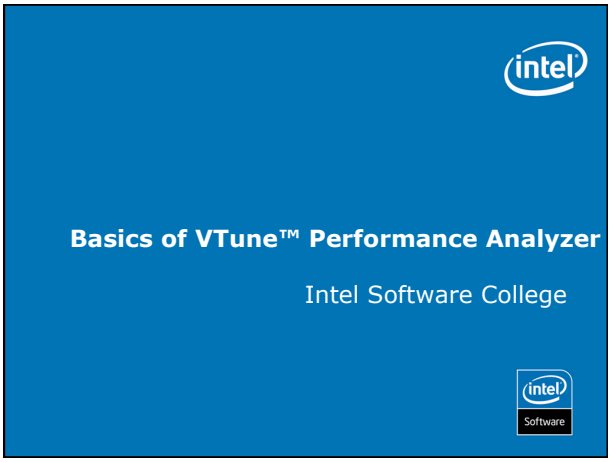


Objectives

At the completion of this module, you will be able to:

- Understand the intended purpose and usage models supported by the VTune™ Performance Analyzer.
- Identify hotspots by drilling down through various sample views.
- Understand how sampling works
- Use callgraph profiling to find hotspots

Agenda

What is the VTune™ Performance Analyzer?
Performance tuning concepts
Using the sampling collector
How sampling works
Sampling Over Time
Call Graph

VTune™ Performance Analyzer

Helps you identify and characterize performance issues by:

- Collecting performance data from the system running your application.
- Organizing and displaying the data in a variety of interactive views, from system-wide down to source code or processor instruction perspective.
- Identifying potential performance issues and suggesting improvements.

Supported Environments

Local and remote data collection

Profile applications that are running on the system that has the analyzer installed on it, or

Run profiling experiments on other systems that are running VTune analyzer remote agents on them

Local Performance Analysis

Intel® IA-32 Processors
- Microsoft Windows* operating systems
- Red Hat Linux*
- SuSE Linux

Itanium® Family Processors
- Microsoft Windows operating systems
- Red Hat Linux
- SuSE Linux

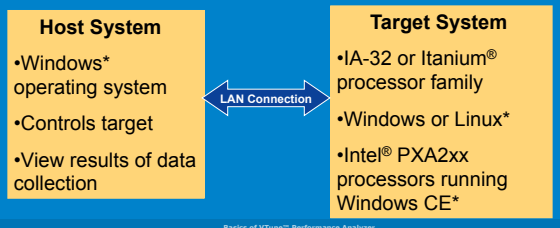
For specific operating systems versions, see the release notes

Host/Target Environment

VTune™ Performance Analyzer supports remote data collection

VTune™ Performance Analyzer installed on host system

Remote agent installed on target system



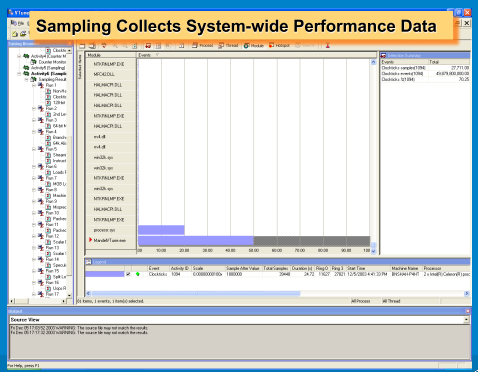
Feature Overview

Sampling

Call graph

VTune™ Analyzer Features and Usage Models

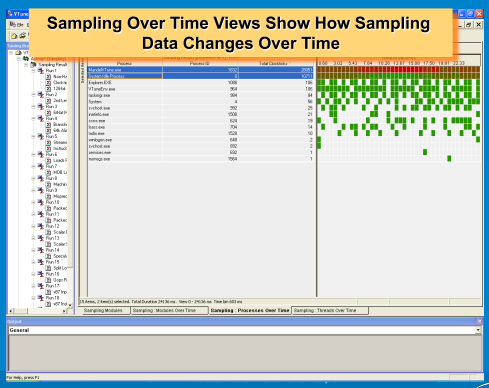
Sampling Collects System-wide Performance Data



9

VTune™ Analyzer Features and Usage Models

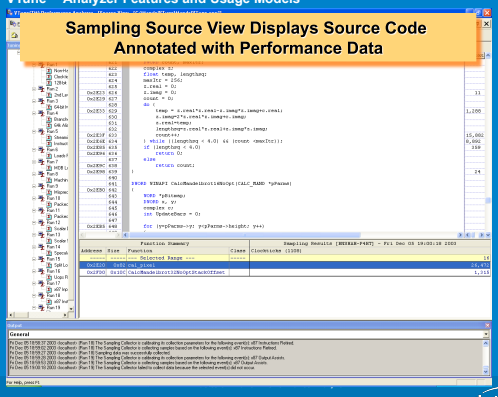
Sampling Over Time Views Show How Sampling Data Changes Over Time



10

VTune™ Analyzer Features and Usage Models

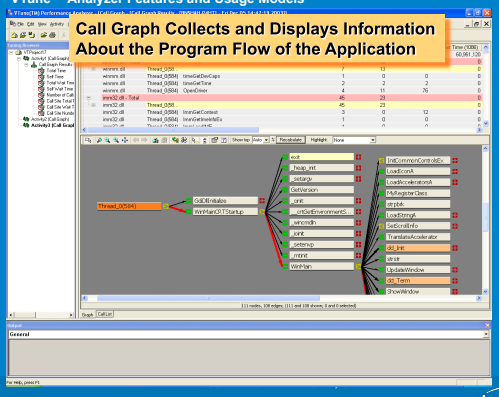
Sampling Source View Displays Source Code Annotated with Performance Data



11

VTune™ Analyzer Features and Usage Models

Call Graph Collects and Displays Information About the Program Flow of the Application



12

What Is a Hotspot?

Where in an application or system there is a significant amount of activity

- Where = address in memory => OS process => OS thread => executable file or module => user function (requires symbols) => line of source code (requires symbols with line numbers) or processor (assembly) instruction
- Significant = activity that occurs infrequently probably does not have much impact on system performance
- Activity = time spent or other internal processor event
  - Examples of other events: Cache misses, branch mispredictions, floating-point instructions retired, partial register stalls, and so on.

Sampling: The Statistical Method of Finding Hotspots

The sampling collector

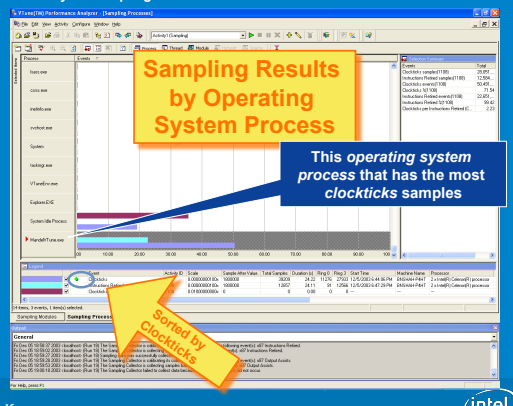
- Periodically interrupts the processor
  - Time-based
  - Event-based: Triggered by the occurrence of a certain number of microarchitectural events
- Collects the execution context
  - Execution address in memory (CS:IP)
  - Operating system process and thread ID
  - Executable module loaded at that address
    - If you have symbols for the module, post-processing can identify the function or method at the memory address.
    - Line numbers from the symbol file can direct you to the relevant line of source code.

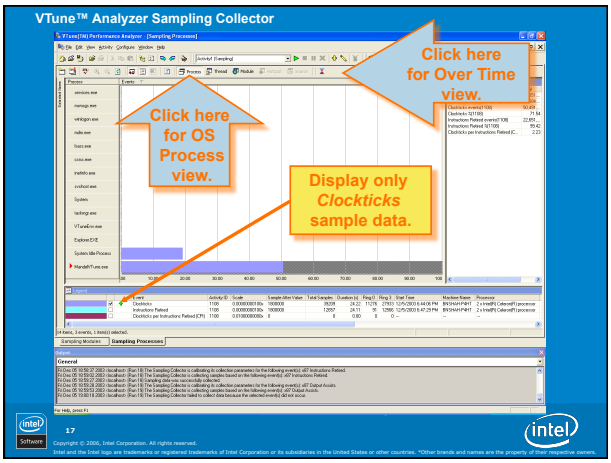
Sampling Collector

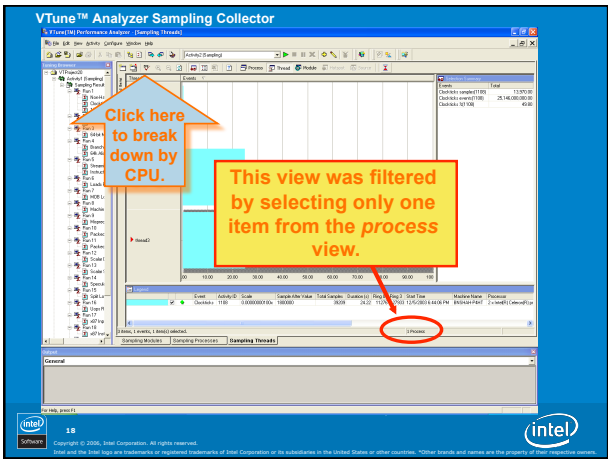
Periodically interrupt the processor to obtain the execution context

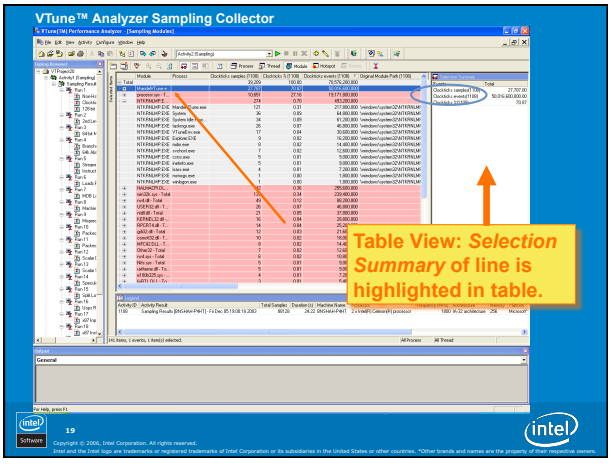
- Time-based sampling (TBS) is triggered by:
  - Operating system timer services
  - Every n processor clockticks
- Event-based sampling (EBS) is triggered by processor event counter overflow
  - These events are processor-specific, like L2 cache misses, branch mispredictions, floating-point instructions retired, and so on

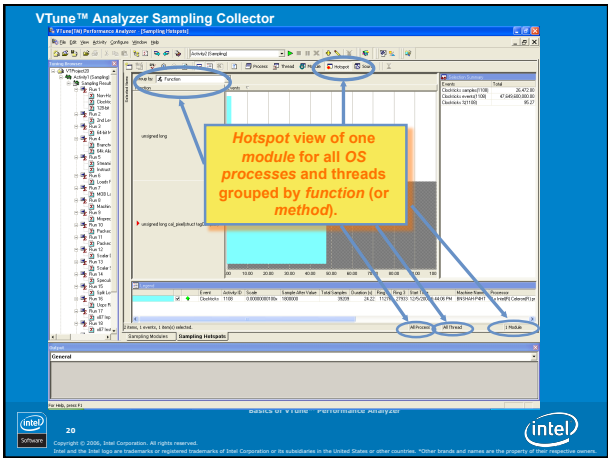
VTune™ Analyzer Sampling Collector



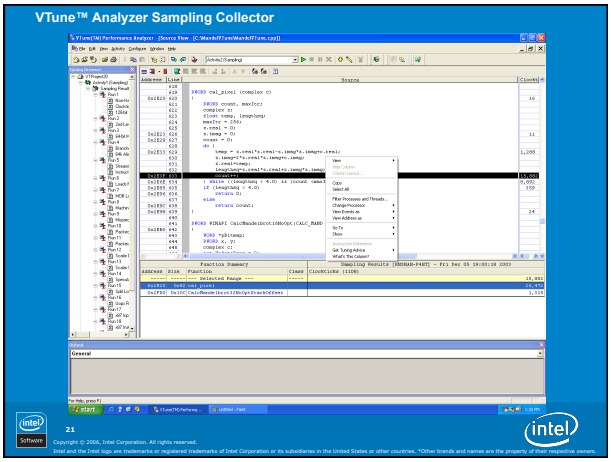

VTune™ Analyzer Sampling Collector
Click here for Over Time view.
Click here for OS Process view.
Display only *Clockticks* sample data.
17


VTune™ Analyzer Sampling Collector
Click here to break down by CPU.
This view was filtered by selecting only one item from the *process* view.
18


VTune™ Analyzer Sampling Collector
Table View: *Selection Summary* of line is highlighted in table.
19

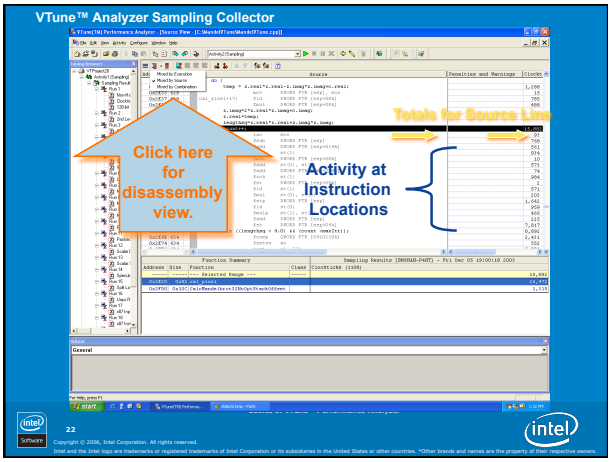

VTune™ Analyzer Sampling Collector
Hotspot view of one *module* for all *OS processes* and threads grouped by *function* (or *method*).
20

VTune™ Analyzer Sampling Collector



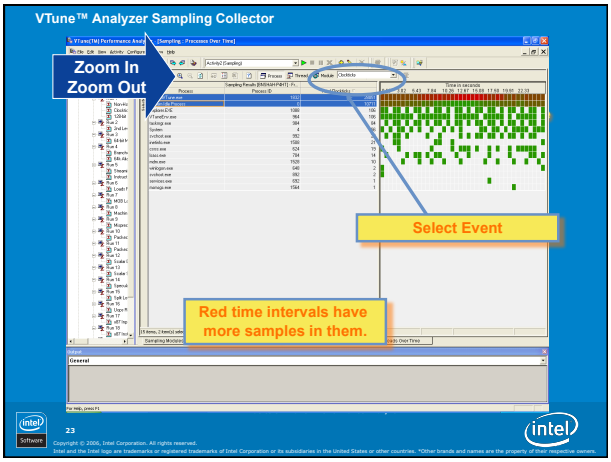
21

VTune™ Analyzer Sampling Collector



22

VTune™ Analyzer Sampling Collector



23

Activity 1: Find the Hotspot

Learn how to identify hotspots with the VTune™ analyzer.

Basics of VTune™ Performance Analyzer

24

Three Key Benefits of Sampling

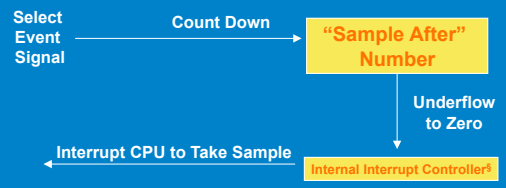
You do not have to modify your code.
- But DO compile/link with symbols and line numbers.
- But DO make release builds with optimizations.

Sampling is system-wide.
- Not just YOUR application.
- You can see activity in operating system code, including drivers.

Sampling overhead is very low.
- Validity is highest when perturbation is low.
- Overhead can be reduced further by turning off progress meters in the user interface.

How else can you reduce sampling overhead?

How Event-based Sampling (EBS) Works Conceptual Diagram



How do you choose a "Sample After" number?

How Many Samples Are Enough?

One million samples for a five-second run?
- Do you have enough samples for it to be statistically significant?
- How much overhead are you causing?

What if you only get 100 samples?
- Is your sample after number 1?
- Are you getting a good profile?

About 1,000 samples per second is a good balance between significance and overhead

Objective: 1,000 Samples Per Second

What is the sample after value for clockticks?
- Dependent upon CPU clock speed
- ANSWER: CPU clock speed in KHz
  - If CPU clock speed = 1,400,000,000 Hz
  - Sample after 1,400,000 clockticks

What is the sample after value for L2 cache read misses?
- It depends on how often you miss the L2 cache!
  - Circular definition? Is not that what you are trying to determine?
- Make an intelligent guess! Estimate!
  - More or less often than the clockticks?
  - 10 times? 100 times? 1000 times?

Calibration

Sets the sample after value to get a reasonable number of samples.

- ~1000 samples per second per logical CPU

Requires the workload to be run twice

Manual Calibration:

- Uncheck Calibrate Sample After value
  - Found on Advanced Activity Configuration dialog
- Start with default value or an estimate
- Run a test
- Modify the sample after value and re-test
- Try to get about a 1000 samples per second per logical CPU

Sampling Over Time

Shows how sample distributions change over time by process, thread, or module

Zoom in on time regions

Useful for:

- Identifying time-variant performance characteristics
- Understanding thread behavior

Sampling Over Time Usage Model

Collect sampling data

Select items of interest from either the process, thread, or modules view

Click 

Highlight region of interest

Click 

Click  to see process/thread/address histogram for time region

Activity 2: Sampling Over Time

Learn how to use the Sampling Over Time view

Call Graph Profiling

Tracks the function entry and exit points of your code at run time

Uses binary instrumentation

Uses this data to determine program flow, critical functions and call sequences

Not system-wide: Only profiles code in applications call path in Ring 3

What Can You Profile?

Win32 applications

Stand-alone Win32* DLLs

Stand-alone COM+ DLLs

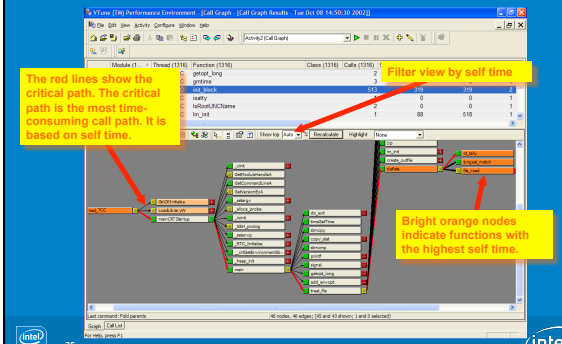
Java applications

.NET* applications

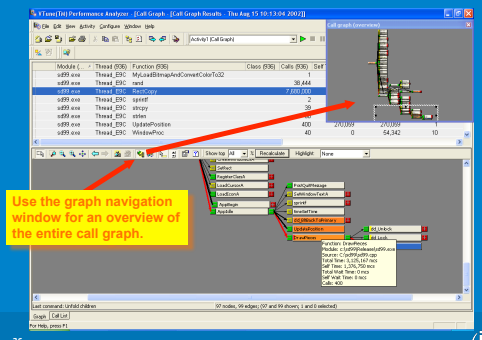
ASP.NET applications

Linux32* applications

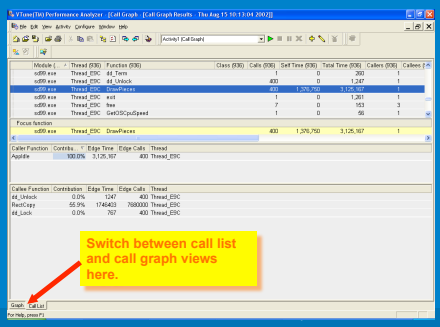
Call Graph View



Call Graph Navigation Window



Call Graph Call List View



Call Graph Metrics

Performance Metric	Description
Self Time	Total time in a function, excluding time spent in its children (includes wait time)
Total Time	Time measured from a function entry to exit point
Total Wait Time	Time spent in a function and its children when the thread is blocked
Wait Time	Time spent in a function when the thread is blocked (excludes blocked time in its children)
Calls	Number of times the function is called

Activity 3: Call Graph

Find the hotspot in gzip using call graph.

Sampling Versus Call Graph

Sampling	Call graph
Low overhead	Higher overhead
System-wide	Ring 3 only on your application call tree
System-wide address histogram	Show function level hierarchy with call counts, times, and the critical path
For function level drill-down, must have debug information	Must re-link with `/fixed:no`, automatically instruments
Can sample based on time and other processor events	Results are based on time

Java* and .NET* Applications

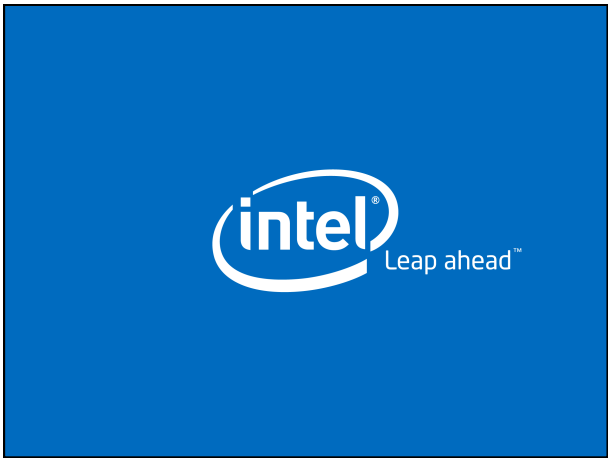
Provides performance data for both managed code and unmanaged code

Gives insight into how managed code calls translate into Win32* calls

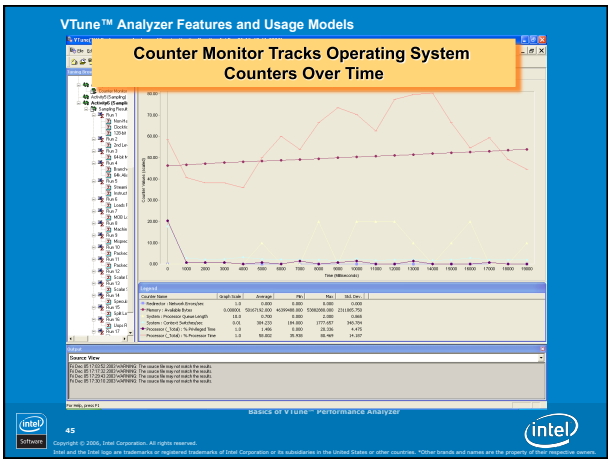
Uses managed code profiling API and binary instrumentation

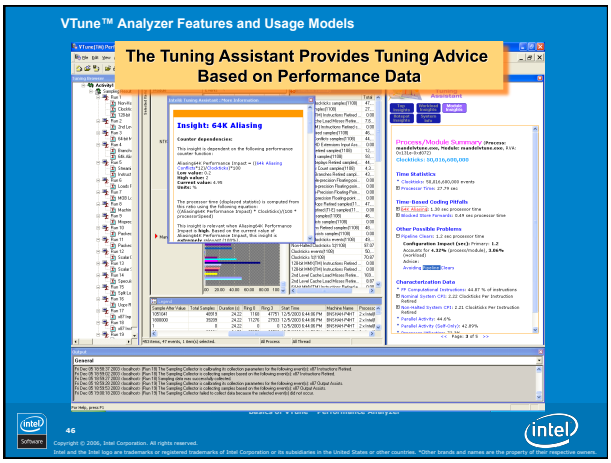
Basics of VTune™ Performance Analyzer What's Been Covered

You can use the different profilers in the VTune™ analyzer to understand the different aspects of the performance of your application.



Extra Slides





Intel® Tuning Assistant

Identifies bottlenecks in:

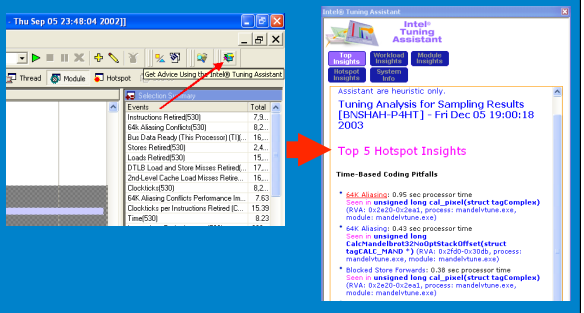
- Pentium® 4, Pentium M®, Itanium® 2, and Pentium® III processors.

Uses EBS and Counter Monitor data.

Shows scaling differences between different runs.

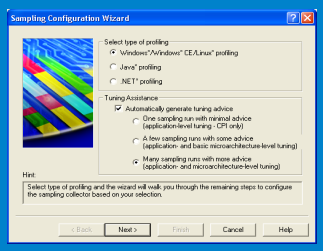
Code Coach is still available but is not enabled by default.

Intel® Tuning Assistant

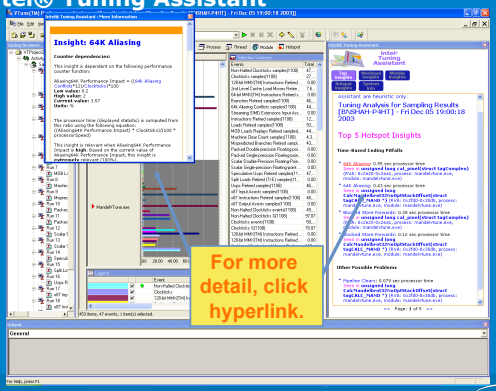


Intel® Tuning Assistant

VTune™ analyzer automatically selects events in the Sampling wizard.



Intel® Tuning Assistant



Lab Activity 3: Getting Tuning Advice

Learn how to get processor-specific tuning advice

Windows* Command Line Interface

Collect sampling data from the command line.

Useful for integrating performance data collection into your automated regression testing.

View the data in the VTune™ Performance Analyzer or export as ASCII text.

Invoke by typing "vtl" at the command line.

Windows* Command Line Interface

Creates hidden project structure

To create an activity: vtl create [activity name] + options

To run an activity: vtl run [activity name]

To view activities type: vtl show

To view results of a particular activity type: vtl view [activityname::result] [options]

To delete the entire project: vtl delete –all

To delete a specific activity: vtl delete <activity name>

Windows* Command Line Interface Examples

Sample on clockticks and instructions retired and launch app matrix.exe:

```
vtl activity a1 –c sampling –app matrix.exe run
```

See the clocktick hotspots in matrix.exe:

```
vtl view a1::r1 –hf –mn matrix.exe
```

See the number of samples in each module system wide:

```
vtl a1::r1 view –modules
```


Windows* Command Line Interface Help

For general command line arguments: `vtl –help`

For sampling command line arguments and events:

```
vtl –help –c sampling
```

For in depth help and examples go to: Start->Programs->Intel® VTune™ Performance Analyzer->Help for the Command Line

Lab Activity 4: Using the Windows* Command Line Interface

Learn how to collect sampling data from the command line

Call Graph Advanced Configuration

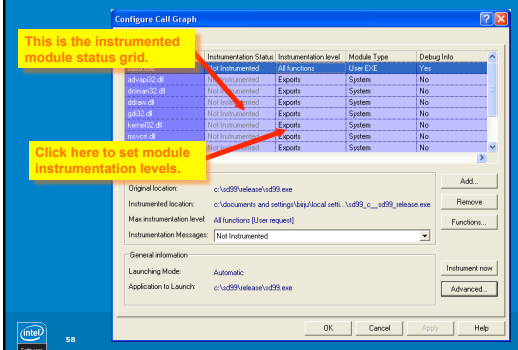
Set instrumentation levels.

- Helps control overhead

Select which functions are instrumented.

- Helps control overhead

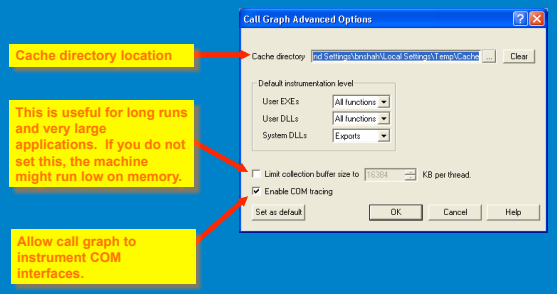
Call Graph Advanced Options



Instrumentation Levels

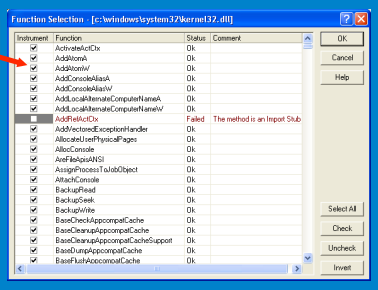
Instrumentation Level	Description	Debug Info Required?
All Functions	Every function in the module is instrumented.	Yes
Custom	You can specify which functions are instrumented	Yes
Export	Every function in the module's export table is instrumented.	No
Minimal	The module is instrumented but no data is collected for it.	No

More Advanced Call Graph Options



Function Selection

Click here to enable or disable instrumentation for a particular function.



Use Sampling and Call Graph Together

Use sampling to find which functions have hotspots.

Use call graph to find out who is calling these functions.

Lab Activity 6: Using Sampling and Call Graph Together

Optimize an application (linpack) using sampling and call graph

Sampling and Call Graph Have Different Hotspots?

Self time includes blocked time.

Event-based sampling (EBS) and time-based sampling (TBS) do not include blocked time in functions (this usually appears in processor.sys).

Hotspots should be the same for self time – wait time (this is non-blocked self time).

What Counter Monitor Does

Collects hardware and software performance counter data

- Windows* Perfmon* counters
- Performance DLL SDK

Correlate counter data with sampling data

Performance DLL SDK

SDK for creating custom performance counters that can be used by counter monitor

Available on the Intel® web site

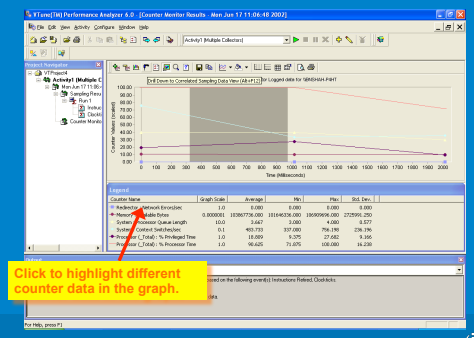
Example: performance counter that measures the transactions per second for a server application

Performance DLL SDK

SDK for creating custom performance counters that can be used by counter monitor

Example: performance counter that measures the transactions per second for a server application

Monitor Window

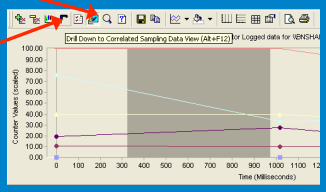


To Correlate Sampling Data

Click the highlight icon and highlight a time slice by dragging over the graph from left to right.

Click on the drill icon.

You should now see the sampling data for that time slice.



Lab Activity 7: Counter Monitor

Use counter monitor to analyze gzip

Trigger API

Allows you to create your own mechanism to programmatically trigger performance counter data collection

Example: collect counter monitor data every time a frame is rendered