



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 <ul style="list-style-type: none">- 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.

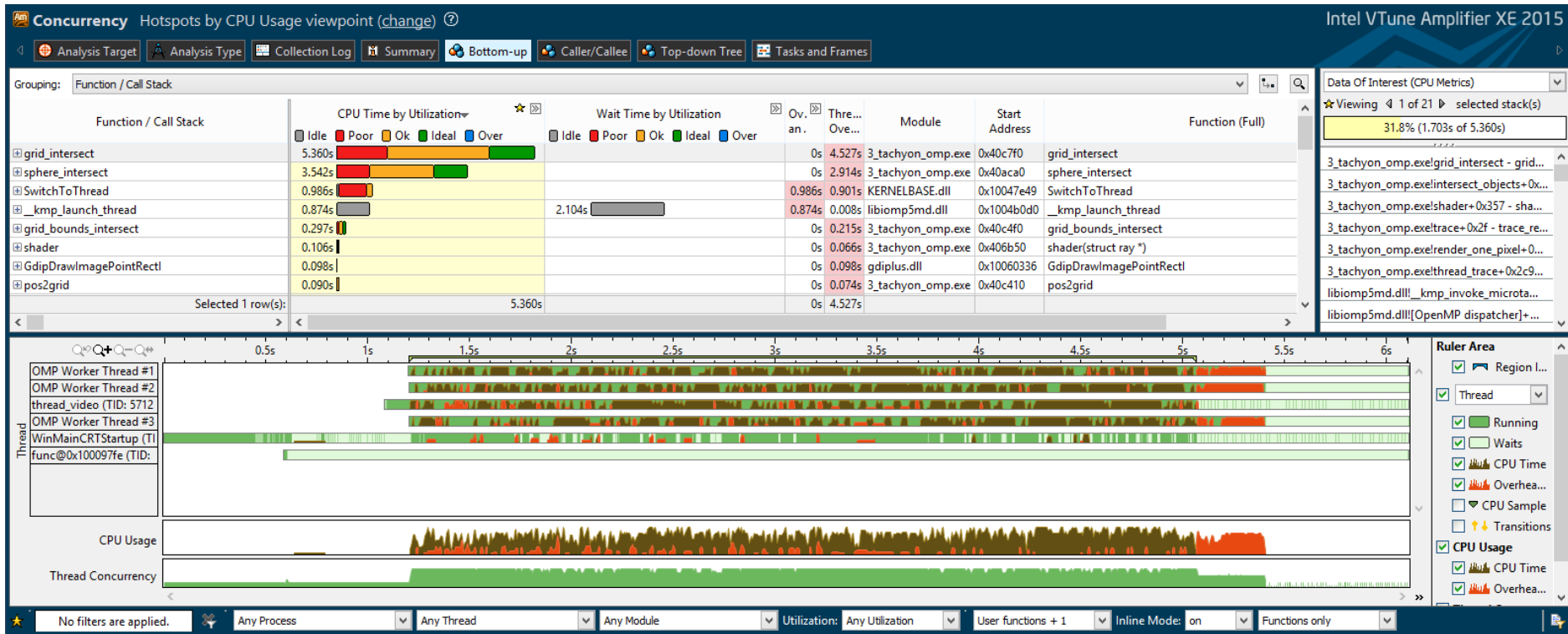
Simultaneously Utilized Logical CPUs	Elapsed Time (s)
0	1.0
1	1.2
2	1.8
3	1.3
4	0.1

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



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Find Answers Fast

Intel® VTune™ Amplifier

Adjust Data Grouping

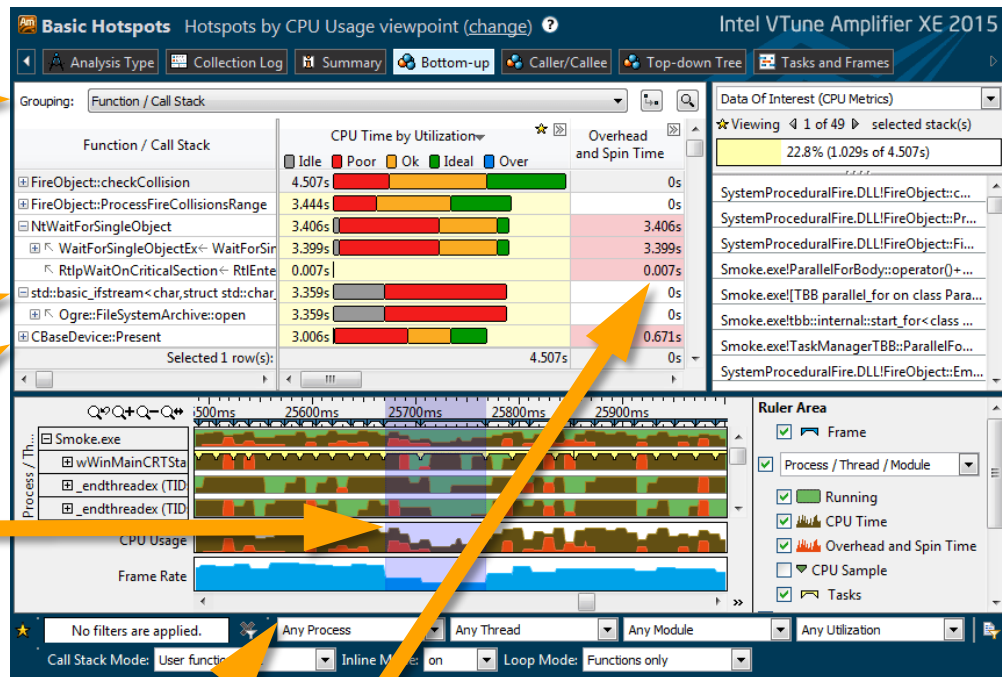
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
- ... (Partial list shown)

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

The screenshot displays the Intel VTune Amplifier XE 2015 interface. The top toolbar includes buttons for 'Source' and 'Assembly'. The 'Source' view on the left shows C++ code with a red bar indicating a hot spot at line 581. The 'Assembly' view on the right shows the corresponding assembly instructions, with a red bar indicating a hot spot at address 0x418b92. A scroll bar 'Heat Map' is visible between the two views. Annotations include arrows pointing to the 'Source' and 'Assembly' tabs, the 'CPU Time' column, the scroll bar, and the assembly instruction at 0x418b92.

Source Line	Source	CPU Time: Total ...	Address	Sour... Line	Assembly	CPU Time: Total ...
579	cur = g->cells[voxindex];	0.017s	0x418b6d	580	cmp dword ptr [ebp-0x190], 0x	0.120s
580	while (cur != NULL) {	0.499s	0x418b74	580	jz 0x418be6 <Block 58>	0.379s
581	if (ry->mbox[cur->obj->id] !=	7.795s	0x418b76		Block 54:	
582	ry->mbox[cur->obj->id] = r	0.547s	0x418b76	581	mov edx, dword ptr [ebp-0x190	0.090s
583	cur->obj->methods->interse	1.769s	0x418b7c	581	mov eax, dword ptr [edx+0x4]	0.020s
584	}		0x418b7f	581	mov ecx, dword ptr [eax]	3.853s
585	cur = cur->next;	0.568s	0x418b81	581	mov edx, dword ptr [ebp+0xc]	2.500s
586	}	0.070s	0x418b84	581	mov eax, dword ptr [edx+0x10]	0.030s
587	curvox.z += step.z;	0.070s	0x418b87	581	mov edx, dword ptr [ebp+0xc]	
588	if (ry->maxdist < tmax.z cu	0.100s	0x418b8a	581	mov eax, dword ptr [eax+ecx*4	0.040s
			0x418b8d	581	cmp eax, dword ptr [edx+0xc]	1.262s
			0x418b90	581	jz 0x418bd6 <Block 57>	
			0x418b92		Block 55:	
			0x418b92	582	mov ecx, dword ptr [ebp-0x190	0.331s
			0x418b98	582	mov edx, dword ptr [ebp-0x4]	0.116s

Scroll Bar "Heat Map" is an overview of hot spots

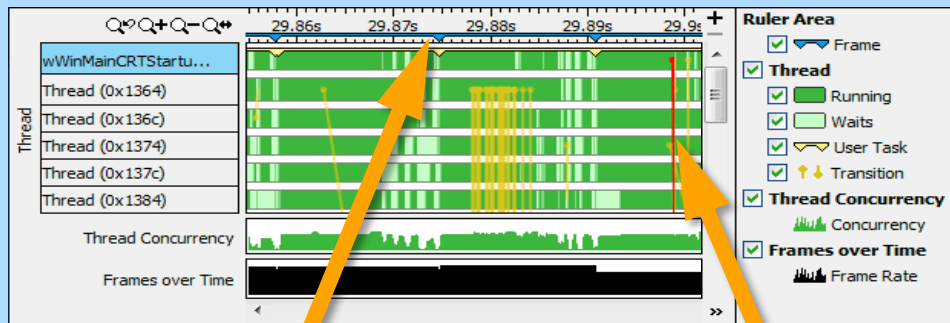
Click jump to scroll Asm

Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier

🔑 Transitions

Locks & Waits



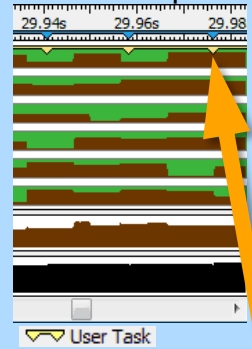
Hovers:

Frame
Frame
Start: 29.858s Duration: 0.017s
Frame: 72
Frame Domain: Smoke::Framework::execute()
Frame Type: Good
Frame Rate: 59.8242179

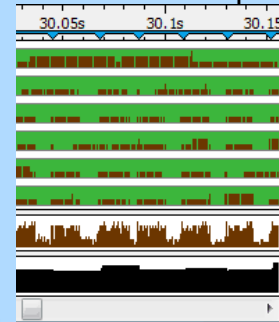
Transition
Transition
wWinMainCRTStartup (0x12d4) to Thread (0x138c) (29.899s to 29.899s)
Sync Object: TBB Scheduler
Object Creation File: taskmanagertbb.cpp
Object Creation Line: 318

CPU Time

Basic Hotspots



Advanced Hotspots



User Task
User Task
Start: 29.958s Duration: 0.018s
Task Type: Smoke::Framework::execute()::Other
Task End Call Stack: Framework::Execute

CPU Time
94.233472%

Optional: Use API to mark frames and user tasks



Optional: Add a mark during collection



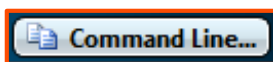
Command Line Interface

Automate analysis

amplxe-cl is the command line:

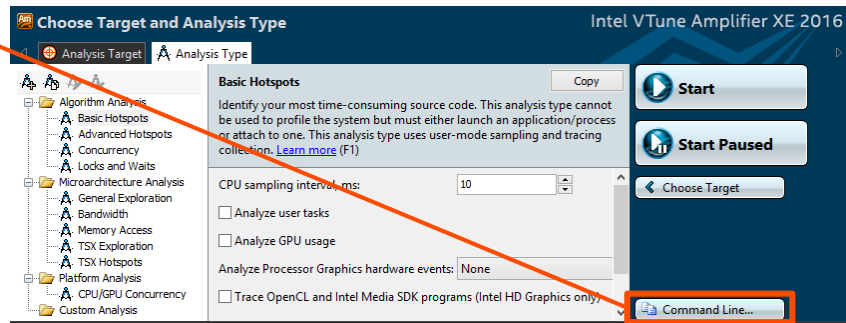
- Windows:** C:\Program Files (x86)\Intel\VTune Amplifier XE\bin[32|64]\amplxe-cl.exe
- Linux:** /opt/intel/vtune_amplifier_xe/bin[32|64]/amplxe-cl

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

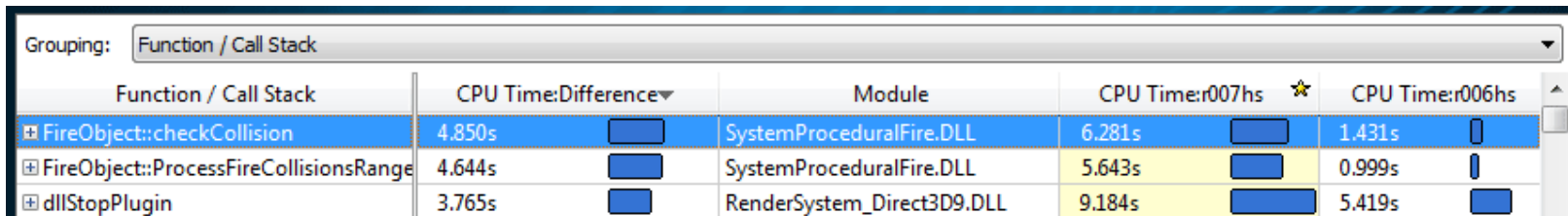
Intel® VTune™ Amplifier

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?

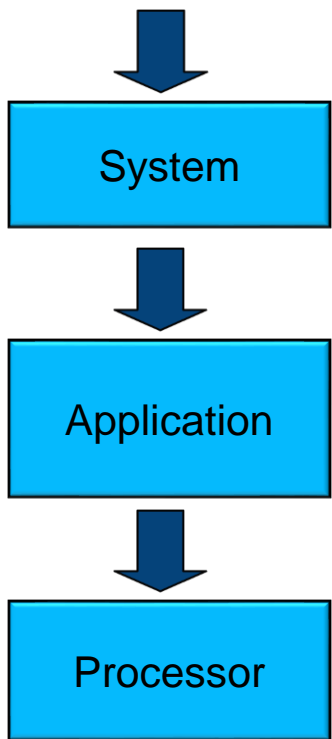
Compare 2 systems – What didn't speed up as much?



The screenshot shows a table with the following columns: Function / Call Stack, CPU Time:Difference, Module, CPU Time:r007hs, and CPU Time:r006hs. The table is sorted by CPU Time:Difference. The first three rows are visible:

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

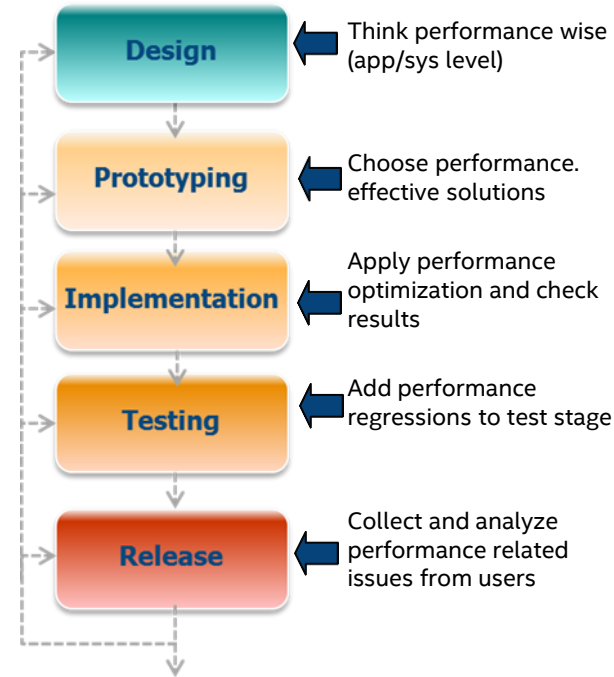


H/W tuning:
BIOS (TB, HT)
Memory
Network I/O
Disk I/O

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

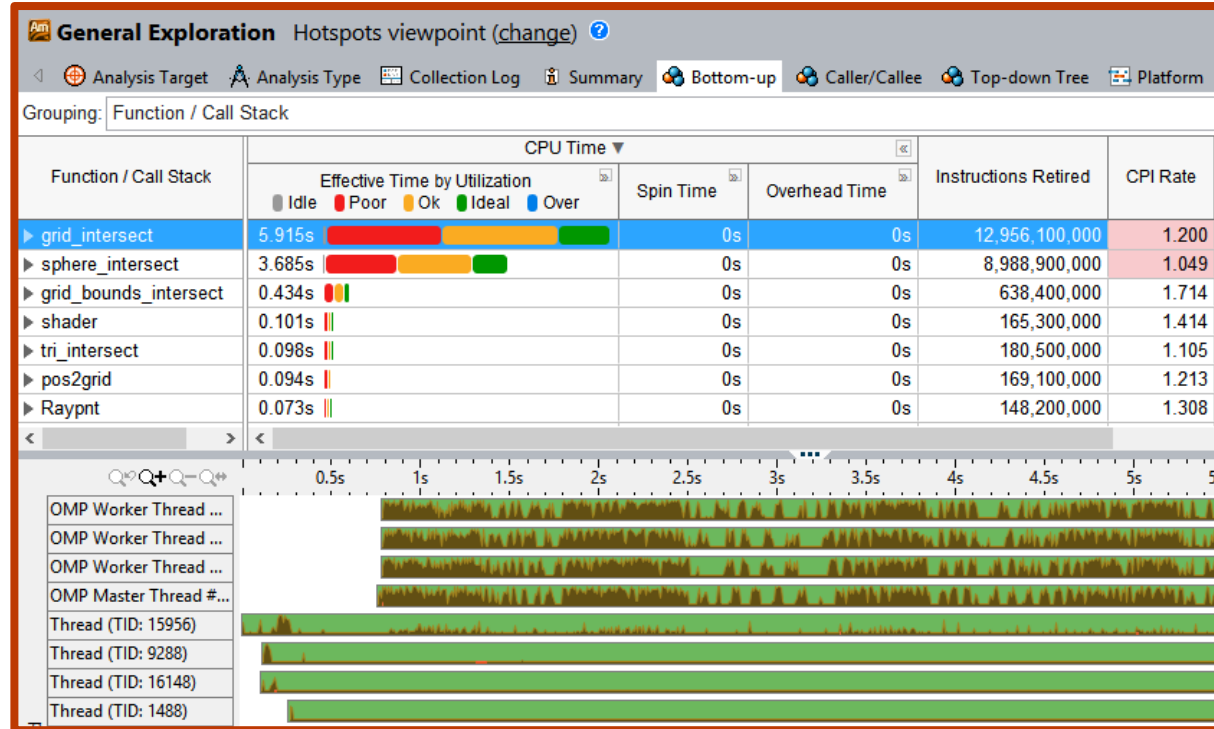
Better application design:
Parallelization
Fast algorithms / data bases
Programming language and RT libs
Performance libraries
Driver tuning

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

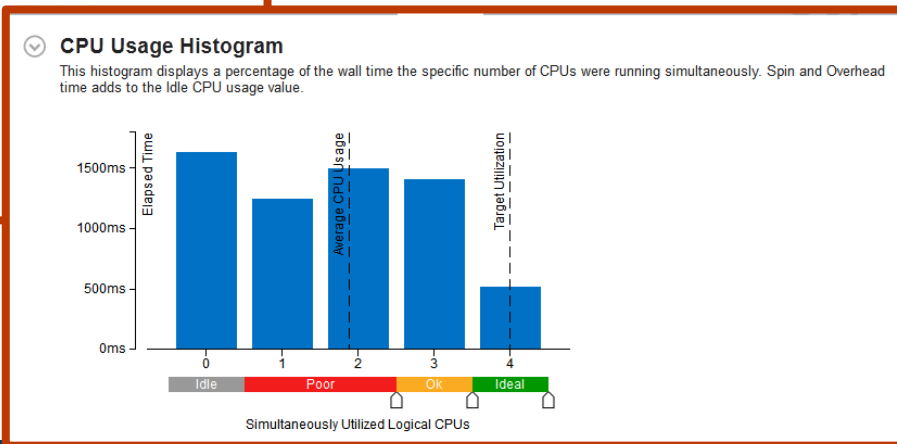
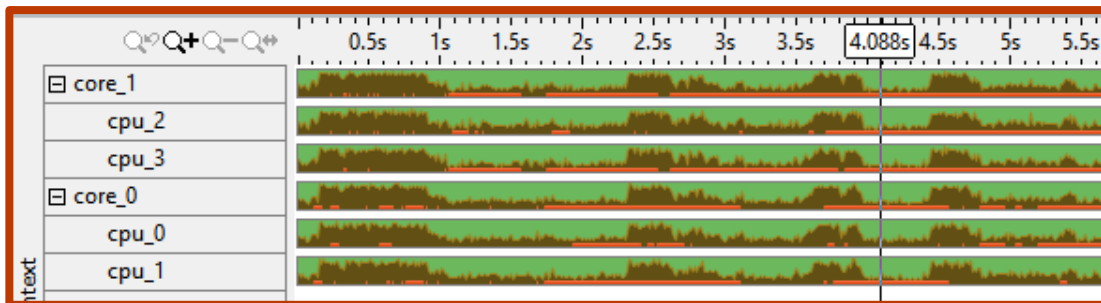
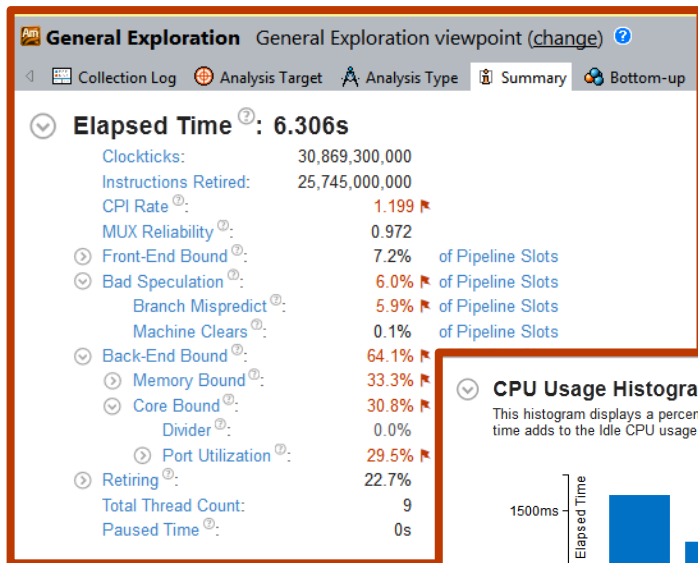


Introduction to Intel VTune Amplifier

- 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



System-Level Profiling – High-level Overviews

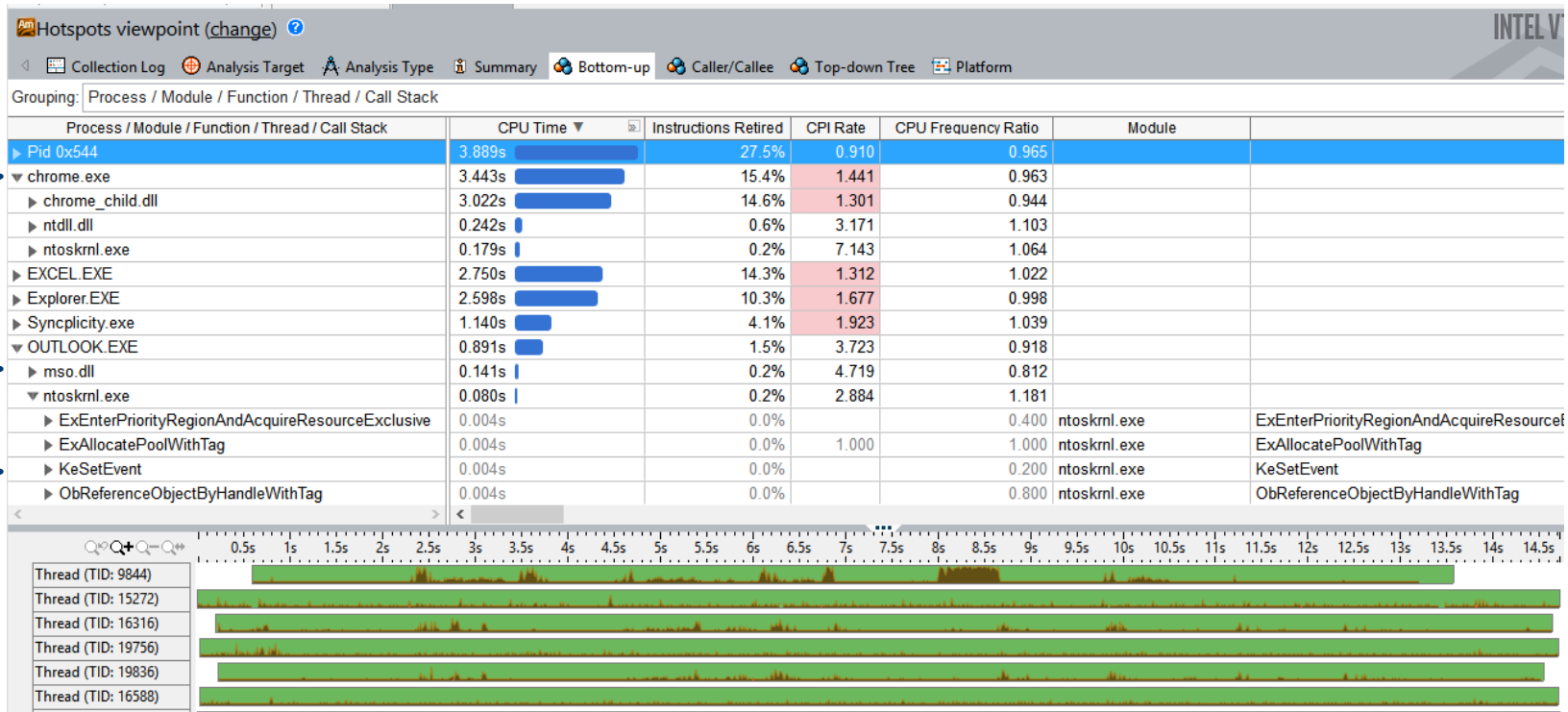


Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



System-Level Profiling – Process/Module Breakdowns



System-Level Profiling – Disk I/O Analysis

Are You I/O Bound or CPU Bound?

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

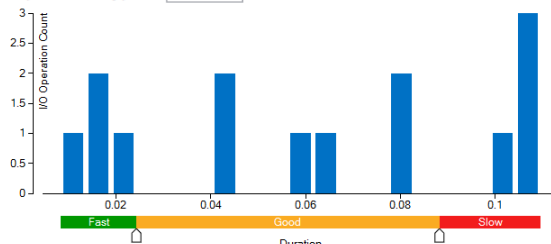
See when CPU is waiting for I/O

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

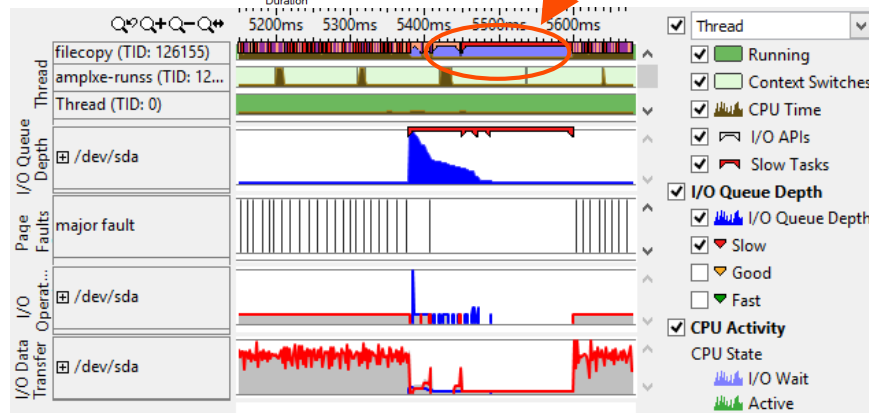
```
> ampxe-cl -collect disk-io -d 10
```

Disk Input and Output Histogram

Operation Type: write



Slow task with I/O Wait



System-Level Profiling – HPC Characterization

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
```

CPU Utilization [Ⓜ]: **60.9%**

Average CPU Usage [Ⓜ]: 14.611 Out of 24 logical CPUs
Serial Time [Ⓜ]: 0.013s (0.1%)

Parallel Region Time [Ⓜ]: **11.986s (99.9%)**

Estimated Ideal Time [Ⓜ]: 8.205s (68.4%)
OpenMP Potential Gain [Ⓜ]: **3.781s (31.5%)**

The time wasted on load imbalance or parallel work arrangement is significant and negatively impacts the application performance and scalability. Explore OpenMP regions with the highest metric values. Make sure the workload of the regions is enough and the loop schedule is optimal.

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region were executed serially.

OpenMP Regions

- [conj_grad](#)
- [MAIN__\\$d](#)
- [MAIN__\\$d](#)
- [MAIN__\\$d](#)
- [MAIN__\\$d](#)
- [\[Others\]](#)

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

Memory Bound [Ⓜ]: **91.8%**

Cache Bound [Ⓜ]: 0.185

DRAM Latency Bound [Ⓜ]: 0.185

DRAM Bandwidth Bound [Ⓜ]: 0.185

This metric represents a fraction of main memory (DRAM). This metric does not include cache memory. Consider improving data locality in NUMA nodes.

NUMA: % of Remote Accesses [Ⓜ]: 0.185

A significant amount of DRAM loads are being accessed from the same core, or at least the same package.

FPU Utilization [Ⓜ]: **1.3%** ⬇

SP FLOPs per Cycle [Ⓜ]: 0.211 Out of 16 ⬇

Vector Capacity Usage [Ⓜ]: **48.3%** ⬇

FP Instruction Mix:

- % of Packed FP Instr.** [Ⓜ]: 93.1%
- % of 128-bit** [Ⓜ]: **93.1%** ⬇
- % of 256-bit** [Ⓜ]: 0.0%
- % of Scalar FP Instr.** [Ⓜ]: 6.9%

FP Arith/Mem Rd Instr. Ratio [Ⓜ]: **0.264** ⬇

FP Arith/Mem Wr Instr. Ratio [Ⓜ]: 6.298

Top 5 hotspot loops (functions) by FPU usage

This section provides information for the most time consuming loops/functions with floating point operations.

Function	CPU Time [Ⓜ]	FPU Utilization [Ⓜ]	Vector Instruction Set [Ⓜ]	Loop Type [Ⓜ]
[Loop at line 575 in conj_grad_omp\$parallel@517]	126.149s	1.6% ⬇	SSE2(128) ⬇	Body
[Loop at line 678 in conj_grad_omp\$parallel@517]	5.004s	1.7%	SSE2(128)	Body
[Loop at line 575 in conj_grad_omp\$parallel@517]	2.678s	2.1%	[Unknown]	Remainder
[Loop at line 573 in conj_grad_omp\$parallel@517]	0.995s	4.0%	SSE2(128)	Body
[Loop at line 661 in conj_grad_omp\$parallel@517]	0.952s	1.3%	SSE(128); SSE2(128)	Body
[Others]	2.437s	N/A*	N/A*	N/A*

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

System-Level Profiling – Memory Bandwidth

HPC Performance Characterization

Analyze important aspects of your application performance, including CPU utilization with additional details on OpenMP efficiency analysis, memory usage, and FPU utilization with vectorization information. For vectorization optimization data, such as trip counts, data dependencies, and memory access patterns, try [Intel Advisor](#). It identifies the loops that will benefit the most from refined vectorization and gives tips for improvements. The HPC Performance Characterization analysis type is best used for analyzing intensive compute applications. [Learn more](#) (F1)

CPU sampling interval, ms:

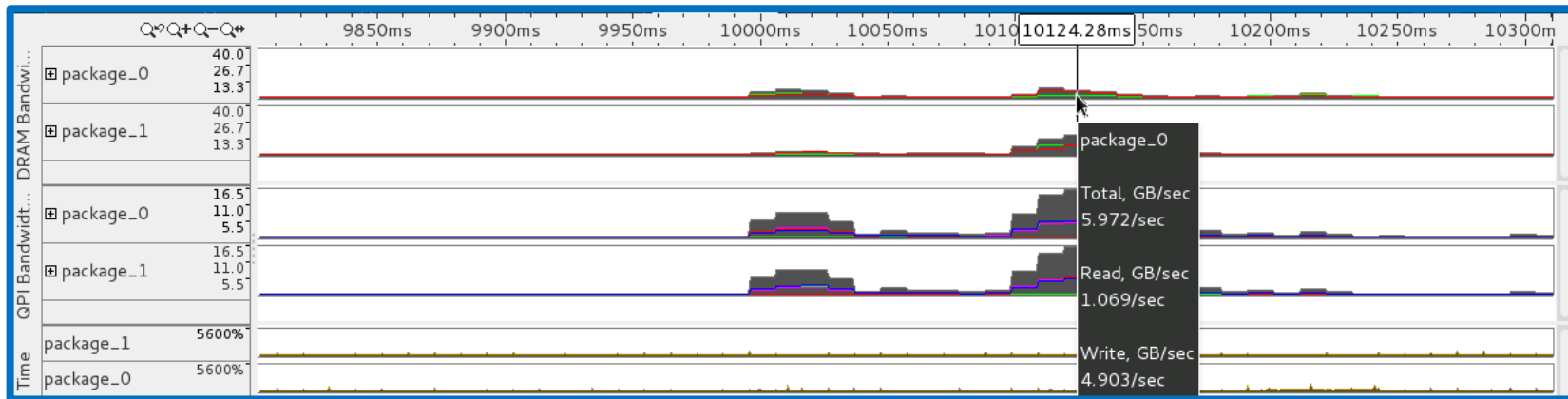
Collect stacks

Analyze memory bandwidth

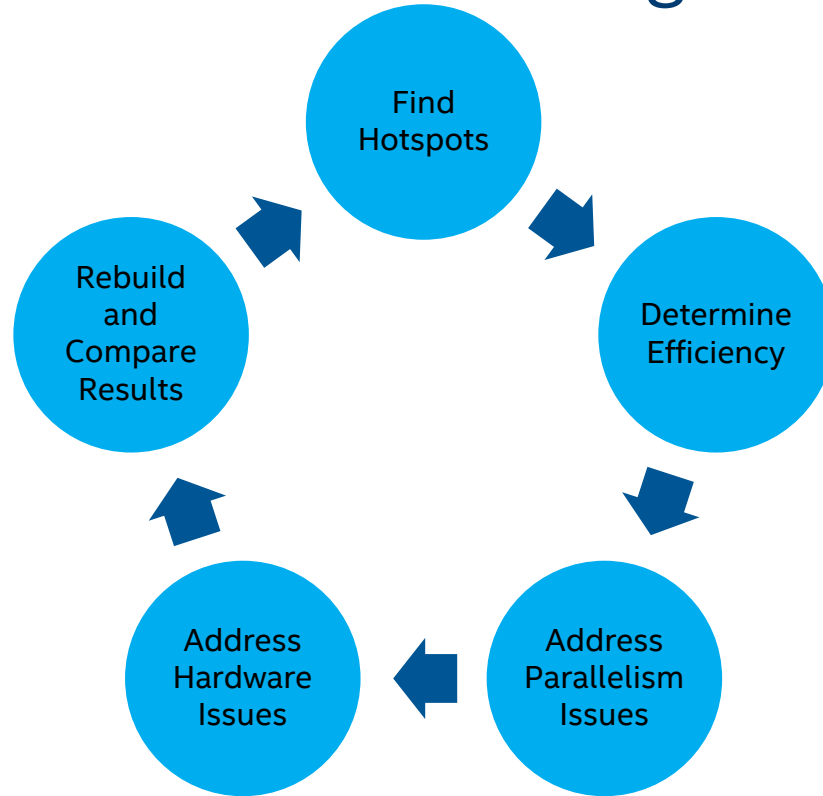
Evaluate max DRAM bandwidth

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

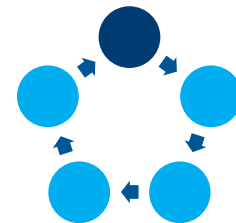
```
-knob collect-memory-bandwidth=true
```



Application Performance Tuning Process



Find Hotspots



Basic Hotspots Hotspots by CPU Usage viewpoint (change)

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

Grouping: Function / Call Stack

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	0x40c2f0
GdiplDrawImagePointRectl	0.172s	gdiplus.dll	GdiplDrawImagePointRectl		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

Viewing: 1 of 19 · selected stack(s)

33.5% (2.033s of 6.063s)

1_tachyon_serial.exe!grid_intersect - ...
1_tachyon_serial.exe!intersect_obje...
1_tachyon_serial.exe!shader+0x346 ...
1_tachyon_serial.exe!trace+0x2e - tr...
1_tachyon_serial.exe!render_one_pi...
1_tachyon_serial.exe!parallel_thread...
1_tachyon_serial.exe!thread_trace+...
1_tachyon_serial.exe!trace_shm+0x...
1_tachyon_serial.exe!trace_region+0...
1_tachyon_serial.exe!renderscene+0...
1_tachyon_serial.exe!rt_renderscene...
1_tachyon_serial.exe!tachyon_video...
1_tachyon_serial.exe!thread_video+...

Thread: thread_video (TID: 171...), WinMainCRTStartup (...)

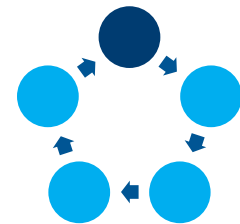
0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s 4.5s 5s 5.5s 6s 6.5s 7s 7.5s 8s 8.5s 9s 9.5s 10s 10.5s 11s 11.5s 12s 12.5s

Thread: Running, CPU Time, Spin and Ov..., CPU Sample, CPU Usage, Spin and Ov...

Functions

Call Stacks

```
> 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

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

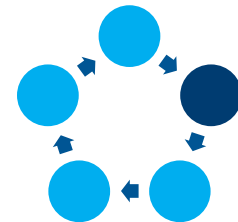
Analysis Target Analysis Type Collection Log Summary Bottom-up Caller/Callee Top-down Tree Platform grid.cpp

Source Assembly

Assembly grouping: Address

Source Line	Source	CPU Time: Total			CPU Time: Self	Source File
		Effective Time by Utilization	Spin Time	Overhead Time		
		<input type="checkbox"/> Idle <input type="checkbox"/> Poor <input type="checkbox"/> Ok <input type="checkbox"/> Ideal <input type="checkbox"/> Over				
562	break;					
563	voxindex += step.x;					
564	tmax.x += tdelta.x;					
565	curpos = nXp;					
566	nXp.x += pdeltaX.x;					
567	nXp.y += pdeltaX.y;					
568	nXp.z += pdeltaX.z;					
569	}					
570	else if (tmax.z < tmax.y) {	0.4%	0.0%	0.0%	0.040s	grid.cpp
571	cur = g->cells[voxindex];	2.9%	0.0%	0.0%	0.321s	grid.cpp
572	while (cur != NULL) {					
573	if (ry->mbox[cur->obj->id] != ry->ser;	22.4%	0.0%	0.0%	2.497s	grid.cpp
574	ry->mbox[cur->obj->id] = ry->serial;	7.3%	0.0%	0.0%	0.817s	grid.cpp
575	cur->obj->methods->intersect(cur->ob	7.9%	0.0%	0.0%	0.406s	grid.cpp
576	}					
577	cur = cur->next;	6.3%	0.0%	0.0%	0.699s	grid.cpp
578	}					
579	curvox.z += step.z;	0.3%	0.0%	0.0%	0.038s	grid.cpp
580	if (ry->maxdist < tmax.z curvox.z ==	0.2%	0.0%	0.0%	0.021s	grid.cpp
581	break;					
582	voxindex += step.z*g->xsize*g->ysize;					
583	tmax.z += tdelta.z;	0.5%	0.0%	0.0%	0.060s	grid.cpp
584	curpos = nZp;					
585	nZp.x += pdeltaZ.x;					
586	nZp.y += pdeltaZ.y;					
...						
Sele...		22.4%	0.0%	0.0%	2.497s	

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				Spin Time	Overhead Time
	Effective Time by Utilization					
	Idle	Poor	Ok	Ideal		
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)

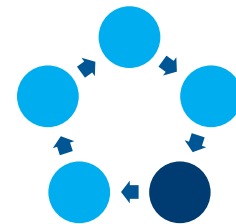
Analysis Target Analysis Type Collection Log Summary Bottom-up

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%	

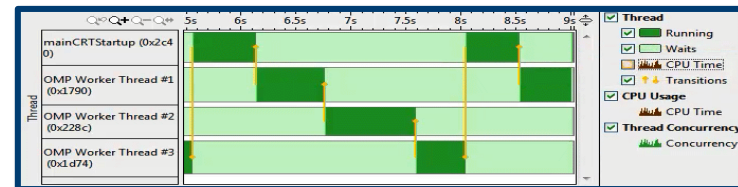
Look for Parallelism, Cycles-per-Instruction (CPI), and Retiring %

Address Parallelism Issues



Coarse-Grain Locks

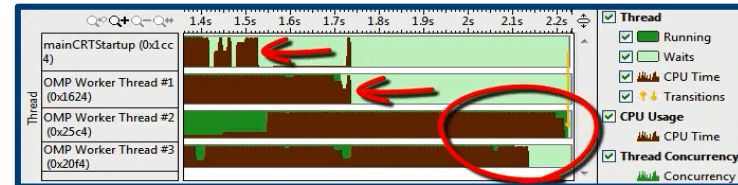
- Use Concurrency Analysis to ensure you're using all your threads as often as possible.



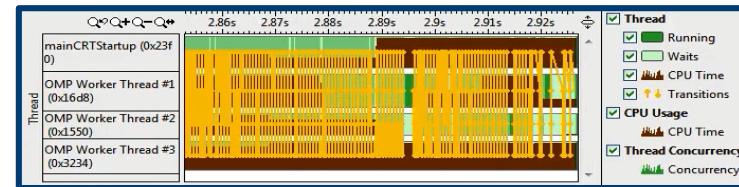
- Common concurrency problems can often be diagnosed in the timeline.

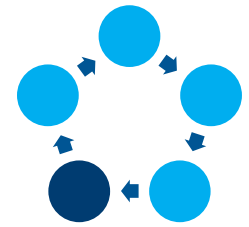
Thread Imbalance

- Switch to the Locks And Waits viewpoint or run a Locks and Waits analysis to investigate contention.

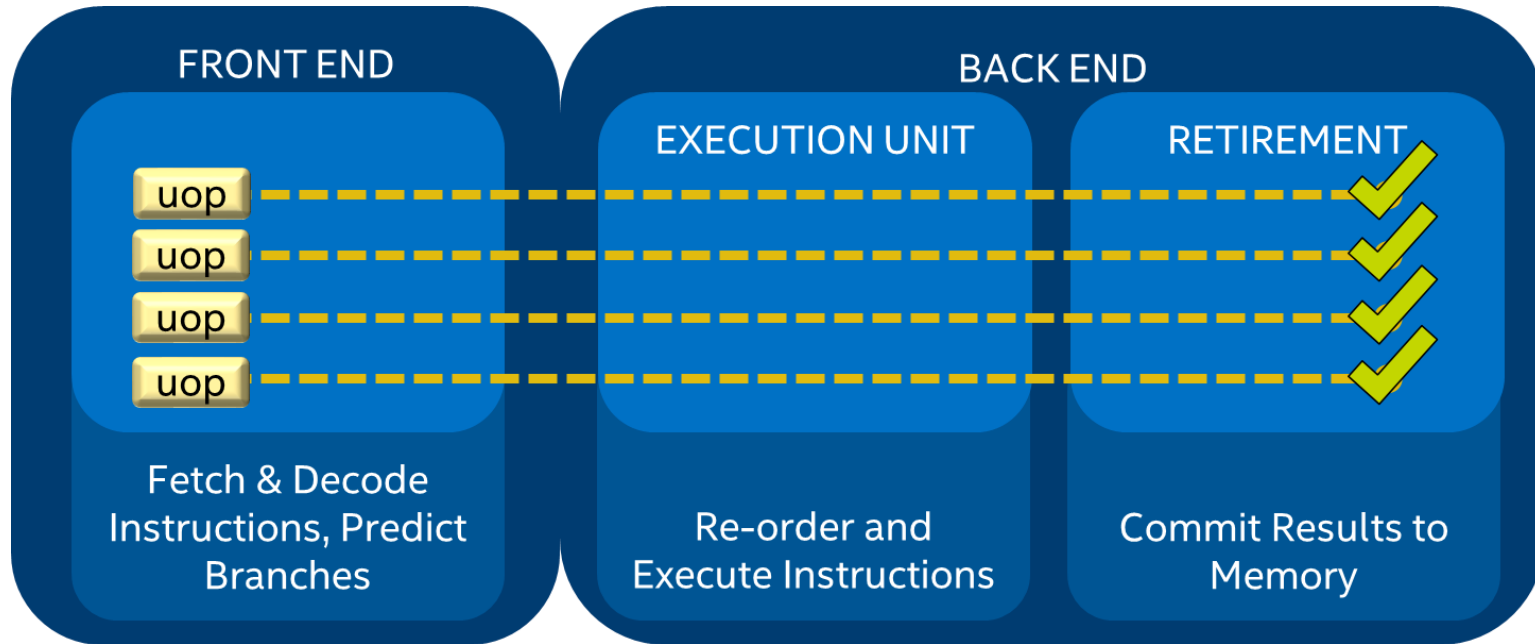


High Lock Contention

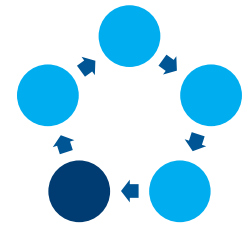




Address Hardware Issues

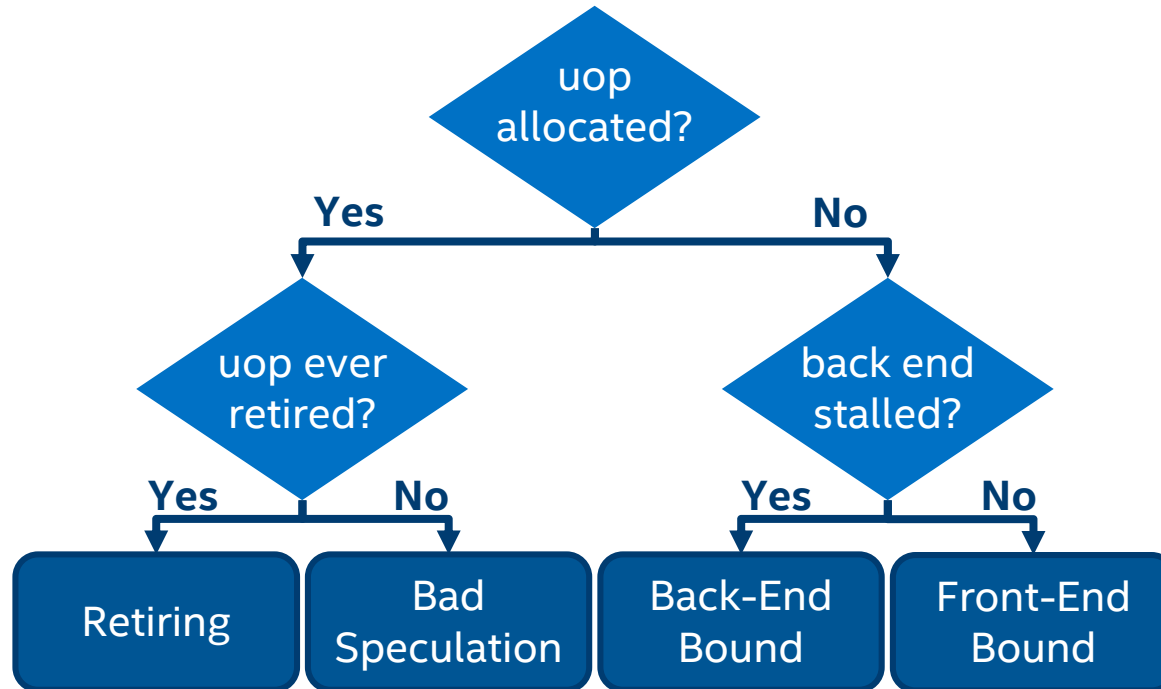


The X86 Processor Pipeline (simplified)

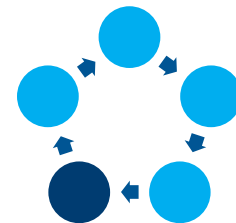


Address Hardware Issues

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

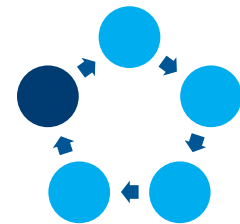
Grouping: Function / Call Stack

Function / Call Stack	Retiring	Front-End Bound	Bad Speculation	Back-End Bound		Module
				Memory Bound	Core Bound	
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

Compare... r002hs primes.cpp r001hs r000hs primes_omp.cpp

Choose Results to Compare

Result 1: r003ah.amplxe

Result 2: r004ah.amplxe

INTEL VTUNE AMPLIFIER XE 2017

These results can be compared. Click the Compare button to continue.

Summary Bottom-up Caller/Callee Top-down Tree

Elapsed Time[?]: 7.420s - 5.541s = 1.879s

Instructions Retired: 24,654,400,000 - 22,868,400,000 = 1,786,000,000

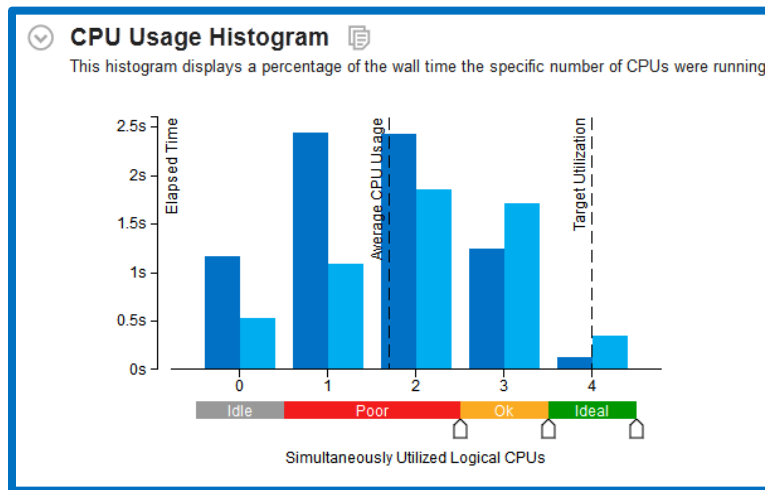
CPI Rate[?]: 1.326 - 1.363 = -0.037

CPU Frequency Ratio[?]: 1.040 - 1.042 = -0.003

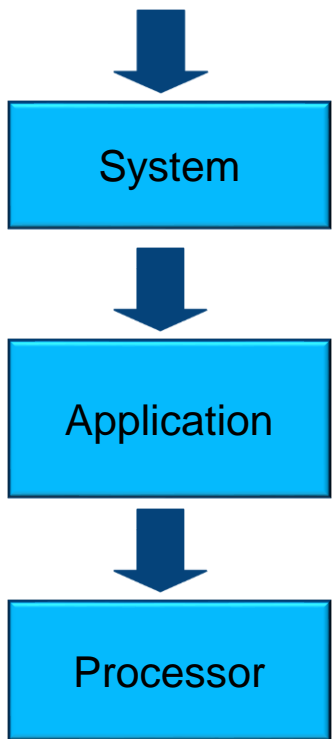
Total Thread Count: Not changed, 4

Paused Time[?]: Not changed, 0s

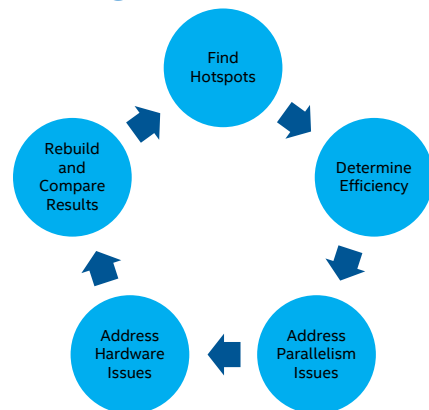
CPU Time[?]: 12.603s - 11.987s = 0.616s



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.

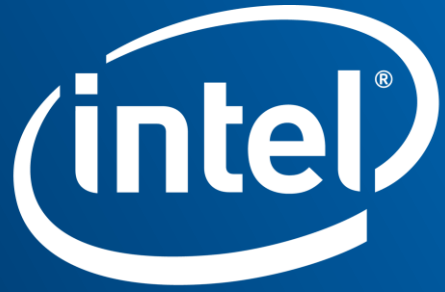
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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



Software