the `mozilla` application and stores data collection files in the `./data` directory.

```
$ vtcgrun data1 -i ignorefile -- mozilla
```

This example runs the data collection for the `mozilla` application and does not profile auxiliary applications used by `mozilla` that are specified in the `ignorefile` file.

```
$ vtcgrun data -cc -- mozilla
```

This example only collects data on function calls in the `mozilla` application. Timing information and call graph is not analyzed.

#### See Also:

[Profiling Configuration](#) (GUI)

[Call Count Analysis Configuration Settings](#) (GUI)

## 6.3.2 Viewing Call Graph Data

To view the collected call graph data from command line, run the `vtcgview` command as follows:

```
vtcgview <experiment_dir> <options>
```

Where:

- `<experiment_dir>` is the directory where the collection results are stored;
- `<options>` are the data collection file and view mode. There are primary and secondary view options. Secondary options can be used only when preceded with the corresponding primary option.

**NOTE:** You can also view the collected exact call graph data in Eclipse.

#### Usage examples:

```
$ vtcgview ./data
```

This example displays the list of collection results stored in the `./data` directory.

```
$ vtcgview ./data1 -p -n 5
```



This example displays the Flat Profile view from the `./data1` experiment directory and limits output to 5 rows. By default, the data are sorted by total time.

```
$ vtcgview ./data1 -p -n 5 --sort-desc=calls
```

This example displays the Flat Profile view from the `./data1` experiment directory, limits output to 5 rows, and sorts the table by the `calls` column in the descending order.

```
$ vtcgview data1/20070406135309525557000_27158.ecg --graph
```

This example displays the Call Graph view for the data collection result stored in the `20070406135309525557000_27158.ecg` file. The file name corresponds to the `data:time:PID` format. To get the name of a particular result, enter: `vtcgview <resultdir>`.

```
$ vtcgview data --convert --flat-profile my-db-file.db
```

This example exports the Flat Profile view to the `my-db-file.db` database file.

**NOTE:** Only the Flat Profile view export is supported.

**See Also:**

[Call Count View](#) (GUI)

## 6.4 Memory Analysis

### 6.4.1 Collecting Heap Profile Data

To collect data on the dynamic memory allocations in your applications, launch the heap profiler from the command line as follows:

```
vthprun <experiment_dir> [options] -- <application> [arguments]
```

Where:

- `<experiment_dir>` is the directory where collection results are stored;
- `[options]` are the data collection configuration settings;
- `<application>` is the application to analyze;
- `[arguments]` are the arguments passed to the application.

You can define custom memory allocation/deallocation functions in the `bin/hp.allocators` file and profile these functions using the `--allocators` (or `-a`) option. For example:

```
$ vthprun /results/data -a=/home/users/iptu/bin/hp.allocators -- myapp
```



The `hp.allocators` file provides the syntax and an example of specifying the allocation/deallocation functions. By default (without the `--allocators` option), the heap profiler analyzes the following memory allocation/deallocation functions: `malloc`, `calloc`, `free`, `realloc`, `_malloc_dbg` (defined in the debug version of the Microsoft\* C runtime), `_free_dbg` (defined in the debug version of the Microsoft C runtime). Be aware that an incomplete list of allocators can lead to the wrong results. Thus, if an application allocates memory using `malloc` and releases using `custom_free`, the heap profiler, by default, profiles only `malloc` but not `custom_free`. This can lead not only to the wrong report on memory leaks and spacetime but to overlapping of block addresses and result in a wrong number of allocated blocks. If this happens, the heap profiler provides an error message.

#### Usage examples:

```
$ vthprun /tmp/vtunedemo_results --exact=no -- ./vtunedemo
```

This example profiles the `vtunedemo` application in the fast mode.

```
$ vthprun /results/app_profile --trace=yes -- ./myapp
```

This example collects information about heap usage over time during data collection.

```
$ vthprun /results/data --trace-children=yes -- ./launcher_script.sh
```

This example profiles the application that is launched via the `launcher_script.sh` script or spawns child processes.

```
$ vthprun /tmp/res --ignore=apps_to_ignore.lst --trace=yes -trace-children=yes -- /home/user/myapp
```

This example excludes applications listed in the `apps_to_ignore.lst` file from profiling.

#### See Also:

[Profiling Configuration](#) (GUI)

[Heap Analysis Configuration Settings](#) (GUI)

## 6.4.2 Viewing Heap Profile Data

To view the collected heap profile data from command line, run the `vthpview` command as follows:

```
vthpview <experiment_dir> [options]
```

Where:

- `<experiment_dir>` is the directory where collection results are stored;
- `[options]` are the options for the data view. There are primary and secondary view options. Secondary options can be used only when preceded with the corresponding primary option.

The command-line viewer displays the collected data at the following levels of detail:





### Time Weighting Information View (`--spacetime`)

This view shows overall memory usage and helps identify logical memory leaks. It introduces the *spacetime* value that links the amount of memory allocated by your application with the time during which this memory was retained. The spacetime is calculated as space (in bytes) multiplied by time (in seconds). After fixing top entries of the report (releasing memory in advance), your application will use less virtual memory overall.

### Memory Footprint View (`--footprint`)

This view shows the *memory footprint* over time. It provides a table that consists of a header enumerating the timestamps for the data listed in the table. Each row in the table describes the amount of memory allocated at the timestamp by a certain thread. If the application is multi-threaded, the table includes several rows, one for each thread. You can use the timestamps enumerated in the header to specify values for the `--begin/--end` options of all other views.

### Allocation Call Graph View (`--allocgraph`)

This view displays a hierarchical representation of allocations by call stack. In each section (separated by a line of dashes), the first listed function is a caller and other functions are callees (functions called by the caller). The Allocation Call Graph view provides the following information:

Column Heading	Description
Index	Index of a function.
Self/Total	Contribution of the function and all its children into the whole memory allocated by the application.
Self % Memory	Amount of memory allocated inside the functions directly and indirectly. The values in parentheses are the percentage contribution into the total values among the functions from one section.
Self% Objects	Number of allocated blocks.
Name [Index] Callee	Function and its callees.

### Function Summary View (`--functions`)

This view shows the summary by the functions. Use this view to quickly identify the number of allocated objects inside the function and estimate amount of allocated memory.

**NOTE:** All options applicable for allocation call graph are applicable for the function summary.

**Memory Leaks View (--memleaks)**

The view shows the number of retained/not released blocks. By default, it shows memory that was not returned by the end of the application run. When trace data is present, you can identify logical memory leaks. To do this, use the `-begin` and/or `-end` options to specify the time period in which all newly allocated memory should be released. All memory leaks are sorted by the amount of retained/not released memory. The memory leaks view provides the following information:

- Summary about the retained/not released block
- ID, thread ID, number of bytes, and number of blocks for the critical function
- Call stack where this memory was retained/not returned

**Block Size Distribution View (--objects)**

This view shows the distribution of memory blocks by their size. The records are grouped by size of allocated memory blocks. For each byte size in which memory was allocated, the allocation memory block table shows the number of allocations of that size (`alloc`) and the number of retained/not released memory blocks of that size (`retained`).

**NOTE:** By default, all views show only the first 10 functions, sorted by the amount of allocated memory. To see all data, use the option `--all-data`.

**Usage examples:**

```
$ vthpview /tmp/ls_results
```

This example displays the default spacetime view.

```
$ vthpview /tmp/vtunedemo_results --list
```

This example displays all heap profile results stored in the experiment directory. Consider using this option for multiple runs or when multiple processes were spawned.

```
$ vthpview /tmp/vtunedemo_results -footprint
```

This example displays memory footprint over time.

```
$ vthpview /tmp/ls_res --memleaks --csv --csv-delimiter="|"
```

This example displays the memory leaks view in the CSV format using `|` as a delimiter.

```
$ vthpview /tmp/vtunedemo_results --allocgraph --begin=1000 --end=5000
```

This example displays the memory allocation graph view in the time range [1000, 5000].

```
$ vthpview /tmp/vtunedemo_results --meleaks --all-data --csv
```

This example displays information about all logical memory leaks (top ten, by default) in the CSV format with the default delimiter.

**See Also:**

[Heap Profiling View](#) (GUI)

## 6.4.3 Collecting Data Access Results

To analyze data-related problems in your application, configure the data collection session and launch the collector from the command line as follows:

```
vtsarun <experiment_dir> -dl -ec <data_profiling_events> [options] [--<application> [arguments]]
```

Where:

- *<experiment\_dir>* is the directory where collection results are stored;
- *<data\_profiling\_events>* is the list of sampling events required for data profiling collection. To view the list of events available on your processor, enter: `vtsarun -el`.
- *[options]* are the data collection configuration settings;
- *<application>* is the application to analyze;
- *[arguments]* are the arguments passed to the application.

When collecting data on IA-32 and Intel® 64 architecture, enable event multiplexing to ensure that all events are collected during one application run. To enable event multiplexing, use the `-em` option.

**NOTE:** [Event multiplexing](#) is not supported on the processors older than the Intel® Core™ 2 Duo processors.

**Usage examples:**

```
$ vtsarun ./exp -dl -em -ec "INST_RETIRED.ANY_P",
"MEM_LOAD_RETIRED.L1D_MISS", "MEM_LOAD_RETIRED.L2_MISS" -- ./my_app
```

This example runs data profiling collection session on the Intel Core 2 Duo processor for the application `./my_app`. The data collection files are stored in the directory `./exp`.

**See Also:**

[Profiling Configuration](#) (GUI)

[Hotspot Analysis Configuration Settings](#) (GUI)

## 6.4.4 Viewing Data Access Results

To view the collected data from the command line, run the `vtdpview` command as follows:



```
vtdpview <experiment_dir> [options]
```

Where:

- `<experiment_dir>` is the directory where the collection results were stored;
- `[options]` are the options for the data view.

#### Usage examples:

```
$ vtdpview ./exp --convert
```

This example performs explicit conversion of all raw data files in the `./exp` directory to a single database.

```
$ vtdpview ./exp -granularity function --filter module,my_module
```

This example displays functions hotspots for the `my_module` module with the profiling data in the `./exp` directory.

```
$ vtdpview ./exp -granularity cacheline --filter process,my_process
```

This example displays cachelines view for the `my_process` process with the profiling data in the `./exp` directory.

#### See Also:

[Data Access View](#) (GUI)

## 6.5 Specifying Search Directories

Each viewer (except for the Heap Profiler and Exact Call Graph) of the Intel Performance Tuning Utility supports command-line options for specifying [search directories](#). To specify a search directory, use the `--search-dir` option. This option can be used multiple times and has the following format:

```
--search-dir bin|sym|src|all[:r|p]=<directory>
```

This option specifies a search directory for binary (`bin`), symbol (`sym`), source (`src`) files, or all above (`all`). Additional options can be specified after colon. Use the `r` option to enable search in subdirectories. Use the `p` option to set the high priority to a search directory.

**NOTE:** Search directories you specify affect only the first run of the viewer in the experiment. To apply new search directories to this experiment, use the `--re-convert` option.

#### Usage examples:

```
$ vtsaview c:\myExperiment -search-dir bin:rp=C:\myProject
```

This example displays sampling data collection results. To resolve binary file associations, the `C:\myProject` directory is searched. The search includes subdirectories and starts prior to [automatic file resolution](#).



```
$ vtsadiff -search-dir src:r C:\MyProject1 C:\Experiment1 -search-dir  
bin=c:\MyProject2\bin C:\Experiment2 -search-dir  
all:rp=C:\CommonFilesForBothExperiments
```

This example displays difference for two sampling data collection results. To resolve file associations, the following settings are applied:

- Binary/source/symbol files for both experiments are searched in the `C:\CommonFilesForBothExperiments` directory and its subdirectories.
- Source files for the first experiment that are not found on step 1 are searched in the `C:\MyProject1\` directory and its subdirectories.
- Binary files for the second experiment that are not found on step 1 are searched in the `C:\MyProject2\bin` directory.

**See Also:**

[Specifying Search Directories](#) (GUI)  
[Predefined Search Directories](#)



## 7 Working with the Graphical Interface

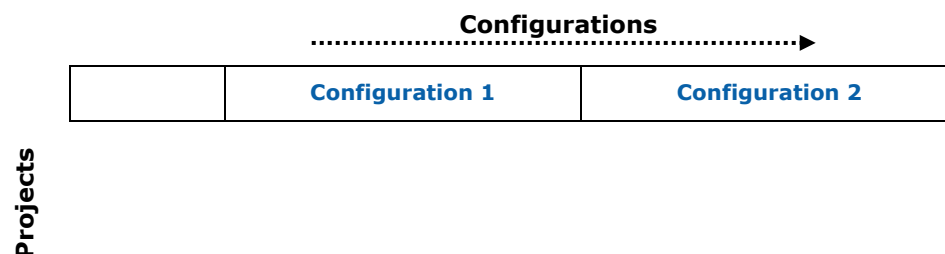
If you are interested only in the graphical interface of the product, you may skip the [Working with the Command-line Interface](#) section. It is easier to use the Intel Performance Tuning Utility via GUI because the GUI version of the product can generate for you a command line for the selected collector and open the viewer after data collection. However, be aware that the following features are available via command line only:

- pause/resume for sampling and statistical call graph
- attach/detach for statistical call graph on Windows
- detailed loop analysis
- exact call graph data collection and analysis

### 7.1 Profiling Concepts

The performance analysis with the Intel Performance Tuning Utility starts with creating a new *Project* where you specify an application as an object of your analysis. You can treat the [Project](#) as a container for the workload you want to profile. Usually, it is an application, application command-line option, and working directory.

Another important concept is a *Profiling Configuration*. [Profiling Configuration](#) is orthogonal to the Project concept: they exist in the Eclipse workspace independently of each other. Profiling Configuration (or just *Configuration*) is a combination of data collection settings. You may use one of default configurations predefined by the Intel Performance Tuning Utility or create your own configuration. By applying a Configuration to a Project you get the *Experiment Result*. Experiment Result is what you get after a data collection session.





▼	Project 1		
	Project 2	Experiment Result	
	Project 3		Experiment Result

What is important is that a custom Configuration you create can be applied to any Project. So, if you create custom configurations and start analyzing a new application, all your custom configurations are available to the new Project.

The concepts of Workspace, Project, and Experiment are reflected directly in the file system. Eclipse workspace is a directory you specify when you start Eclipse for the first time. Projects are located as sub-directories of the workspace directory. [Experiment directories](#) are located in a project directory.

Profiling with the Intel Performance Tuning Utility consists of the following steps:

1. Create and configure a project.
2. Select a basic configuration or create your own configuration.
3. Run data collection.
4. View the collected results.

You may iterate all these steps several times to analyze the application performance deeper.

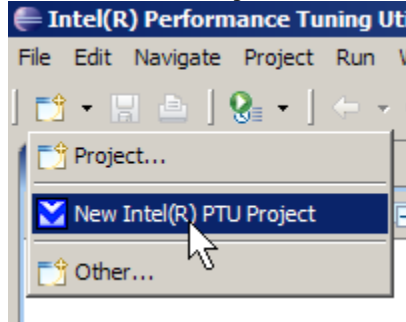
## 7.1.1 Project

Project identifies a workload that you want to analyze and contains application settings. A Project can contain one application only.

It is important to differentiate between the Intel Performance Tuning Utility Project and the VTune analyzer Project. In the VTune analyzer, the Project can contain a number of Activities where each Activity corresponds to an individual application workload. In the Intel Performance Tuning Utility, the Project corresponds to one workload. Think of it as of "something to run".

**To create a new Project:**

1. Click the **New Project** button arrow and select **New Intel(R) PTU Project**.



The **New Project Wizard** dialog box opens.


2. In the **Project name** and **Location** fields, specify the name and location of your Project. Usually, the name of the Project is the name of the executable you want to analyze.
3. Click **Next**.
4. In the **Application** field, enter a full path to the application you want to analyze. The **Working directory** field is assigned automatically as you change the **Application** field. It defaults to the directory where the application resides. Change the default working directory, if required. If you want to launch the application yourself, select the **Profile without launching an application** option.
5. In the **Application parameters** field, enter application arguments, if necessary.
6. In the **Workload duration** field, specify the duration of data collection. The default session duration is 20 seconds.
7. Click **Finish**.  
Your project appears in the **Tuning Navigator**.

Next step is to apply a Profiling Configuration to the newly created Project.

## 7.1.2 Profiling Configuration

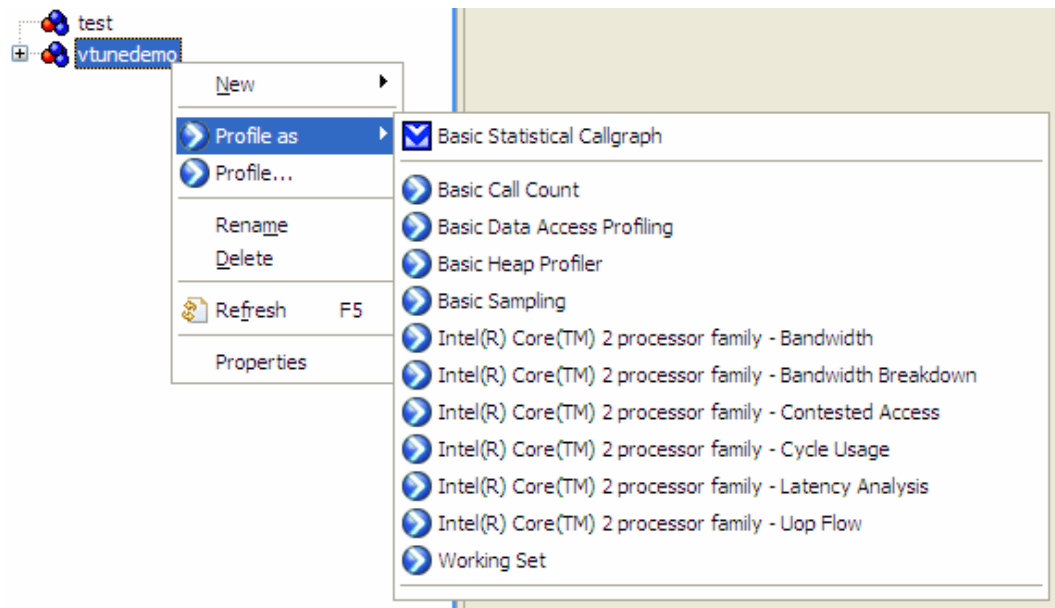
You may use predefined or custom configurations to collect Experiment Results for a Project.

### To use a predefined configuration:

1. Click the arrow on the  **Profile** toolbar button.
2. From the drop-down menu, select **Profile As**.  
Another drop-down menu opens.
3. Select one of the available configurations.  
Data collection starts.

You may also select a default configuration by right-clicking a Project node in the **Tuning Navigator** and selecting the **Profile As** cascade menu.





Based on the default configuration, you may create your own configuration that would use, for example, some specific hardware events.

#### To create/modify a profiling configuration:

1. Click the arrow on the **Profile** button.
2. From the drop-down menu, select **Profile....**  
The **Profile** dialog box opens.
3. Click the **New Launch Configuration** button to create a new configuration.
4. From the right pane, select the tab with the type of analysis to perform and modify [configuration settings](#) as required.
5. Click **Apply** to save your settings.
6. Click **Profile** to start data collection.  
The collection starts for the Project selected in the **Profile project** combo box. It may happen that the current configuration cannot be applied to a particular Project. In this case, an error message pops up and the **Profile** button is disabled.

## 7.2 Configuration Settings

There are two types of settings that affect your configuration:

- **Analysis settings:** To set up analysis settings, use the **Profile** dialog box and select the type of analysis you wish to perform for the current configuration.  
The following types of analysis are available:
  - Hotspot Analysis
  - Heap Analysis



- Call Count Analysis

Click the required tab and modify settings for the existing configuration or specify them for a new one.

The **Profile** dialog box also includes the **Description** tab. For advanced predefined configurations provided with the Intel Performance Tuning Utility, the **Description** tab contains information on each configuration and events it uses. You may create/modify the descriptions for your own configurations and exchange them to share performance tuning expertise.

- [Configuration-specific project settings](#)

## 7.2.1 Hotspot Analysis Configuration Settings

The hotspot analysis combines [sampling](#), [statistical call graph](#), and [data access profiling](#) analysis. To select the hotspot analysis for your configuration and edit configuration settings, check the **Enable Hotspot Analysis** box.

Hotspot Analysis | Call Count Analysis | Heap Analysis | Description

☒ Enable Hotspot Analysis

☐ Time-based sampling

Sampling interval, msec: 10

☒ Event-based sampling

Configured events for CPU: Intel(R) Core(TM) i7 Processor pmn.xml

Event Name	Sample After	LBR Filter
BR_INST_EXEC.NEAR_CALLS	20 000	All Branches
BR_INST_RETIRED.NEAR_CALL_R3	20 000	All Calls

☐ Enable statistical callgraph ☒ Enable loop analysis ☐ Enable data access profiling

Profile project: test Properties... Get commands

- **Time-based sampling:** Enable the time-based sampling mechanism.
- **Sampling interval, msec:** Set up the period of sampling for time-based mechanism.
- **Event-based sampling:** Enable the event-based sampling mechanism.
- **Configured events for CPU:** For event-based sampling, select the processor you use to perform the hotspot analysis.



- **Event Name** column: For event-based sampling, click the **Add** button to select events to monitor. Selected events are added to the **Event Name** column.
- **Sample After Value** column: Specify the number of events after which the Intel Performance Tuning Utility interrupts the processor to collect data.
- **LBR Filter** column: Click a row to select a filter for an event and enable the collection of filtered Last branch records (LBRs). This column is available for Intel® Core™ i7 processor family only.
- **Enable statistical call graph**: Check this box to analyze call sequences in your application.
- **Enable loop analysis**: Check this box to have the Intel Performance Tuning Utility display [hints on loops](#) in the functions.
- **Enable data access profiling**: For event-based sampling, check this box to collect additional information required to display [data access profiling](#) information.

**NOTE:** When collecting data profiling information, make sure to select the event multiplexing option in the configuration-specific project properties. This enables collecting all needed information in one run, which is especially important when analyzing multi-threaded applications.

To see the command lines used to collect data with this configuration for the chosen Project, use the **Get Commands** button. In case of the event-based sampling mechanism, command lines depend on the CPU currently selected from the CPU drop-down list. This feature is useful for generating command lines for a CPU which is different from the CPU on the host machine.

**NOTE:** You can select event-based sampling and statistical call graph analysis in one experiment. In this case the collection will be run two times.

**See Also:**

[Collecting Sampling Data](#) (CLI)

[Collecting Statistical Call Graph Data](#) (CLI)

[Collecting Data Access Results](#) (CLI)

### 7.2.1.1 Configuring for Viewing Call Sites

On the Intel® Core™ i7 processor family, you can create/modify a profiling configuration to collect and view sampling hotspot data with call sites.

**To enable collecting call site data:**

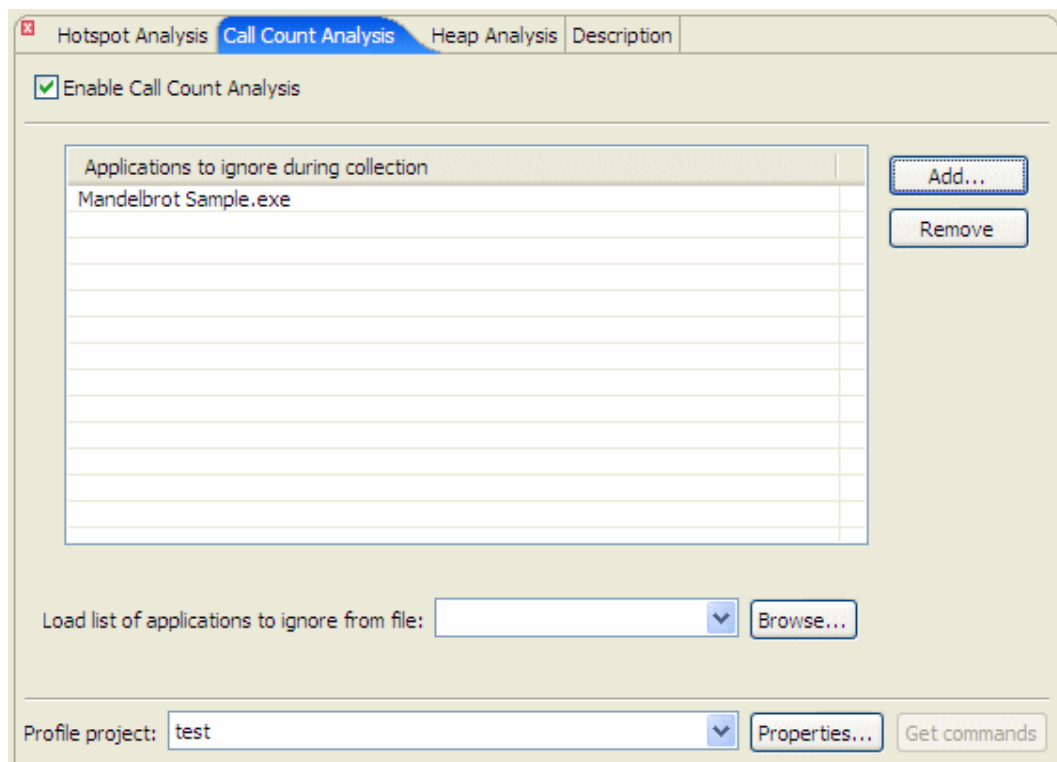
1. [Create a profiling configuration](#).
2. From the **Hotspot Analysis** tab, select **Event-based sampling** and choose the required Intel® Core™ i7 processor from the drop-down menu.



3. Click the **Add...** button.  
The **Select events to add** dialog box opens.
4. In the **Select events to add** dialog box, select events. Make sure to include one of the following events: BR\_INST\_RETIRED.NEAR.CALL or BR\_INST\_RETIRED.NEAR.CALL\_R3.
5. Click **OK**.  
Selected events are added to the **Event Name** column in the **Hotspot Analysis** tab.
6. In the **LBR Filter** column, apply any filter to the selected event.
7. Click **Apply** to save your settings.
8. Click **Profile** to run the experiment.  
The Intel PTU will open the [sampling Hotspot view](#) with call site data for hotspot functions.

## 7.2.2 Call Count Analysis Configuration Settings

To select the [call count analysis](#) for your configuration and edit configuration settings, check the **Enable Call Count Analysis** box.



Click the **Add** button to specify applications to ignore during data profiling. You may also click the **Browse** button and specify a file listing applications to ignore.

**See Also:**

[Collecting Call Graph Data](#) (CLI)

## 7.2.3 Heap Analysis Configuration Settings

To select the [heap analysis](#) for your configuration and edit configuration settings, check the **Enable Heap Analysis** box.

- **Exact collection mode:** Collect detailed data. The collected data is more accurate but requires more time to collect.
- **Fast collection mode:** Collect less detailed data. The collected data is less accurate but requires less time to collect.
- **Applications to ignore:** List applications to ignore while collecting data.
- **Profile spawned child processes:** Check this box to profile all applications spawned by the first profiled process.
- **Collect trace:** Check this box to collect overtime information. It may require more time to collect.

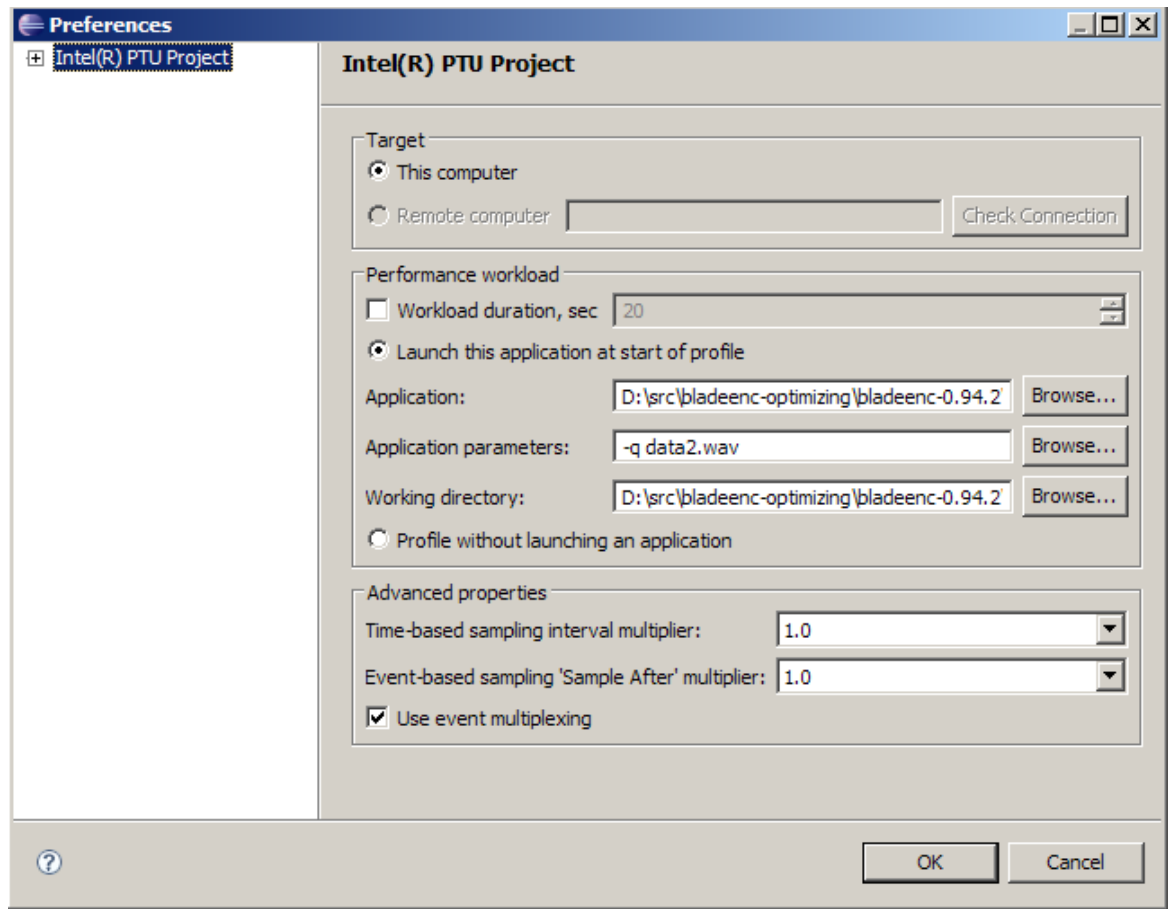
**See Also:**

[Collecting Heap Profile Data](#) (CLI)



## 7.2.4 Configuration-specific Project Properties

Sometimes you need to adjust the behavior of a profiling configuration without changing the configuration itself. Configurations are supposed to be a kind of “recipes” and you are not expected to change them often. But what if a particular collection mechanism has a parameter that enables you to affect the collection process? To address this, the Intel Performance Tuning Utility introduces a concept of configuration-specific project properties. You can set these parameters in the **Advanced Properties** section of the project property page:



You can also set configuration-specific properties when creating a new project by expanding the **More...** bar:



**New Intel(R) PTU Project**

**Workload Details**  
Specify what you want to profile

**Target**

☒ This computer

☐ Remote computer

**Performance workload**

☒ Workload duration, sec

☐ Launch this application at start of profile

Application:  

Application parameters:  

Working directory:  

☒ Profile without launching an application

**More...**

**Advanced properties**

Time-based sampling interval multiplier:

Event-based sampling 'Sample After' multiplier:

☐ Use event multiplexing



The following configuration-specific properties are available:

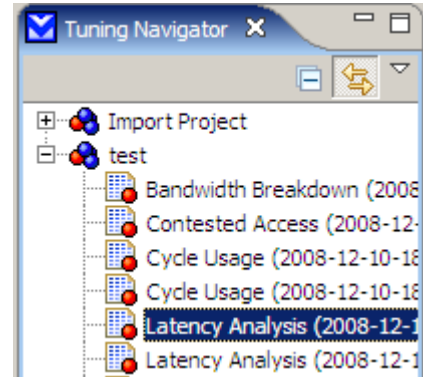
- **Time-based sampling interval multiplier.** Use this property on the per-project basis to affect the value of the time-based sampling interval. For example, if you set the multiplier to 0.1 and, then, run the project with a profiling configuration configured for the time-based sampling with 20 millisecond interval, the actual interval will be  $20 * 0.1 = 2$  msec.
- **Event-based sampling "Sample After" multiplier.** This property is similar to the previous one but affects the Sample After value for event-based sampling configurations.
- **Use event multiplexing.** You can select/deselect check box to enable/disable event multiplexing for event-based sampling.

## 7.3 Workspace and Tuning Navigator

The **Tuning Navigator** displays a hierarchical list of project contents. This panel enables you to view and manage all projects and related Experiments. To open the **Tuning Navigator**, select **Window > Show View>Tuning Navigator**.


The **Tuning Navigator** displays two items: *Project* and *Experiment*:

-  **Project** includes an application profile that contains information about how an application is executed. Typically, Project contains one or more Experiments. Double-click the project node to expand/collapse its contents.
-  **Experiment** represents data collection session with a specific [configuration](#) and provides data collection results. The Intel Performance Tuning Utility creates an Experiment node automatically when data collection starts. The default name of an Experiment consists of the configuration name used to collect it and timestamp of its creation, for example: `Basic Sampling (2008-12-11-11-56-08)`.



You can manage your projects and experiments by creating, renaming, or deleting them. To rename an experiment, right-click it and select **Rename**. To delete an experiment, press **Delete** or right-click the node and select **Delete**.

## 7.4 Data Views

Upon the data collection completion, the Intel Performance Tuning Utility automatically displays experiment results in a data view related to the profiling type. You can also invoke it for any experiment by double-clicking the  **Experiment** icon. The following data views are available:

- [Sampling view](#)
- [Statistical call graph view](#)
- [Call count view](#)
- [Heap profiling view](#)
- [Data access view](#)

For an event-based sampling, click the experiment result node in the **Tuning Navigator** to see the Experiment Summary view. The Experiment Summary view opens at the bottom of the Eclipse window and provides data on the analyzed





application, processor, and event count in user and kernel mode occurred for a particular event.

You can configure most of the views in the Intel Performance Tuning Utility by using **Window > Preferences... > Intel® Performance Tuning Utility >** and selecting a view you want to configure.

## 7.4.1 Filtering Controls

The Filter bar located the bottom of each central data view enables filtering the data presented in the table. The following controls are available:

- **Granularity:** The hotspot display depends on the granularity level. Function level is default.  
If application modules have debug information, you can rely on functions shown as hotspots. When debug information is incomplete or missing at all, you may see a number of <unknown> functions or samples collected on internal functions of a module might be attributed to exported functions.  
Use the **Granularity** drop-down menu to change the grouping level. For sampling data, select between Process, Thread, Module, Source, Function, Basic Block, Relative Address, and Address granularity levels. For statistical call graph, only Function granularity is available.
- **Limit:** Use the **Limit** drop-down menu to focus on the most critical hotspots. Specify the percent of hotspots to display (95% is default) manually or choose one of the recommended thresholds from the drop-down menu.  
Percent of hotspots indicates the percent of samples collected in a hotspot arranged in a decreasing order. Selecting a percent threshold filters the data view to display the hotspots whose total number of samples is lower than the selected threshold of total samples. The Hotspot view for multiple events is combined from the hotspot list per each event.  
For the statistical call graph, use the **Limit** drop-down menu to display an absolute number of hotspots.
- **CPU:** Use the **CPU** drop-down menu to view the event count distributed per CPU. By default, the **Total** value is selected and the event column displays the total number of samples collected for this event on all CPUs. To display event count for each logical CPU, select the **Each** value. To display event count for a particular logical processor, select the required CPU number from the drop-down menu (for example, 3). To view event count for each CPU package, select the **Per Socket** value. If the Hyper-Threading Technology is enabled on your system, select the **Per HW Thread** value to view the event count per each CPU hardware thread. To view maximum event value across logical processors, select **Max(Each)**.  
In case of incorrect mapping between CPU views, see the [Troubleshooting the Hotspot View](#) section.



- **Module:** Select the module name in the **Module** drop-down menu to limit displayed hotspots to specific module only. Module selection limits the list of processes in the **Process** filter control to the processes that loaded the selected module during the experiment.
- **Process:** Select the process name in the **Process** drop-down menu to limit displayed hotspots to specific process only.  
Process selection limits the list of modules in the Module filter control to the modules that were loaded in the selected process during the experiment.  
Process selection limits the list of threads to the threads that were executed in the selected process during the experiment.
- **Thread:** Select the thread name in the **Thread** drop-down menu to limit displayed hotspots to specific thread only.

## 7.4.2 Sampling View

The Sampling view consists of the Hotspot view, Events over IP view, and Advanced Profile view.

### 7.4.2.1 Hotspot View

The Hotspot view opens in the central Eclipse window and provides data collection results in the table format. It displays a hotspot, number of events (or samples) that hit the hotspot during the experiment, and other related information.

Hotspot is a code region that generated a statistically meaningful number of samples. In general, a hotspot could be expressed at different levels of source code details (or granularity), for example, as a module, function, or source code line. Thus, in case of a hotspot as a function, all events that occurred on instructions of the specific function are grouped (summed up) together and attributed to the function. If a hotspots is presented as a loop, all events occurred on the instructions of the specific loop (including the nested loops) are aggregated to this loop. The lowest level of detail for a hotspot is the address of one machine instruction.

For sampling data, a hotspot can be presented as a module, function, basic block, relative virtual address, or virtual address. Hotspot representation is controlled by the Granularity control located at the filter bar at the bottom of the data view. Select the required hotspot granularity level(s) to identify the hotspots. For example, **VA** is the finest hotspot granularity. It shows hotspots by virtual address. As opposed to the **RVA** granularity, it helps differentiate module instances with different load addresses even within one process and, therefore, may contain more hotspot items.



By default, the hotspot column is the leftmost column of the table. Other columns can be considered as hotspot attributes, for example: a module where a hotspot function is located or a function where a hotspot basic block is located.

You can sort the Hotspot view table by any column. By default, it is sorted in the descending order by one of the **<Events>** columns.


To identify the total number of samples occurred in several hotspots for a particular event, select the hotspots and see the **Total Selected** line value.

Right-click and select the **Show Data as** pop-up menu command to choose between the following types of sampling data representation:

- **Samples:** Number of samples collected for the selected event.
- **Events:** Event count which is equal to **Samples** value \* Sample after value for the selected event.
- **% of Experiment:** Percent of samples with respect to total samples collected during the entire duration of the experiment.

The Intel® Core™ i7 processor family has specific events that enable advanced capabilities for experiment analysis. When the EBS profiles include these events, the sampling Hotspot view displays additional information.

If the EBS profile contains BR\_INST\_RETIRED.ALL\_BRANCHES with the **All Branches** LBR filter enabled, the Intel PTU statistically calculates basic block execution counts for the analyzed application. The sampling Hotspot view displays this data in the BB\_EXEC column when the granularity is **Basic Block** or **Loop**.

If the list of events includes BR\_INST\_RETIRED.NEAR.CALL or BR\_INST\_RETIRED.NEAR.CALL\_R3 with LBR filters enabled, the sampling Hotspot view displays [call site](#) data for hotspot functions. To view call sites, choose the **Function** granularity and click the triangle symbol  at the hotspot function name. For each call site, the **RVA** column displays a particular address from where a call to the hotspot function was made. For hotspot functions, RVA corresponds to the beginning of the function.

BR\_INST\_RETIRED.NEAR.CALL or BR\_INST\_RETIRED.NEAR.CALL\_R3 event count can also serve as function call count approximation. To view call count numbers, switch data representation to **Events**.

**NOTE:** You can export the current content of the Hotspot view into a comma-separated text file by using the pop-up menu command. If the experiment contains both event-based sampling and statistical call graph data and the granularity is set to "Function" the Hotspot view merges them and shows in one view.

**See Also:**




[Viewing Sampling Data \(CLI\)](#)  
[Configuring for Viewing Call Sites](#)

### 7.4.2.2 Hotspot View with Loop Granularity

The Intel Performance Tuning Utility can display a hierarchy of loops for the functions of the binaries compiled with debug information. To view loop information from the Hotspot view, set the [granularity](#) level to **Loop**.

The Hotspot view displays hot top-level loops, sorted in the descending order. The **Address** column of the view displays addresses of basic blocks belonging to a loop. The **Function** column shows the name of the function the loop belongs to.

Each loop is indicated by a *loop line* with a small triangle  in front of the function name. It contains the address of the loop header basic block that uniquely identifies the loop in the binary. All events occurred on the basic blocks and nested loops inside a higher level loop are aggregated to its loop line.

To expand a loop and see its basic blocks and nested loops, click the triangle in the loop line. The view updates to display next-level nested loops and basic blocks of the expanded loop. A basic block that belongs to the loop but has not collected any samples does not appear among the loop basic blocks.

The Intel PTU may display the header block address twice. The first instance denotes the loop; the second one shows the basic block itself with samples occurred on its instructions.

Address	Function	Module	CPU_CLK_UNHALT...	L2_LINES_IN.SELF.ANY	MEM_LOAD_RETIRED...
0x36CC	▶ quantum_toffoli	libquantum.exe	60,184	28,792	3,631
0x3FEB	▶ quantum_sigma_x	quantum.exe	38,375	11,561	5,259
0x3D1C	▶ quantum_cnot	libquantum.exe	14,498	6,451	857
0x3f	▶ quantum_swaptheads	libquantum.exe	2,238	9	9
0x3249	▼ quantum_swaptheads o...	libquantum.exe	802	343	44
0x344C	▼ quantum_swaptheads_o...	libquantum.exe	284	113	14
0x344C	quantum_swapthelea...	libquantum.exe	205	85	14
0x349	quantum_swapthelea...	libquantum.exe	52	13	0
0x346B	quantum_swapthelea...	libquantum.exe	23	12	0
0x347B	quantum_swapthelea...	quantum.exe	4	3	0
0x3585	▶ quantum_swaptheads_o...	libquantum.exe	261	116	15
0x3313	▶ quantum_swaptheads_o...	libquantum.exe	257	114	15
0x1EBE	▶ quantum_gate1.	libquantum.exe	380	110	0
0x1EF7	▶ quantum_gate1.	libquantum.exe	119	40	21

1	An expandable loop line with the header basic block address uniquely identifying the loop in the binary.
2	A triangle to expand the loop.
3	A nested loop.
4	A duplicated basic block address indicating the loop and the basic block itself.
5	A basic block inside the nested loop.

**See Also:**  
[Hotspot View](#)



### 7.4.2.3 Advanced Profile View. Automatic Identification of Performance Issues

The Intel Performance Tuning Utility has a mechanism for an automatic identification of performance issues. It is based on collecting predefined profiles and calculation of [event ratios](#).

When a ratio has a value exceeding the predefined threshold, it signals a potential performance problem. In the Hotspot view, a cell corresponding to the [key event of that ratio](#) is highlighted in pink:

Function	CPU...	CPU_CLK_UNHAL...	INST_RE...	MEM_LOAD_RETIRED.L2_LINE_MISS	BUS_TRANS_BURST.SELF	BUS...
compute_rhs	7,480	3,790	4,337	203	1,125	
z_solve	3,984	2,003	2,118	73	600	
y_solve	2,969	1,485	1,747	63	467	
x_solve	2,830	1,425	1,731	45	472	
lhsx	1,978	996	642	18	326	
lhsy	1,969	1,011	589	16	318	
lhsz	1,797	900	603	11	261	
<unknown(s)>	1,555	797	1,383	0	20	
_kmpc_end_ma...	1,263	636	519	0	47	
adi	1,015	514	465	40	175	
_kmp_wait_sleep	977	490	399	1	10	
...	...	...	...	...	...	...

Limit	95%	Grav...	Func...	Process	All	Thread	All	Module:	All	Cpu	Max(Ei...
-------	-----	---------	---------	---------	-----	--------	-----	---------	-----	-----	-----------

Experiment Summary	Console	Memory Hotspots	Tuning Navigator	Advanced Profile Info
--------------------	---------	-----------------	------------------	-----------------------

Function : compute_rhs - sp.c			
Event	Samples	Events	Issue
CPU_CLK_UNHALTED.CORE	7,480	14,960,000,000	
INST_RETIRED.ANY	4,337	8,674,000,000	Clocks per Instructions Retired - CPI = 1.7247
BUS_TRANS_BURST.SELF	1,125	225,000,000	Bandwidth Limitation = 1.1873
MEM_LOAD_RETIRED.L2_LINE_MISS	203	20,300,000	L2 load driven misses = 0.2714
CPU_CLK_UNHALTED.CORE max(ICPU)	3,790	7,580,000,000	
INST_RETIRED.ANY max(ICPU)	2,192	4,384,000,000	
MEM_LOAD_RETIRED.L2_LINE_MISS max(ICPU)	105	10,500,000	
BUS_TRANS_BURST.SELF max(ICPU)	575	115,000,000	

The Sampling Hotspot view uses a smart column ordering based on the priority of identified issues and shows the most important event columns on the left. This makes events signaling potential performance issues more immediately visible. Such an ordering is especially helpful when a large (predefined) profile configuration containing many events is used. The limited width of a GUI display cannot show all of them at once.

The Intel PTU also provides the Advanced Profile Info view. To enable it, select a function in the Hotspot view and right-click to choose the **View Issue Details** from the pop-up menu. This shows values of all the events collected for this hotspot and annotates those that are recognized as signaling potential issues. The rows of this view are ordered in the same manner as the columns of the Hotspot view, but use only the events associated with the selected function. The issue value in the **Issue** column is exactly the value of the associated event ratio.



The lower pane shows the Advanced Profile information for the function selected in the upper Hotspot view window. Highlighting a different function automatically changes the data in the Advanced Profile view to correspond to the data for the newly selected function.

To analyze a specific issue, use the buttons located at the right of the **Advanced Profile Info** pane:

- **Explain Issue:** Click to view the tooltip with a detailed explanation of the issue.
- **Explain Event:** Click to display the description of the selected event.
- **Explain Ratio:** Click to display the formula for the calculated ratio. This button is available only if the event ratio was calculated.

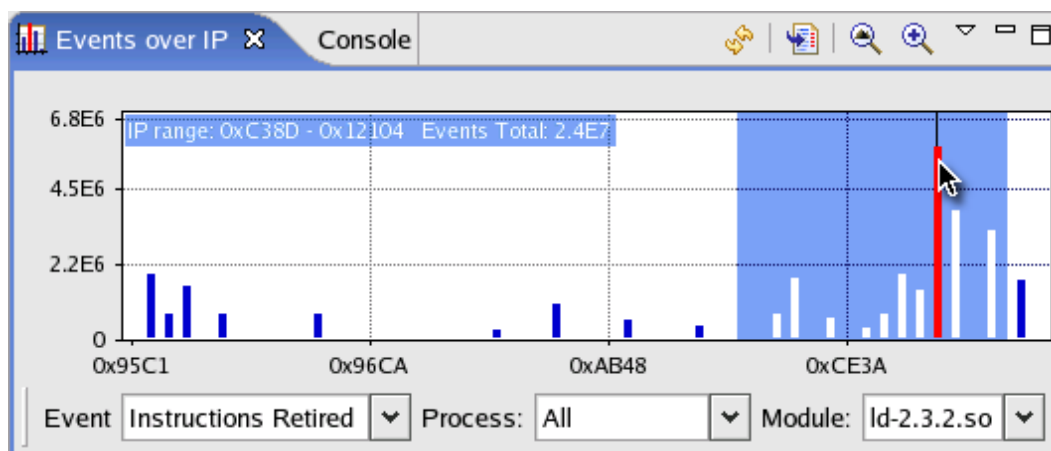
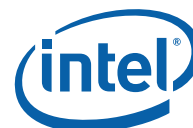
#### Example:

In the figure above, 2 ratios, “L2 load driven misses” and “Bandwidth limitations”, exceed their threshold values. This identifies potential performance issues in, at least, four hot functions in the Hotspot view. The Hotspot view indicates this by highlighting the cells corresponding to MEM\_LAOD\_RETIRED.L2\_LINE\_MISS and BUS\_TRANS\_BURST.SELF events in pink. It means that four top functions (`compute_rhs`, `z_solve`, `y_solve`, `x_solve`) experience memory problems as front-side bus saturation and load-driven Level 2 cache misses. So, to improve the code performance, consider optimizing data layout and/or the way you traverse through the data.

### 7.4.2.4 Events over IP View

While the [Hotspot view](#) displays significant hotspots over the whole experiment, you might want to see a picture for a specific module. A histogram of Events over IP view can help you with this.

From the Hotspot view for a sampling experiment, select a row associated with the module you are interested in and launch the Events over IP view using the pop-up menu. The Events over IP view opens at the bottom of the Eclipse window and displays the sample or event distribution per IP range for the selected module and event of interest. Analyze the histogram to identify major hotspots in a module.



The horizontal axis displays relative virtual addresses in the selected module.

**NOTE:** If you select a particular process in the **Process** drop-down menu and **All** modules for this process in the **Module** drop-down menu, the horizontal axis displays virtual addresses (not *relative* virtual addresses).

The scale of IPs over horizontal axis is not uniform. All IPs where there are no samples occurred are omitted.

The vertical axis shows a number of events (or samples). The height of the bar indicates the total number of events (or samples) occurred on the region of IPs or on a single IP (this depends on the zoom).

By default, the Events over IP view displays the following summary information: entire IP region of the module where samples occurred of the module. To zoom into a specific IP range and get more detailed information per IP address, select a range of interest and click the **Zoom In** button. The updated chart displays samples data for the selected range of IPs. To return to the default summary view, click the **Zoom Out** button. To analyze the hottest IPs in the module, select a region and drill-down to the source code of the highest hotspot by clicking the **Source View** button.

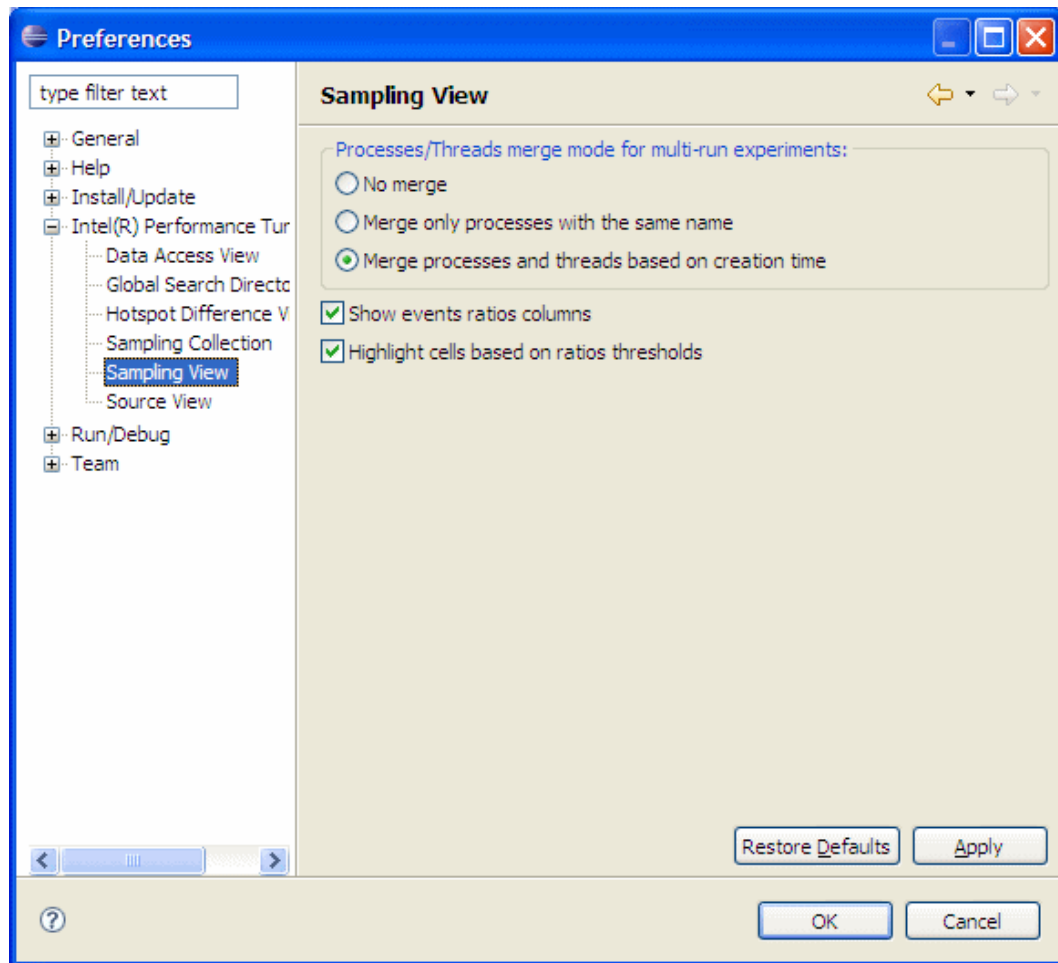
The filter bar of the Events over IP view enables filtering the displayed data.

When switching between different experiments, the Events over IP view resets its context and updates only filter bar items. To refresh a histogram, press the **Update** button in the view-specific toolbar or modify a filter bar selection.

### 7.4.2.5 Sampling View Preferences

To configure the sampling view, go to **Window > Preferences... > Intel® Performance Tuning Utility > Sampling View** preference page.

You can configure the following preferences:



- **Processes/Threads merge mode for multi-run experiments:** Select merge criteria for processes/threads.
- **Show event ratios columns:** Check the box to display ratio values in the sampling Hotspot view.
- **Highlight cells based on ratios thresholds:** Enable highlighting cells with key events for ratios exceeding the predefined threshold.

#### See Also:

[Event Ratios](#)

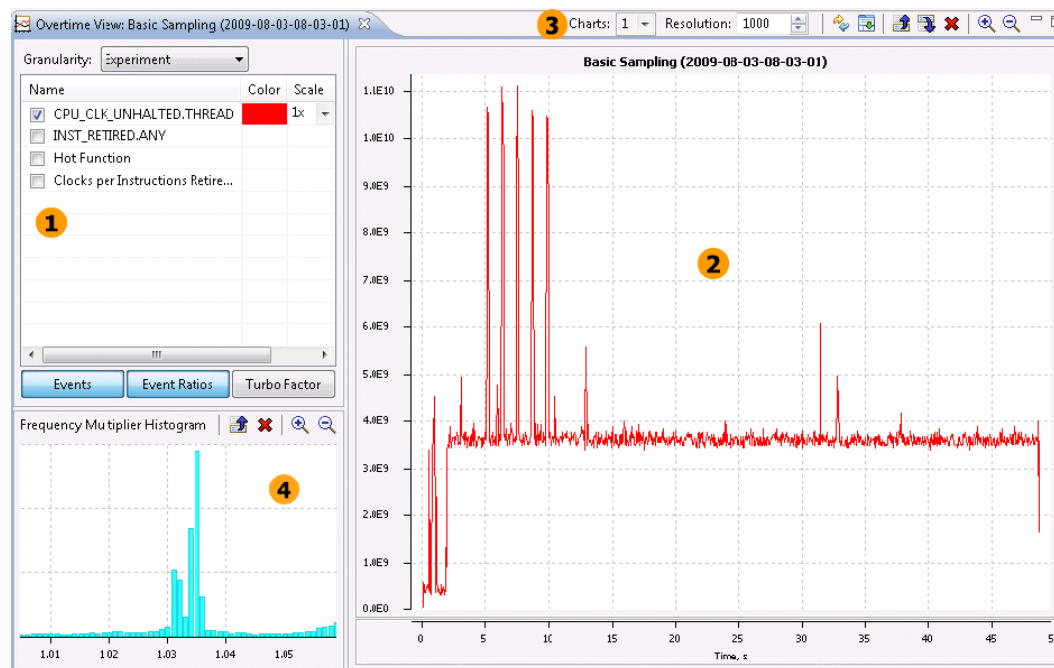
[Advanced Profile View](#)





## 7.4.3 Overtime View

To open the Overtime view, right-click the Hotspot pane and select Show Overtime View from the pop-up menu. The Overtime view appears at the bottom of the Eclipse window.



<b>1</b>	<a href="#">Events/Ratios Selection pane</a>	<b>3</b>	<a href="#">Overtime Toolbar</a>
<b>2</b>	<a href="#">Time Charts pane</a>	<b>4</b>	<a href="#">Frequency Multiplier Histogram pane</a>

The Overtime view displays the sampling data collected during the experiment chronologically. By default, data for the hottest event is displayed.

Use the Overtime view to:

- compare time distribution of events/ratios;
- analyze changes in CPU operating frequency during the sampling collection using a ratio between the actual operating frequency and the reference CPU frequency ([turbo factor](#)).

**NOTE:** Turbo factor data is useful for processors with Intel® Turbo Boost Technology that can adjust the operating frequency of the processor cores depending on performance requirements and energy consumption level. This feature is available starting from the Intel® Core™ i7 processor family.



### 7.4.3.1 Events/Ratios Selection Pane

Use the Events/Ratios Selection pane to control and configure the information in the Overtime view time charts.

Item	Description
<b>Granularity</b> drop-down menu	<p>View data distribution for various granularity levels:</p> <ul style="list-style-type: none"> <li>• Experiment (default)</li> <li>• CPU</li> <li>• Core</li> <li>• Socket</li> <li>• Process</li> <li>• Thread</li> <li>• Module</li> <li>• Source</li> <li>• Function</li> <li>• Basic Block</li> <li>• RVA</li> </ul> <p>To change the default granularity, go to <b>Window &gt; Preferences &gt; Intel Performance Tuning Utility &gt; Overtime View</b> preference page and select the required level from the <b>Default Granularity:</b> drop-down menu.</p>
Events/Ratios table	<p>View events and ratios collected during the experiment.</p> <p>Check the box in a corresponding row to view the</p>

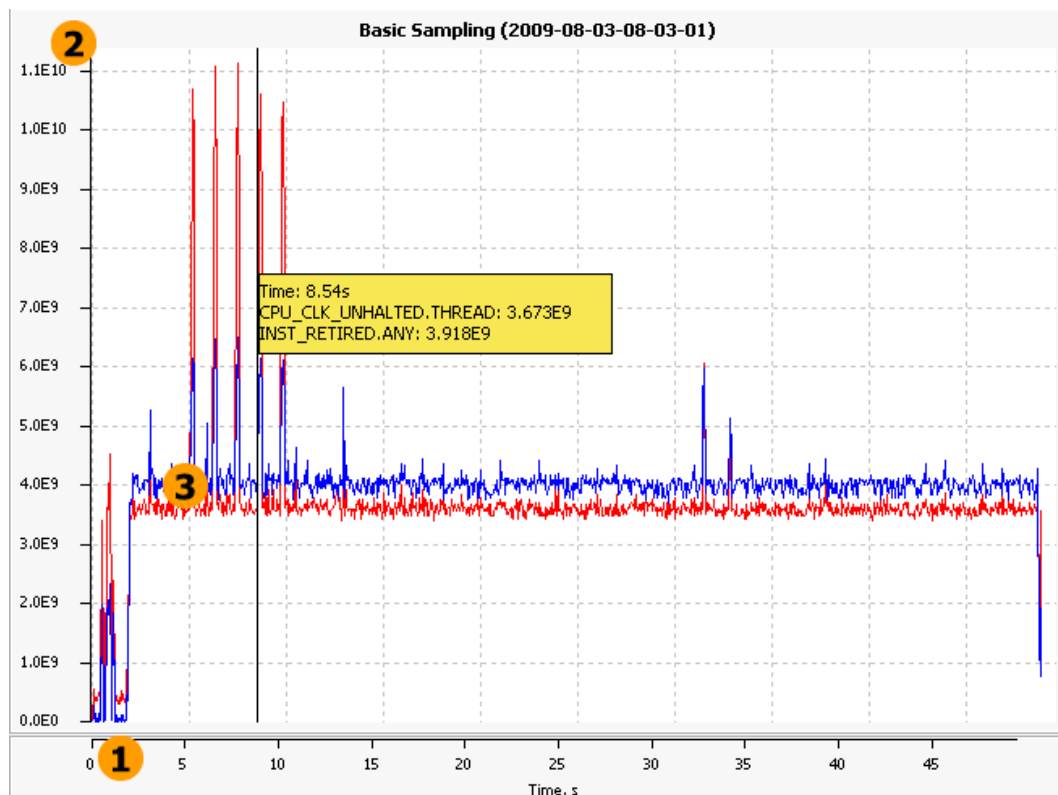


	<p>distribution of a particular event/ratio in the time charts.</p> <p>The <b>Color</b> column of the table defines the color for each event/ratio displayed in the time charts.</p> <p>The <b>Scale</b> column defines the scale in which an event/ratio is displayed. Use the drop-down menu to change the current scale.</p>
<b>Events</b> button	Click to add/remove events from the list.
<b>Event Ratios</b> button	Click to add/remove ratios from the list.
<b>Turbo Factor</b> button	Click to display CPU frequency deviation from the reference frequency.

**See Also:**
[Time Charts Pane](#)

### 7.4.3.2 Time Charts Pane

The time charts show the distribution of events/ratios and the CPU frequency level as function of time. This allows you to identify performance problems in relation to particular time ranges of your experiment.





<b>1</b>	The X-axis of the chart displaying time of the experiment, in seconds.
<b>2</b>	<p>The Y-axis displaying the number of events/ratios collected during a particular time range.</p> <p><b>TIP:</b> For different charts, the Y-axis scale may be different depending on the event/ratio values displayed in the chart. To compare the event/ratio distribution for different charts, you can make the scales equal. Right-click a chart and select <b>Equal Y-axis scales</b> from the pop-up menu. All scales are equaled by the largest scale.</p>
<b>3</b>	The graphs in different colors showing event/ratio distribution over time of the experiment. By default, the Intel PTU displays data as line graphs. To display data as bar graphs, right-click the chart and select <b>Show Data as &gt; Bars</b> from the pop-up menu.

For different granularities, a different number of charts may be displayed. By default, the Intel PTU displays four charts, if they are available for the selected granularity. Use the **Charts:** drop-down menu on the [Overtime Toolbar](#) to specify the number of charts displayed at a time.

To select the charts to display:

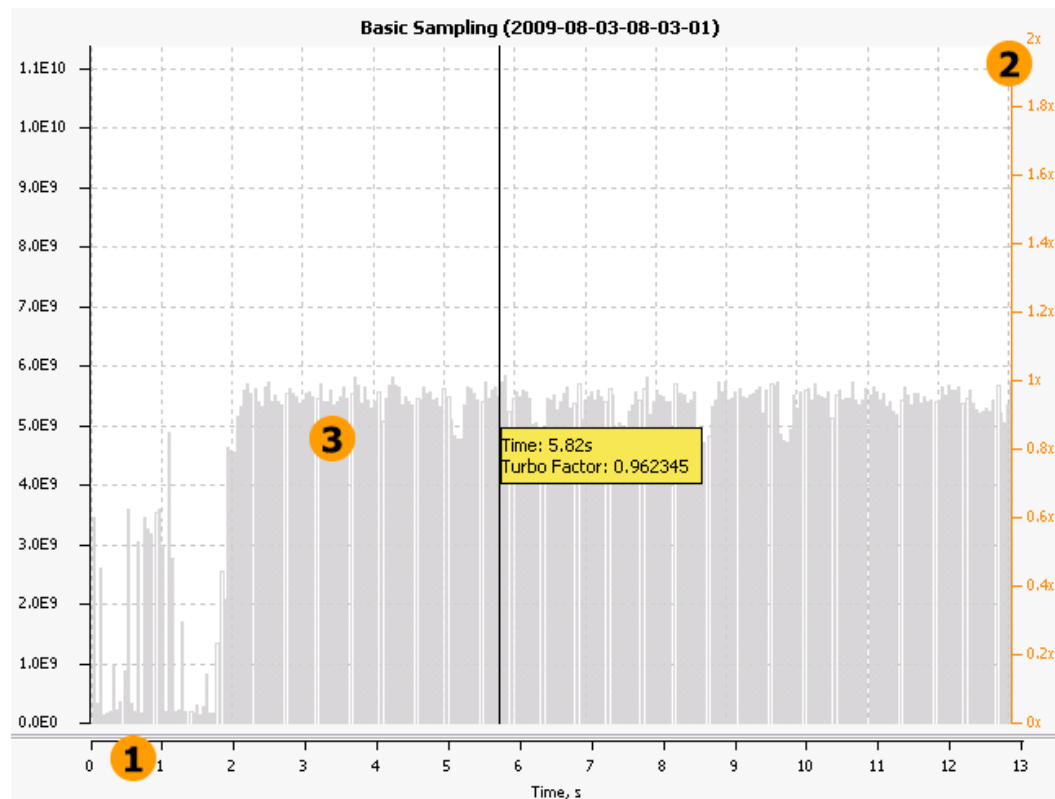
1. Right-click a chart and choose **Select charts...** from the pop-up menu.  
The **Chart Selection** dialog box opens listing all the charts available. The charts currently displayed are highlighted in grey. By default, the charts are sorted in the descending order, starting from the one corresponding to the hottest item in the Hotspot pane. If the granularity level for Hotspot and Overtime views is the same, the order of the charts corresponds to the order of hotspots in the Hotspot pane.
2. Drag and drop the required chart to the grey area.  
The selected chart replaces the one it was dropped onto.

You can compare the over-time distribution of several events/ratios in the chart. The Intel PTU displays each event/ratio in a color specified in the Events/Ratios Selection pane, which makes the comparison easy and demonstrative. Hover the mouse over the chart to see the information on the number of events/ratios detected at a particular moment.

When you compare event and ratio distribution, the difference in values can be too great to view the graphs simultaneously. To assess their distribution in relation to each other, you can change the scale of the graphs by selecting the required value from the **Scale** drop-down menu available in the [Events/Ratios Selection pane](#) in the same row with the selected event/ratio.



To view changes in CPU operating frequency level during the sampling collection, click the **Turbo Factor** button in the Events/Ratios Selection pane. The time charts update to display [turbo factor](#) data:



<b>1</b>	The X-axis displaying time of the experiment, in seconds.
<b>2</b>	The Y-axis displaying the ratio between the reference frequency of your processor and the actual operating frequency. The values above 1 indicate that the available CPUs are running faster than the reference frequency.
<b>3</b>	The grey bars representing changes in <a href="#">turbo factor</a> over time of the experiment. Hover over the bar to view the turbo factor value for a particular moment of the experiment.

**See Also:**

[Events/Ratios Selection Pane](#)

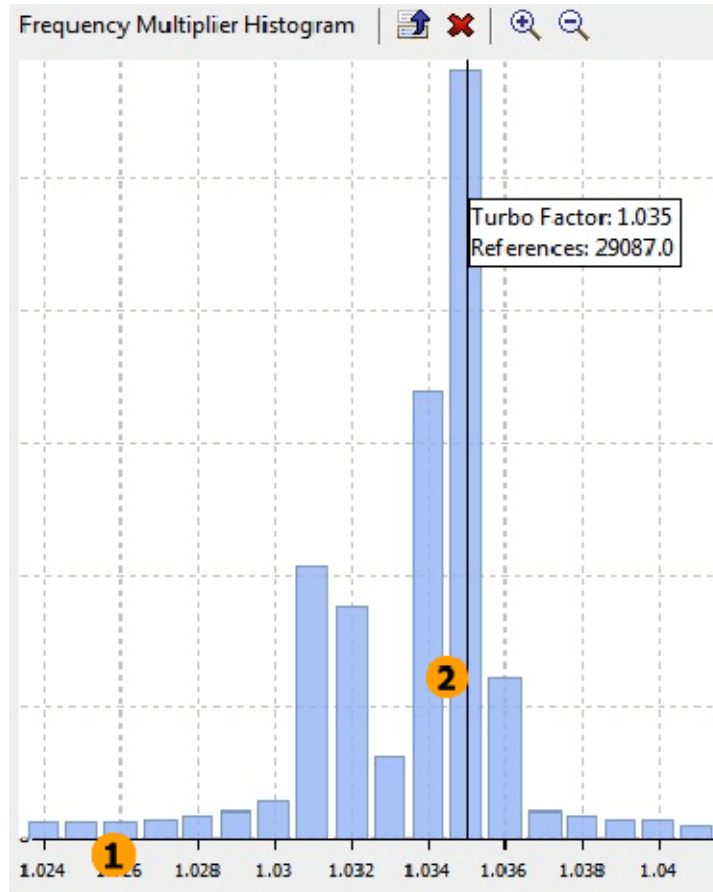
[Managing the Overtime View](#)

[Frequency Multiplier Histogram Pane](#)




[Overtime View Preferences](#)

### 7.4.3.3 Frequency Multiplier Histogram Pane

During the experiment, the Intel PTU refers to available CPUs at a certain interval to check the current CPU frequency. The **Frequency Multiplier Histogram** displays the operating frequency levels at which your application was running.



1	The X-axis displaying the <a href="#">turbo factor</a> data for your processor.
2	The histogram bars representing the number of references counted at different frequency ranges. The longer the bar, the more references occurred at this frequency. To see the number of references for an exact turbo factor value, hover the mouse over the bar.

To view the hotspots that occurred at a particular frequency level only, select this level and click the  **Filter hotspots by selected turbo factor** button above the histogram. The Hotspot pane updates to include only the hotspots that occurred while the CPUs were running at the selected frequency. To cancel the filtering, click the  **Cancel turbo factor filtering** button. To restore the default view of the histogram, use the  **Zoom Out** button.


**See Also:**
[Time Charts Pane](#)
[Managing the Overtime View](#)

### 7.4.3.4 Managing the Overtime View

You can manage the information displayed in the Overtime view using the controls located on the Overtime Toolbar.

<div> Charts: 1 Resolution: 1000  </div>	
Toolbar Item	Description
<b>Charts:</b> drop-down menu	Specify the number of charts displayed. By default, the Intel PTU displays four charts if they are available for the selected granularity.
<b>Resolution:</b> spin box	Change the chart width, in points. The default width is 1000 points.
<b>Load detailed data for the selected time range</b> button	Click to view details for the zoomed-in time range and restore chart resolution to the default value.  <b>TIP:</b> To automatically load details when the selected time range is less than a specified value, enable this option in the <a href="#">Overtime View Preferences</a> .
<b>Synchronize Overtime View with the current experiment</b> button	Display overtime data for the experiment in focus.
<b>Filter hotspots by current time range</b> button	Click to view the hotspots only that occurred during the time range selected in the chart.
<b>Filter Overtime view by selected hotspots</b> button	Click to view the time range when a particular hotspot occurred. The Frequency Multiplier histogram and the time charts update to display only the information related to the hotspot currently selected in the Hotspot pane.
<b>Cancel time filtering</b> button	Click to cancel the applied filtering.
<b>Zoom in to selection</b> button	Click to view a selected time range in more detail.  <b>TIP:</b> To automatically load details when the selected time



	range is less than To automatically zoom in when selecting a time range, right-click the time chart and enable the <b>Use mouse selection to zoom in</b> option in the pop-up menu.
 <b>Zoom Out</b> button	Click to view a larger time range with less detail.

You can use hot keys to manage the display of the selected time range:

- **CTRL+Up/CTRL+Down** - magnify/minify the view.
- **CTRL+Right/CTRL+Left** - scroll the graph right or left.
- **F5** - load detailed data for the selected range.

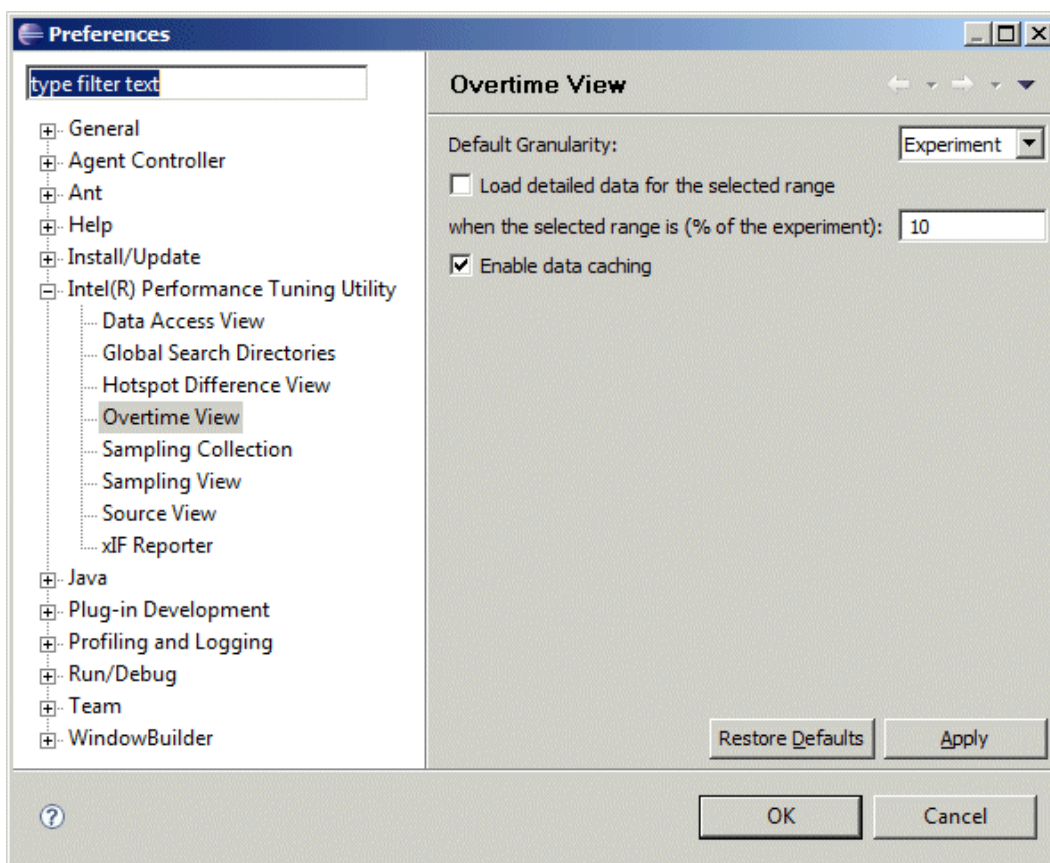
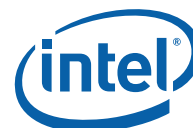
**See Also:**

[Time Charts Pane](#)

### 7.4.3.5 Overtime View Preferences

To configure the Overtime view, go to **Window > Preferences... > Intel® Performance Tuning Utility > Overtime View** preference page.





- **Default Granularity:** Use this drop-down menu to select the granularity level for the time charts. The default value is **Experiment**.
- **Load detailed data for the selected range when the selected range is smaller (% of the experiment):** Check this box to automatically load detailed data when zooming in to a time range making up less than the specified percent of the experiment. The default value is 10 percent. When data is reloaded, the chart resolution reaches the value specified in the **Resolution** spin box on the Overtime Toolbar.
- **Enable data caching:** Check this box to cache data every time you query overtime data from the experiment database. This enables faster data loading next time you open the Overtime view with the same parameters (granularity, filtering, and others). When this option is on (default), the Intel PTU reloads this data from cache saved in `<experiment_dir>/cache_overtime` directory. The event selection is also stored in cache.

**See Also:**

[Time Charts Pane](#)

[Managing the Overtime View](#)



## 7.4.4 Statistical Call Graph View

The statistical call graph view consists of the Hotspot view, Caller/Callee view, and Hotpath view.

### 7.4.4.1 Hotspot View

Like the [sampling Hotspot view](#), the statistical call graph view displays hotspots and the number of events (or samples) occurred in the hotspot during an experiment. In the statistical call graph Hotspot view, a hotspot is presented as a function.

For the statistical call graph experiment, the Hotspot view contains more data than just hotspots. For each hotspot, it displays aggregated stack data that tells you what the call sequences were when a sample occurred in the hotspot. In the data table, each hotspot is a row with an embedded tree. Expanding the selected function hotspot displays callers of the selected hotspot function, then callers of the first caller(s), and so on.

In the statistical call graph Hotspot view, the sample values provided in the **Samples** column for a hotspot and underlying stack items are different. For hotspots, it is a number of samples occurred in the hotspot function. For stack functions, it is a number of times when the function was detected on the stack restored for the hotspot.

**NOTE:** You can export the current content of the Hotspot view into a comma-separated text file by using the pop-up menu command.

### 7.4.4.2 Caller/Callee View

The Caller/Callee view opens at the bottom of the Eclipse window. It displays function call information only for [statistical call graph collection](#) if the [Hotspot view](#) is active in the main window, otherwise, it is empty.

The Caller/Callee view displays all callers and callees of a function selected in the Hotspot view. As soon as you focus on another function in the Hotspot view, the Caller/Callee view is updated. The name of the focus function is in bold. The caller functions are located above the focus function. The callee functions are located below the focus function. To move a caller/callee function to a focus, double-click its name in the list.

This view is reconstructed from the statistical stack data displayed in the Hotspot view, which means that information in this view is also only statistically correct.



Caller / Callee		Console			
Function	Hint	Module	Total samples	Self samples	
<a href="#">VDMPEG::DecodeBlock_C</a>		VirtualDub.exe	162	86	
<a href="#">VDMPEG::DecodeBlock_Y</a>		VirtualDub.exe	271	154	
<a href="#">VDMPEG::CopyPredictionFor...</a>		VirtualDub.exe	3	3	
<a href="#">VDMPEG::DecodeBlockNonPr...</a>		VirtualDub.e...	438	245	
<a href="#">isse_idct_intra</a>		VirtualDub.exe	56	53	
<a href="#">isse_idct_nonintra</a>		VirtualDub.exe	135	4	
<a href="#">__security_check_cookie</a>		VirtualDub.exe	2	2	

The Caller/Callee view displays the following data:


	Total samples	Self samples
<b>Caller function</b>	Total number of samples occurred directly in the focus function and all its callees when it was called from this caller function	Number of samples occurred directly in the focus function when it was called from this caller function
<b>Focus function</b>	Total number of samples occurred directly in this function and total samples of all its callees.	Number of samples occurred directly in this function when called from all callers.
<b>Callee function</b>	Number of total samples for the callee function when it was called from the focus function.	Number of samples occurred directly in this callee function when it was called from the focus function.

Be aware that samples information in the Caller rows is related to the focus function. Namely, samples information of a caller function shows samples of the focus function when it is called from the caller function. The Caller/Callee view enables you to deduct distribution of the focus function work that it did for different callers.

The Intel Performance Tuning Utility enables you to detect *recursion* in your code by using the statistical call graph. A function is identified as recursive on a stack if it appears several times on this stack. So, a function can appear both as recursive and non-recursive in different stacks.

*Self Samples* of a recursive function is a sum of Self samples of all instances of this function on a stack. *Total Samples* of a recursive function is a sum of Self Samples of all instances of this function on a stack and all their children.



Recursive functions are marked with the  **Recursion cycle** [hints](#) in the Hotspot, Hotpath, and Caller/Callee views. In the Caller/Callee view, repeated callers/callees (that could appear in case of recursion) are folded into a single caller/callee.

### 7.4.4.3 Hotpath View

The Hotpath view opens to the right of the [Hotspot view](#) when it shows statistical call graph experiment results. The Hotpath view displays the most performance-critical path for the hotspot function selected in the Hotspot view.




The first row in the **Function** column provides the name of a hotspot function. The list below hotspot function shows the caller functions that contributed the most number of samples while calling the previous function in the list.

The **Samples** column shows the number of samples collected in the hotspot function. For caller functions, the **Samples** column shows the number of samples collected in the hotspot function when it was detected on the part of stack from the selected caller function to the hotspot function.

Function selection in the Hotpath view re-focuses the Hotspot view to the same function, and vice versa.

### 7.4.4.4 Hint Icons

There are hint icons that may appear for a hotspot in the statistical call graph Hotspot view:

Icon	Description
	Function has a loop that falls into top hot loops category.
	Function is called from the one of the hot loops. The caller function with a loop may not be necessarily an immediate parent function.
	Recursive function.

Top hot loops are defined as top 20% loops and evaluated as total samples per loop. This means that samples for all nested loops are summed up.

#### See Also:

[Viewing Statistical Call Graph Data](#) (CLI)

[Identifying Loops](#) (CLI)

[Filtering Controls](#) (GUI)



## 7.4.5 Call Count View

The Call count view opens in the central Eclipse window and displays a limited set of [exact call graph](#) data: a function name, module name, and call count, which is a number of times the function was called.

**NOTE:** To view the call count data collected from the command line in the graphical interface, go to **File > Import** and select the **Import Intel(R) PTU Experiment** from the wizard.

**See Also:**

[Viewing Call Graph Data](#) (CLI)

## 7.4.6 Heap Profiling View

The Heap profiling view opens in the central Eclipse window and displays the following data:

Column	Description
<b>Function</b>	Name of the function.
<b>Hint</b>	Recursion <a href="#">hint icon</a> .
<b>Module</b>	Name of the module the function belongs to.
<b>Spacetime</b>	<a href="#">Spacetime</a> value.
<b>Leaked memory</b>	Amount of memory (in bytes) not returned to the heap when the application exited.
<b>Leaked objects</b>	Number of memory blocks not returned to the heap when the application exited.
<b>Memory allocated</b>	Amount of memory (in bytes) allocated by the application in this function and its callees in the call stacks during the application run.
<b>Objects allocated</b>	Number of blocks allocated by the application in this function and all function callees in the call stacks during the application run.

**See Also:**

[Viewing Heap Profile Data](#) (CLI)

## 7.4.7 Data Access View

If you collected data using the Basic Data Access Profiling configuration or configured data memory related events by yourself and checked **Enable data access profiling** check box in the **Profile** configuration dialog box, you are prompted to choose between the Data Access view and Sampling Hotspot view when opening the experiment results.

Extending the Sampling Hotspot view, the Data Access view presents data memory hotspots as well as code hotspots. The Data Access View shows data for the process

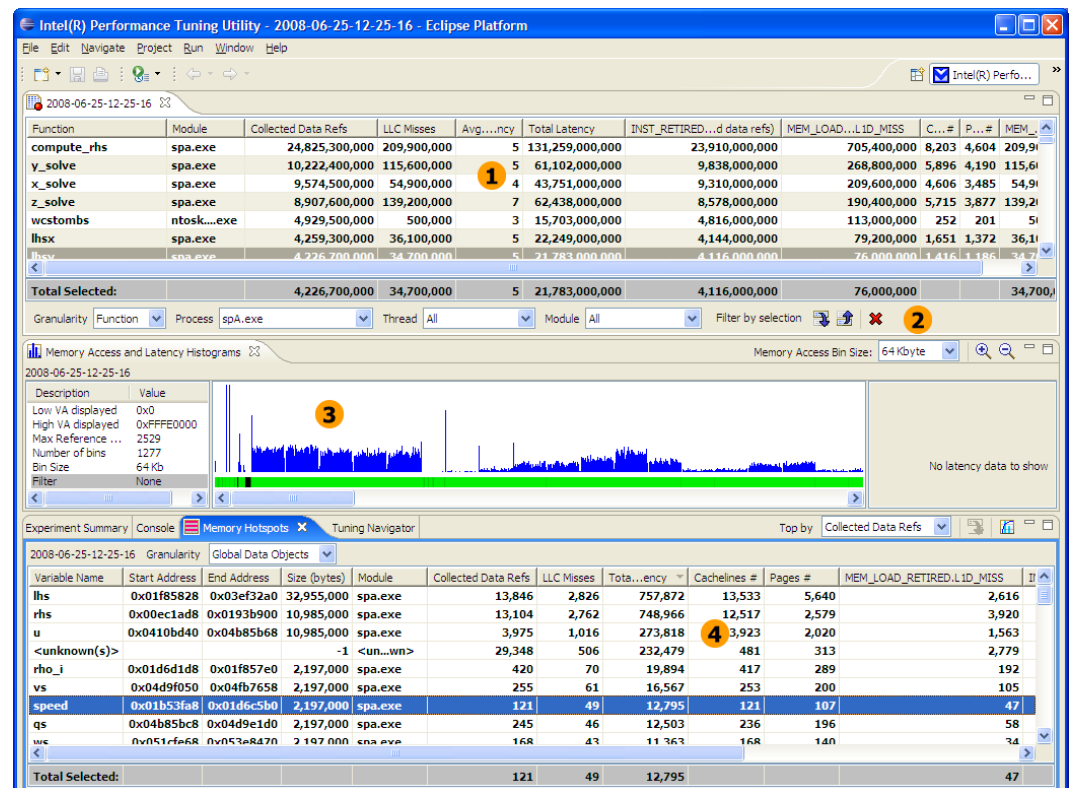


that collected most of the data referenced in the experiment. You can choose another process in the filter bar or view data for all processes together. Viewing data for all the processes together is useful if many processes have shared memory space.

The Data Access view consists of the Filter toolbar, [Hotspot pane](#), [Memory Hotspots pane](#), [Memory Access and Latency histogram pane](#). By default, the Hotspot pane and Memory Hotspots pane appear at the end of data collection. To open the Memory Access and Latency Histogram, right-click the Hotspot pane and choose **Memory Access and Latency Histogram** from the pop-up menu.

The Data Access view shows “raw” sampling events and metrics calculated from the raw events, for example, total number of references and number of accessed cachelines. The Data Access view also provides extended filtering capabilities that enable displaying data memory hotspots only for specific code sub-sets and vice versa.

When you open several Data Access profiling views, the Memory Hotspots and Memory Access and Latency histogram panes synchronize with the Hotspot pane in focus. However, if collected events do not provide precise latency information, the Latency Histogram is empty.



1

Hotspot pane

3

Memory Access and Latency  
Histogram pane



<b>2</b>	Filter bar	<b>4</b>	Memory Hotspots pane
----------	------------	----------	----------------------

### 7.4.7.1 Hotspot Pane

The Hotspot pane of the Data Access view is similar to the Sampling Hotspot view. But, in this Hotspot pane, the Intel Performance Tuning Utility calculates data memory-related metrics from sampling events and presents them in the table.

The code regions, such as functions and basic blocks with high values in the **Data Refs** (data references), **Total Latency**, and **Average Latency** columns have major impact on the performance. To sort the columns by the values in a certain column, click on the column header.

By default, the metrics values are shown as number of events. To change the value metric to samples right-click and select **Samples** from the pop-up menu.

To drill-down to the Source view of the selected code region, follow these steps:

1. Set the granularity level to Function or RVA
2. Double-click the required function or address or use the View Source pop-up menu.

### 7.4.7.2 Memory Hotspots Pane

The Memory Hotspots pane has a notion of **Granularity** similar to the code Hotspot pane. The granularity could be **Cachelines** and **Global Data Objects**. To switch between cachelines and global data objects, use the **Granularity** control on the top left of the Memory Hotspots pane.


When **Granularity** = **Cacheline**, the pane shows memory chunks with the highest number of references from the code. The size of the chunks in this pane is aligned to the size of the processor cache line size. Therefore, they are called *cachelines*. Each chunk corresponds to one cacheline that loaded to the CPU cache as a whole. For IA-32 and Intel® 64 architecture, the cacheline size is 64 bytes. For the Intel® Itanium® architecture, the cacheline size is 128 bytes.

The metrics in the Memory Hotspots pane are shown as samples.

By default, the Memory Hotspots pane displays 200 hottest cachelines. To change the number of cachelines displayed, go to **Window > Preferences > Intel® Performance Tuning Utility > Data Access View**.

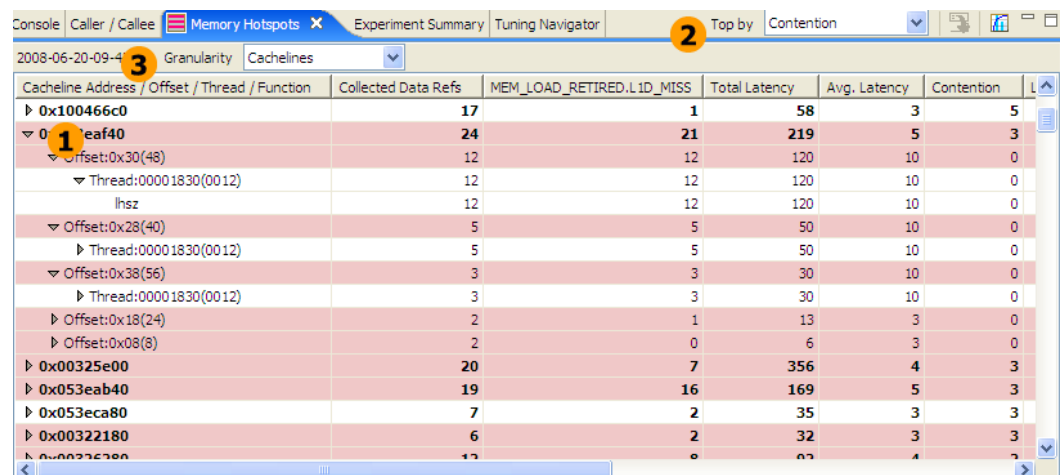
When in the cacheline granularity, the Memory Hotspots pane provides information on the accessed memory offsets within the cacheline and the list of threads and functions



that accessed a specific offset. To view this data, click the small triangle  in the cacheline address.

The Memory Hotspots pane displays the cachelines based on the criteria selected in the **Top by** drop-down menu. By default, this selects the 200 cachelines with the most collected data references. You can change the criteria to select the top lines by: LLC misses, Average Latency, Total Latency, Contention, or by any of the precise memory access events that were used in the profile. The Contention condition is defined as the number of references by secondary threads. This is evaluated with the difference of (total references – references by the primary thread), where the primary thread for each line is the thread with the most references to that line.

If you change the granularity to global data objects, the Memory Hotspots pane shows global data objects defined in the modules of the focus process and associate them with metrics corresponding to events collected on these objects.



Cacheline Address / Offset / Thread / Function	Collected Data Refs	MEM_LOAD_RETIRED.L1D_MISS	Total Latency	Avg. Latency	Contention
▶ 0x100466c0	17	1	58	3	5
▼ 0x100466c0	24	21	219	5	3
▼ Offset:0x30(48)	12	12	120	10	0
▼ Thread:00001830(0012)	12	12	120	10	0
lhsz	12	12	120	10	0
▼ Offset:0x28(40)	5	5	50	10	0
▶ Thread:00001830(0012)	5	5	50	10	0
▼ Offset:0x38(56)	3	3	30	10	0
▶ Thread:00001830(0012)	3	3	30	10	0
▶ Offset:0x18(24)	2	1	13	3	0
▶ Offset:0x08(8)	2	0	6	3	0
▶ 0x00325e00	20	7	356	4	3
▶ 0x053eab40	19	16	169	5	3
▶ 0x053eca80	7	2	35	3	3
▶ 0x00322180	6	2	32	3	3
▶ 0x00322180	12	8	82	4	2

1	Triangle to expand/collapse the cacheline address
2	Top by drop-down menu to query and sort the cachelines by events
3	Granularity control to switch between cachelines and global data objects

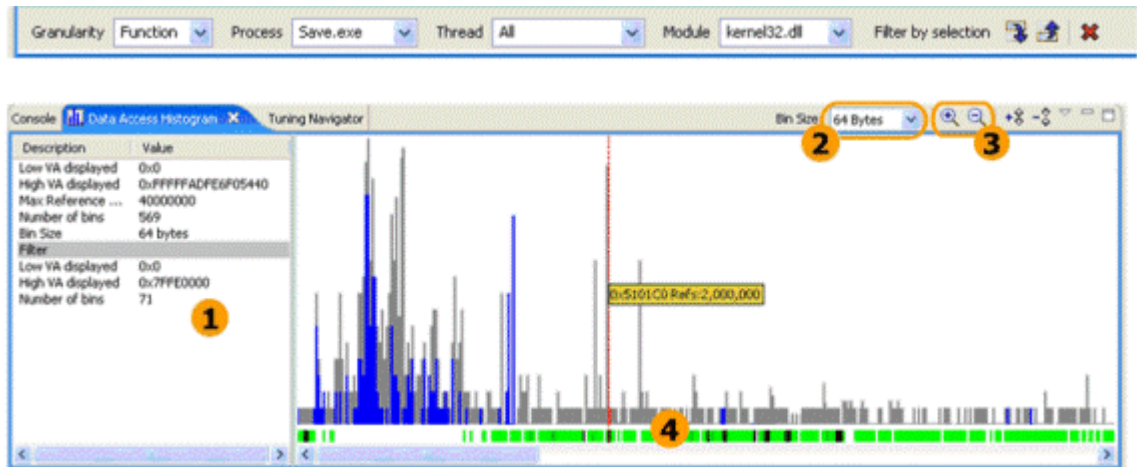
### 7.4.7.3 Data Access and Latency Histogram Pane

The histogram view opens at the bottom of the Eclipse window and consists of two charts and a data summary. The chart on the right shows a number of references occurred with a specific latency value. This chart is empty if no events providing precise latency information are collected. Such events are present on the Intel® Itanium processors family only. The chart in the middle shows data reference





distribution over memory virtual space. The grid on the left provides summary information on the data presented by the histogram. This chart is also referred to as the Memory Access histogram.



1	Histogram summary	3	<b>ZoomIn/ZoomOut</b> buttons to magnify/minify the histogram view
2	Drop-down menu to change current bin size	4	Distance legend

The X-axis of the histogram displays memory locations. The Y-axis displays the number of memory references. Grey bars show references for the whole process. Blue bars show references according to the current filter setting. Thus, on the figure above, blue bars show references for the `kernel32.dll` module selected in the **Module** drop-down menu above.

The histogram presents memory locations in *bins*. A bin represents a continuous virtual memory chunk, of fixed size, and sums all references made to the addresses it covers. One bar represents one bin. The size of a bin can differ from the size of a cacheline (64 or 128 bytes) up to 16 Mbytes. The bin address space is defined by the start address and size. To change the level of details shown for a memory region, change the bin size in the histogram. The bigger the bin size is, the less number of bins is shown for the same fixed region.

Bins without any memory references are omitted from the histogram. This may cause different distance between adjacent bins.

The *distance legend* under the bars represent the spatial distribution of the bins. Different shades of green, white, or black indicate spatial separation:

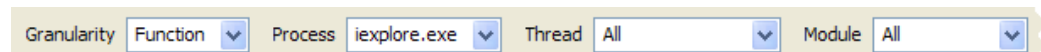
- Black bars indicate regions that are separated by more than 1GB

- White bars indicate adjacent regions
- Green bars indicate regions that are not adjacent, but the space between them is less than 1GB. Darker shades of green indicate a larger space

Hover the mouse over a memory chunk to see a tooltip with the summary information on this chunk. To get information about a particular memory region, select the required region by pressing the mouse button and dragging the cursor over the memory axis. You can also filter the code hotspots for a particular region using the [filtering options](#).

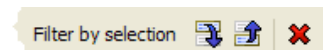
#### 7.4.7.4 Filtering the Data Access View


The Data Access view filtering options consist of standard and advanced view-specific options. The standard options include the **Process**, **Thread**, and **Module** controls residing on the central [filterbar](#) that enable displaying the code and memory hotspots for a specific process, thread, or module.



Data access filtering helps you analyze specific source code or assembly instruction(s) responsible for high memory latencies or for data reference peaks. To do this, drill-down to the [Source view](#) and analyze the exact code associated with specific data address access, high latencies, or, in general, any reference peak(s).

For advanced filtering, select items either in the code [Hotspot pane](#), [Memory Hotspots pane](#), or a region on the [Data Access and Latency Histogram pane](#) and use the **Filter by Selection** buttons on the filterbar. These buttons enable filtering code hotspots by selection in the data pane or histogram, and vice versa.



Use the advanced filtering options to highlight only those memory references that were made by specific module(s), thread(s), function(s), or separate instructions. And vice versa, select memory hotspots (data object or cacheline(s)) or a memory region to filter the hotspots that affected the selected memory. To return to the default non-filtered state, click the  **Cancel All Filters** button on the filterbar.

#### 7.4.7.5 False Sharing Hints

False sharing is one of the typical problems that limits application performance on multiprocessor systems. False sharing happens when processors write to a shared cacheline but not at the same location. As a result, the cache coherence protocol



makes each processor invalidate a cacheline and forces the cacheline reload at each write.

The Data Access analysis identifies possible false sharing candidates by monitoring different offsets of the same cachelines accessed by different threads. The cachelines identified with high probability as false-shared are highlighted in pink in the [Memory Hotspots pane](#).

Cacheline Address / Offset / Thread / Function	Refs (%Total)	Avg. Latency	Total Latency (...)	Contributors
▶ 0x000000000000f680	8,000,000 (0.7...		3 24,000,000 (0.0...	Offsets: 1 Threads: 1
▼ 0x000000014011e380	4,000,000 (0.4...		3 13,400,000 (0.0...	Offsets: 3 Threads: 3
▼ Offset:0x20(32)	2,000,000 (0.2%)		3 6,000,000 (0.1%)	Threads: 1
▶ Thread:000008fc(0007)	2,000,000 (0.2%)		3 6,000,000 (0.1%)	Functions: 1
▼ Offset:0x30(48)	2,000,000 (0.2%)		3 6,000,000 (0.1%)	Threads: 1
▶ Thread:00000b50(0005)	2,000,000 (0.2%)		3 6,000,000 (0.1%)	Functions: 1
▶ Offset:0x38(56)	200,000 (0.0%)		10 2,000,000 (0.0%)	Threads: 1
▶ 0x000000014007a480	4,000,000 (0.4...		3 12,000,000 (0.0...	Offsets: 2 Threads: 1
▶ 0x0000000140156c80	4,000,000 (0.4...		3 12,000,000 (0.0...	Offsets: 1 Threads: 2
▶ 0x000000014007b380	4,000,000 (0.4...		3 12,000,000 (0.0...	Offsets: 2 Threads: 2
<b>Total Selected:</b>	<b>2,000,000 (0.2...</b>		<b>3 6,000,000 (0.1...</b>	


#### See Also:

[Viewing Data Access Results](#) (CLI)

## 7.4.8 Source View

For better understanding of a performance problem, it is important to associate a hotspot with the source code and exact machine instruction(s) that caused this hotspot. To do this, you can use the Source view.

You can drill-down to the Source view as follows:

- From the [Hotspot view](#), do one of the following:
  - Right-click the required hotspot and select the **Drill-down to source view** pop-up menu command.
  - Set the granularity level to Function, Basic block, or Address and double-click a hotspot.
- From the [Events over IP view](#), select a region of IPs and press the  **Source View** button.
- From the [Caller/Callee view](#), right-click and use the pop-up menu.

In the current version, the Source view displays only the function the selected hotspot belongs to.



The Source view contains the following panes: [Source](#), [Disassembly](#), and [Control Flow Graph](#). These panes are synchronized with each other so that selecting a line or basic block in one pane highlights corresponding line(s) or basic block(s) in the others. For example, selecting an assembly instruction in the Disassembly pane highlights a corresponding code line in the Source pane and a related basic block in the Control Flow Graph.

The Source and Disassembly panes display accurate information provided that:

- Your code is compiled with the debug information and debug information is written correctly in the binary file on Linux or debug information file (or symbol file) on Windows.
- Source code file exists.

If the source file is unavailable, only Disassembly pane is displayed. If there is no correct debug information or symbol file is unavailable, the disassembly view might be incorrect. In this case, the Source view uses heuristics to define function boundaries in the binary module.

The information available from the Source view helps you find all local hotspots within a function and identify code sections that require optimization. Navigate between hotspots using the Hotspot navigation buttons.

**NOTE:** Press **Ctrl-F** to search for a text pattern in the source or assembly code, bookmark all occurrences of this text, and iterate over them.

### 7.4.8.1 Source Pane

The Source pane displays source code of the function, line information, and number of processor events (or their samples) associated with a code line.

In the Source pane, code lines of the selected hotspot function are displayed in blue while the other code in the source file is displayed in black. For black regions, no information is provided.

To export the Source pane content to a comma-separated text file, right-click and use the pop-up menu.



Source		
Line	Source	Samples
14	for(i = 3.0; i < 1000000; i += 2.0)	44
15	{	
16	pi -= 4.0 / i;	126
17	i += 2.0;	7
18	pi += 4.0 / i;	117
19	}	
20	return pi;	
21	}	
22	void funcA (void) { printf("PI = %f\n", funcC...	
Total Selected:		250

### 7.4.8.2 Disassembly Pane

Disassembly pane displays assembler instructions for the selected hotspot function grouped by *basic blocks*. The basic block lines are marked in bold. You can expand/collapse basic blocks by right-clicking the Disassembly pane and selecting the **Expand/Collapse All** command from the pop-up menu.

To export the current view to a comma-separated text file, also use the pop-up menu.

Address	Line	Assembly	INST_RETIRED.ANY_P	MEM...	MEM_LO
0x7729	1346	add r15,08h			
0x772c	1346	cmp r15,r14			
0x7730	1346	jnge compute_rhs+04df4h			
▼ Block 210 compute_rhs+04ee6:					
0x7736	1346	lea r14,QWORD PTR [r12+rbp*8]			
0x773a	1346	lea rbp,QWORD PTR [r12+r10*8]			
▼ Block 211 13... compute_rhs+04eee:					
0x773e	1347	movsd xmm4,MWORD PTR [r14 + .I0x3a...	2	176	7
0x7747	1347	movhpd xmm4,MWORD PTR [r14 + .I0x3...	1		
0x7750	1347	movaps xmm13,XMMWORD PTR [rip+01d1a8h]			
0x7758	1347	divpd xmm13,xmm4	2		
0x775d	1349	movsd xmm11,MWORD PTR [r14 + .I0x3...	10		
0x7766	1349	movhpd xmm11,MWORD PTR [r14 + .I0x...			
0x776f	1350	movsd xmm8,MWORD PTR [r14 + .I0x3e...		2	
0x7778	1350	movhpd xmm8,MWORD PTR [r14 + .I0x3...		3	
0x7781	1351	movsd xmm9,MWORD PTR [r14 + .I0x41...			
0x778a	1351	movhpd xmm9,MWORD PTR [r14 + .I0x4...			
Total Selected (2 instructions):			3		

1	Source line number
2	Basic block name or Relative Virtual Address of instruction



<b>3</b>	Assembly code of instruction (instruction name and parameters)
<b>4</b>	Number of samples occurred during instruction or basic block execution
<b>5</b>	Number of samples occurred during the execution of the selected instructions

If you collected the experiment using the **Enable data access profiling** option, you can view the statistics for register values for any instruction or a set of instructions that have samples for precise events. To do this, select an instruction (or a set of instructions), right-click and select the **Show registers values statistics** menu item from the pop-up menu:

0xAD01	push	rsi	546		
0xAD02	push	rbp	198		
0xAD03	push	rbx	597		
0xAD04	sub	rsi	<b>Registers Stats</b>		
0xAD08	mov	rbx			
0xAD0E	mov	rbx			
0xAD11	mov	r11			
0xAD14	mov	r11			
0xAD17	mov	rax			
0xAD1F	movsxd	rdi			
0xAD22	mov	rcx			
0xAD25	mov	r11			
0xAD2D	mov	r11			
0xAD35	movsxd	r10			
0xAD38	movsxd	r11			
0xAD38	lea	rdi			
0xAD42	mov	r8			
0xAD48	mov	QWORD [rsi]			
0xAD50	shl	rdi			
0xAD54	mov	r9d, 01h	24		

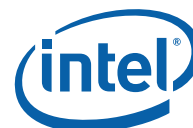
Name	Min	Max	Average	Std. Dev
rax	5369645016	5369645016	5369645016	0
rbx	5369645028	5369645028	5369645028	0
rcx	5369644464	5369644860	5369644658	184
rdx	5369645032	5369645032	5369645032	0
rsi	1	18432	1143	3350
rdi	3	3	3	0
rbp	5369645024	5369645024	5369645024	0
rsp	41347504	367057072	221775021	99848022
r8	5369645028	5369645028	5369645028	0
r9	5369645024	5369645024	5369645024	0
r10	5369644992	5369644992	5369644992	0
r11	5369645016	5369645016	5369645016	0
r12	18432	184467440737095515...	1253096358903611392	4641691216271772672
r13	41350488	41350704	41350623	104
r14	5369644464	5369644860	5369644658	184

Click outside or press <Esc> to close

For each general purpose register, minimum, maximum, average and std. dev. values are shown.

On Core™ i7 systems running an application compiled for Intel® 64 architecture systems, it is possible to use this option to display statistical metrics of the function integer arguments, when they are passed through registers in accordance with the calling conventions for the OS (different for Windows\* and Linux\*). You can do this by collecting the BR\_INST\_RETIRED.NEAR\_CALL or BR\_INST\_RETIRED.NEAR\_CALL\_R3 event and using only the instructions with these precise events.

Loop trip count metrics can also be extracted from the registers statistics view. If the BR\_INST\_RETIRED.ALL\_BRANCHES or BR\_INST\_RETIRED.CONDITIONAL event is captured, enabled for data access profiling (capturing register values with the events), then the register values at the first instruction of the loop and the disassembly for the comparison logic controlling the terminating conditional branch (end of the loop) can be used to determine the trip count value. Most counted loops are compiled to compare a loop index with a maximum value which is held in a register. Even for non-



counted loops (for example, while loops), there is almost always an induction variable in the loop logic and the mean value of the induction variable will be half the average trip count.

### 7.4.8.3 Control Flow Graph

Control Flow Graph (CFG) displays the function execution flow, where each node indicates a basic block. Control Flow Graph is always displayed for one function only. To view the Control Flow Graph, click the **Control Flow Graph** button at the top of the Source View window.

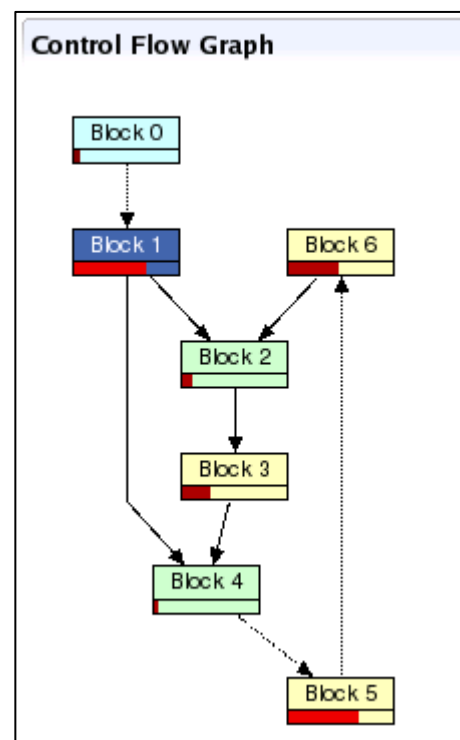
Each node has a “thermometer” bar at the bottom. It displays the ratio of number of events (for the Event of Interest) occurred during execution of this basic block to the number of events occurred during execution of the hottest basic block of this function. For the hottest basic block, the whole “thermometer” bar is filled in red.

Control Flow Graph represents the flow of function execution by means of connectors (edges) between nodes. Most of graph edges are shown as solid lines. If a basic block ends with a call instruction, the edge to the next block is shown as a dotted line.

You can zoom in/out the Control Flow Graph by using the pop-up menu commands.

The selected basic block is colored in grey or in saturated blue when CFG is in focus. The caller and callee blocks are highlighted in light blue and light green respectively.

Basic block tooltip shows assembly instructions contained in the block with event values. Press **F2** to retain the tooltip on the screen. You can scroll extensive basic blocks or copy the content to clipboard by using the tooltip menu.



### 7.4.8.4 Source View Preferences

When drilling down to the Source view, the Intel Performance Tuning Utility checks for the debug information. If it is incomplete or missing, the **Mode Selection** dialog box opens and prompts you to specify the viewing mode for the target function. The Intel Performance Tuning Utility uses the following modes in case of incomplete debug information:



- **Source file and disassembly recovered from debug line information.** In this mode, the Intel Performance Tuning Utility restores the disassembly boundaries based on the available debug line number information of the source file. This mode is available and recommended if information about symbol size is missing in the module but line information exists. This mode enables viewing a source file in the Source view.
- **Pseudo function heuristically determined by/from disassembly.** Use this mode if there is no debug information found for the module. In this mode, the Intel Performance Tuning Utility uses an algorithm to restore function boundaries from the static analysis of disassembly.
- **Fixed range of disassembly.** Use this mode if there is no debug information found for the module. In this mode, the Source view opens with a region surrounding the target address. The default range size is 1024 bytes of code. A larger portal size enables viewing a larger area that is more likely to include functions. However, it takes longer to load. This mode is recommended if pseudo function mode gives inadequate results.

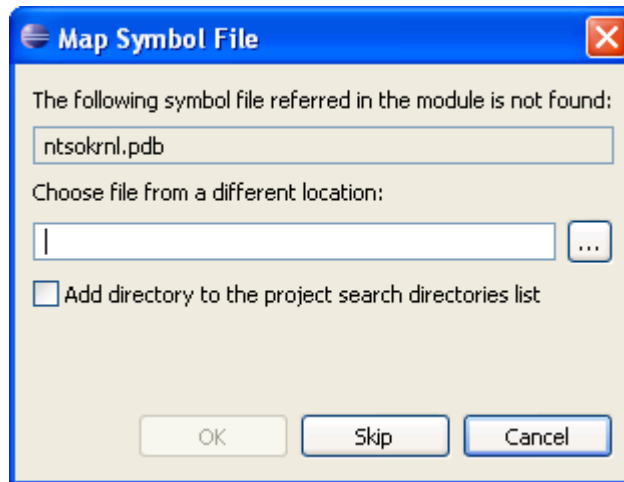
Select a preferred mode for the selected function. If you want to make this mode default for all functions in the project, select the **Use these values as defaults and never show this dialog for this project again** check box. You may also select any of these modes as default from the **Source View** preference page. To access this page, go to **Window>Preferences> Intel® Performance Tuning Utility> Source View**.

The **Source View** preference page enables you to set up a preferred assembly syntax style for the Disassembly view on Intel® 64 and IA-32 architecture. To do this, click the corresponding radio button to select between **Intel®** and **AT&T\*** syntax style. **Intel®** syntax is used by default.

### 7.4.8.5 Mapping Files

When you request to open the [Source view](#), the Intel Performance Tuning Utility tries to locate the binary file and source file corresponding to the selected hotspot and open source code file with the disassembly section of corresponding region in the binary file. To proceed with disassembly and define hotspot function boundaries, the Intel Performance Tuning Utility requires a symbol file. If some of the files cannot be located during [automatic resolution](#) and [search directories look-up](#) while drill-downing to the source, the **Map <...> File** dialog box opens.





Use this dialog box to specify the correct filename and full path to it. To add the directory you specified to the corresponding Directory search group, select the **Add directory to the directories search list** check box. You may also choose **Skip** if no such a file is available. In this case, the Source view runs a heuristic algorithm to define function boundaries.

If a source file cannot be found, the **Map Source File** dialog box opens. To locate the file, you can either specify a path to the correct source file or click the **Assembly Only** button. In the latter case, only Disassembly and Control Flow Graph pane opens. You may choose to skip this dialog box for all unresolved source files by selecting the **Never show this dialog for this project again (show assembly only instead)** check box. Only Disassembly view will be available.

By default, the Source view may find a wrong (or modified) version of the source file. In this case, press **SHIFT** while drilling down from a previous view to the Source view. The Source view prompts you to specify the source file location and this is your responsibility to provide the correct source file related to the experiment.

During module resolution, the Intel Performance Tuning Utility compares the modification date for the binary and source file. If the source file is modified later than the binary file, the **Source File Changed** dialog box opens and prompts you to specify the source file to use. If you want the Intel Performance Tuning Utility to always choose the latest source file, select the **Never show this dialog for this project again (always use changed file instead)** check box.

You can control the module resolution dialog boxes via the **Source View** property page. To access the property page, right-click the project in the **Tuning Navigator** and select **Properties of... > Intel(R) PTU Project > Source View**. This page enables you to disable the selected types of the module resolution dialog boxes and use the default options instead.

## 7.4.9 Hotspot Difference Views

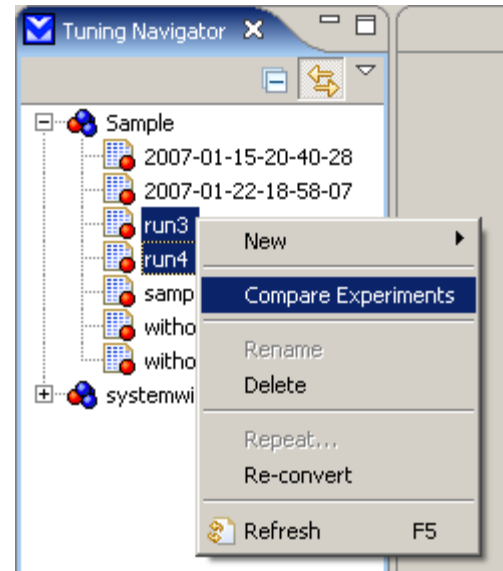
After you measured the performance of your application and optimized the code, you may run an experiment with the same workload and identify the performance difference against the benchmarks. The Intel Performance Tuning Utility provides an option to compare two sampling results of selected experiments and view the difference in sample count (or time for Clockticks) for each event.

To compare results of two sampling experiments, select the required experiments from the **Tuning Navigator**, right-click and select **Compare** **Experiments** from the pop-up menu. The Hotspot Difference view opens.

Each hotspot line in the Hotspot Difference view displays difference between Experiment 1 and Experiment 2 for each event as follows:

<Difference Value> = <Experiment #1 Value> – <Experiment #2 Value>

The time difference is evaluated in milliseconds as this allows comparing the performance of two different Intel® microprocessors, running at different frequencies.



You can expand a hotspot line to view experiment specific data: the first row provides event values for Experiment 1 and the second row - for Experiment 2.

You can filter the Hotspot Difference view to focus only on the functions with the defined threshold difference. From the **Delta** drop-down menu on the filterbar, select a threshold value. The Hotspot Difference view updates to include only the hotspots with the difference not smaller than the defined threshold, by absolute value.

Basic-Sampling-2009-08-25-18-04-17 - Basic-Sampling-2009-08-25-18-06-55				
Function	Module	Time(msec)	INST_RETIRED.ANY	
▶ test_if	vtunedemo.exe	20	-10	
▶ test_memset	vtunedemo.exe	11	-16	
▶ divd_rout	vtunedemo.exe	6	-19	
▼ test_if1	vtunedemo.exe	-18	36	
Basic-Sampling-2009-08-25-18-04-17		157	287	
Basic-Sampling-2009-08-25-18-06-55		175	251	
▶ CmUnRegisterCallback	ntoskrnl.exe	4	-10	
▶ get_fcsr	msvcr80d.dll	9	10	
Granularity	Function	Process	All	Module All Delta 10



Double-click a hotspot line in bold to drill down to the Source Difference view that displays the sample count difference between Experiment 1 and Experiment 2 for each code line of the selected function.

**NOTE:** Address and basic block granularity is available for identical binaries only. Different binaries are automatically excluded from the view. Function and module granularity is available for both identical and different binaries.

To drill down to the experiment specific Source view of the selected function, double-click the corresponding row under the hotspot function.

### 7.4.9.1 Modifying the CPU Frequency

With the Hotspot Difference view, you can analyze how the CPU frequency change affects the code execution time. Typically, the CPU frequency is automatically detected during the data collection. But sometimes it is impossible to detect the frequency properly or you may want to know the performance of your system/application with a different frequency. In this case, you can modify the CPU frequency using the Hotspot Difference view pop-up menu and analyze the **Time(msec):Diff** column value. The **Time(msec)** value is calculated as Clockticks total value / (CPU frequency \* 1000) and corresponds to the time spent for execution of the respective unit (module, function, basic block, or instruction). The **Time(msec):Diff** value is a difference between Experiment 1 and Experiment 2:  $\text{Time(msec):Diff} = \text{Exp1 Time} - \text{Exp2 Time}$ .

To specify a different frequency value for an experiment(s):

1. Right-click the Hotspot Difference view and select the **Override CPU Frequency** pop-up menu command.  
The **Override CPU Frequency** dialog box opens.
2. Modify the CPU frequency value for both or one of the experiments and click **OK**.  
The Time values in the Hotspot Difference view are recalculated according to the formula.

#### See Also

[Comparing Two Sampling Experiments](#) (CLI)

### 7.4.9.2 Handling Modules with Identical Names

In the Hotspot Difference view, you may want to combine modules with the same name into one ignoring the paths to the modules. The performance data for the new alias module presents the summed up data from the combined modules. This technique could be useful, for example, if you compare experiments with identical workload collected on different systems.

To automatically combine modules with identical names into one, go to the **Preferences>Intel® Performance Tuning Utility>Hotspot Difference View** and



select the **Treat such modules as the same module** option in the **Handling modules/functions with identical names** section.

### 7.4.9.3 Comparing Functions with Identical Names

From the Hotspot Difference view, you can compare functions from the same source files but different modules. This option could be useful for compiler regression analysis. Thus, you can compile the same source file twice with different compiler options and, then, compare the same functions by drilling down to the Source view from the Hotspot Difference view. In this case, the Source view displays a single source pane and two assembly panes for each module: **Assembly (for 1st exp.)** and **Assembly (for 2nd exp.)**. The Control Flow Graph is not available.

The screenshot shows the Intel Performance Tuning Utility interface. The top bar displays the file path '2007-07-22-00-09-51 - 2007-07-22-00-10-17' and the file name 'vtunedemo.c'. Below the bar are tabs for 'Source', 'Assembly (1st exp.)', and 'Assembly (2nd exp.)'. The 'Source' pane shows a C code snippet with line numbers 166 to 171. Line 169 is selected, and the corresponding assembly blocks are highlighted in the assembly panes. The 'Assembly (1st exp.)' pane shows Block 16 (testmain+015ah) and Block 17 (testmain+0167h). The 'Assembly (2nd exp.)' pane shows Block 5 (testmain+045h). The 'Total Selected' row at the bottom of each pane indicates the number of selected lines and blocks.

L...	Source	T...	I...
166	// Show the speedup of a local value		
167	for(i = 0; i < 100; i++)	3	
168	{		
169	Global_Test_if_putp = i%2; // set g...	-1	-2
170	test_if1(ia, ib, ic, 90);	2	
171	}		
Total Selected:		-1	-2

Address	L...	Assembly (1st ex...	T...	I...
▼ Block 16 testmain+015ah:				
0x1BBA	169	mov ...		
0x1BC0	169	and ...		
0x1BC5	169	jns ...		
▼ Block 17 testmain+0167h:				
0x1BC7	169	dec eax		
0x1BC8	169	or ...		
0x1BCB	169	inc eax		
▼ Block 18 169 testmain+016ch:				
0x1BCC	169	mov ...	1	1
Total Selected (...)			1	1

Address	L...	Assembly (2nd ...)	T...	I...
▼ Block 5 169 testmain+045h:				
0x1105	169	mov ...	1	3
0x1107	169	and ...		
0x110A	169	add ...		
0x110D	169	cmp ...		1
0x1110	169	mov ...		1
0x1116	169	jnge ...		1
Total Selected...			1	3

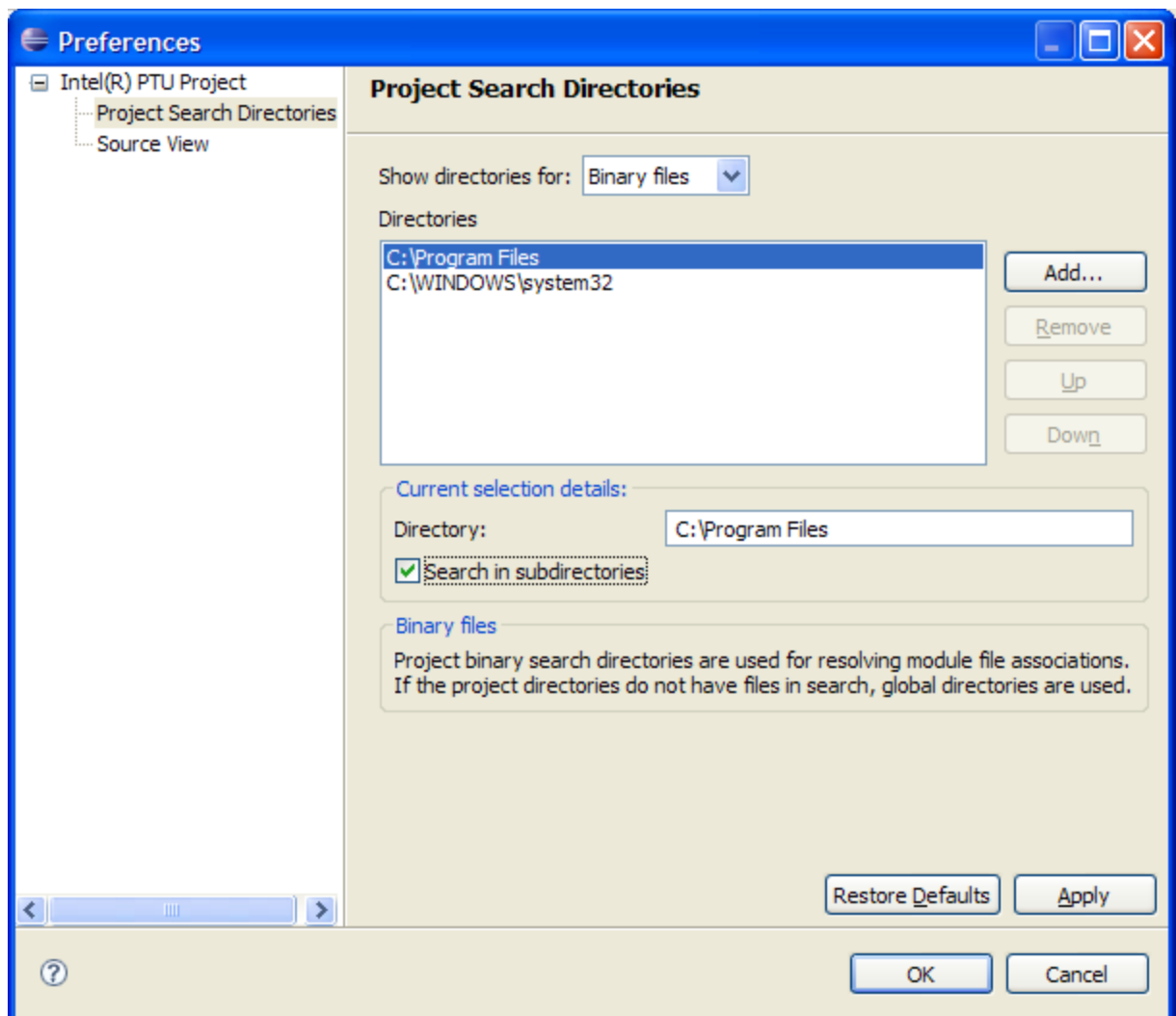
Selecting lines in the source pane filters corresponding basic blocks in the assembly panes. You can export the selected source lines from the source pane and filtered basic blocks in the assembly pane to a CSV file via the **Export selected source and associated basic blocks** option of the pop-up menu. The CSV file will include three spreadsheets separated by headers.



## 7.5 Specifying Search Directories

You can specify the [search directories](#) in the **Project Search Directory** page and **Global Search Directory** page. Typically, you use the global search directory list to specify directories specific for your machine, for example, a directory that contains PDB symbol files for system libraries. The project search directory list is usually used for directories specific to the current project, for example, a directory where the project sources are located. Project search directories are always treated as high priority.

To access **the Project Search Directories** property page, right-click the project name in the **Tuning Navigator** and select **Properties of...** > **Intel(R) PTU Project** > **Project Search Directories**.





Lists of the global directories are controlled via the **Global Search Directories** preference page. To access this preference page, go to **Window>Preferences>Intel® Performance Tuning Utility>Global Search Directories**.

## 7.6 Importing Data

You can import existing experiments and projects that were created in other workspaces using the GUI, or from the command-line interface.

To import the experiment into a specific project in your workspace:

1. Copy the experiments into the project directory in the workspace.  
In case you have no projects in this workspace, create a new empty project.
2. Open Intel(R) PTU and the **Tuning Navigator**.
3. Select the project into which you copied the data.
4. Press **F5**.  
You will see all the copied experiments under the project node.

You can also import an experiment into the current workspace using **File > Import > Import Intel(R) PTU Experiment** dialog box. When the importing process finishes, the experiment appears in the **Tuning Navigator** under the **Import Project** node.

You can analyze the data with the hotspot view at this stage.

If you need access to binary, symbol or source files to perform the analysis, you should define search directories for the analyzed data. For your convenience, group the experiments, collected on the same workload, under one project directory. This grouping enables you to [define the search directories](#) for all the experiments once in the project properties. Otherwise, you can set search directories for each experiment separately as [predefined search directories](#).

To import a whole project created in another workspace, select **File > Import > General > Existing Projects into Workspace**. In this case the original project settings are preserved.

**TIP:** If you want to import several projects from the same location, set this location as the root directory in the **Import** dialog box.

**NOTE:** If you collected the data using the GUI, you can find the location of project or experiment by switching to the **Resource** perspective:

1. Select **Window > Open perspective> Resource**.
2. In the opened **Navigators** right-click the required node and select **Properties** from the pop-up menu.



## 8 Collecting and Analyzing Data on Different Systems

---

With the Intel Performance Tuning Utility you can collect data on one machine and analyze them on another machine. Computer used to collect data is called a *collection machine*. Computer used to analyze the collected data is called an *analysis machine*.

Each data collection and analysis session includes the following steps:

1. Collect data.
2. Convert data.
3. Copy the collected data to the analysis machine and analyze it.

### 8.1 Collecting Data

During data collection session raw data are gathered to the experiment directory.

You may choose to collect data either using the graphical interface or command-line interface:

- In the GUI, you can collect the data using a predefined Configuration (for example, Basic Sampling) or create your own.
- In the command-line interface, use the [vt\\*run](#) commands to collect data. For example:

```
$ vtsarun <experiment_dir> -s -- <application> <application arguments>
```

**NOTE:** You may want to use the Intel Performance Tuning Utility GUI to [generate the command line for collection](#).

### 8.2 Converting Data

Raw data gathered during the data collection should be converted to a database. Symbols resolution, attributing performance data to various levels of analysis granularity (modules, basic blocks, functions, classes) are particular phases of data conversion process.



**NOTE:** Data conversion should be performed on the collection machine. To have the data conversion process working correctly on the analysis machine, copy all the modules referenced by the experiment to the analysis machine and configure search directories properly using the `--search-dir` option.

In the GUI, you do not need to explicitly convert the data because it is done automatically upon the data collection completion.

In the [command-line interface](#), convert the collected raw data so that the data can now be moved to another machine. The command for data conversion must correspond to the `vt*run` command used for collection. For example:

```
$ vtsaview <experiment_dir> --convert
```

## 8.3 Copying Data to the Analysis Machine

If you want to analyze the data using the command-line interface, copy the experiment directory from the collection machine to the analysis machine and use [vt\\*view tools](#).

If you want to analyze the data from the GUI you should copy and import your data. See [Importing Data](#) for more information.

To be able to drill-down to the source and disassembly on the analysis machine, you need to make sure that the original modules, symbols, and sources can be found. To view hotspot data only, [conversion](#) step is enough.

Consider using one of the following strategies to point to modules, symbols, and sources:

- [Specifying search directories](#)
- Using [predefined search directories](#)

It may be useful to read the whole chapter [File Resolution](#) to fully understand how it works. This can be critical for converting and analyzing data on a machine different from the machine where collection was performed.





## 9 Troubleshooting

---

### 9.1 Troubleshooting the Hotspot View

**Symptom:** When sampling on multi-processor machine, the collector reports incorrect [mapping](#) between logical processor number and corresponding processor socket, core and HW thread. Data provided in the **Per Socket** and **Per HW Thread** views is incorrect.

**Solution:** You may provide correct mapping in the `cpumapping.txt` file. The file format is the following:

```
<cpu_num>,<cpu_node>,<socket_num>,<core_num><hw_thread_num>
<cpu_num>,<cpu_node>,<socket_num>,<core_num><hw_thread_num>
...
```

This example of the `cpumapping.txt` file content defines mapping for a system with 8 logical processors:

```
#cpu,node,socket,core.hw_thread
0,0,0,0,0
1,0,0,1,0
2,0,1,0,0
3,0,1,1,0
4,0,2,0,0
5,0,2,1,0
6,0,3,0,0
7,0,3,1,0
```

You can place the `cpumapping.txt` file to the following locations:

- Experiment result directory. First, the sampling viewer searches for the `cpumapping.txt` file in the experiment result directory. It is recommended to place the file to this directory if you analyze data collected on a different system.
- `<install_dir>/bin` directory. If the Intel Performance Tuning Utility doesn't find the `cpumapping.txt` file in the experiment result directory, it searches in the `<install_dir>/bin` directory. It is recommended to place the file to this directory if you collect and analyze data on the same machine.





# Appendix A: Command-line Reference

**NOTE:** To get the most recent command-line reference, enter `vt* -help`.

**Table 1 vtsarun Options**

Primary Options	
<code>--version   -V</code>	Display product version number.
<code>--help   -h   ?</code>	Display help message and exit.
Secondary Options	
<code>--cpu   -c &lt;cpu_name&gt; &lt;cpu_alias&gt;</code>	Specify processor to configure collection for or display events available.
<code>--cpu-list   -cl</code>	Display currently supported processors.
<code>--events-list   -el</code>	Display events to monitor.
<code>--modifiers-list   -ml &lt;event_name&gt;</code>	Display modifiers/constraints available for the specified event.
<code>--start   -s</code>	Enable EBS sampling collection.
<code>--dry-run   -n</code>	Print configuration on the output without running real sampling collection.
<code>--cpu-mask   -cm &lt;cpus&gt;</code>	Specify CPU(s) to collect data on. For example, mask "1-3,7,9-12" means that only CPUs 1 to 3, 7, 9 to 12 are sampled.
<code>--avail-cpu-mask   -am &lt;cpus&gt;</code>	Specify currently available CPU(s) to collect data on. For example, mask "0-1,4" means that only CPUs 0 to 1 and 4 are sampled.
<code>--duration   -d &lt;num&gt;</code>	Specify duration of sampling collection.
<code>--sampling-delay   -sd &lt;delay&gt;</code>	Specify number of seconds to delay sampling while application is being executed. For example, if you set duration to 10 seconds ( <code>-d 10</code> ) and delayed sampling on 5 seconds ( <code>-sd 5</code> ), the collector waits for 5 seconds and then collects data during 10 seconds.
<code>-ec &lt;event parameters&gt;</code>	Specify collection parameters for a particular event. Event parameters syntax: " <code>&lt;event name&gt;</code> "[: <code>&lt;modifier name&gt;</code> = <code>value</code> ]/[ <code>&lt;constraint name&gt;</code> = <code>{[:&lt;modifier name&gt;=value]}</code> ],...
<code>--stop</code>	Stop EBS collection.
<code>--pause</code>	Pause EBS collection.
<code>--resume</code>	Resume EBS collection.
<code>--data-latency   -dl</code>	Enable data latency specific profiling.
<code>--event-multiplexing   -em [-dts &lt;time slice per event group in milliseconds&gt;]</code>	Multiplex events while performing sampling collection.



Table 2 vtsaview Options

Primary Options	
--help   -h	Display help message and exit.
--version   -V	Display product version number.
Secondary Options	
--convert   -c	Put raw data to database.
--summary	Display summary information for the experiment.
--threshold   -t = <value>	Display heaviest hotspots contributing to the experiment in total $\geq$ <value> percent of samples. Threshold <value> is an integer from 1 to 100. Using a threshold enables you to filter out hotspots with minor contributions to the experiment and focus on the hotspots with larger contributions.  For example, set 90% (-t=90) to view the hotspots contributing to at least 90% of the experiment in total.
--granularity   -g = <value>	Define hotspot granularity. Supported values are va, rva, basicblock(b), function(f), source(s), module(m), thread(t), process(p).
--filter   -f = <column>,<string>...	Print data that contain defined substring in the specified <column>. Supported columns are function, thread, process, and module.
--FILTER   -F <column>, <string>,...	Print data that do not contain defined substring in the specified <column>. Supported columns are function, thread, process, and module.
--rows-limit   -n = <value>	Limit the number of output rows to specified value. Default is 100. To see the heaviest hotspots in an experiment result, use the --threshold(-t) option.  <b>Note:</b> This option is ignored when used with --threshold(-t).
--aggregate   -a = <level>	Group the output by <level>. Supported values are thread(t), process(p), and experiment(e).
--sort   -s = <column>	Sort the output in the descending order. Supported columns are function, thread, process, module, and event name.
--event-ratios   -er	Show event ratios.
--SORT   -S = <column>	Sort the output in the ascending order. Supported columns are function, thread, process, module, and events name.
--show-events-as   -sea = <value>	Specify format of events values. Supported formats are s(samples), t(totals), and p(percents). Default is samples.
--re-convert	Delete the existing database and rebuild it from raw data applying new search directories and binary/symbol files location.
--cpu=<value>	Specify how to display sample count for an



	<p>event distributed per processor. Possible values are:</p> <ul style="list-style-type: none"> <li>total - for all processors in one column (default);</li> <li>each - for each logical processor;</li> <li>per-socket - for each processor socket;</li> <li>per-ht - for each hardware thread;</li> <li>&lt;n1&gt;, &lt;n2&gt;, ... - for specified logical processors;</li> <li>max-each - for maximum value across logical processors.</li> </ul>
<pre>--search-dir=&lt;value&gt; where &lt;value&gt; is &lt;all   bin   src   sym[:&lt;p   r&gt;]&gt;=&lt;directory&gt;</pre>	<p>In &lt;directory&gt;, search files of the specified category: binary files (bin), source files (src), symbol files (sym), or all above (all). Additional options are the following:</p> <ul style="list-style-type: none"> <li>:r for recursive search;</li> <li>:p for high search priority (for example, lookup by absolute name).</li> </ul>
<pre>--view-overtime=&lt;value&gt;</pre>	View over-time data for separate time ranges of the experiment. <value> is the number of time points dividing the time of the experiment into separate time ranges.
<pre>--view-overtime-mode=&lt;value&gt;</pre>	<p>Specify the over-time display mode. Possible values are:</p> <ul style="list-style-type: none"> <li>rate - display events per second metrics for each time point (default);</li> <li>raw - display the total number of events collected up to the time point from the start of the experiment.</li> </ul> <p>This option works with the <code>view-overtime</code> option only.</p>
<pre>--filter-time=&lt;value&gt;</pre>	<p>Filter output by specified time range. By default, time is displayed in milliseconds. Use <code>ns</code> suffix to specify time in nanoseconds. Option format:</p> <pre>--filter-time start_time,end_time</pre>
<pre>--filter-turbo=&lt;value&gt;</pre>	<p>Filter output by specified turbo factor range. Option format:</p> <pre>--filter-turbo start,end</pre>
<pre>--turbo-histogram</pre>	Display Frequency Multiplier histogram.

**Table 3 vtsadiff Options**

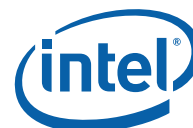
Primary Options	
<pre>--help   -h</pre>	Display help message and exit.
<pre>--version   -V</pre>	Display product version number.
<pre>--granularity   -g &lt;level&gt;</pre>	Define hotspot granularity. Supported values are <code>va</code> , <code>rva</code> , <code>basicblock(b)</code> , <code>function(f)</code> , <code>source(s)</code> , <code>module(m)</code> , <code>thread(t)</code> , <code>process(p)</code> . <code>function</code> is default.
Secondary Options	
<pre>--filter-module &lt;module1&gt;[,module2,...]</pre>	Filter collected data by module name.
<pre>--filter-process &lt;process1&gt;[,process2,...]</pre>	Filter collected data by process name.



<code>--filter-thread &lt;thread1&gt;[,thread2,...]</code>	Filter collected data by thread name.
<code>--filter-function &lt;function1&gt;[,function2,...]</code>	Filter collected data by function name.
<code>--module-dir   -m &lt;dir1&gt;[,dir2,...]</code>	Search modules in the directories list <code>&lt;dir1,dir2,...&gt;</code> .
<code>--csv</code>	Provide output in the comma-separated view format.
<code>--csv-delimiter &lt;delimiter&gt;</code>	Specify a delimiter in the comma-separated view format. <code>\t</code> delimiter is default.
<code>--show-clockticks</code>	Show clockticks as samples. Time in ms is shown by default.
<code>--ignore-module-paths</code>	Ignore module paths.
<code>--sea, --show-events-as=&lt;value&gt;</code>	Show events according to specified <code>&lt;value&gt;</code> : <code>s</code> (samples) or <code>e</code> (events).
<code>--delta=&lt;value&gt;</code>	Display only hotspots for the compared results with the difference exceeding the defined value for at least one event.

**Table 4 vtssrun Options**

Option	Description
<code>--help   -h</code>	Display help message and exit.
<code>--version   -V</code>	Display product version number.
<code>--interval   -i &lt;value&gt;</code>	Set sampling interval in milliseconds. Default <code>&lt;value&gt;</code> is 10 ms.
<code>--notrace   -nt</code>	Do not profile processes by default. To select individual processes, use the <code>-s</code> option.
<code>--signal   -g &lt;signum&gt;</code>	Change the signal for pause/resume. By default, the pause/resume options use system signal 35.
<code>--bind   -b</code>	Resolve dynamically linked symbols at startup. This option can be helpful if application hangs under profiling.
<code>--strategy   -s &lt;name&gt;[HOW]</code>	Specify how to profile a process <code>&lt;name&gt;</code> . <i>HOW</i> can be: : <code>notrace   nt[,children   c]</code> - do not profile the process, but profile its children : <code>trace   t[,children   c]</code> - profile the process and its children (default) : <code>notrace   nt[,nochildren   nc]</code> - do not profile the process and its children : <code>trace   t,nochildren   nc</code> - profile the process but do not profile its children
<code>--log_level   -e &lt;value&gt;</code>	Set logging level. Possible values are: CRITICAL, ERROR, WARNING, INFO, TRACE. CRITICAL is default.
<code>--list   -l</code>	Display profiling status of running processes in an experiment.
<code>--pause   -p [PID [PID...]]</code>	Pause profiling of <code>[PID]</code> or entire experiment if <code>all</code> is specified instead of PID. Use this option to focus data collection on specific code



	sections. If you launch an application with this option, the statistical call graph collector starts the experiment in a paused state.
<code>--resume   -r [PID [PID...]]</code>	Resume profiling of <code>[PID]</code> or entire experiment if <code>all</code> is specified instead of PID.
<code>--stop   -s</code>	Stop data collection and terminate profiled applications.
<code>--duration   -d</code>	Set duration of profiling collection in seconds.
<code>--sampling-delay   -sd &lt;seconds&gt;</code>	Specify the number of seconds to delay sampling while the application is executing. Default is 0 seconds.
<b>Windows-specific Options</b>	
<code>--attach   -a &lt;PID&gt;</code>	Attach to process with the specified PID.
<code>--detach   -dt &lt;PID&gt;</code>	Detach process with the specified PID. Use <code>-dt all</code> command to detach from all processes
<code>--event-config   -ec</code>	See the same option for <a href="#">vtsarun</a> . Note that only one event can be provided to statistical call graph collector.

**Table 5 vtssview Options**

<b>Primary Options</b>	
<code>--help   -h</code>	Display help message and exit.
<code>--version   -V</code>	Display product version number.
<code>--convert   -c</code>	Convert raw data collection files to a single database file.
<code>--flat-profile   -p</code>	Open Flat Profile view.
<code>--graph   -g</code>	Open Call Graph view.
<code>--hot-stack   -t</code>	Open Hot Stack view.
<code>--aggregate   -a=&lt;level&gt;</code>	Aggregate the output data by <code>&lt;level&gt;</code> : t - by thread; p - by process; e - by experiment.
<code>--get-by-id &lt;ID&gt;</code>	Get data by defined <code>&lt;ID&gt;</code> .
<code>--print-debug-info</code>	Display debug information.
<code>--csv</code>	Print all data in CSV format.
<code>--loops   -l</code>	Display loops.
<b>Secondary Options</b>	
<code>--rows-limit   -n &lt;number&gt;</code>	Display top <code>&lt;number&gt;</code> of hotspot functions.
<code>--sort   -s &lt;column name&gt;</code>	Sort the table by <code>&lt;column name&gt;</code> . By default, columns are sorted in the descending order. Supported columns are total, self, function, thread, process, and module.
<code>--SORT   -S=&lt;column name&gt;</code>	Sort the Flat Profile or Hot Stack views in the ascending order by <code>&lt;column name&gt;</code> . Supported columns are total, self, function, thread, process, and module.
<code>--function-decoration   -m (mangled m)   (demangled d)</code>	Choose the function decoration form: mangled or demangled.



<code>--filter   -f &lt;column name&gt;,&lt;substring&gt;[,&lt;column name&gt;,&lt;substring&gt;[, ... ]]</code>	Display the data for functions that contain a defined substring in the specified <i>&lt;column name&gt;</i> . Supported columns are function, thread, process, and module.
<code>--FILTER   -F &lt;column name&gt;,&lt;substring&gt;[,&lt;column name&gt;,&lt;substring&gt;[, ... ]]</code>	Display the data for functions that do not contain a defined substring in the specified <i>&lt;column name&gt;</i> . Supported columns are function, thread, process, and module.
<code>--width   -w &lt;column name&gt;,&lt;number&gt;[,&lt;column name&gt;,&lt;number&gt;[,...]]</code>	Set width for <i>&lt;column name&gt;</i> . Supported columns are index, total, self, function, thread, process, module.
<code>--flat-profile-format &lt;column name&gt;:&lt;width&gt;:&lt;column name&gt;:&lt;width&gt;:...</code>	Set Flat Profile view columns order and width. <i>&lt;width&gt;</i> can be <i>&lt;number&gt;</i> or a combination of <i>&lt;number&gt;.&lt;align&gt;</i> . For example: function:15:process:15: module:20:total:10:self:10.right
<code>--cg-view-format &lt;column name&gt;:&lt;width&gt;:&lt;column name&gt;:&lt;width&gt;:...</code>	Set Call Graph view columns order and width. Supported columns are function, thread, process, and module.
<code>--hotstack-view-format &lt;column name&gt;:&lt;width&gt;:&lt;column name&gt;:&lt;width&gt;:...</code>	Set Hot Stack view columns order and width. Supported columns are function, thread, process, and module.
<code>--re-convert</code>	Delete the existing database and rebuild it from raw data applying new search directories and binary/symbol files location.

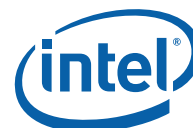
**Table 6 vtcgrun Options**

Option	Description
<code>--ignore   -i &lt;file&gt;</code>	Do not profile applications listed in the <i>&lt;file&gt;</i> . Use this option to reduce collection overhead and focus on an application of interest only.
<code>--callcount   -cc</code>	Collect data on function calls only.

**Table 7 vtcgview Options**

Option	Description
<code>--help   -h   ?</code>	Display help message and exit.
<code>--version   -v</code>	Display product version number.
<code>--graph   -g</code>	Display call graph view.
<code>--flat-profile   -p</code>	Display or export flat profile view.
<code>--module   -m</code>	Display module view.
<code>--merge-threads   -mt</code>	Merge all threads into one.
<code>--group-by-threads   -bt</code>	Display each thread separately.
<code>--convert</code>	Convert all trace files to a database file in the directory specified with the <i>&lt;experiment_dir&gt;</i> argument.
<code>--re-convert</code>	Delete the existing database and rebuild it from raw data applying new search directories and





	binary/symbol files location.
<code>--rows-limit   -n &lt;value&gt;</code>	Limit the number of output rows to specified <code>&lt;value&gt;</code> .
<code>--sort-asc   -s</code>	Specify columns for sorting in the ascending order.
<code>--sort-desc   -S</code>	Specify columns for sorting in the descending order.
<code>--verbose   -V</code>	Print additional information while collecting data.
<code>--quiet   -Q</code>	Print as little information on data collection as possible.
<code>--flat-profile-format</code>	Set the flat profile view columns width and adjustment. Possible columns are: total, self, calls, name, thread, module, process, pid, rva, number-of-parents, number-of-callees. For example: name:30.left:self:20.left:total:30.left:calls:15.left:module:20.left:process:30.left:pid:10.center
<code>--cg-view-format</code>	Set the call graph view columns width and adjustment. Possible columns are: index, total, self, calls, name, thread, module, process, pid, rva, number-of-parents, number-of-callees. For example: index:10.center:total:20.left:self:20.left:name:30.left:thread:10.center:calls:10.center:module:30.left
<code>--module-view-format</code>	Set the module view columns width and adjustment.
<code>-C   --config-file</code>	Specify configuration file. By default, it is taken from the current directory.
<code>--csv</code>	Output in csv format.
<code>--csv-delimiter</code>	Specify csv delimiter.

Table 8 vthprun Options

Option	Description
<code>--trace   -t=no   yes</code>	Set <code>yes</code> to collect trace of all allocations and releases with time stamps. Set <code>no</code> to compress information on all allocations during the whole application run. Trace enables analyzing a period of time and identifying logical memory leaks. <b>NOTE:</b> Trace data may generate large result files.
<code>--ignore   -i=&lt;file&gt;</code>	Do not profile applications listed in <code>&lt;file&gt;</code> and reduce overhead due to concentrating only on applications of interest.
<code>--trace-children   -c=no yes</code>	Set <code>yes</code> to profile all child applications. Set <code>no</code> (default) to focus on the first executed application only.
<code>--exact   -e=no yes</code>	Set <code>yes</code> to enable precise profiling on IA-32



	systems. Set <code>no</code> to enable the fast mode profiling and reduce overhead of data collection.
<code>--allocators   -a=&lt;file&gt;</code>	Profile memory management functions defined in the file <code>&lt;file&gt;</code> . Exact mode only.

**Table 9 vthview Options**

Primary Options	
<code>--spacetime</code>	Show time weighting allocated memory.
<code>--footprint</code>	Show the heap usage over the time.
<code>--allocgraph</code>	Show memory allocation for all the children of a function.
<code>--functions</code>	Show allocation data on the number/size of allocated objects and memory in every function.
<code>--memleaks</code>	Show allocated memory not freed before the end of the application or timestamp specified with the <code>-end</code> option.
<code>--objects</code>	Show information about allocated memory based on block size.
<code>--list</code>	List the results and applications names for the experiment directory.
Secondary Options	
<code>--all-data</code>	Show all data generated by the view. By default, all views show top 10 items sorted by amount of allocated memory.
<code>--begin=&lt;value&gt;</code>	If data contains a trace, display the data subset starting from the specified <code>&lt;value&gt;</code> of time. See the value in the footprint view.
<code>--end=&lt;value&gt;</code>	If data contains a trace, display data up to the specified <code>&lt;value&gt;</code> of time. See the value in the footprint view.
<code>--csv</code>	Provide output in the CSV format.
<code>--csv-delimiter=&lt;symbol&gt;</code>	Specify a non-default delimiter for CSV output. TAB is default.

**Table 10 vtdpview Options**

Options	Description
<code>--help   ?   h</code>	Display help message and exit.
<code>--version   -V</code>	Display product version number.
<code>--convert   -c</code>	Convert raw data collection files to a single database file.
<code>--re-convert</code>	Delete the existing database and rebuild it from raw data applying new search directories and binary/symbol files location.
<code>--threshold   -t &lt;value&gt;</code>	Display output data threshold in percents. 95 is default.



<code>--granularity   -g &lt;value&gt;</code>	Display data according to granularity level <i>&lt;value&gt;</i> . Supported values are <i>process</i> , <i>thread</i> , <i>module</i> , <i>function</i> (default), and <i>cacheline</i> .
<code>--aggregate   -a &lt;level&gt;</code>	Aggregate the output data by <i>&lt;level&gt;</i> : t - by thread p - by process (default) e - by experiment
<code>--column-delimiter   -cd &lt;value&gt;</code>	Specify a delimiter in the comma-separated view format. <code>\t</code> delimiter is default.
<code>--filter   -f &lt;column name&gt;,&lt;substring&gt;[,&lt;column name&gt;,&lt;substring&gt;[, ... ]]</code>	Display the data for functions that contain a defined substring in the specified <i>&lt;column name&gt;</i> . Supported columns are <i>thread</i> , <i>process</i> , and <i>module</i> .
<code>--FILTER   -F &lt;column name&gt;,&lt;substring&gt;[,&lt;column name&gt;,&lt;substring&gt;[, ... ]]</code>	Display the data for functions that do not contain a defined substring in the specified <i>&lt;column name&gt;</i> . Supported columns are <i>thread</i> , <i>process</i> , and <i>module</i> .
<code>--cacheline &lt;value&gt;</code>	Filter output by the specified cachelines.
<code>--latency &lt;value&gt;</code>	Filter output by the specified min/max latency.
<code>--data-address &lt;value&gt;</code>	Filter output by the specified address range.
<code>--rows-limit   -n &lt;value&gt;</code>	Limit the number of output rows to the specified value. 100 is default.

## Appendix C: Glossary

---

### EBS (Event-based Sampling)

Sampling collection when program is interrupted based on PMU event counter overflow.

### Basic Block

A straight-line piece of code that starts with a jump target and ends with a jump. Basic blocks form nodes in the Control Flow Graph.

### Bottleneck

A bottleneck is an area in your code that consumes a visible fraction of the whole application execution time and potentially could be optimized to take less time. Bottlenecks appear as [hotspots](#) in the [Hotspot view](#). All bottlenecks are hotspots but all hotspots need not necessarily be bottlenecks.

### Clockticks

The event that initiates time-based sampling by setting the counters to count the processor's clockticks. Processor clocktick is the smallest unit of time recognized by



the processor. The term is also used to indicate the time required by the processor to execute an instruction.

### **Experiment Result**

A file system directory that contains the results of a data collection session. In the **Tuning Navigator** pane, Experiment Result nodes appear under Project nodes. When created from GUI, Experiment Result directory also contains configuration details about how this Experiment was created. It enables you to repeat a particular experiment from GUI.

### **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.

### **Instructions Retired**

The event that indicates the number of instructions that retired or executed completely. This does not include partially processed instructions executed due to branch mispredictions.

### **PMU (Performance Measurement Unit)**

Special unit on CPU that is responsible for measuring various events occurring on the CPU during program execution.

### **Precise Events**

Precise events are events for which the exact instruction addresses that caused the event are available. Some events for the Pentium(R) 4 processor are precise events.

### **Profiling Configuration**

Set of data collection settings that can be applied to a Project to get Experiment Results.

### **Project**

Includes workload specification that contains information on how an application is executed. Eventually, Project contains one or more Experiment Results.

### **Recursion cycle**



Recursion is a programming method in which a function calls itself or another function that directly or indirectly calls the original function. Recursion cycle is a function sequence that is repeated several times.

**Reference frequency**

A constant frequency equal to the frequency of the time-stamp counter read by the RDTSC instruction. The RDTSC change between sequential interrupts for the core frequency on a single processor is used for calculating the [turbo factor](#) value. Starting from the Intel® Core™ i7 processor family, the actual operating frequency of a core (on a single processor) can differ from the reference frequency, depending on performance requirements and energy consumption level.

**Sample After Value**

The frequency or the number of events after which the Intel Performance Tuning Utility interrupts the processor to collect a sample during EBS.

**Self time**

Time (microseconds) spent inside a function, including time spent waiting between execution activities. It does not include time spent in calls to other instrumented functions.

**TBS (Time-based Sampling)**

Sampling collection when program is interrupted based on the OS timer.

**Total time**

Time (microsecond) elapsed between the time a function starts execution until the time it terminates execution. This is the sum of this function's Self Time and all its callees Total Time.

**Turbo Factor**

A ratio between the actual operating frequency and the reference CPU frequency. If the turbo factor value is below 1, the available CPUs are running slower than the reference frequency. If it is above 1, the CPUs are running faster than the reference frequency. Turbo factor data is useful for processors with Intel® Turbo Boost Technology that can adjust the operating frequency of the processor cores depending on performance requirements and energy consumption level. This feature is available starting from the Intel® Core™ i7 processor family.