



Intel[®] VTune[™] Amplifier

Performance Profiler

Faster, Scalable Code, Faster

Intel® VTune™ Amplifier Performance Profiler

Accurate Data - Low Overhead

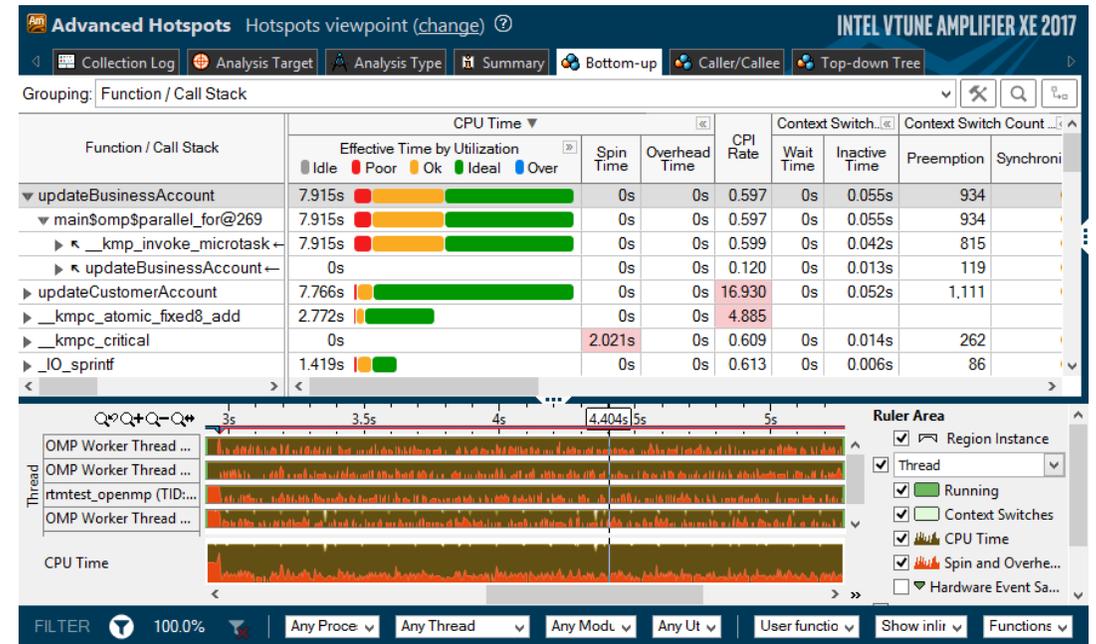
- CPU, GPU, FPU, threading, bandwidth...

Meaningful Analysis

- Threading, OpenMP region efficiency
- Memory access, storage device

Easy

- Data displayed on the source code
- Easy set-up, no special compiles



“Last week, Intel® VTune™ Amplifier helped us find almost 3X performance improvement. This week it helped us improve the performance another 3X.”

Claire Cates
Principal Developer
SAS Institute Inc.

<http://intel.ly/vtune-amplifier-xe>

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

Find Answers Fast

Intel® VTune™ Amplifier

Adjust Data Grouping

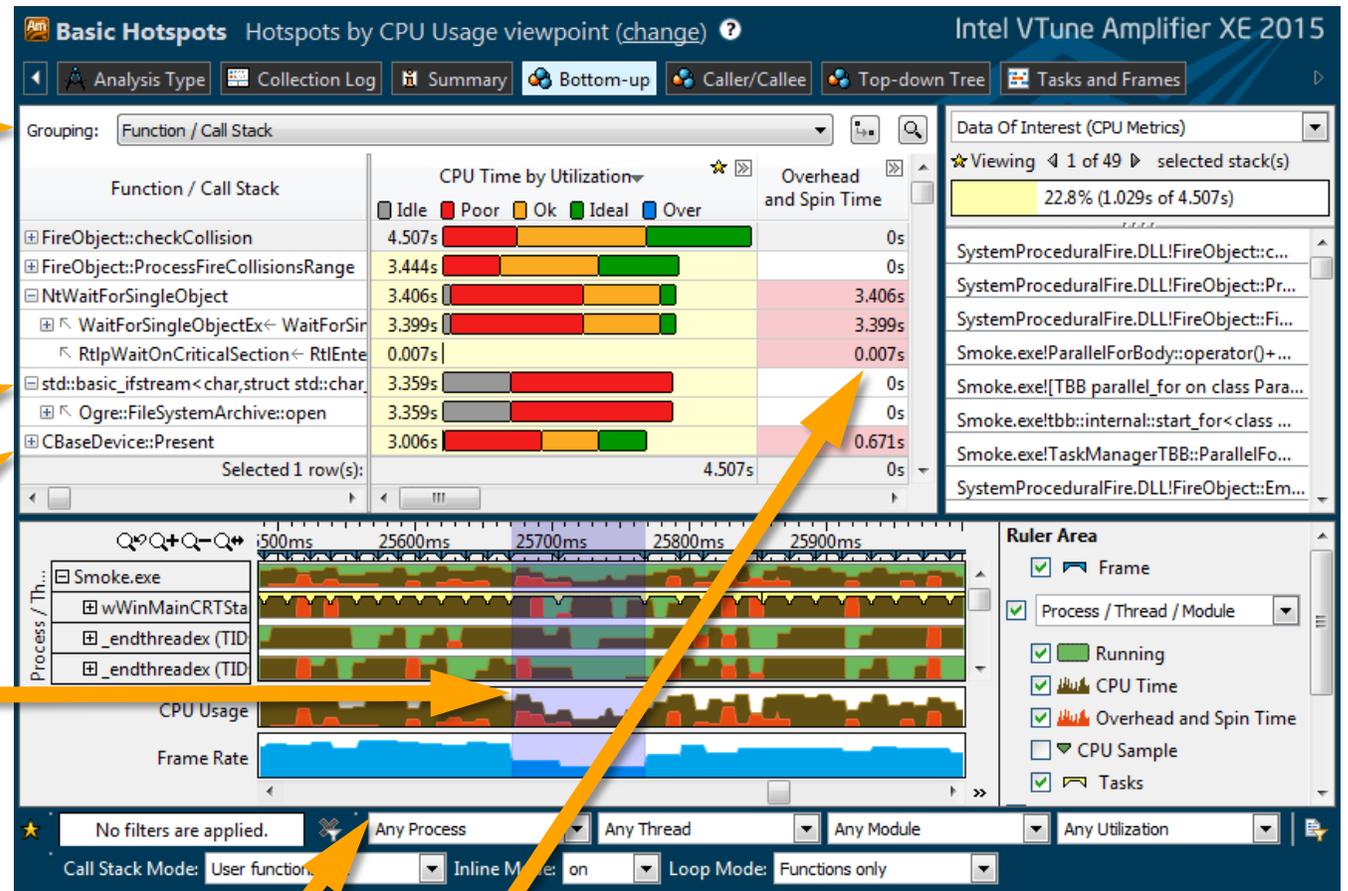
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
- ... (Partial list shown)

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

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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 shows the Intel VTune Amplifier XE 2015 interface. The 'Hotspots by CPU Usage viewpoint (change)' is active. The 'Source' view on the left shows C++ code with a 'Heat Map' scroll bar on the right. The 'Assembly' view on the right shows the corresponding assembly instructions. Annotations include: an arrow pointing to the 'Source' tab; an arrow pointing to the 'CPU Time: Total ...' column; an arrow pointing to the 'Assembly' view; an arrow pointing to the 'Heat Map' scroll bar; an arrow pointing to the 'jz 0x418bd6 <Block 57>' instruction; and an arrow pointing to the 'jz 0x418bd6 <Block 57>' instruction in the assembly view.

Source Line	Source	CPU Time: Total ...	Address	Sour... Line	Assembly	CPU Time: Total ...
579	cur = g->cells[voxindex];	0.30s	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	581	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	582	Block 55:	
			0x418b92	582	mov ecx, dword ptr [ebp-0x190]	0.331s
			0x418b98	582	mov edx, dword ptr [ecx+0x4]	0.116s

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

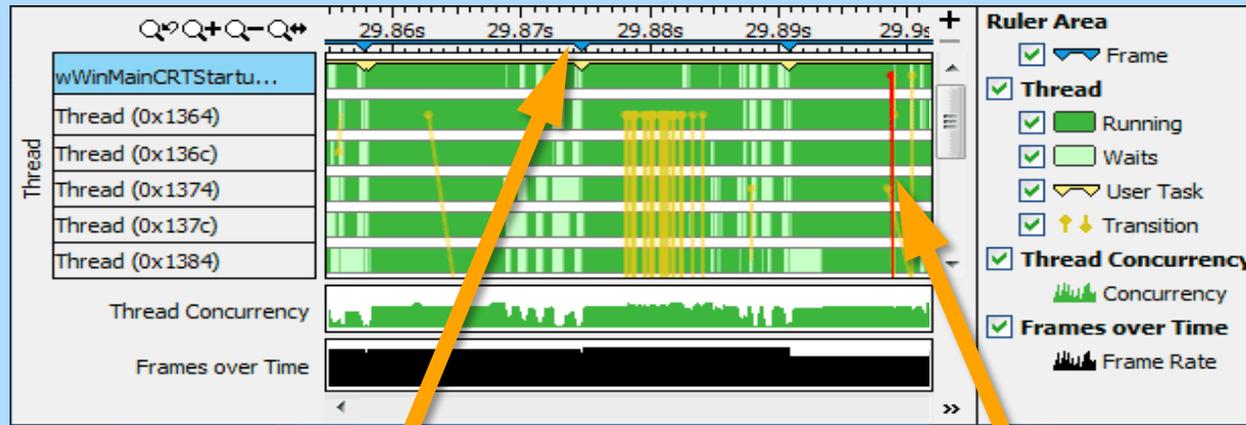
Click jump to scroll Asm

Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier

🔑 Transitions

Locks & Waits



Hovers:

Frame

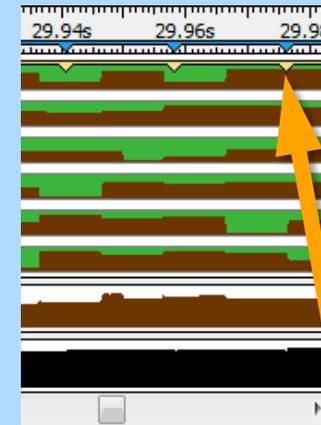
Frame
Start: 29.858s Duration: 0.017s
Frame: 72
Frame Domain: Smoke::Framework::execute()
Frame Type: Good
Frame Rate: 59.8242179

Transition

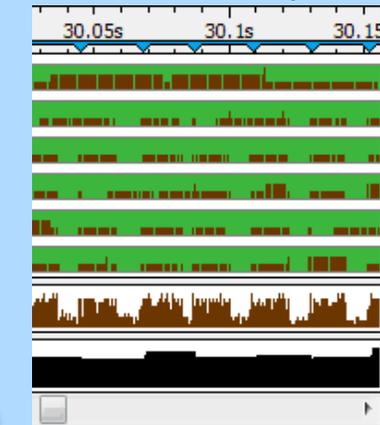
Transition
wWinMainCRTStartup (0x12d4) to Thread (0x138c) (29.899s to 29.899s)
Sync Object: TBB Scheduler
Object Creation File: taskmanagertbb.cpp
Object Creation Line: 318

📊 CPU Time

Basic Hotspots



Advanced Hotspots

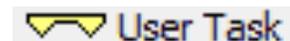


User Task

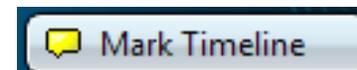
User Task
Start: 29.958s Duration: 0.018s
Task Type: Smoke::FrameWork::execute()::Other
Task End Call Stack: Framework::Execute

CPU Time
94.233472%

Optional: Use API to mark frames and user tasks



Optional: Add a mark during collection



Optimization Notice

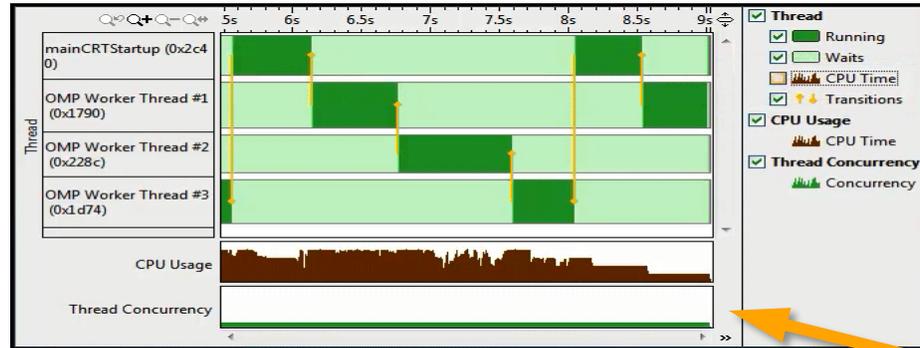
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



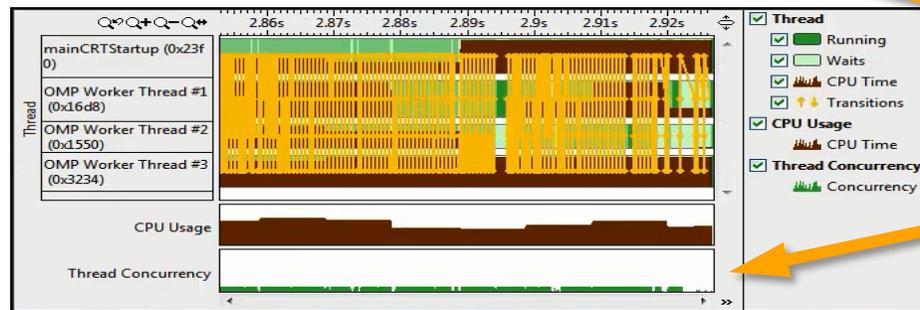
Visualize Parallel Performance Issues

Look for Common Patterns

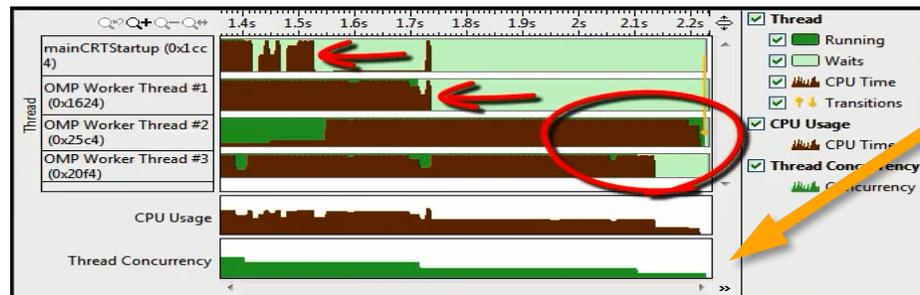
Coarse Grain
Locks



High Lock
Contention



Load
Imbalance



Low
Concurrency

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

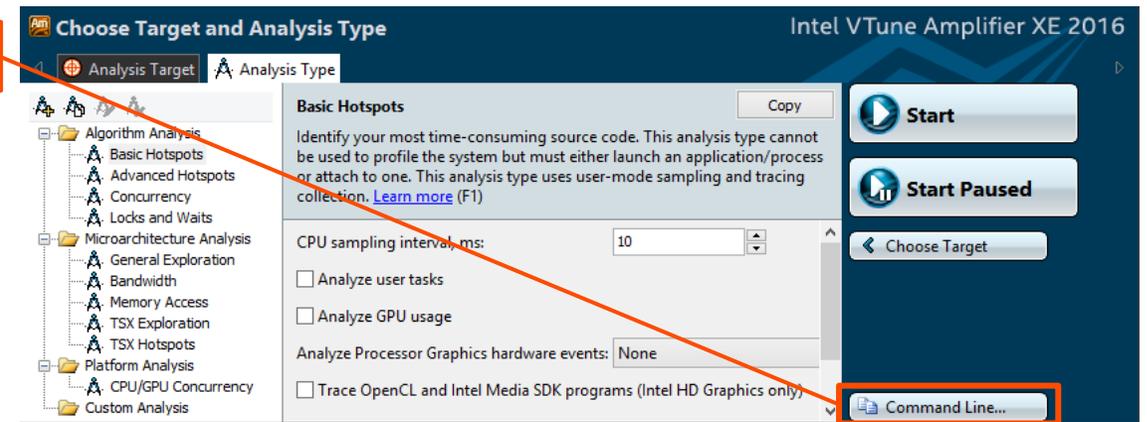
-**Linux:** /opt/intel/vtune_amplifier_xe/bin[32|64]/amplxe-cl

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

MPI Analysis

Command line:

```
> mpirun -n 16 -ppn 4 -l amplxe-cl -collect advanced-hotspots -  
trace-mpi -result-dir my_result -- my_app.a
```

Or use gtool:

```
> mpirun -gtool "amplxe-cl -collect memory-access -result-dir  
my_result:7,5" my_app.a
```

Each process data is presented for each node they were running on:

```
my_result.host_name1 (rank 0-3)  
my_result.host_name2 (rank 4-7)  
my_result.host_name3 (rank 8-11)  
my_result.host_name4 (rank 12-15)
```

Elapsed Time [?]: 120.101s

CPU Time [?]: 7085.857s

Effective Time [?]: 2227.628s

Spin Time [?]: 4589.471s

A significant portion of CPU time is spent waiting. Use this metric to discover which synchronizations are spinning. Consider adjusting spin wait parameters, changing the lock implementation (for example, by backing off then descheduling), or adjusting the synchronization granularity.

Overhead Time [?]: 268.758s

Instructions Retired: 3,333,315,033,755

CPI Rate [?]: 2.757

Wait Rate [?]: 191.043

CPU Frequency Ratio [?]: 1.000

Context Switch Time [?]: 609.769s

Total Thread Count: 139

Paused Time [?]: 60.056s

OpenMP Analysis. Collection Time [?]: 60.046

Serial Time (outside any parallel region) [?]: 43.097s (71.8%)

Serial Time of your application is high. It directly impacts application Elapsed Time and scalability. Explore options for parallelization, algorithm or microarchitecture tuning of the serial part of the application.

Parallel Region Time [?]: 16.949s (28.2%)

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 was optimized to have no load imbalance asst

OpenMP Region	OpenMP Potential Gain [?]	(%) [?]	OpenMP Region Time [?]
allocGauge_D\$omp\$parallel:128@/panfs/projects/milc_wg/ruizi/QPHIX/mbench/ks_long_dslash.cpp:231:234	0.340s	0.6%	1.033s
qphix_ks_dslash_D\$omp\$parallel:128@/panfs/projects/milc_wg/ruizi/QPHIX/mbench/ks_long_dslash.cpp:541:627	0.301s	0.5%	3.573s
allocGauge18_D\$omp\$parallel:128@/panfs/projects/milc_wg/ruizi/QPHIX/mbench/ks_long_dslash.cpp:231:234	0.278s	0.5%	0.894s
ks_congrad_parity_qphix_D\$omp\$parallel:128@/panfs/projects/milc_wg/ruizi/milc_qcd_1/milc_qcd/ks_imp_rhmc/./generic_ks/d_congrad5_fn_qphix_P.c:188:191	0.159s	0.3%	0.713s
ks_congrad_parity_qphix_D\$omp\$parallel:128@/panfs/projects/milc_wg/ruizi/milc_qcd_1/milc_qcd/ks_imp_rhmc/./generic_ks/d_congrad5_fn_qphix_P.c:240:243	0.144s	0.2%	0.741s
[Others]	N/A*	N/A*	9.996s

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

Top Hotspots

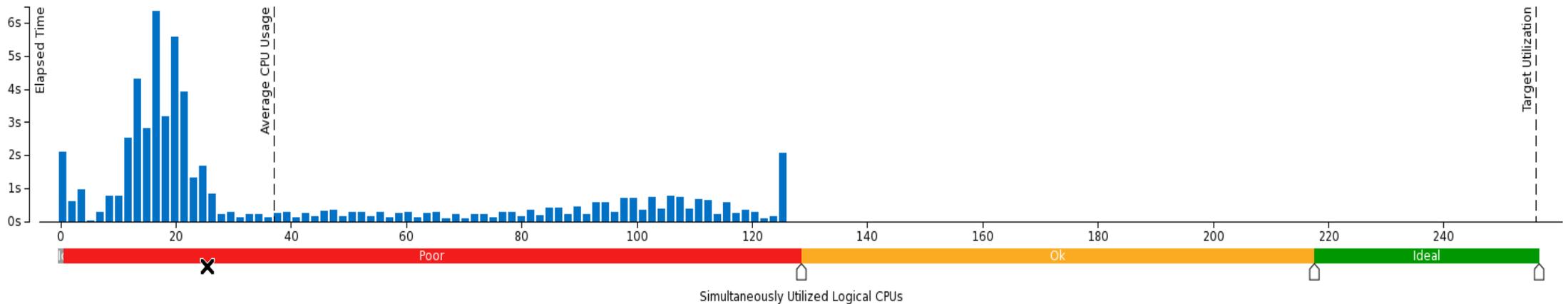
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time ?
_kmp_hyper_barrier_release	libiomp5.so	3791.856s
[vmlinux]	vmlinux	671.755s
_kmp_wait_template<kmp_flag_64>	libiomp5.so	425.421s
ks_long_dslash_vec_noinline	su3_rhmc_hisq_d_knl_mpi	360.775s
_kmp_release_template<kmp_flag_64>	libiomp5.so	263.734s
[Others]	N/A*	1572.316s

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

CPU Usage Histogram

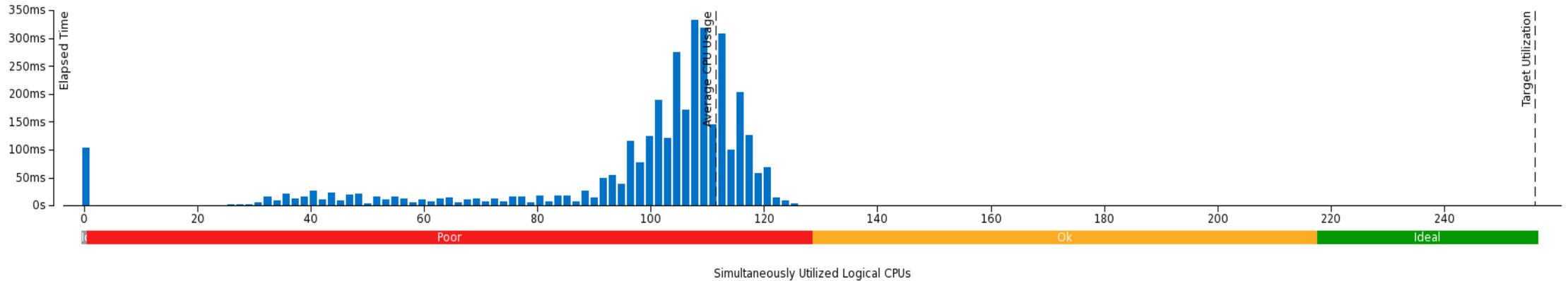
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.



OpenMP Region CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously in an OpenMP region. Spin and Overhead time adds to the Idle CPU usage value. OpenMP regions in the drop-down list are sorted by Potential Gain (Elapsed Time) so it is recommended to start exploration from the top.

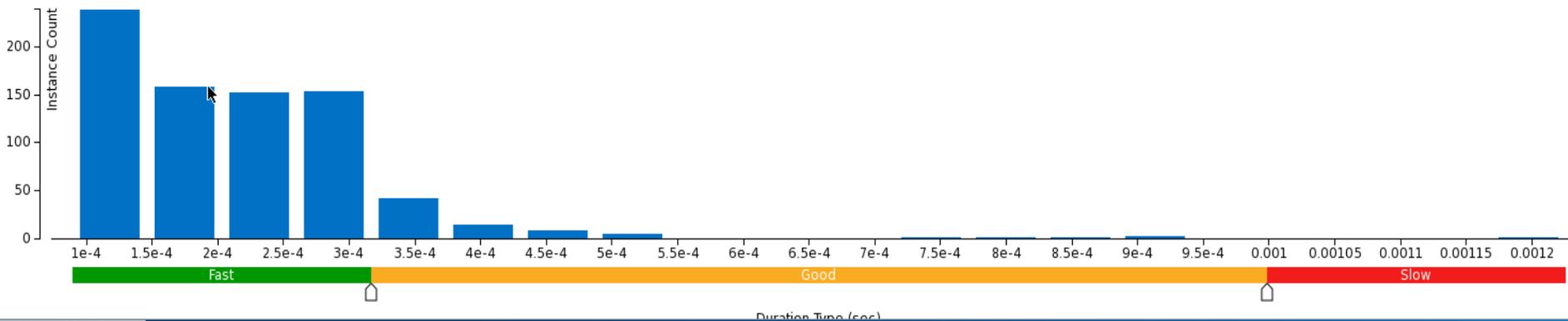
OpenMP Region: `qphix_ks_dslash_Dompparallel:128@/panfs/projects/milc_wg/ruizi/QPHIX/mbench/ks_long_dslash.cpp:541:627`



OpenMP Region Duration Histogram

This histogram shows the total number of region instances in your application executed with a specific duration. High number of slow instances may signal a performance bottleneck. Explore the data provided in the Bottom-up, Top-down Tree, and Timeline panes to identify code regions with the slow duration.

8/18/2016 11:15 AM OpenMP Region: `declare_strided_gatherompparallel:128@/panfs/projects/milc_wg/ruizi/milc_qcd_1/milc_qcd/ks_imp_rhmc/./generic/com_mpi.c:1755:1757`



Optimization Notice

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

Grouping: OpenMP Region / Module / Function / Call Stack

OpenMP Region / Module / Function / Call Stack	OpenMP Potential Gain							Elapsed Time	Number of OpenMP threads	Instance Count	CPU Time				Instructions Retired	CPI Rate	Freq R
	Imbalance	Lock Conten...	Crea...	Sch...	Red.	Ato.	Other				Effective Time by Utilization						
											Idle	Poor	Ok	Ideal			
Serial - outside any region							0s	43.097s			431.896s	4301.079s	248.733s	82.5%	2.350		
qphix_ks_dslash_D\$omp\$par	0.285s	0.000s	0s	0s	0s	0s	0.016s	3.573s	128	986	398.284s	33.112s	2.076s	4.0%	4.195		
QPHIX_D3_asqtad_create_L	0.054s	0.000s	0s	0.000s	0s	0s	0.002s	1.831s	128	72	224.716s	6.077s	0.321s	0.8%	11.549		
QPHIX_D3_asqtad_create_L	0.054s	0s	0s	0.000s	0s	0s	0.002s	1.833s	128	72	222.434s	5.956s	0.311s	0.9%	9.502		
QPHIX_D3_asqtad_invert_m	0.102s	0.000s	0s	0.004s	0s	0s	0.011s	1.096s	128	421	124.911s	11.431s	1.865s	1.2%	4.593		
allocGauge_D\$omp\$parallel	0.323s	0s	0s	0.000s	0s	0s	0.017s	1.033s	128	144	90.620s	37.008s	2.176s	1.0%	4.895		
allocGauge18_D\$omp\$parallel	0.265s	0.000s	0s	0.001s	0s	0s	0.013s	0.894s	128	144	81.035s	30.401s	1.715s	0.9%	5.059		
create_qphix_raw4_D_G_fro	0.059s	0s	0s	0.000s	0s	0s	0.004s	0.756s	128	144	87.157s	7.701s	0.481s	1.4%	2.697		
ks_congrad_parity_qphix_D\$	0.137s	0.000s	0s	0.000s	0s	0s	0.007s	0.741s	128	260	76.315s	15.362s	0.862s	0.7%	5.399		
ks_congrad_parity_qphix_D\$	0.151s	0.000s	0s	0.001s	0s	0s	0.007s	0.713s	128	260	70.767s	16.826s	0.983s	0.6%	5.757		
allocKS_D\$omp\$parallel:128	0.124s	0.000s	0s	0.001s	0s	0s	0.007s	0.546s	128	550	53.183s	15.020s	1.063s	0.9%	3.077		
ks_congrad_parity_qphix_D\$	0.097s	0s	0s	0.001s	0s	0s	0.006s	0.545s	128	260	56.553s	11.672s	0.913s	0.6%	4.842		
ks_congrad_parity_qphix_D\$	0.097s	0.000s	0s	0.001s	0s	0s	0.005s	0.539s	128	260	56.548s	11.401s	0.762s	0.5%	5.223		
create_qphix_D_V_from_fie	0.065s	0.000s	0s	0.000s	0s	0s	0.004s	0.438s	128	204	45.273s	7.590s	0.541s	0.7%	2.827		
QPHIX_D3_create_V_from_r	0.062s	0.000s	0s	0.001s	0s	0s	0.003s	0.278s	128	204	25.751s	7.340s	0.501s	0.4%	2.993		
unload_anhix_D_V_to_fie	0.039s	0.000s	0s	0.000s	0s	0s	0.002s	0.218s	128	139	21.640s	4.241s	0.321s	0.4%	2.791		
Selected 1 row(s):							0s	43.097s			431.896s	4301.079s	248.733s	82.5%	2.350		

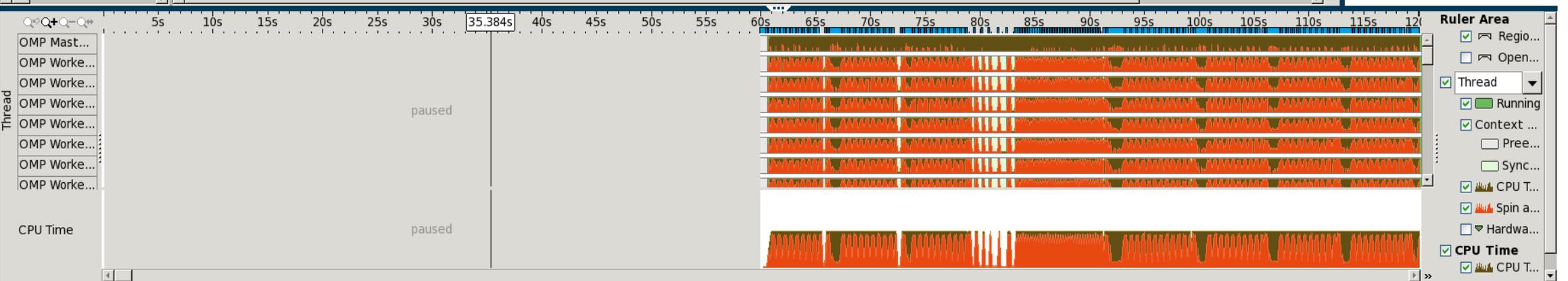
CPU Time

Viewing 1 of 148 selected sta...

71.8% (3577.134s of 4981.719s)

libc-2.12.so!libc-2.12.so! - [unknow...

libc-2.12.so!Unknown stack frame(...



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



Memory Access Analysis

Tune data structures for performance

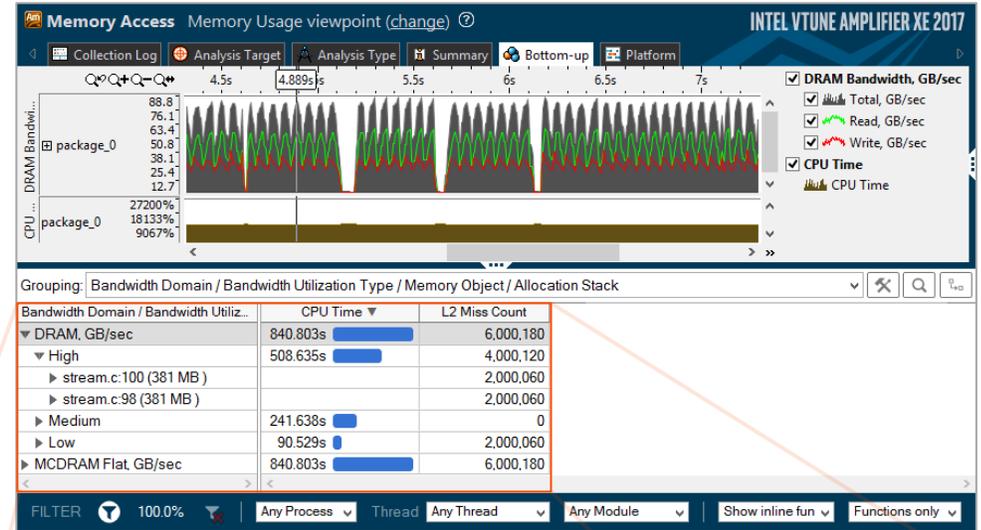
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

Optimize NUMA latency & scalability

- True & false sharing optimization
- Auto detect max system bandwidth
- Easier tuning of inter-socket bandwidth

Easier install, Latest processors

- No special drivers required on Linux*
- Intel® Xeon Phi™ processor MCDRAM (high bandwidth memory) analysis



Bandwidth Domain / Bandwidth Utiliz...	CPU Time	L2 Miss Count
▼ DRAM, GB/sec	840.803s	6,000,180
▼ High	508.635s	4,000,120
▶ stream.c:100 (381 MB)		2,000,060
▶ stream.c:98 (381 MB)		2,000,060
▶ Medium	241.638s	0
▶ Low	90.529s	2,000,060
▶ MCDRAM Flat, GB/sec	840.803s	6,000,180

Elapsed Time [?]: 3.821s

CPU Time [?] :	282.064s
Memory Bound [?] :	84.6% of Pipeline Slots
L1 Bound [?] :	30.5% of Clockticks
L2 Bound [?] :	0.0% of Clockticks
L3 Bound [?] :	27.1% of Clockticks
DRAM Bound [?] :	15.0% of Clockticks
DRAM Bandwidth Bound [?] :	0.0% of Elapsed Time
Memory Latency:	
Remote / Local memory Ratio [?] :	0.000
Local DRAM [?] :	0.0% of Clockticks
Remote DRAM [?] :	0.0% of Clockticks
Remote Cache [?] :	24.4% of Clockticks
Loads:	43,544,206,287
Stores:	5,444,263,323
LLC Miss Count [?] :	95,405,724
Average Latency (cycles) [?] :	39
Total Thread Count:	116
Paused Time [?] :	0s

Top Memory Objects by Latency

This section lists memory objects that introduced the highest latency to the overall application execution.

Memory Object	Total Latency	Loads	Stores	LLC Miss Count [?]
[Unknown]	56.4%	26,823,504,681	2,909,787,291	59,403,564
linear_regression_pthread.c:136 (7 KB)	40.4%	6,193,985,814	2,146,564,395	36,002,160
[Stack]	1.7%	5,949,178,470	91,802,754	0
lreg-pthread!main (54 MB)	1.1%	4,015,020,447	0	0
[vmlinux]	0.3%	502,215,066	296,108,883	0
[Others]	0.0%	60,301,809	0	0

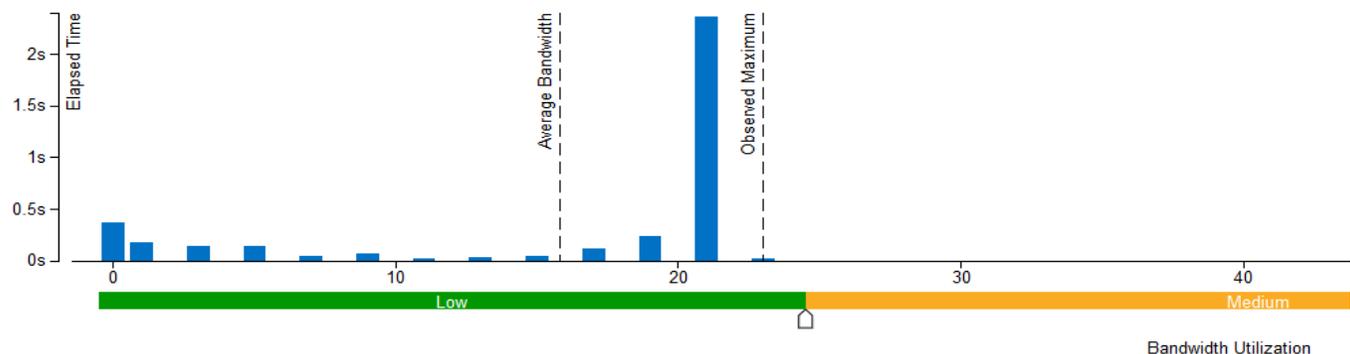
Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

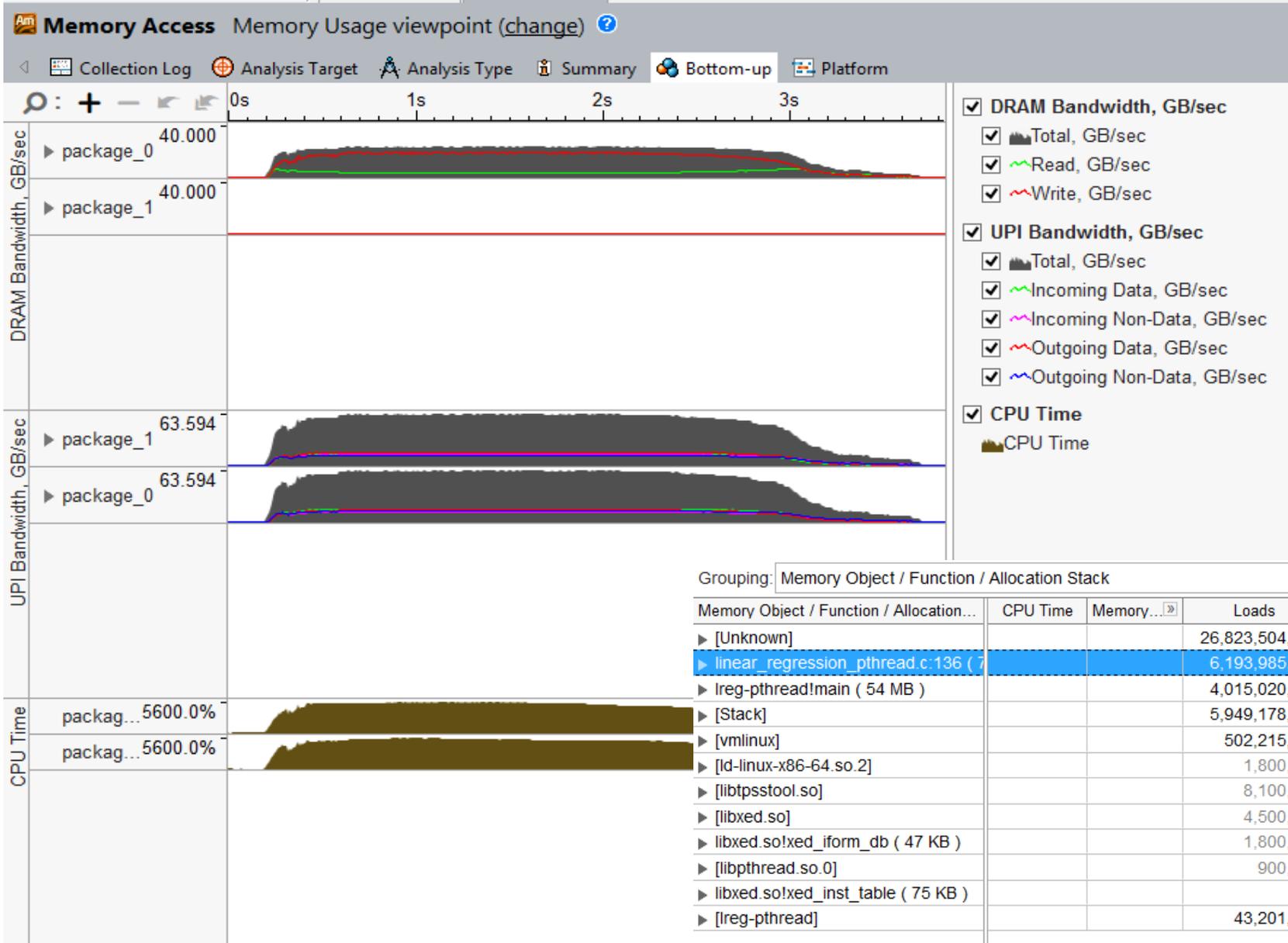
Bandwidth Domain:

Bandwidth Utilization Histogram

This histogram displays the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure



Optimization Notice



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



Memory Access Memory Usage viewpoint (change) ⓘ

Collection Log Analysis Target Analysis Type Summary Bottom-up Platform bt.f

Source Assembly Assembly grouping: Address

S. ▲	Source
46	
47	c-----
48	program BT
49	c-----
50	
51	include 'header.h'
52	include 'mpi_stuff.h'
53	
54	integer num_zones
55	parameter (num_zones=x_zones*y_zones)
56	
57	integer nx(num_zones), nxmax(num_zones), ny(num_zones),
58	\$ nz(num_zones)
59	
60	c-----
61	c Define all field arrays as one-dimensional arrays, to be reshaped
62	c-----
63	double precision
64	> u (proc_max_size5),
65	> us (proc_max_size),
66	> vs (proc_max_size),
67	> ws (proc_max_size),
68	> qs (proc_max_size),
69	> rho_i (proc_max_size),
70	> square (proc_max_size),
71	> rhs (proc_max_size5),
72	> forcing (proc_max_size5),
73	> qbc_ou (proc_max_bcsiz),
74	> qbc_in (proc_max_bcsiz)

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time
▼ z_solve_omp\$parallel@43	361.646s
▼ bt.f:64 (41 MB)	
▶ u	
▼ bt.f:68 (8 MB)	
▶ qs	
▼ bt.f:70 (8 MB)	
▶ square	
▼ work_lhs_vec.h:11 (1 MB)	
▶ lhs	
▶ work_lhs_vec.h:10 (657 KB)	
▶ x_solve_vec.f:46 (3 MB)	
▶ x_solve_vec.f:46 (3 MB)	
▶ bt.f:71 (41 MB)	

General Exploration Analysis

- Predefined Analysis Type that collects different types of CPU performance events
- Good for first look at whether any CPU event categories are affecting performance
- GUI highlights those events and functions that have performance problems
- Execution pipeline slots distribution by Retiring, Front-End, Back-End, Bad Speculation
- Second level metrics for each aspect of execution pipeline to understand the reason of stalls

The screenshot shows the 'General Exploration' tool interface. The main display area lists various performance metrics under the 'Elapsed Time' category. The metrics are organized into expandable sections: Front-End Bound, Bad Speculation, Back-End Bound, Memory Latency, Memory Reissues, and Retiring. The 'Back-End Bound' section is highlighted in red, indicating a significant performance issue at 63.6%. Other metrics like 'Retiring' are also highlighted in red at 29.1%. The 'Front-End Bound' is at 6.5%, and 'Bad Speculation' is at 0.7%. The 'Memory Latency' section shows several metrics with red arrows indicating performance degradation, such as L2 Hit Bound (0.157) and SIMD Compute-to-L2 Access Ratio (15.439). The 'Retiring' section shows a Total Thread Count of 185 and a Paused Time of 0s.

Metric	Value
Elapsed Time	24.340s
Clockticks	1,834,214,751,318
Instructions Retired	1,032,885,549,326
CPI Rate	1.776
MUX Reliability	1.000
Front-End Bound	6.5%
ICache Misses	0.030
ITLB Overhead	0.017
BACLEARS	0.033
MS Entry	0.008
ICache Line Fetch	0.017
Bad Speculation	0.7%
Branch Mispredict	0.7%
SMC Machine Clear	0.000
MO Machine Clear Overhead	0.000
Back-End Bound	63.6%
Memory Latency:	
L1 Hit Rate	0.938
L2 Hit Rate	0.938
L2 Hit Bound	0.157
L2 Miss Bound	0.141
UTLB Overhead	0.009
SIMD Compute-to-L1 Access Ratio	0.894
SIMD Compute-to-L2 Access Ratio	15.439
Contested Accesses (Intra-Tile)	0.000
Page Walk	0.027
Memory Reissues:	
Split Stores	0.002
Loads Blocked by Store Forwarding	0.009
Retiring	29.1%
Total Thread Count	185
Paused Time	0s

Example: General Exploration

Bottom-up View

General Exploration General Exploration viewpoint (change) ? INTEL VTUNE AMPLIFIER XE 2017

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

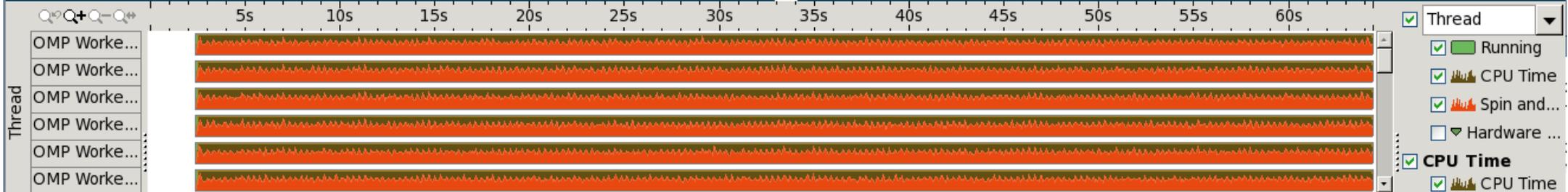
Grouping: Function / Call Stack

Function / Call Stack	Clockticks ▼	Instructions Retired	CPI Rate	Front-End Bound	Bad Speculation		Back-End Bound		Retiring	
					Branch Mispredi...	Machine Clears	Memory Bound	Core Bound		
▶ grid_intersect	14,293,700,000	11,109,300,000	1.287	7.1%	4.2%	0.4%	27.0%	42.3%	19.0%	3_tachyo
▶ sphere_intersect	8,569,000,000	7,885,000,000	1.087	6.2%	0.0%	7.4%	18.2%	45.8%	22.4%	3_tachyo
▶ func@0x1002e59d	374,300,000	311,600,000	1.201	5.1%	0.0%	15.2%	0.0%	28.9%	50.8%	libiomp5r
▶ shader	271,700,000	195,700,000	1.388	10.5%	0.0%	0.0%	31.6%	47.4%	14.0%	3_tachyo
▶ KeDelayExecutionThread	220,400,000	110,200,000	2.000	17.2%	0.0%	17.2%	0.0%	31.0%	34.5%	ntosknl.e
▶ func@0x1401514f0	212,800,000	26,600,000	8.000	31.3%	0.0%	22.3%	0.0%	19.6%	26.8%	ntosknl.e
▶ tri_intersect	195,700,000	210,900,000	0.928	4.9%	0.0%	9.7%	0.0%	61.2%	24.3%	3_tachyo
▶ __kmp_x86_pause	144,400,000	43,700,000	3.304	19.7%	0.0%	0.0%	0.0%	34.2%	72.4%	libiomp5r
▶ [wow64cpu.dll]	104,500,000	41,800,000	2.500	18.2%	0.0%	0.0%	0.0%	63.6%	54.5%	wow64cp
▶ light_intersect	87,400,000	57,000,000	1.533	10.9%	0.0%	10.9%	0.0%	67.4%	10.9%	3_tachyo
▶ func@0x10013010	81,700,000	247,000,000	0.331	0.0%	0.0%	0.0%	32.6%	32.6%	46.5%	gdiplus.d
▶ VNorm	66,500,000	34,200,000	1.944	14.3%	0.0%	14.3%	28.6%	28.6%	14.3%	3_tachyo
▶ func@0x10009c00	58,900,000	125,400,000	0.470	0.0%	0.0%	16.1%	0.0%	83.9%	0.0%	gdiplus.d
▶ SleepEx	58,900,000	17,100,000	3.444	0.0%	0.0%	16.1%	0.0%	51.6%	32.3%	kernelba:
▶ __libm_sse2_pow	55,100,000	26,600,000	2.071	17.2%	0.0%	17.2%	0.0%	65.5%	0.0%	libmmd.d
▶ func@0x140159900	55,100,000	11,400,000	4.833	0.0%	0.0%	17.2%	0.0%	65.5%	17.2%	ntosknl.e

Find real CPU stalls due to cache misses, instruction fetch misses, branch misprediction, and a lot more.

Grouping: OpenMP Region / Function / Call Stack

OpenMP Region / Function / Call Stack	OpenMP Potential Gain	Elapsed Time	Number of OpenMP threads	Instance Count	CPU Time			Spin Time	Overhead Time	Instructions Retired	CPI Rate	CPU Frequency	
					Effective Time by Utilization								
					Idle	Poor	Ok	Ideal	Over				
▶ z_solve_omp\$parallel:16@unknown:43:454	23.424s	73.882s	16	51,456	795.740s				359.793s	4.878s	31.0%	1.484	1.0
▶ y_solve_omp\$parallel:16@unknown:43:442	13.360s	62.365s	16	51,456	755.842s				217.641s	4.417s	27.8%	1.395	1.0
▶ x_solve_omp\$parallel:16@unknown:46:446	13.291s	61.117s	16	51,456	740.960s				213.383s	4.287s	26.5%	1.438	1.0
▶ compute_rhs_omp\$parallel:16@unknown:28:43	6.235s	25.777s	16	51,712	279.954s				118.079s	7.382s	7.1%	2.273	0.0
▶ [Serial - outside any region]	0s	17.738s			107.527s				156.761s	8.329s	5.6%	1.927	1.0
▶ add_omp\$parallel:16@unknown:22:33	0.913s	2.964s	16	51,456	27.503s				17.559s	1.763s	0.7%	2.839	0.0
▶ copy_x_face_omp\$parallel:16@unknown:255:26	0.591s	1.066s	16	102,912	2.902s				12.895s	1.019s	0.3%	2.224	0.0
▶ copy_y_face_omp\$parallel:16@unknown:215:22	0.554s	0.952s	16	102,912	2.611s				11.484s	0.968s	0.3%	2.219	1.0
▶ copy_x_face_omp\$parallel:16@unknown:244:25	0.609s	1.040s	16	102,912	2.445s				13.051s	1.005s	0.3%	2.095	0.0
▶ copy_y_face_omp\$parallel:16@unknown:204:23	0.599s	1.010s	16	102,912	2.303s				12.656s	0.983s	0.3%	2.163	1.0
▶ initialize_omp\$parallel:16@unknown:28:204	0.049s	0.208s	16	512	2.217s				0.989s	0.031s	0.1%	1.195	0.0
▶ exact_rhs_omp\$parallel:16@unknown:21:357	0.061s	0.195s	16	256	1.979s				1.067s	0.015s	0.1%	1.681	1.0
▶ error_norm_omp\$parallel:16@unknown:27:54	0.005s	0.021s	16	256	0.201s				0.116s	0.011s	0.0%	2.104	1.0
▶ rhs_norm_omp\$parallel:16@unknown:86:107	0.004s	0.011s	16	256	0.074s				0.040s	0.014s	0.0%	2.024	1.0



Optimization Notice

Example – Hardware Events Viewpoint

General Exploration Hardware Events viewpoint (change) ?

Collection Log Analysis Target Analysis Type Summary Event Count Sample Count Uncore Event Count Caller/Calle

Hardware Events

Hardware Event Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample
BACLEAR.S.ALL	141,602,124	59	200003
CPU_CLK_UNHALTED.REF_TSC	73,650,110,475	36,825	2000003
CPU_CLK_UNHALTED.THREAD	79,282,118,923	39,641	2000003
CPU_CLK_UNHALTED.THREAD_P	79,248,118,872	3,302	2000003
CYCLES_DIV_BUSY.ALL	0	0	2000003
FETCH_STALL.ICACHE_FILL_PENDING_CYCLES	285,604,284	119	200003
INST_RETIRED.ANY	10,042,015,063	5,021	2000003
MACHINE_CLEAR.S.FP_ASSIST	0	0	200003
MACHINE_CLEAR.S.MEMORY_ORDERING	0	0	200003
MACHINE_CLEAR.S.SMC	0	0	200003
MEM_UOPS_RETIRED.ALL_LOADS	2,913,643,704	1,214	200003
MEM_UOPS_RETIRED.ALL_STORES	974,414,616	406	200003
MEM_UOPS_RETIRED.HITM	0	0	200003
MEM_UOPS_RETIRED.L1_MISS_LOADS	1,048,815,732	437	200003
MEM_UOPS_RETIRED.L2_HIT_LOADS_PS	993,614,904	414	200003
MEM_UOPS_RETIRED.L2_MISS_LOADS_PS	28,802,016	24	100007
MEM_UOPS_RETIRED.UTLB_MISS_LOADS	40,800,612	17	200003
MS_DECODED.MS_ENTRY	4,800,072	2	200003
NO_ALLOC_CYCLES.MISPREDICTS	19,805,097,072	8,252	200003
NO_ALLOC_CYCLES.NOT_DELIVERED	7,183,307,748	2,993	200003
PAGE_WALKS.CYCLES	271,204,068	113	200003
PAGE_WALKS.I_SIDE_CYCLES	36,000,540	15	200003
REHABQ.LD_SPLITS_PS	840,012,600	350	200003
UOPS_RETIRED.ALL	10,320,015,480	430	2000003
UOPS_RETIRED.MS	96,000,144	4	2000003
UOPS_RETIRED.PACKED_SIMD	4,437,666,564	1,849	200003
UOPS_RETIRED.SCALAR_SIMD	124,801,872	52	200003

Uncore Event Count

Uncore Event Type	Uncore Event Count
UNC_M_CAS_COUNT.RD[UNIT0]	121,667,400
UNC_M_CAS_COUNT.RD[UNIT1]	121,277,947
UNC_M_CAS_COUNT.RD[UNIT2]	122,910,148
UNC_M_CAS_COUNT.RD[UNIT3]	122,185,974
UNC_M_CAS_COUNT.RD[UNIT4]	120,999,173
[Others]	285,397,113

44% of packed and scalar SIMD over all Uops.retired

Example – Hardware Events Viewpoint

General Exploration Hardware Events viewpoint (change) ? INTEL VTUNE AMPLIFIER XE 2017

Collection Log Analysis Target Analysis Type Summary **Event Count** Sample Count Caller/Callee Top-down Tree Platform

Grouping: Function / Call Stack

Function / Call Stack	INST_RETIRED.ANY ▼	CPU_CLK_UNHALTED.THREAD	CPU_CLK_UNHALTED.REF_TSC	RECYCLEQ.LD_BLOCK_ST_FORWARD_PS	CPU_CLK_UNHALTED.THREAD_F
▶ binvrhs	962,516,100,000	840,516,300,000	837,484,700,000	6,908,303,623	825,683,238,52
▶ __kmp_wait_template<kmp_flag_64>	611,201,500,000	1,210,440,400,000	1,214,688,800,000	5,200,078	1,193,479,790,21
▶ z_solve_omp\$parallel@43	347,128,600,000	479,744,200,000	478,821,200,000	2,405,036,075	466,622,699,93
▶ y_solve_omp\$parallel@43	339,554,800,000	449,736,300,000	448,412,900,000	2,233,433,501	442,780,664,17
▶ x_solve_omp\$parallel@46	309,962,900,000	435,763,900,000	432,552,900,000	2,589,638,844	421,590,632,38
▶ matmul_sub	203,911,500,000	466,079,900,000	463,160,100,000	9,445,941,687	453,986,680,97
▶ compute_rhs_omp\$parallel@28	132,531,100,000	340,174,900,000	338,747,500,000	941,214,118	331,864,497,79
▶ matvec_sub	102,785,800,000	192,093,200,000	191,388,600,000	20,800,312	188,552,282,82
▶ [vmlinux]	77,374,700,000	235,730,300,000	238,954,300,000	405,606,084	336,024,504,03
▶ __kmp_yield	76,654,500,000	87,428,900,000	87,451,000,000	2,600,039	83,954,125,93
▶ __kmp_x86_pause	20,668,700,000	28,120,300,000	28,431,000,000	0	26,754,040,13
▶ binvrhs	16,421,600,000	20,715,500,000	22,124,700,000	145,602,184	19,708,029,56
▶ kmp_basic_flag<unsigned long long>::notdone_c	14,040,000,000	72,924,800,000	74,201,400,000	0	70,954,106,43
▶ __kmp_x86_pause	13,910,000,000	6,919,900,000	7,147,400,000	0	6,110,009,16
▶ add_omp\$parallel_for@22	6,510,400,000	29,164,200,000	28,827,500,000	257,403,861	28,314,042,47
▶ __kmp_wait_template<kmp_flag_64>	5,522,400,000	12,256,400,000	13,812,500,000	0	11,492,017,23

Example – Gflops and FPU utilization

General Exploration HPC Performance Characterization viewpoint (change) ?

Collection Log Analysis Target Analysis Type Summary Bottom-up

Elapsed Time[?]: 64.488s

GFLOPS Upper Bound[?]: 92.897

⊖ **CPU Utilization[?]: 16.5%** 📉

Average CPU Usage[?]: 42.213 Out of 256 logical CPUs

⊖ **Top OpenMP Processes by MPI Communication Spin Time**

This section lists processes sorted by MPI Communication Spin time. The lower MPI Communication Spin time, the more efficiency metrics by MPI processes laying on the critical path

Process	PID	OpenMP Potential Gain [?] (%) [?]	Serial Time [?] (%) [?]
bt-mz.C.4 (rank 0)	238538	15.057s 📉 24.3% 📉	3.962s 6.4%
bt-mz.C.4 (rank 2)	238540	13.994s 📉 22.5% 📉	4.492s 7.2%
bt-mz.C.4 (rank 3)	238542	14.726s 📉 23.7% 📉	4.558s 7.3%
bt-mz.C.4 (rank 1)	238550	15.920s 📉 25.6% 📉	4.726s 7.6%

⊖ **CPU Usage Histogram**

⊖ **Back-End Bound[?]: 39.6%**

L2 Hit Bound[?]: 0.080
L2 Miss Bound[?]: 0.167 📉

⊖ **FPU Utilization Upper Bound[?]: 3.7%** 📉

⊖ **GFLOPS Upper Bound[?]: 92.897**

Scalar GFLOPS Upper Bound[?]: 24.134
Packed GFLOPS Upper Bound[?]: 68.763

⊖ **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 Upper Bound [?]	Loop Characterization [?]
binvcrhs	645.711s	3.8% 📉	
z_solve_\${omp\$parallel}@43	369.177s	3.2% 📉	
matmul_sub	357.102s	13.6% 📉	
y_solve_\${omp\$parallel}@43	345.732s	3.3% 📉	
x_solve_\${omp\$parallel}@46	333.504s	3.2% 📉	
[Others]	675.905s	N/A*	

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

The HPC Performance Characterization Analysis

Threading: CPU Utilization

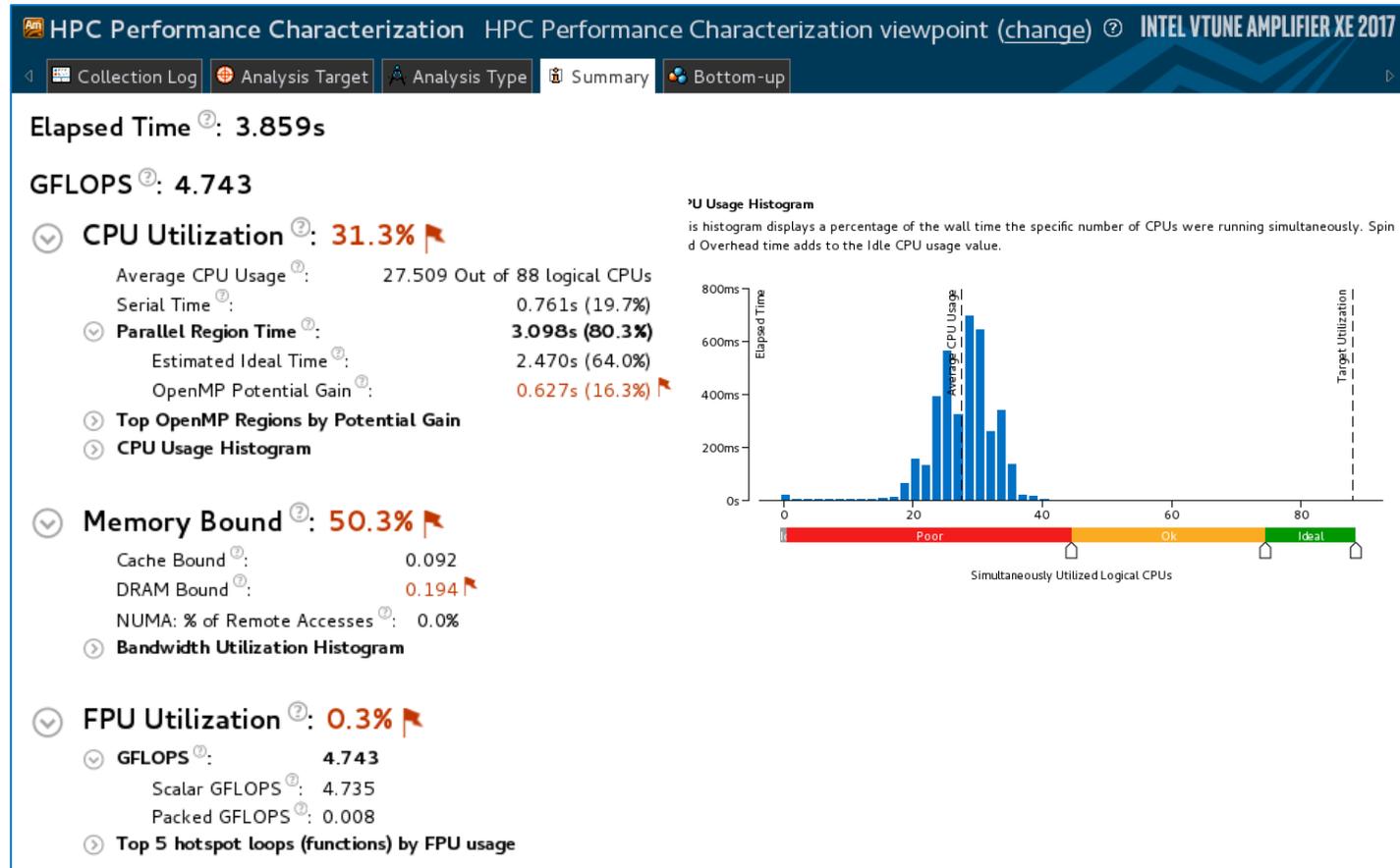
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use hotspot OpenMP region analysis for more detail

Memory Access Efficiency

- Stalls by memory hierarchy
- Bandwidth utilization
- Tip: Use Memory Access analysis

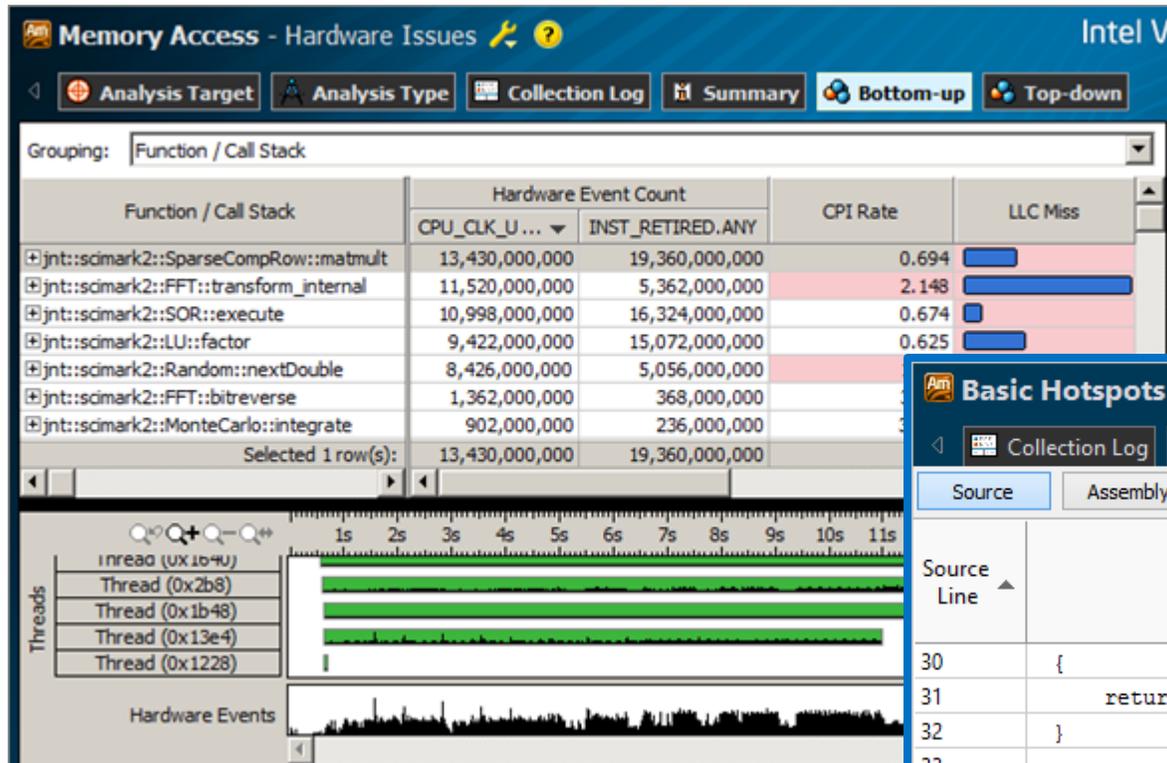
Vectorization: FPU Utilization

- FLOPS[†] estimates from sampling
- Tip: Use Intel Advisor for precise metrics and vectorization optimization

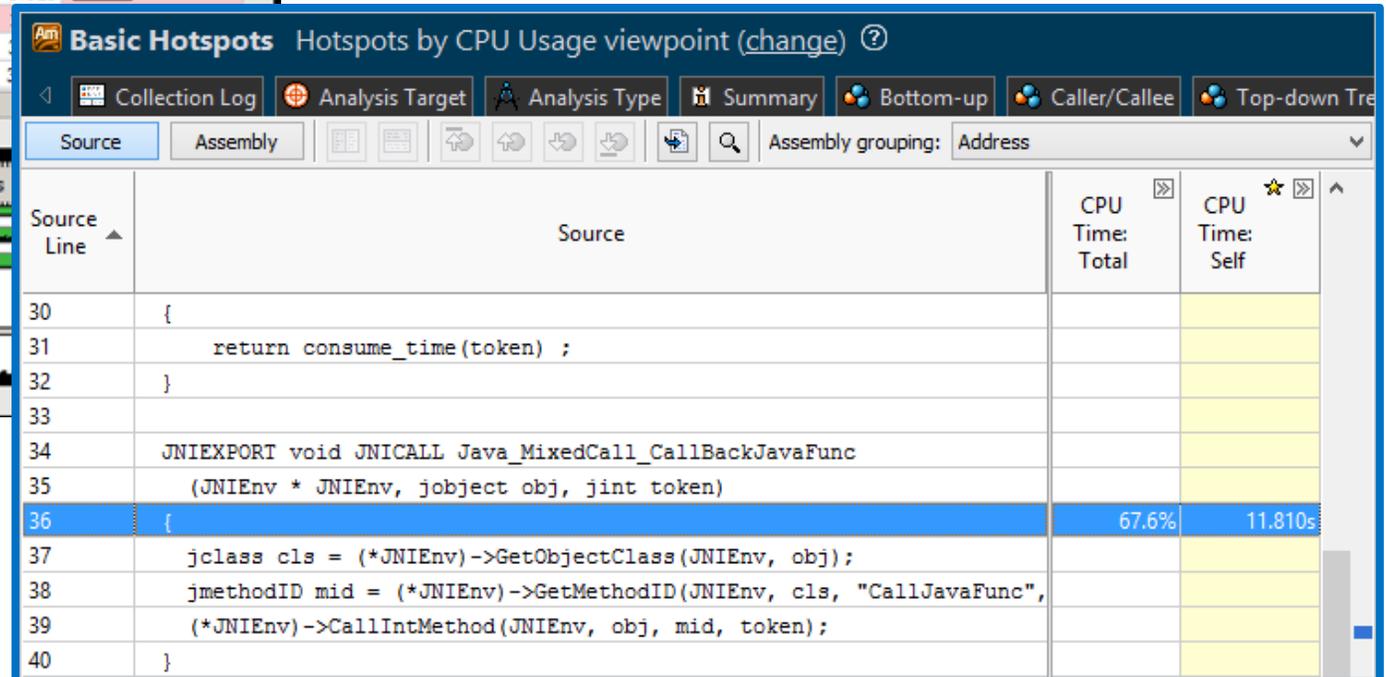


[†] For 3rd, 5th, 6th Generation Intel® Core™ processors and second generation Intel® Xeon Phi™ processor code named Knights Landing.

Java Analysis



- Multiple simultaneous JVMs
- Sampling is fast / unobtrusive
- Mixed Java / C++ / Fortran
- See results on the Java source



Optimize Private Cloud-Based Applications

Profile Enterprise Applications

- Native C, C++, Fortran*
- Attach to running Java* services (e.g., Mail)
- Profile Java daemons without restart

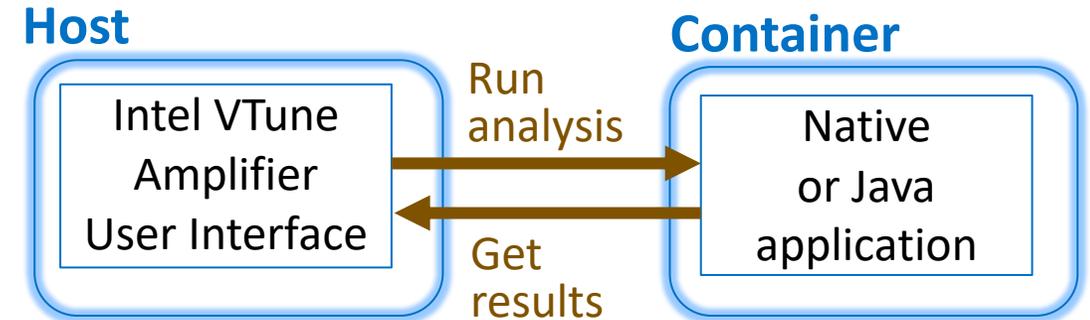
Accurate, Low-Overhead Data Collection

- Advanced hotspots and hardware events
- Memory analysis
- Accurate stack information for Java and HHVM*

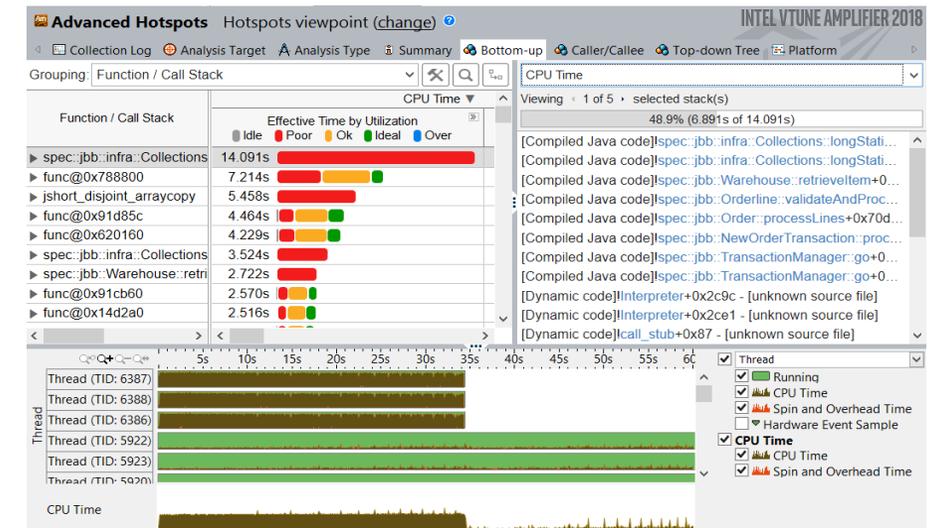
Popular Containers Supported

- Docker*
- Mesos*

Software collectors (e.g., locks & waits) and Python* profiling are not currently available for containers.



- No container configuration required
- Detection of the container is automatic



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.

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

Storage Device Analysis (HDD, SATA or NVMe SSD)

Intel® VTune™ Amplifier

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

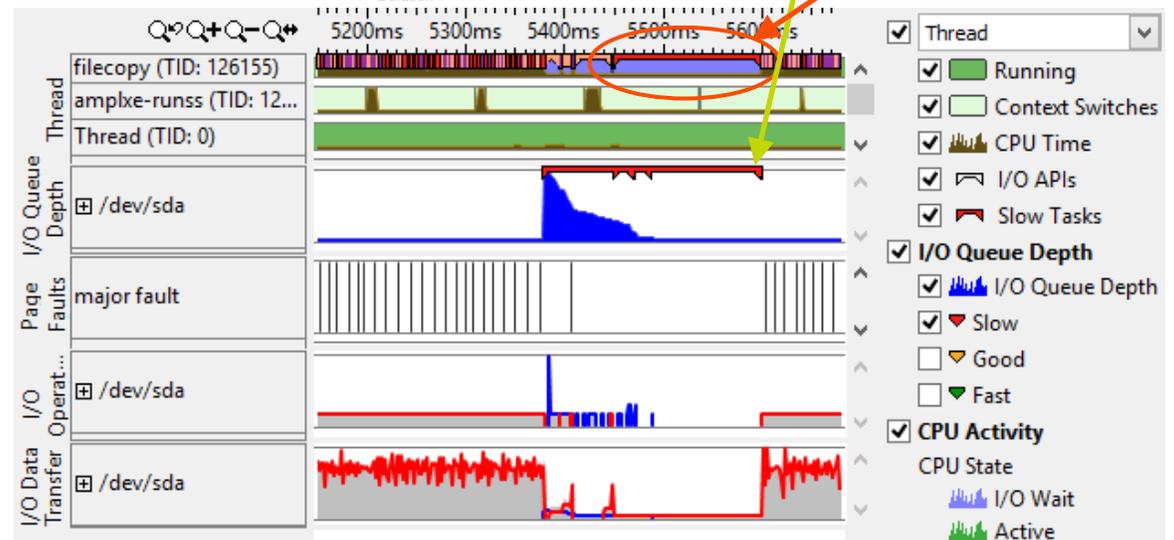
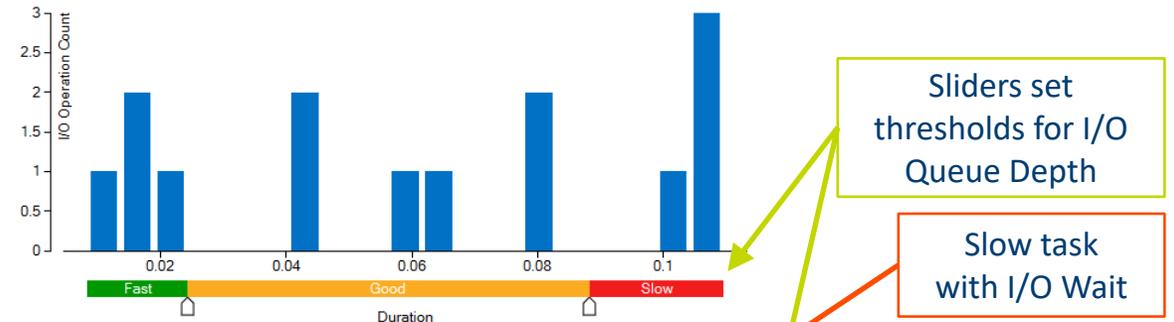
Latency analysis

- 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



Application Performance Snapshot (APS)

High-level **overview** of application performance

Identify primary optimization areas and **next steps** in analysis

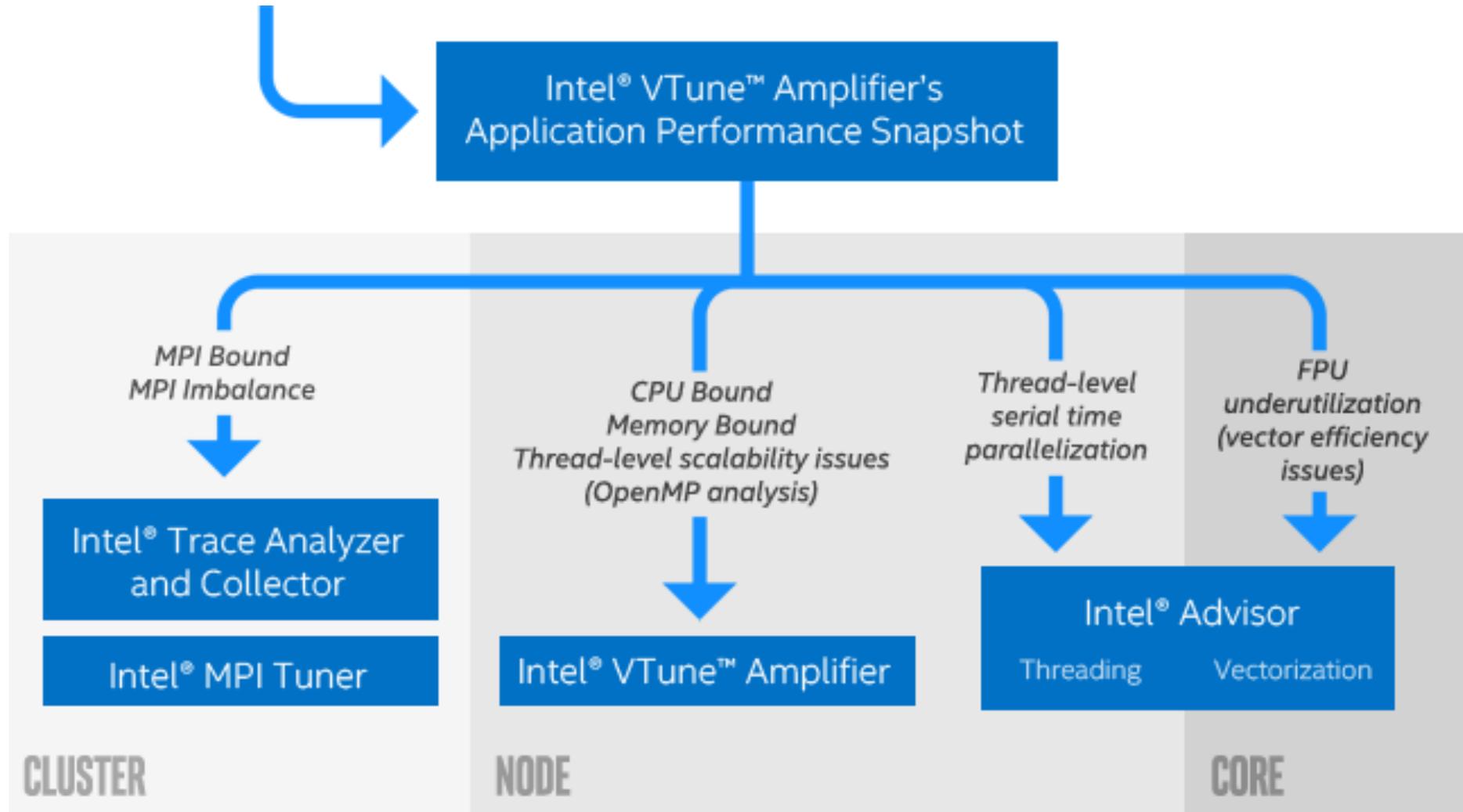
Easy to install, run, explore results with CL or HTML reports

Scales to large jobs

Multiple methods to obtain

- Part of Intel® Parallel Studio XE 2018
 - <http://intel.ly/intel-parallel-studio-xe-2018-beta>
- Separate **free** download (110Mb) from APS page
 - <https://software.intel.com/sites/products/snapshots/application-snapshot/>

Performance Optimization Workflow based on APS



APS Usage

Setup Environment

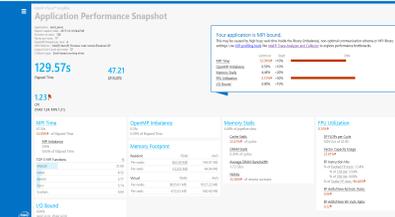
- `>source <APS_Install_dir>/apsvars.sh`

Run Application

- `>aps <application and args>`
- MPI: `>mpirun <mpi options> aps <application and args>`

Generate Report on Result Folder

- `>aps -report <result folder>`

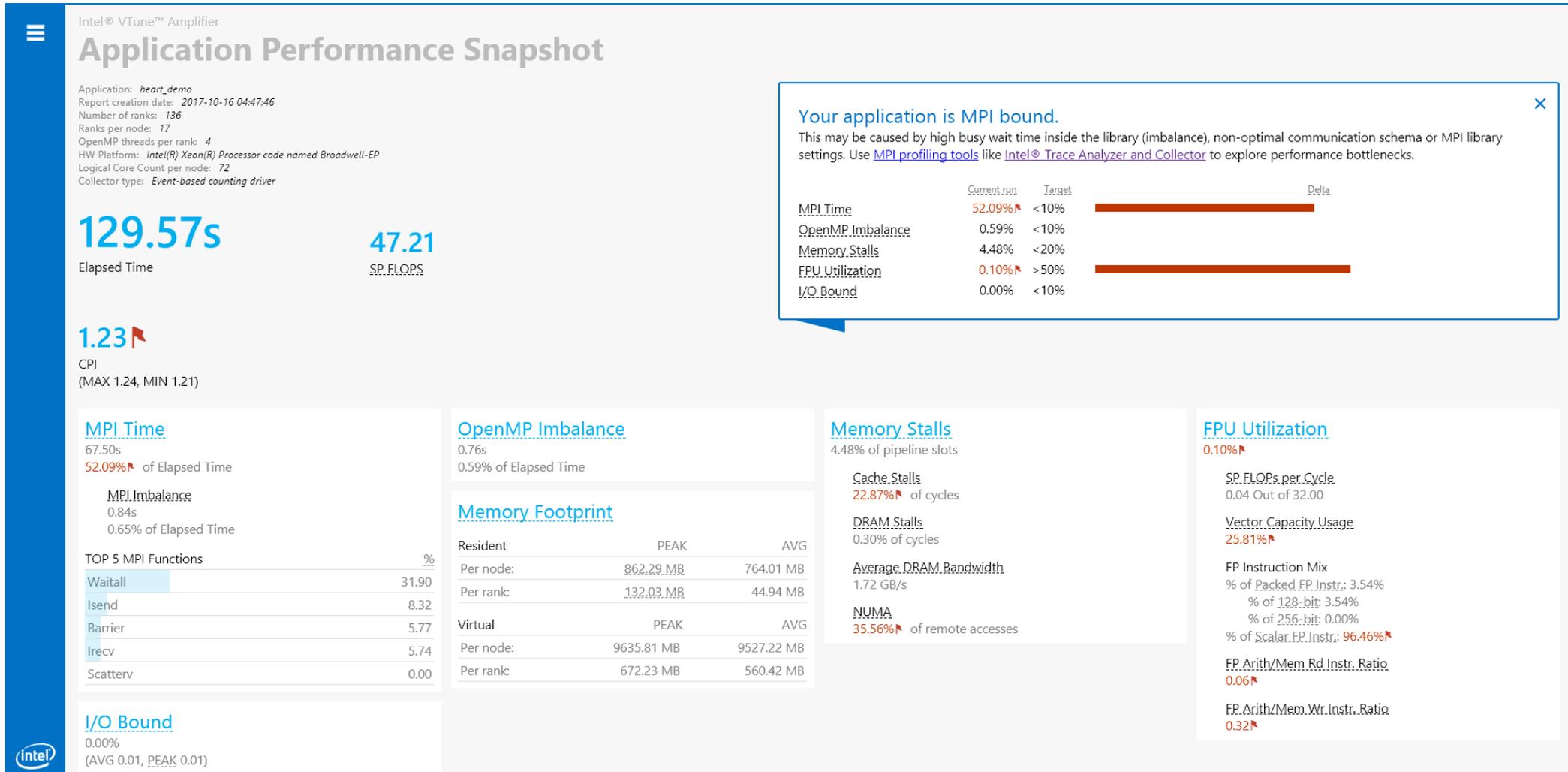


Generate CL reports with detailed MPI statistics on Result Folder

- `aps-report -<option> <result folder>`

Rank	Rank	Volume (MB)	Volume (s)	Transfers
0020	--> 0024	84.55	1.56	13477
0026	--> 0028	84.55	1.56	13477
0024	--> 0020	84.55	1.56	13477
0028	--> 0024	83.43	1.55	13477
0022	--> 0023	83.43	1.54	13477
[Filtered out 16 lines]				
0012	--> 0011	69.60	1.29	13477
0020	--> 0019	69.59	1.29	13477
0026	--> 0025	69.78	1.27	13477
0025	--> 0024	68.38	1.27	13477
0022	--> 0023	69.38	1.27	13477
[Filtered out 17 lines]				
0016	--> 0015	58.01	1.09	13477
0026	--> 0025	57.69	1.05	13477
0007	--> 0008	56.98	1.05	13477
0030	--> 0031	54.74	1.03	13477
0006	--> 0007	54.44	1.01	13477
[Filtered out 1100 lines]				
TOTAL		5403.22	100.00	1418619
AVG		4.47	0.00	12.4

APS HTML Report



APS Command Line Reports - Summary

```
| Summary information
|-----
| Application      : heart_demo.test02
| Number of ranks : 8
| Used statistics  : stat_20170502/
| Creation date   : 2017-05-02 11:44:27
|
| Your application has significant OpenMP imbalance. Use OpenMP profiling tools like Intel(R) VTune(TM) Amplifier
| to see the imbalance details.
|
| Elapsed time:          73.19 sec
| CPI Rate:              4.01
| The CPI value may be too high.
| This could be caused by such issues as memory stalls, instruction starvation,
| branch misprediction, or long latency instructions.
| Use Intel(R) VTune(TM) Amplifier General Exploration analysis to specify
| particular reasons of high CPI.
| MPI Time:              11.48 sec          15.69%
| Your application is MPI bound. This may be caused by high busy wait time
| inside the library (imbalance), non-optimal communication schema or MPI
| library settings. Explore the MPI Imbalance metric if it is available or use
| MPI profiling tools like Intel(R) Trace Analyzer and Collector to explore
| possible performance bottlenecks.
| MPI Imbalance:         3.36 sec          4.59%
| OpenMP Imbalance:     22.52 sec         30.77%
| The metric value can indicate significant time spent by threads waiting at
| barriers. Consider using dynamic work scheduling to reduce the imbalance where
| possible. Use Intel(R) VTune(TM) Amplifier HPC Performance Characterization
| analysis to review imbalance data distributed by barriers of different lexical
| regions.
```

Tip:

>aps -report=<my_result_dir> | grep -v "|"
eliminating verbose descriptions

APS Command Line Reports – Advanced MPI statistics (1/3)

- MPI Time per rank
 - `aps-report -t <result>`

```
MPI Time per Rank
```

Rank	LifeTime(sec)	MPI Time(sec)	MPI Time (%)	Imbalance(sec)	Imbalance (%)
0007	72.52	14.31	19.74	4.84	6.67
0004	72.53	11.57	15.96	3.26	4.50
0005	72.52	11.40	15.72	3.20	4.42
0006	72.51	11.11	15.32	3.17	4.37
0000	72.49	11.08	15.29	4.33	5.97
0001	72.52	10.95	15.10	3.01	4.15
0002	72.49	10.79	14.88	2.57	3.55
0003	72.50	10.64	14.68	2.50	3.45
TOTAL	580.07	91.86	15.84	26.88	4.63
AVG	72.51	11.48	15.84	3.36	4.63

APS Command Line Reports – Advanced MPI statistics (2/3)

Message Size Summary by all ranks

- `aps-report -m <result>`

```
| Message Sizes summary for all ranks  
|-----  
| Message size(B)      Volume (MB)      Volume (%)      Transfers      Time (sec)      Time (%)  
|-----  
|           8           1.49           0.09           195206         27.79           37.93  
|          176           0.41           0.02           2420           27.67           37.78  
|           4           0.00           0.00           1150           15.55           21.22  
|        100264         115.89           6.94           1212            0.27            0.37  
|         98400         113.74           6.81           1212            0.19            0.26  
|         66256          38.29           2.29            606            0.17            0.23  
| [filtered out 57 lines]  
|-----  
| TOTAL                1670.60         100.00         265160         73.25           100.00  
|
```

APS Command Line Reports – Advanced MPI statistics (3/3)

Data Transfers for Rank-to-Rank Communication

- `aps-report -x <result>`

And many others – check

- `aps-report -help`

```
|-----|
| Rank --> Rank           Volume (MB)           Volume (%)           Transfers
|-----|-----|-----|-----|
| 0023 --> 0024           84.35             1.56                13477
| 0025 --> 0026           84.35             1.56                13477
| 0024 --> 0025           84.15             1.56                13477
| 0021 --> 0022           83.84             1.55                13477
| 0022 --> 0023           83.43             1.54                13477
| [filtered out 16 lines]
| 0012 --> 0011           69.60             1.29                13477
| 0020 --> 0019           69.29             1.28                13477
| 0026 --> 0025           68.78             1.27                13477
| 0025 --> 0024           68.38             1.27                13477
| 0022 --> 0021           68.38             1.27                13477
| [filtered out 17 lines]
| 0016 --> 0015           58.81             1.09                13477
| 0028 --> 0027           57.69             1.07                13477
| 0007 --> 0008           56.98             1.05                13477
| 0030 --> 0031           54.74             1.01                13477
| 0006 --> 0007           54.44             1.01                13477
| [filtered out 1108 lines]
|-----|-----|-----|-----|
| TOTAL                   5403.22           100.00              1415619
| AVG                     4.67              0.09                1224
```

Collection Control API

Since 2018 Update 2

To measure a particular application phase or exclude initialization/finalization phases use:

MPI:

- Pause: `MPI_Pcontrol(0)`
- Resume: `MPI_Pcontrol(1)`

MPI or Shared memory applications:

- Pause: `__itt_pause()`
- Resume: `__itt_resume()`
 - See [how to configure](#) the build of your application to use itt API

Tip: use `aps -start-paused` option allows to start application without profiling and skip initialization phase

Python

Profiling Python is straightforward in VTune™ Amplifier, as long as one does the following:

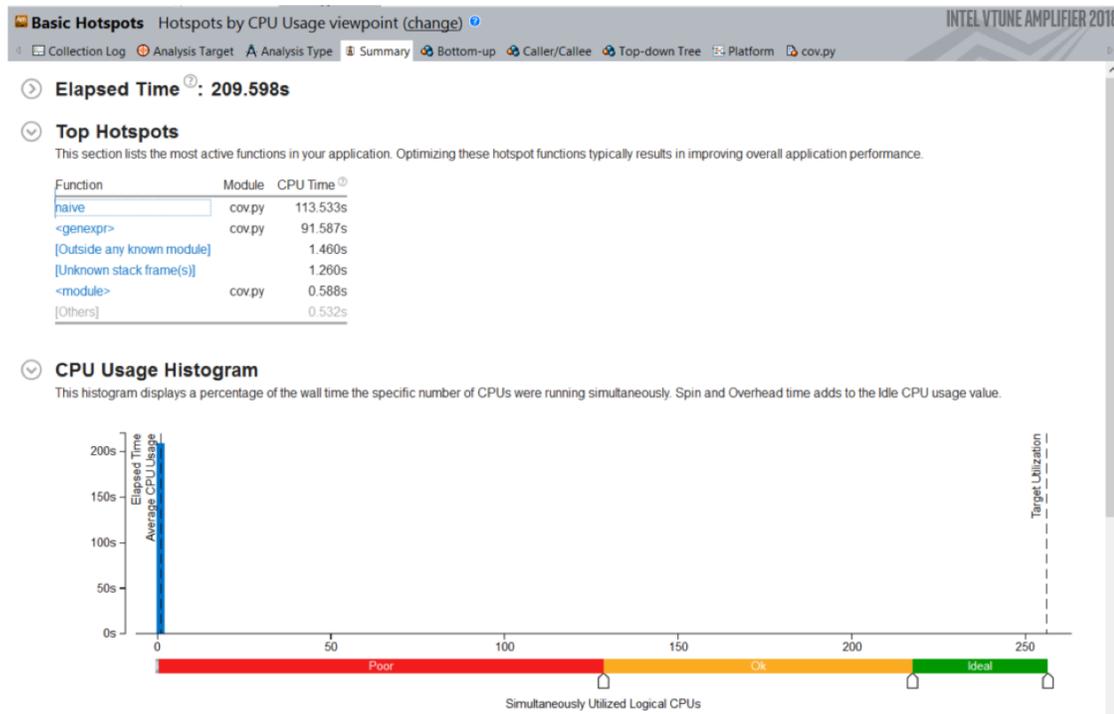
- The “application” should be the full path to the python interpreter used
- The python code should be passed as “arguments” to the “application”

Example:

```
mpirun -n 1 -N 1 amplxe-cl -c hotspots -r res_dir \  
      -- /usr/bin/python3 mycode.py myarguments
```

Simple Python Example

```
mpirun -n 1 -N 1 amplxe-cl -c hotspots -r vt_pytest \  
-- /usr/bin/python ./cov.py naive 100 1000
```



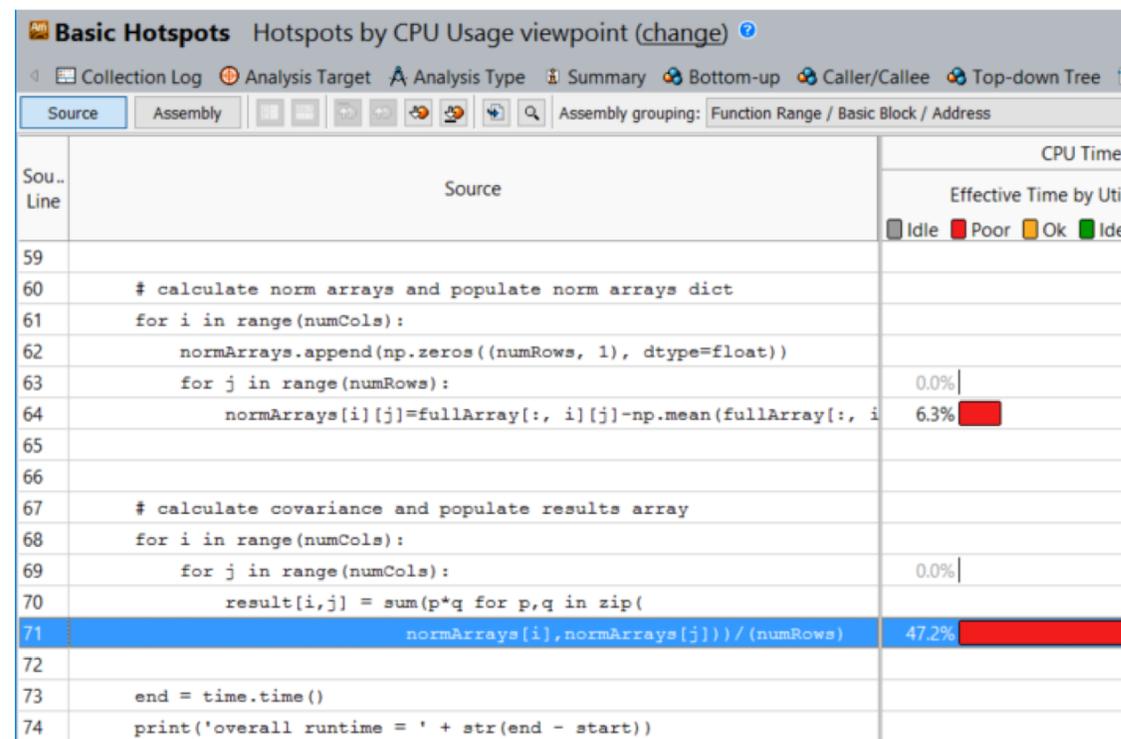
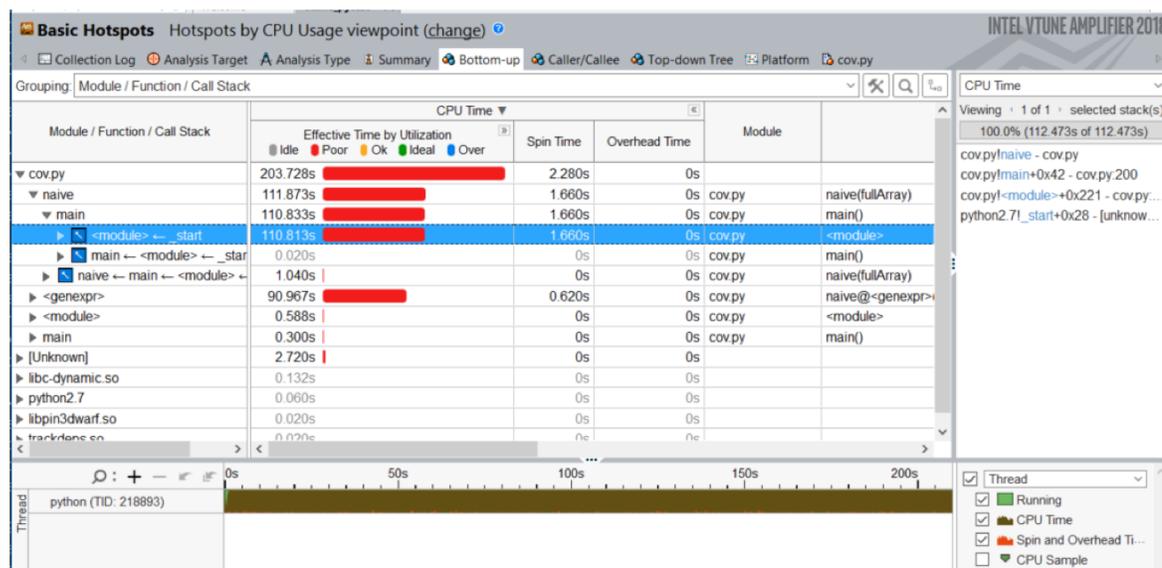
Naïve implementation of the calculation of a covariance matrix

Summary shows:

- Single thread execution
- Top function is “naive”

Click on top function to go to Bottom-up view

Bottom-up View and Source Code



Inefficient array multiplication found quickly
We could use numpy to improve on this

Note that for mixed Python/C code a Top-Down view can often be helpful to drill down into the C kernels

VTune Command Line Analysis Configuration

How to Run VTune on MPI Applications

```
><mpi_launcher> -n N <vtune_command_line> ./app_to_run
```

- `>srun -n 48 -N 16 amplxe-cl -collect memory-access -trace-mpi -r result_dir ./my_mpi_app`
 - `>mpirun -n 48 -ppn 16 amplxe-cl -collect advanced-hotspots -r result_dir ./my_mpi_app`
 - Encapsulates ranks to per-node result directories suffixed with hostname
 - `result_dir.hostname1` with 0-15, `result_dir.hostname2` with 16-31, `result_dir.hostname3` with 32-47
-  Add `-trace-mpi` option for VTune CL to enable per-node result directories for non-Intel MPIS
- Works for software and Intel driver-based collectors

VTune Command Line Analysis Configuration

Selective Rank Profiling

Superposition of application to launch and VTune command line for selective ranks to reduce trace size

Example: profile rank 1 from 0-15:

```
>mpirun -n 1 ./my_app : -n 1 <vtune_command_line> -- ./my_app : -n 14 ./my_app
```

- In the case of Intel MPI launcher `-gtool` option can be used:

Example: profile ranks 3, 7, 11-13 from 0-15:

```
>mpirun -gtool "amplxe-cl -collect advanced-hotspots -r result_dir:3,7,11-13" ./my_app
```

Analysis Workflow

Result finalization and viewing on KNL target might be slow

Use the recommended workflow:

1. Run collection on KNL deferring finalization to host:

```
>amplxe-cl -collect memory-access -no-auto-finalize -r <my_result_dir> ./my_app
```

2. Finalize the result on the host

- Provide search directories to the binaries of interest for resolving with `-search-dir` option

```
>amplxe-cl -finalize -r <my_result_dir> -search-dir <my_binary_dir>
```

3. Generate reports, work with GUI

```
>amplxe-cl -report hotspots -r <my_result_dir>
```

Legal Disclaimer & Optimization Notice

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.

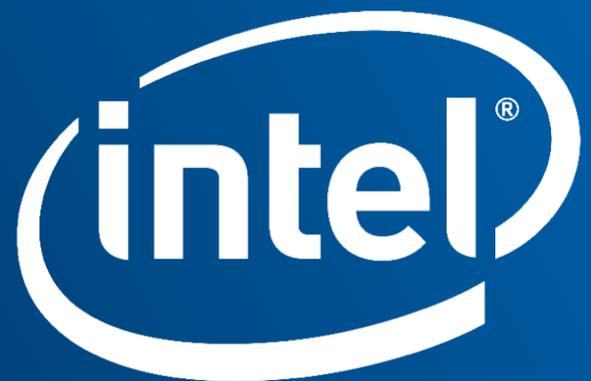
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may 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.

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