Software

INTEL® VTUNE™ AMPLIFIER

Jackson Marusarz - Intel

Agenda

- Introduction to Performance Tuning
- Introduction to Intel VTune Amplifier
- System-Level Profiling
  - HPC Characterization
  - Disk I/O Analysis
- Application Performance Tuning Process
  - Find Hotspots
  - Determine Efficiency
  - Address Parallelism Issues
  - Address Hardware Issues
  - Rebuild and Compare
- Summary

Two Great Ways to Collect Data

Intel® VTune™ Amplifier

Software Collector	Hardware Collector
Uses OS interrupts	Uses the on chip Performance Monitoring Unit (PMU)
Collects from a single process tree	Collect system wide or from a single process tree.
~10ms default resolution	~1ms default resolution (finer granularity - finds small functions)
Either an Intel® or a compatible processor	Requires a genuine Intel® processor for collection
Call stacks show calling sequence	Optionally collect call stacks
Works in virtual environments	Works in a VM only when supported by the VM (e.g., vSphere*, KVM)
No driver required	Requires a driver - Easy to install on Windows - Linux requires root (or use default perf driver)

No special recompiles - C, C++, C#, Fortran, Java, Assembly

A Rich Set of Performance Data

Intel® VTune™ Amplifier

Software Collector	Hardware Collector
Basic Hotspots Which functions use the most time?	**Advanced Hotspots** Which functions use the most time? Where to inline? – Statistical call counts
Concurrency Tune parallelism. Colors show number of cores used.	**General Exploration** Where is the biggest opportunity? Cache misses? Branch mispredictions?
Locks and Waits Tune the #1 cause of slow threaded performance: – waiting with idle cores.	**Advanced Analysis** Memory-access, HPC Characterization, etc…
Any IA86 processor, any VM, no driver	Higher res., lower overhead, system wide

No special recompiles - C, C++, C#, Fortran, Java, Assembly

Example: Hotspots Analysis

Summary View

General Exploration Hotspots viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up

Elapsed Time: 5.554s

CPU Time:	**10.504s**
Instructions Retired:	21,698,000,000
CPI Rate:	1.257
CPU Frequency Ratio:	1.041
Total Thread Count:	9
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
grid_intersect	3_tachyon_omp.exe	5.539s
sphere_intersect	3_tachyon_omp.exe	3.247s
func@0x1002e59d	libiomp5md.dll	0.148s
shader	3_tachyon_omp.exe	0.117s
KeDelayExecutionThread	ntoskrnl.exe	0.091s
[Others]	N/A*	1.361s

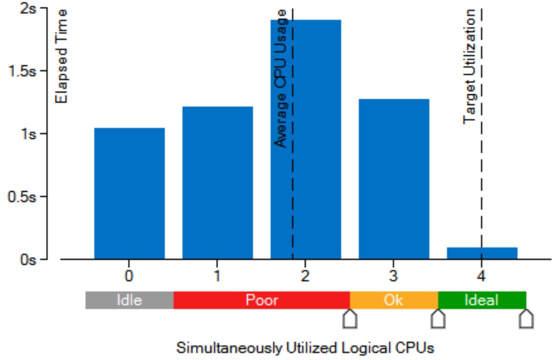
*N/A is applied to non-summable metrics.

Average Bandwidth

Package	Total, GB/sec	Read, GB/sec	Write, GB/sec
package_0	5.715	3.504	2.212

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.

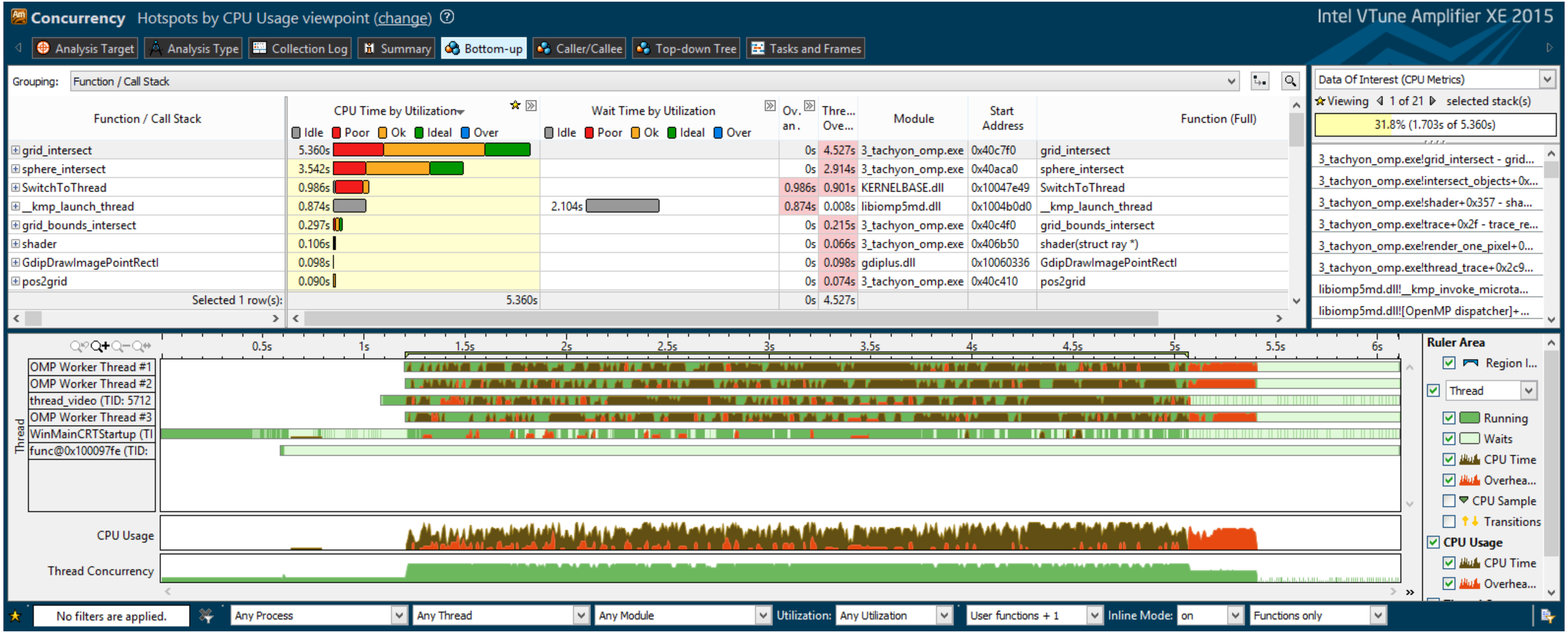


Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Example: Concurrency Analysis

Bottom-up View



Find Answers Fast

Intel® VTune™ Amplifier

Adjust Data Grouping

Function - Call Stack
Module - Function - Call Stack
Source File - Function - Call Stack
Thread - Function - Call Stack

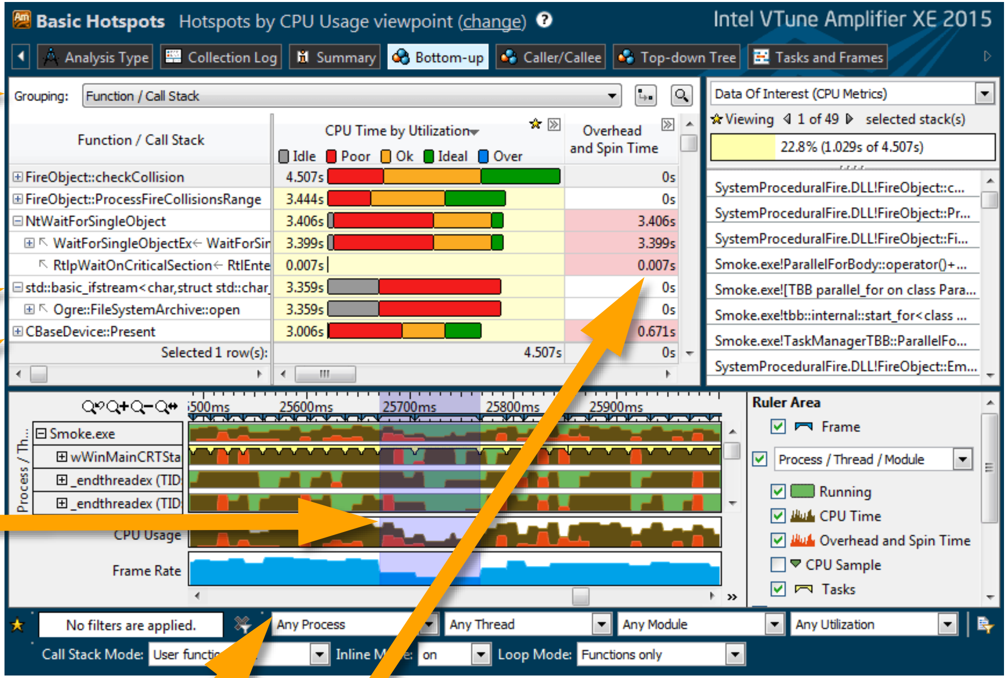
… (Partial list shown)

Double Click Function to View Source

Click [+] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Zoom In And Filter On Selection
Filter In by Selection
Remove All Filters



Filter by Process & Other Controls

Tuning Opportunities Shown in Pink. Hover for Tips

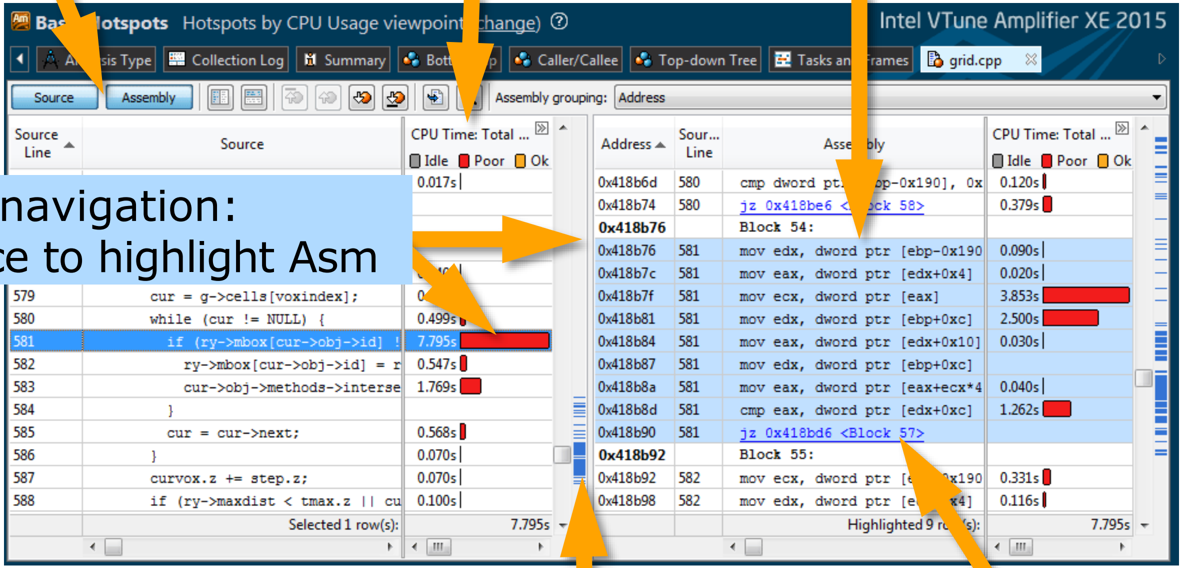
See Profile Data On Source / Asm

Double Click from Grid or Timeline

View Source / Asm or both

CPU Time

Right click for instruction reference manual



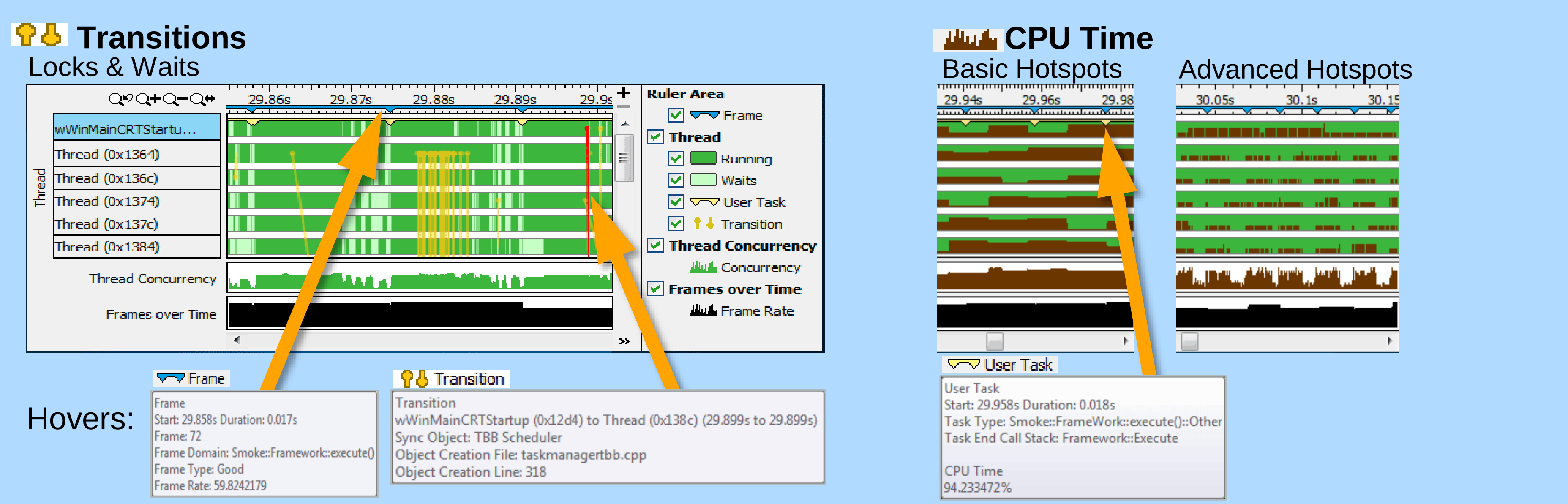
Quick Asm navigation:
Select source to highlight Asm

Scroll Bar “Heat Map” is an overview of hot spots

Click jump to scroll Asm

Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier



Optional: Use API to mark frames and user tasks   Frame   User Task

Optional: Add a mark during collection   Mark Timeline

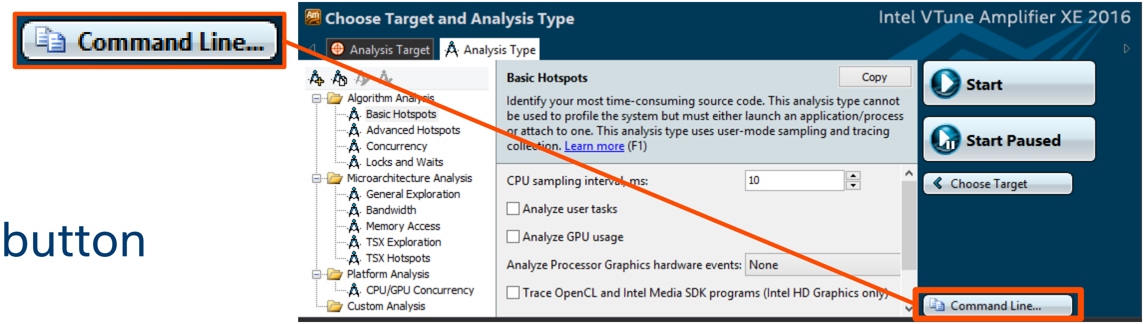
Command Line Interface

Automate analysis

amplxe-cl is the command line:

- **Windows:** `C:\Program Files (x86)\Intel\VTune Amplifier XE\bin[32|64]\amplxe-cl.exe`
- **Linux:** `/opt/intel/vtune_amplifier_xe/bin[32|64]/amplxe-cl`

Help: `amplxe-cl –help`



Use UI to setup

1) Configure analysis in UI
2) Press “Command Line…” button
3) Copy & paste command

Great for regression analysis – send results file to developer
Command line results can also be opened in the UI

Compare Results Quickly - Sort By Difference

Intel® VTune™ Amplifier

Quickly identify cause of regressions.

- Run a command line analysis daily
- Identify the function responsible so you know who to alert

Compare 2 optimizations – What improved?

Compare 2 systems – What didn't speed up as much?

Grouping: Function / Call Stack

Function / Call Stack	CPU Time:Difference	Module	CPU Time:r007hs	CPU Time:r006hs
FireObject::checkCollision	4.850s	SystemProceduralFire.DLL	6.281s	1.431s
FireObject::ProcessFireCollisionsRange	4.644s	SystemProceduralFire.DLL	5.643s	0.999s
dllStopPlugin	3.765s	RenderSystem_Direct3D9.DLL	9.184s	5.419s

Introduction to Performance Tuning

System

Application

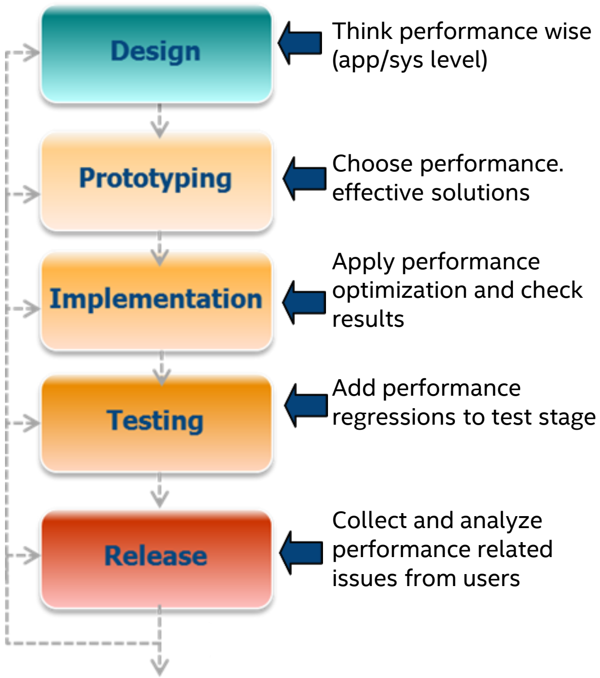
Processor

H/W tuning:
- BIOS (TB, HT)
- Memory
- Network I/O
- Disk I/O

OS tuning:
- Page size
- Swap file
- RAM Disk
- Power settings
- Network protocols

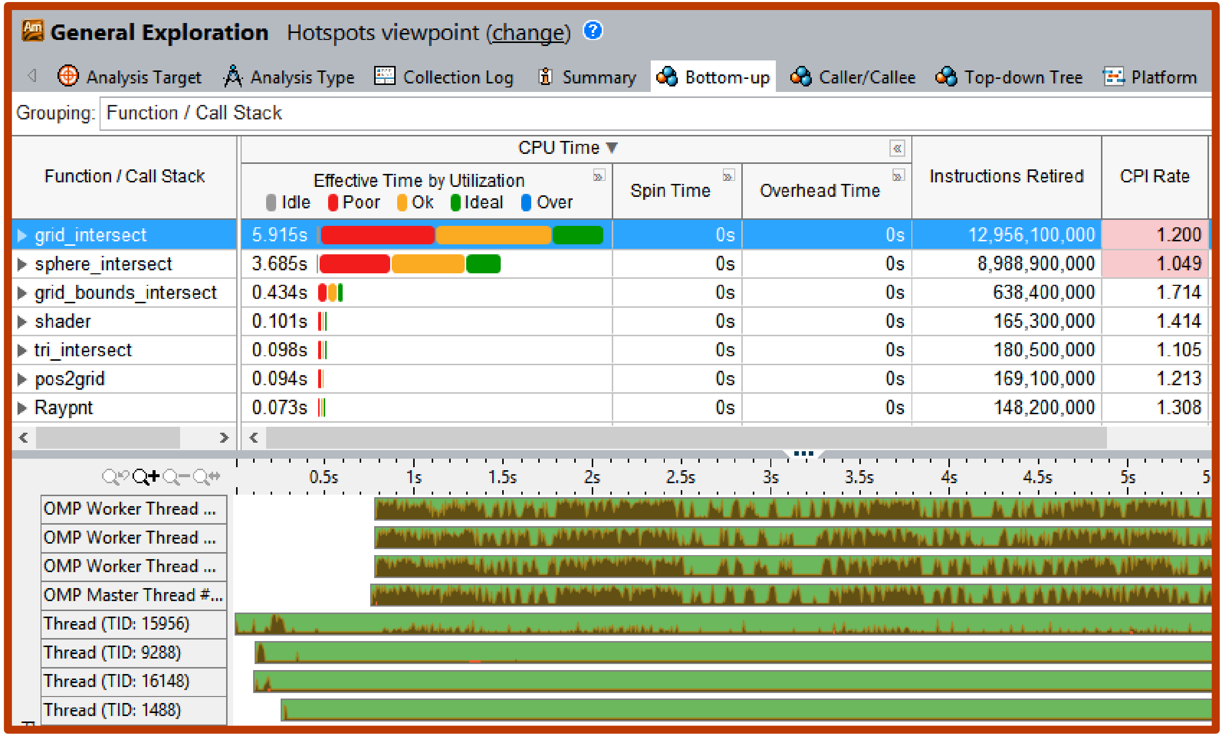
Better application design:
- Parallelization
- Fast algorithms / data bases
- Programming language and RT libs
- Performance libraries
- Driver tuning

Tuning for Microarchitecture:
- Compiler settings/Vectorization
- Memory/Cache usage
- CPU pitfalls



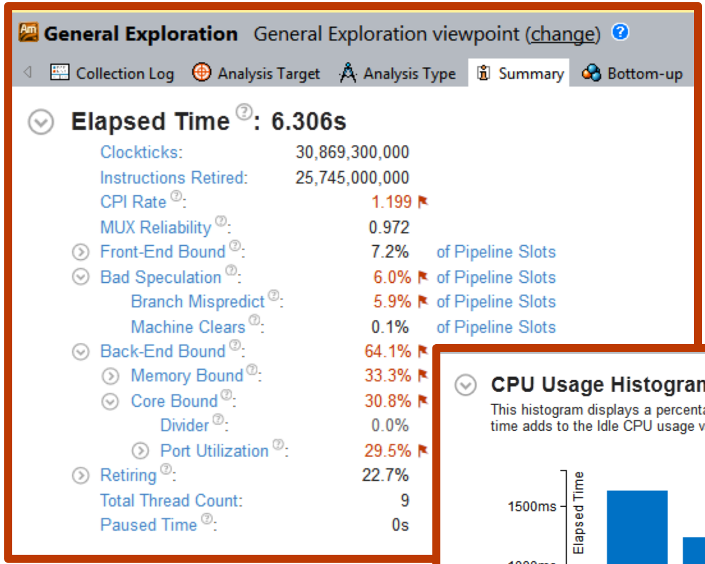
Introduction to Intel VTune Amplifier

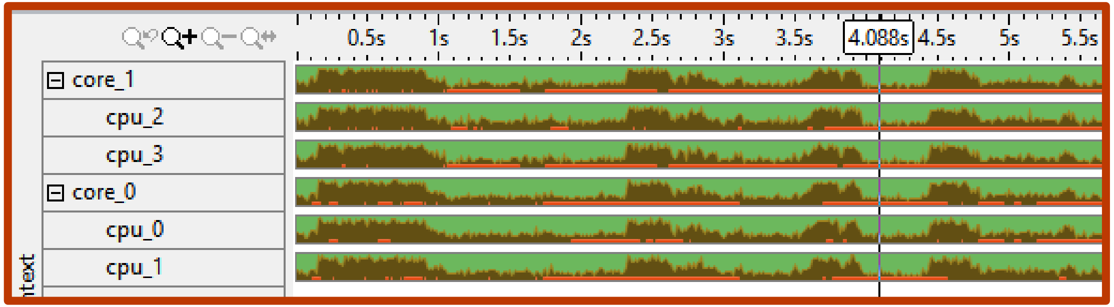
- Accurate Data - Low Overhead
  - CPU, GPU, FPU, threading, bandwidth, and more…
  - Profile applications or systems
- Meaningful Analysis
  - Threading and hardware utilization efficiency
  - Memory and storage device analysis
- Easy
  - Data displayed by source code
  - Expert advice built-in
  - Easy set-up, no special compiles

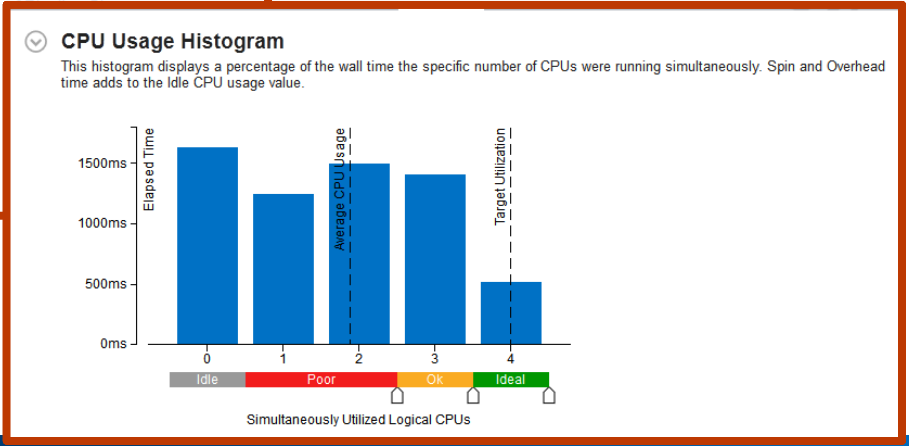



```
> amplxe-cl -help collect
```

System-Level Profiling – High-level Overviews







System-Level Profiling – Process/Module Breakdowns

Hotspots viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

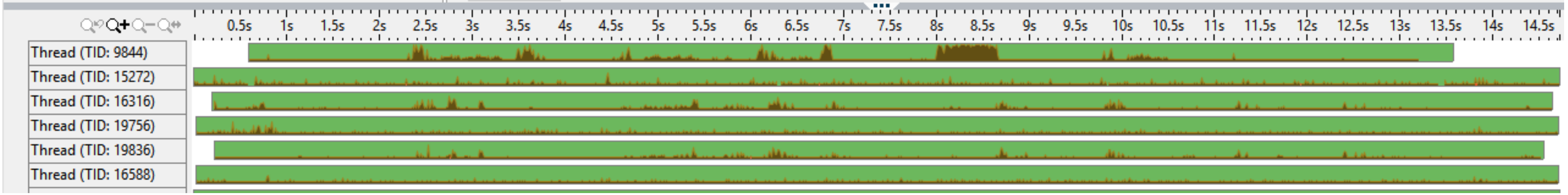
Grouping: Process / Module / Function / Thread / Call Stack

Processes

Modules

Functions

Process / Module / Function / Thread / Call Stack	CPU Time ▼	Instructions Retired	CPI Rate	CPU Frequency Ratio	Module	
▶ Pid 0x544	3.889s	27.5%	0.910	0.965		
▼ chrome.exe	3.443s	15.4%	1.441	0.963		
▶ chrome_child.dll	3.022s	14.6%	1.301	0.944		
▶ ntdll.dll	0.242s	0.6%	3.171	1.103		
▶ ntoskrnl.exe	0.179s	0.2%	7.143	1.064		
▶ EXCEL.EXE	2.750s	14.3%	1.312	1.022		
▶ Explorer.EXE	2.598s	10.3%	1.677	0.998		
▶ Syncplicity.exe	1.140s	4.1%	1.923	1.039		
▼ OUTLOOK.EXE	0.891s	1.5%	3.723	0.918		
▶ mso.dll	0.141s	0.2%	4.719	0.812		
▼ ntoskrnl.exe	0.080s	0.2%	2.884	1.181		
▶ ExEnterPriorityRegionAndAcquireResourceExclusive	0.004s	0.0%		0.400	ntoskrnl.exe	ExEnterPriorityRegionAndAcquireResourceE
▶ ExAllocatePoolWithTag	0.004s	0.0%	1.000	1.000	ntoskrnl.exe	ExAllocatePoolWithTag
▶ KeSetEvent	0.004s	0.0%		0.200	ntoskrnl.exe	KeSetEvent
▶ ObReferenceObjectByHandleWithTag	0.004s	0.0%		0.800	ntoskrnl.exe	ObReferenceObjectByHandleWithTag



System-Level Profiling – Disk I/O Analysis

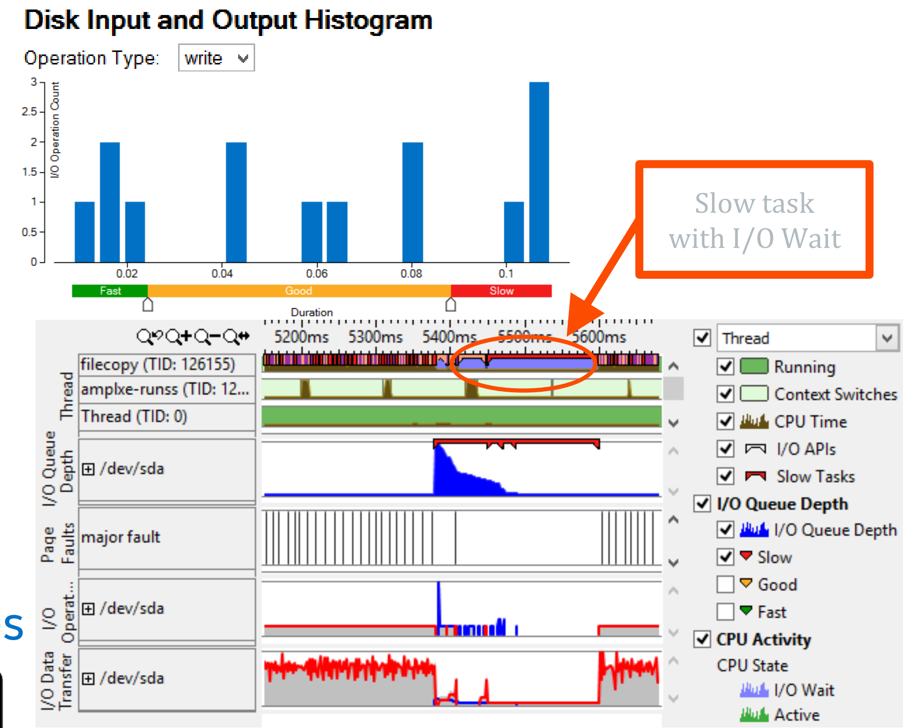
Are You I/O Bound or CPU Bound?

- Explore imbalance between I/O opera (async & sync) and compute
- Storage accesses mapped to the source code

See when CPU is waiting for I/O

- Measure bus bandwidth to storage
- Latency analysis
- Tune storage accesses with latency histogram
- Distribution of I/O over multiple devices

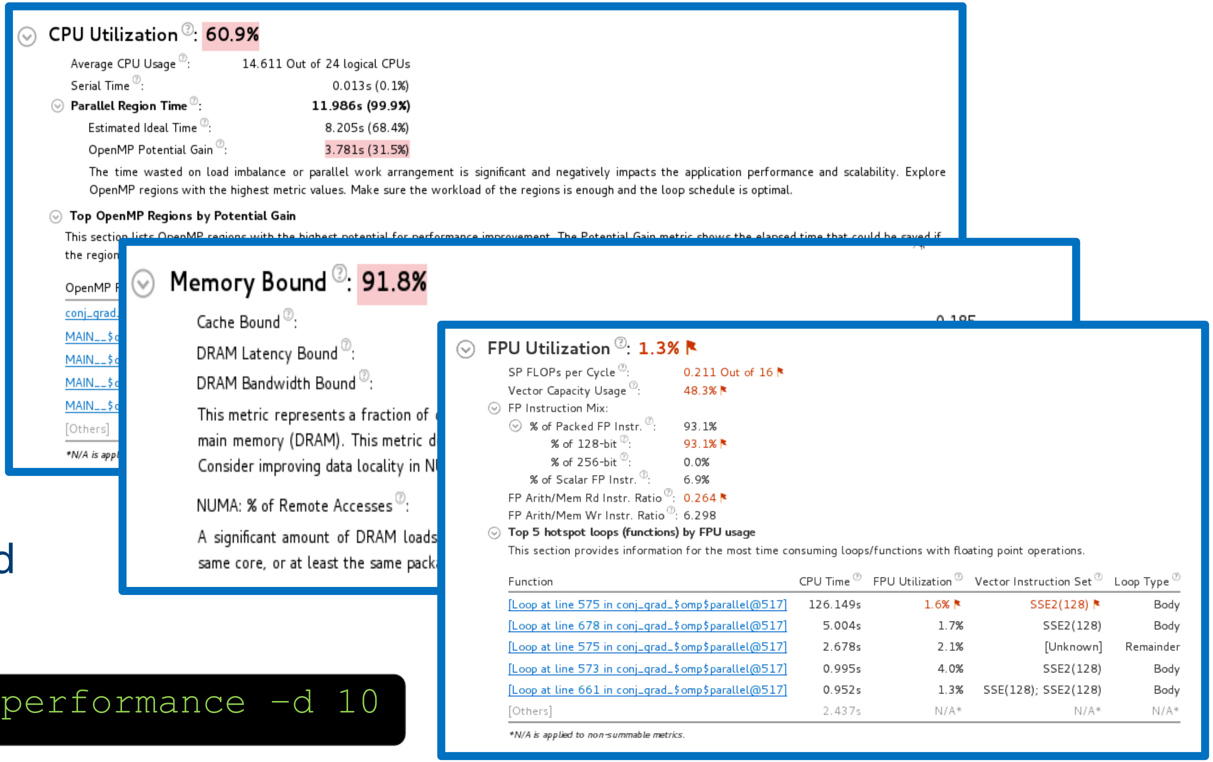
```
> amplxe-cl -collect disk-io -d 10
```



System-Level Profiling – HPC Characterizaton

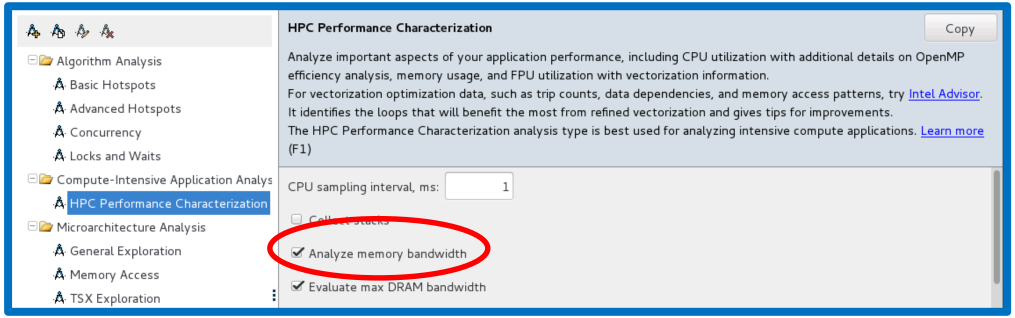
Three Metric Classes

- CPU Utilization
  - Logical core % usage
  - Includes parallelism and OpenMP information
- Memory Bound
  - Break down each level of the memory hierarchy
- FPU Utilization
  - Floating point GFLOPS and density



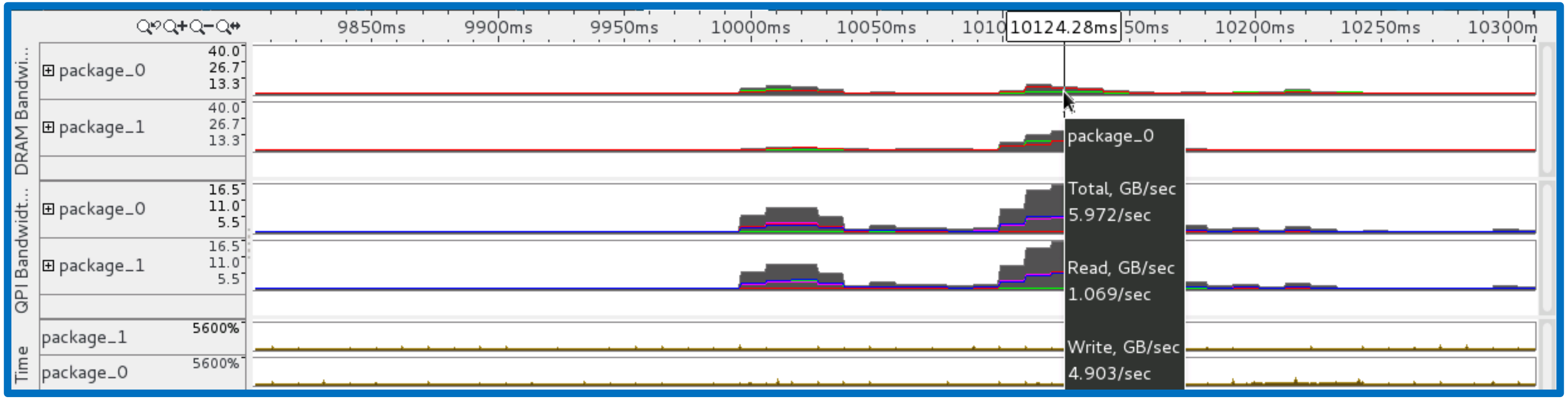

```
> amplxe-cl -collect hpc-performance -d 10
```

System-Level Profiling – Memory Bandwidth

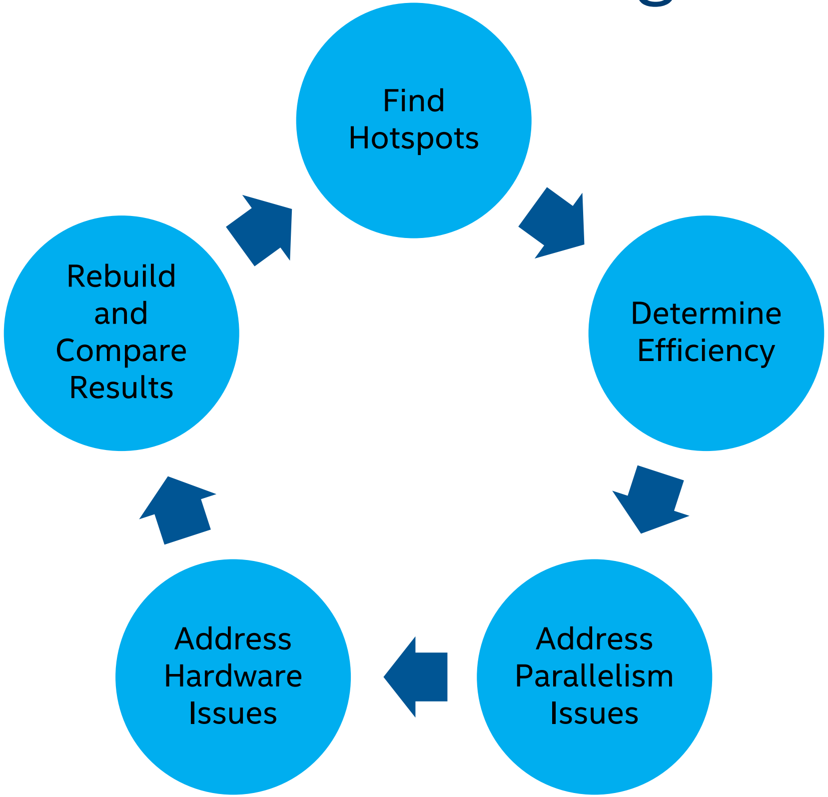


Find areas of high and low bandwidth usage. Compare to max system bandwidth based on Stream benchmarks.

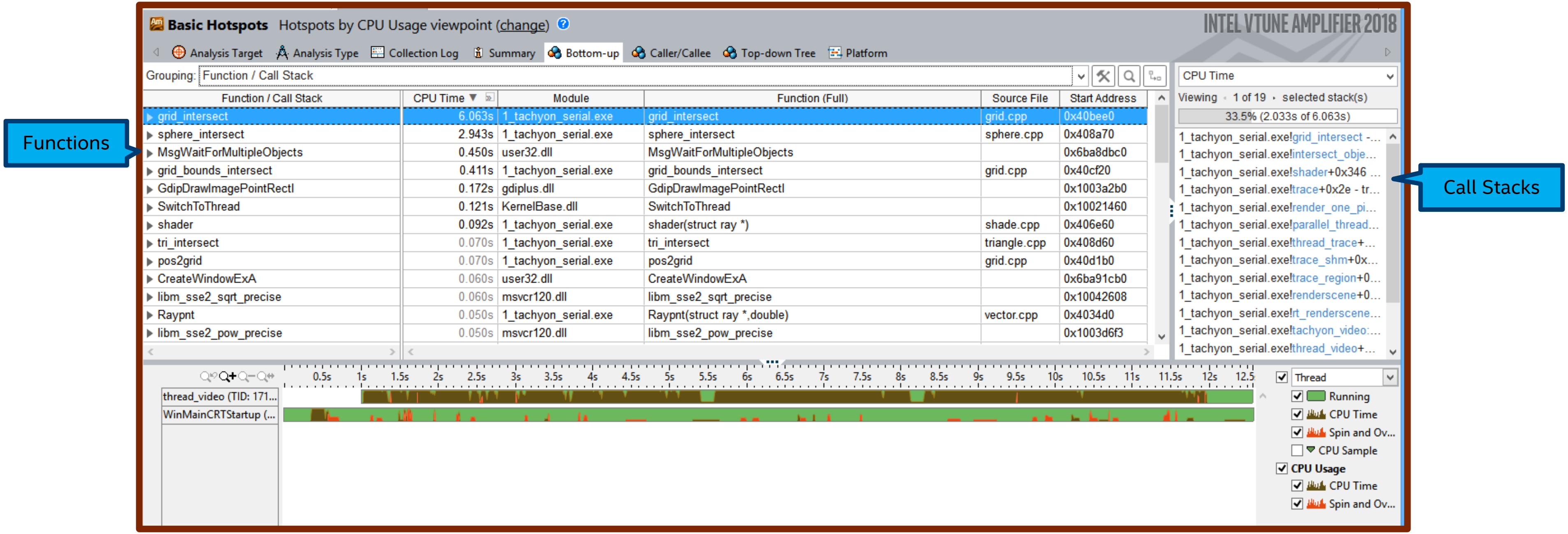
```
-knob collect-memory-bandwidth=true
```



Application Performance Tuning Process



Find Hotspots



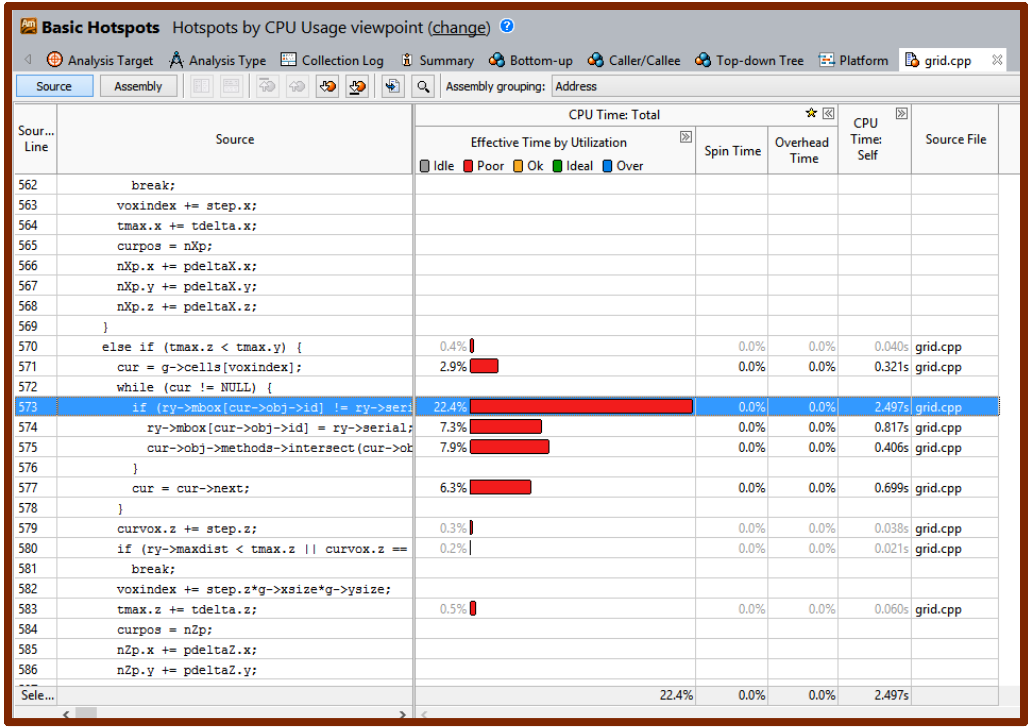
Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
grid_intersect	6.063s	1_tachyon_serial.exe	grid_intersect	grid.cpp	0x40bee0
sphere_intersect	2.943s	1_tachyon_serial.exe	sphere_intersect	sphere.cpp	0x408a70
MsgWaitForMultipleObjects	0.450s	user32.dll	MsgWaitForMultipleObjects		0x6ba8dbc0
grid_bounds_intersect	0.411s	1_tachyon_serial.exe	grid_bounds_intersect	grid.cpp	0x40cf20
GdipDrawImagePointRectI	0.172s	gdiplus.dll	GdipDrawImagePointRectI		0x1003a2b0
SwitchToThread	0.121s	KernelBase.dll	SwitchToThread		0x10021460
shader	0.092s	1_tachyon_serial.exe	shader(struct ray *)	shade.cpp	0x406e60
tri_intersect	0.070s	1_tachyon_serial.exe	tri_intersect	triangle.cpp	0x408d60
pos2grid	0.070s	1_tachyon_serial.exe	pos2grid	grid.cpp	0x40d1b0
CreateWindowExA	0.060s	user32.dll	CreateWindowExA		0x6ba91cb0
libm_sse2_sqrt_precise	0.060s	msvcr120.dll	libm_sse2_sqrt_precise		0x10042608
Raypnt	0.050s	1_tachyon_serial.exe	Raypnt(struct ray *,double)	vector.cpp	0x4034d0
libm_sse2_pow_precise	0.050s	msvcr120.dll	libm_sse2_pow_precise		0x1003d6f3

```
> amplxe-cl -collect basic-hotspots -- ./myapp.out
```

Find Hotspots

- Drill to source or assembly
- Hottest areas easy to ID
- Is this the expected behavior
- Pay special attention to loops and memory accesses

- Learn how your code behaves
- What did the compiler generate
- What are the expensive statements



Determine Efficiency

General Exploration Hotspots viewpoint (change)

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time
grid_intersect	5.915s	0s	0s
sphere_intersect	3.685s	0s	0s
grid_bounds_intersect	0.434s	0s	0s
shader	0.101s	0s	0s
tri_intersect	0.098s	0s	0s
pos2grid	0.094s	0s	0s
Raypnt	0.073s	0s	0s

General Exploration General Exploration viewpoint (change)

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up

Grouping: Function / Call Stack

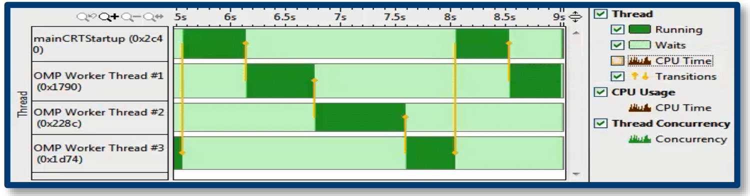
Function / Call Stack	CPI Rate	Retiring	Fro
grid_intersect	1.200	22.5%	
sphere_intersect	1.049	23.9%	
grid_bounds_intersect	1.714	16.5%	
shader	1.414	16.3%	
pos2grid	1.213	50.9%	
tri_intersect	1.105	23.8%	
Raypnt	1.308	39.2%	
func@0x140150ef0	9.714	80.9%	
libm_sse2_sqrt_precise	2.241	0.0%	

Look for Parallelism, Cycles-per-Instruction (CPI), and Retiring %

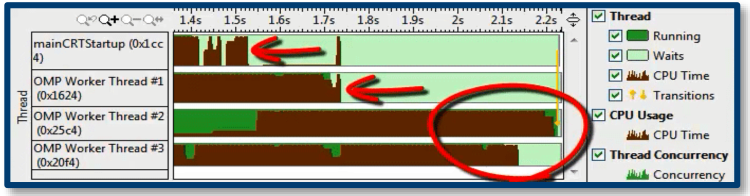
Address Parallelism Issues

- Use Concurrency Analysis to ensure you're using all your threads as often as possible.
- Common concurrency problems can often be diagnosed in the timeline.
- Switch to the Locks And Waits viewpoint or run a Locks and Waits analysis to investigate contention.

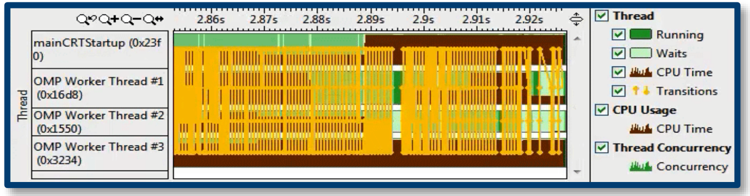
Coarse-Grain Locks



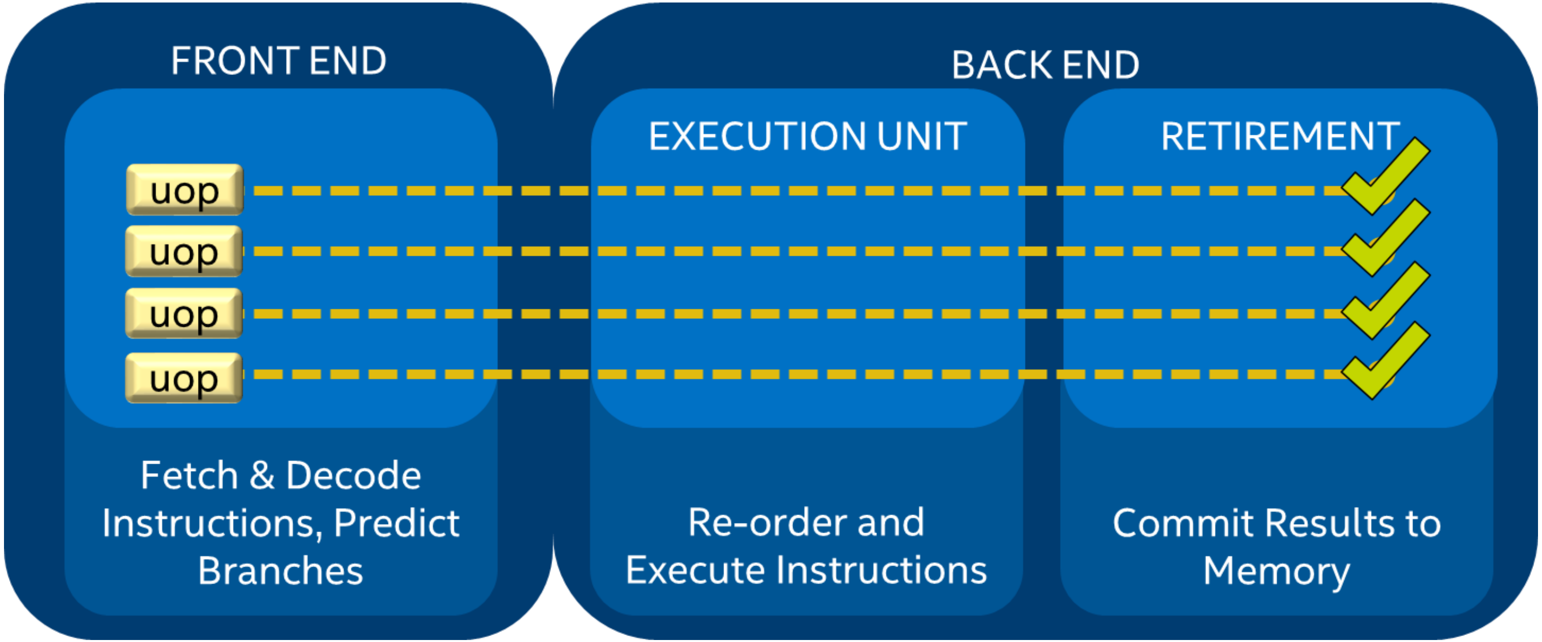
Thread Imbalance



High Lock Contention



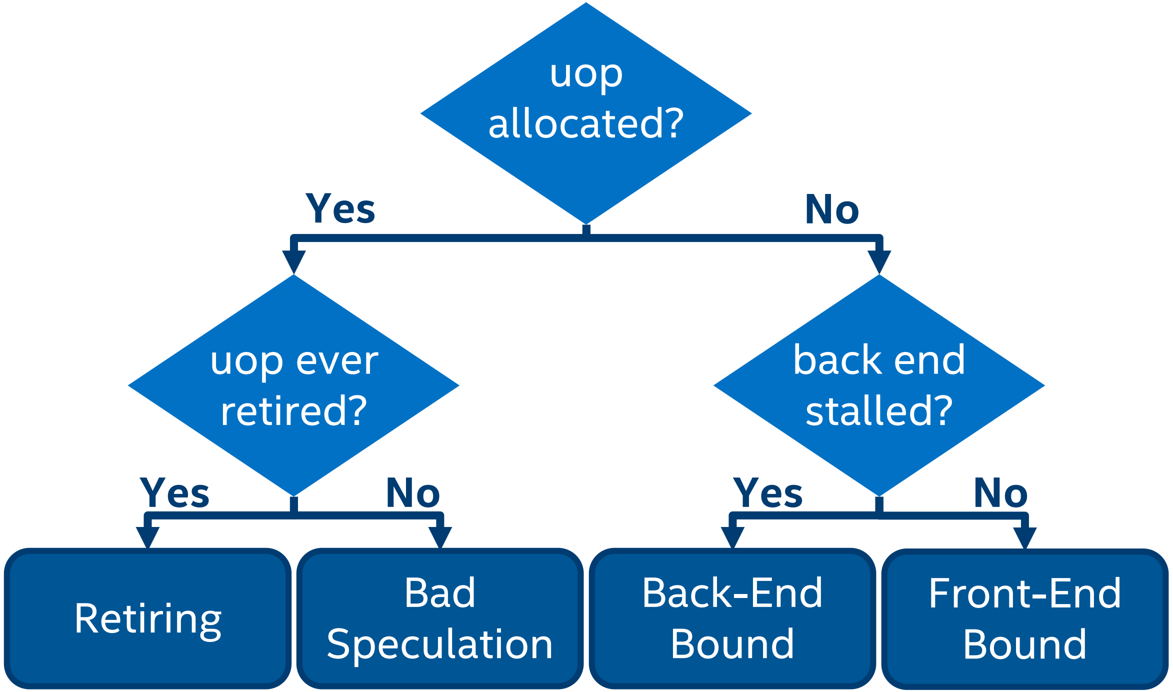
Address Hardware Issues



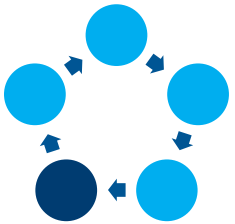
The X86 Processor Pipeline (simplified)

Address Hardware Issues

For each pipeline slot on each cycle:



Address Hardware Issues



General Exploration General Exploration viewpoint (change)

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Event Count | Platform

Grouping: Function / Call Stack

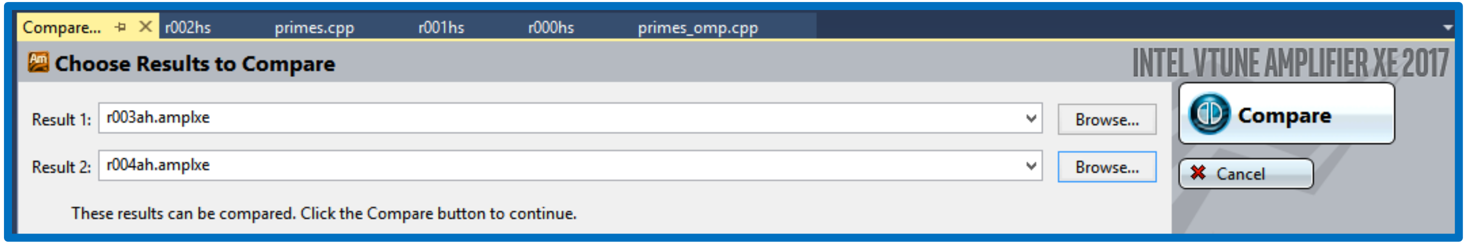
Function / Call Stack	Retiring	Front-End Bound	Bad Speculation	Back-End Bound: Memory Bound	Back-End Bound: Core Bound	Module
grid_intersect	22.5%	6.5%	4.5%	34.6%	31.8%	3_tachyon_omp.exe
sphere_intersect	23.9%	6.2%	11.5%	29.0%	29.4%	3_tachyon_omp.exe
grid_bounds_intersect	16.5%	11.3%	8.7%	31.8%	31.8%	3_tachyon_omp.exe
shader	16.3%	20.3%	4.1%	100.0%	0.0%	3_tachyon_omp.exe
pos2grid	50.9%	4.6%	0.0%	72.2%	0.0%	3_tachyon_omp.exe
tri_intersect	23.8%	14.3%	0.0%			3_tachyon_omp.exe
Raypnt	39.2%	4.9%	0.0%	0.0%	90.2%	3_tachyon_omp.exe
func@0x140150ef0	80.9%	0.0%	0.0%	15.6%	10.9%	ntoskrnl.exe
libm_sse2_sqrt_precise	0.0%	30.8%	38.5%	0.0%	30.8%	msvcr120.dll
aullrem	46.9%	0.0%	0.0%	26.6%	26.6%	libiomp5md.dll
func@0x10013010	41.0%	16.4%	0.0%	0.0%	50.8%	gdiplus.dll
_kmp_linear_barrier_release	33.3%	0.0%	41.7%	7.1%	17.9%	libiomp5md.dll
libm_sse2_pow_precise	0.0%	9.1%	18.2%			msvcr120.dll
ColorScale	30.6%	0.0%	0.0%			3_tachyon_omp.exe
intersect_objects	20.8%	10.4%	0.0%	0.0%	100.0%	3_tachyon_omp.exe
func@0x10009c00	35.7%	23.8%	0.0%	0.0%	64.3%	gdiplus.dll

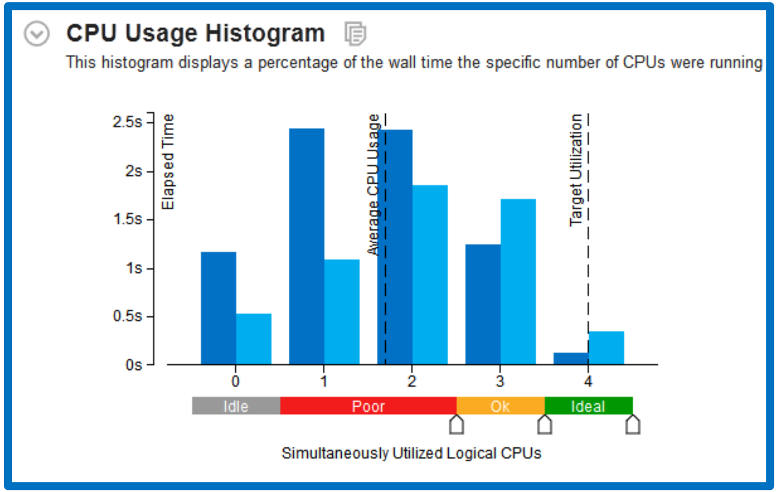
This data is collected statistically with event multiplexing. Gray data has low confidence levels.

General Exploration Analysis Shows the Hardware Bottleneck in the Application

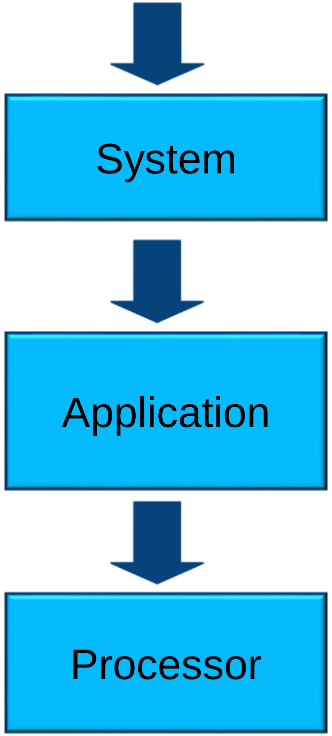
```
> amplxe-cl -collect general-exploration -- ./myapp.out
```

Rebuild and Compare Results

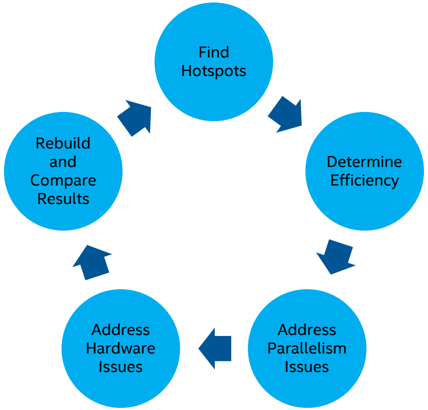




Summary



- Start with the lowest hanging fruit for performance tuning
- Use Intel® VTune™ Amplifier for system and application profiling
- Hotspots, HPC Characterization, and General Exploration are good starting points
- Performance tuning is an iterative process



Legal Disclaimer & Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

intel®

Software