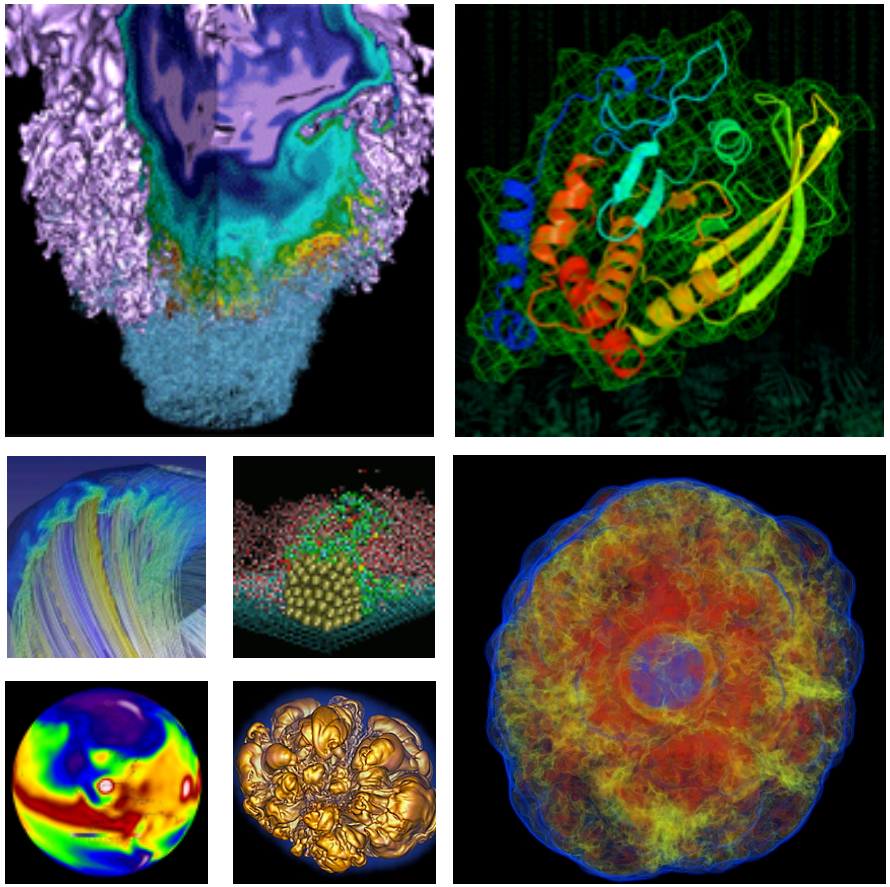
Using VTune on Cori KNL Nodes



Zhengji Zhao
User Engagement Group

Cori KNL Training (Spring 2017)
June 9, 2017, Berkeley CA

VTune Introduction

- **An Intel profiling tool**
  - Focus on on-node performance
  - Works with serial and parallel codes
- **Provides both command line interface "amplxe-cl" and a gui "amplxe-gui"**
  - Recommended to use the command line tool to collect data via batch jobs, and then use the gui to display data later on a login node
  - NX is recommended to speedup the gui display
- **Available via the vtune modules on Cori**
  - The current default is 2017.up2 on Cori
  - VTune interfaces could be different from different versions

Running VTune on Cori KNL nodes

1. **Compile your codes with −g and link the code dynamically with the −dynamic option for ftn, cc or CC**
   - The **-debug inline-debug-info** is recommended to generate enhanced debug information for the inlined codes.
   - Use optimization flags as in the production use (-O0 is not required)
   - Intel compilers are recommended

```
module swap craype-haswell craype-mic-knl
ftn  -dynamic −qopenmp -O3 −g -debug inline-debug-info jacobi.f90  -o jac.x
```

Running VTune on Cori KNL nodes --continued

2. **To run VTune, you have to:**
   - Use #SBATCH --perf=vtune or salloc −perf=vtune with a batch or interactive job
   - Load a vtune module before submit jobs or using salloc to request nodes
   - Use a Lustre file system, $SCRATCH

```
#Continued …
mkdir $SCRATCH/vtune_runs
cp −p jac.x $SCRATCH/vtune_runs
cd $SCRATCH/vtune_runs
Module load vtune
salloc -N2 −C knl,quad,cache −qos=interactive −t 1:00:00 −perf=vtune
 wait …
```

Running VTune on Cori KNL nodes --continued

- **Sample job script to run VTune on Cori KNL**

```
#!/bin/bash
#SBATCH -J vtune_test
#SBATCH -N 2
#SBATCH -C knl,quad,cache  #running in quad,cashe mode
#SBATCH −p debug
#SBATCH -t 00:30:00
#SBATCH --perf=vtune
#SBATCH -o myjob.o%j

module load vtune

srun -n 128 -c 4 amplxe-cl -collect general-exploration -r res_dir -trace-mpi −
finalization-mode=none  -- ./a.out
```

Running VTune on Cori KNL nodes --continued

3. Collect data (on compute nodes)

```
#Continued …

module load vtune
export OMP_PROC_BIND=true
export OMP_PLACES=threads
export OMP_NUM_THREADS=8

# to run memory-access analysis:
srun -n 8  -c 32 --cpu_bind=cores amplxe-cl −collect memory-access -r
res_dir -trace-mpi −finalization-mode=none -- ./jac.x
```

- The -finalization-mode=none is used so to finalize data later on a faster node (e.g, on a login node)
- The -data-limit=0 option can be used to get >500MB profiling data
- do amplxe-cl −help collect for the available analysis types, and command line options; do amplxe-cl −help collect <analysis name> for the knob option fore the analysis.

Running VTune on Cori KNL nodes --continued

- **Available analyses**

```
advanced-hotspots  Advanced Hotspots
concurrency  Concurrency
cpugpu-concurrency  CPU/GPU Concurrency
disk-io            Disk Input and Output
general-exploration  General Exploration
gpu-hotspots  GPU Hotspots
hotspots           Basic Hotspots
hpc-performance  HPC Performance Characterization
locksandwaits  Locks and Waits
memory-access  Memory Access
sgx-hotspots  SGX Hotspots
tsx-exploration  TSX Exploration
tsx-hotspots  TSX Hotspots
```

Example amplxe-cl command line with more knob options:

srun -n 4 -c 24 amplxe-cl -collect memory-access -knob analyze-openmp=true -knob analyze-mem-objects=true -r res_dir -trace-mpi -- ./a.out

Running VTune on Cori KNL nodes --continued

4. **Display the collected data using the VTune gui on a login node**
   - amplxe-gui -- > Open results, then the Vtune gui finalize the data first and then display
   - Or you can finalize the data outside the gui by

```
amplxe-cl −finalize −finalization-mode=full −r <your result dir>
-search-dir <path to your binary>
```

Memory access – Memory usage viewpoint

Memory Access Memory Usage viewpoint (change)

INTEL VTUNE AMPLIFIER XE 2017

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

Memory Bound:

L2 Hit Rate	89.0%	
L2 Hit Bound	9.6%	of Clockticks
L2 Miss Bound	16.2%	of Clockticks
MCDRAM Cache Bandwidth Bound	4.0%	
MCDRAM Flat Bandwidth Bound	4.9%	
DRAM Bandwidth Bound	0.0%	of Elapsed Time
L2 Miss Count	7,038,211,140	
MCDRAM Hit Rate	98.4%	
MCDRAM HitM Rate	94.3%	
Total Thread Count	65	
Paused Time	0s	

Memory Usage

This view uses hardware event-based metrics to characterize how effectively your application uses memory resources and identify potential memory access related issues like excessive access to remote memory on NUMA platforms, hitting DRAM or Intel QPI bandwidth limit, and others. It provides various performance metrics for both the application code and memory objects arrays.

☑ Show the analysis description when result opens

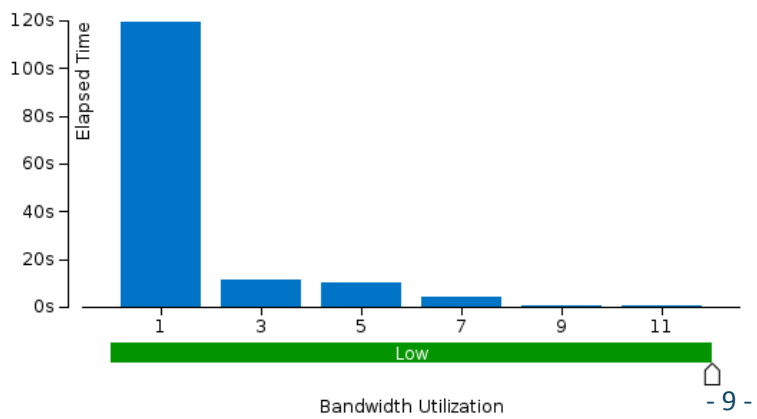
Bandwidth Utilization

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

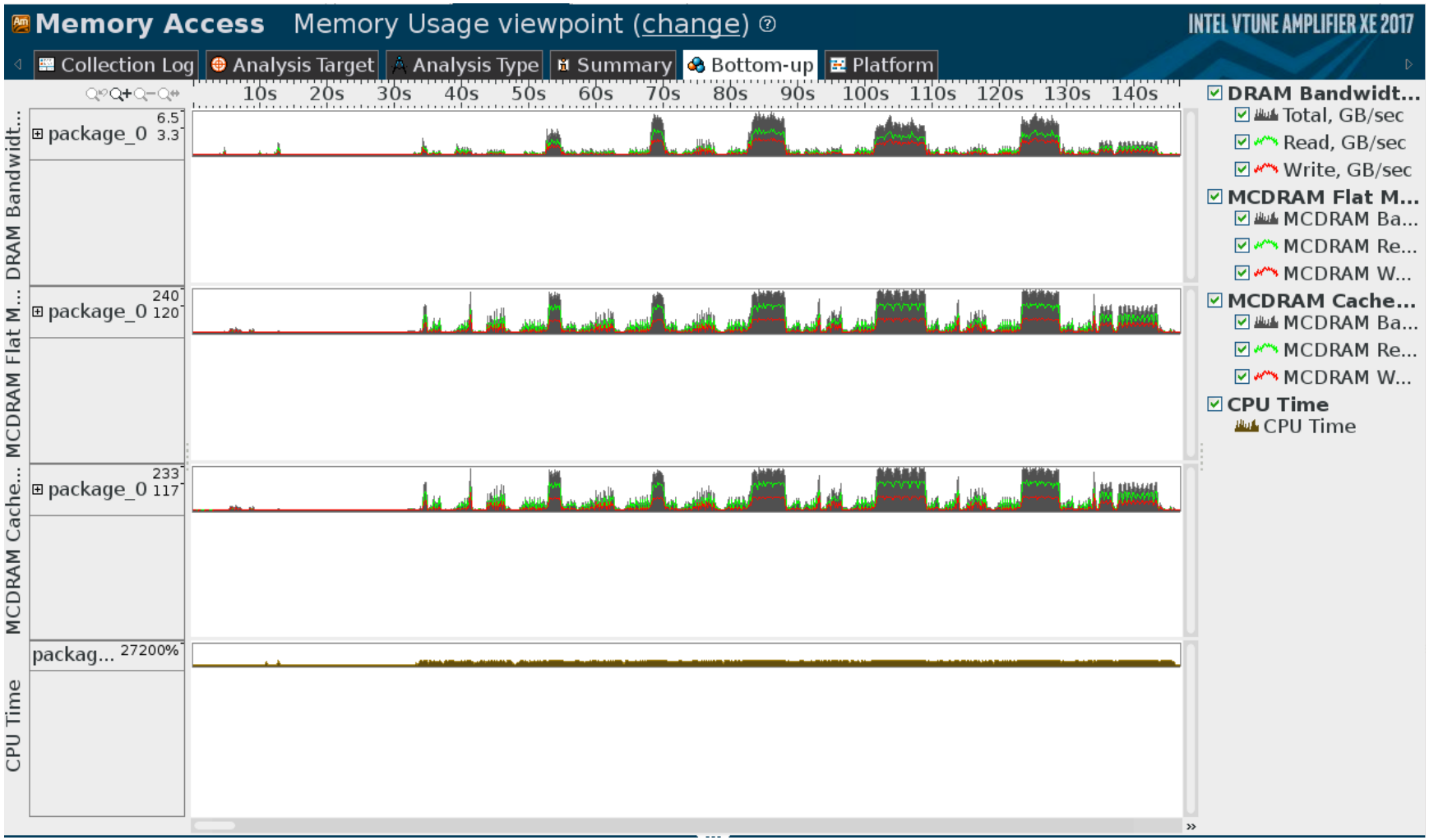
Bandwidth Domain: DRAM, GB/sec

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 High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and QPI bandwidth.



Memory Access –Memory usage bottom-up viewpoint



Memory access – Hotspots viewpoint

Memory Access Hotspots viewpoint (change) INTEL VTUNE AMPLIFIER XE 2017

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

Elapsed Time: 146.850s

CPU Time:	6696.230s
Instructions Retired:	6,609,218,000,000
CPI Rate:	1.517
CPU Frequency Ratio:	1.069
Total Thread Count:	65
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
[libiomp5.so]	libiomp5.so	2983.694s
[vasp_std]	vasp_std	2559.634s
[vmlinux]	vmlinux	538.931s
[libmpich_intel.so.3.0.1]	libmpich_intel.so.3.0.1	304.960s
[libugni.so.0.6.0]	libugni.so.0.6.0	157.160s
[Others]		151.850s

Average Bandwidth

Package	Total, GB/sec	Read, GB/sec	Write, GB/sec
package_0	1.137	0.718	0.419

Hotspots

This view uses hardware event-based collection to display code regions (modules, functions, and so on) that consume a lot of CPU time (hotspots).

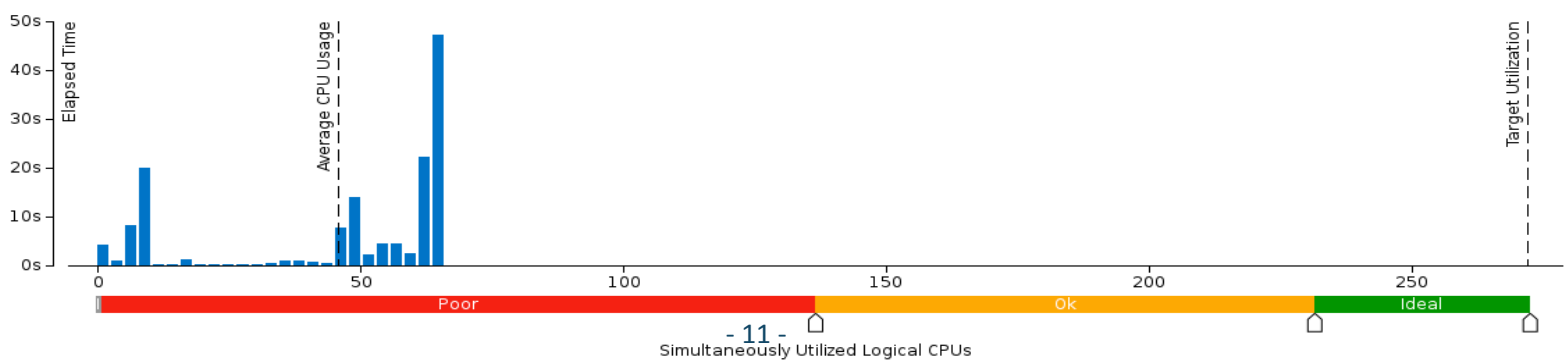
Use this view to:
- Identify and analyze hotspots with the highest CPU time values and high CPI rate.
- Filter hotspots to identify hotspots for a specific time region.
- Analyze the source by double-clicking a hotspot and identify its most critical code lines.

Press **F1** for help on each window.

Show the analysis description when result opens

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.



General exploration – General exploration viewpoint

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

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

Elapsed Time: 93.154s

Clockticks:	6,401,472,000,000	
Instructions Retired:	3,850,662,200,000	
CPI Rate:	1.662	
MUX Reliability:	0.999	
Front-End Bound:	32.3%	of Pipeline Slots
Bad Speculation:	2.5%	of Pipeline Slots
Back-End Bound:	29.1%	of Pipeline Slots
Retiring:	36.1%	of Pipeline Slots
Total Thread Count:	69	
Paused Time:	0s	

General Exploration

This view uses hardware event-based metrics to show code regions that experienced potentially significant architectural bottlenecks. Hover over a metric name in the grid for the metric description.

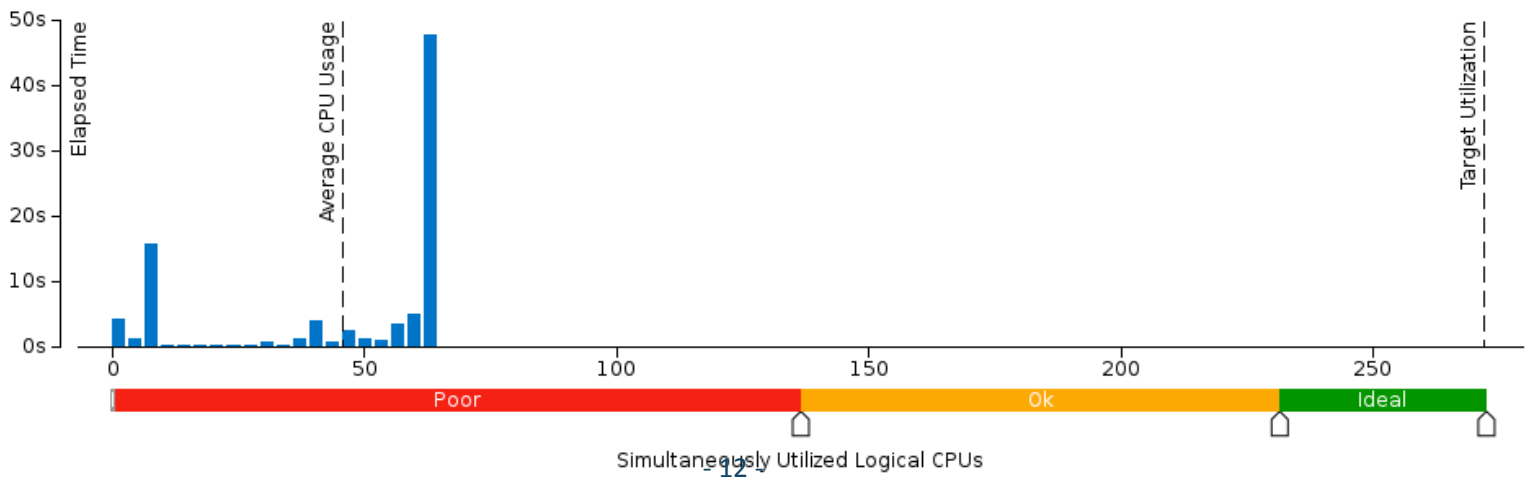
Use this view to:
- Identify code regions (modules, functions, and so on) with the highest execution time.
- Analyze detected hardware issues highlighted by pink cells and get tuning recommendations.

Press **F1** for help on each window.

☑ Show the analysis description when result opens

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.



General exploration –hotspots

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

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

Elapsed Time: 93.154s

CPU Time:	4267.914s
Instructions Retired:	3,850,662,200,000
CPI Rate:	1.662
CPU Frequency Ratio:	1.071
Total Thread Count:	69
Paused Time:	0s

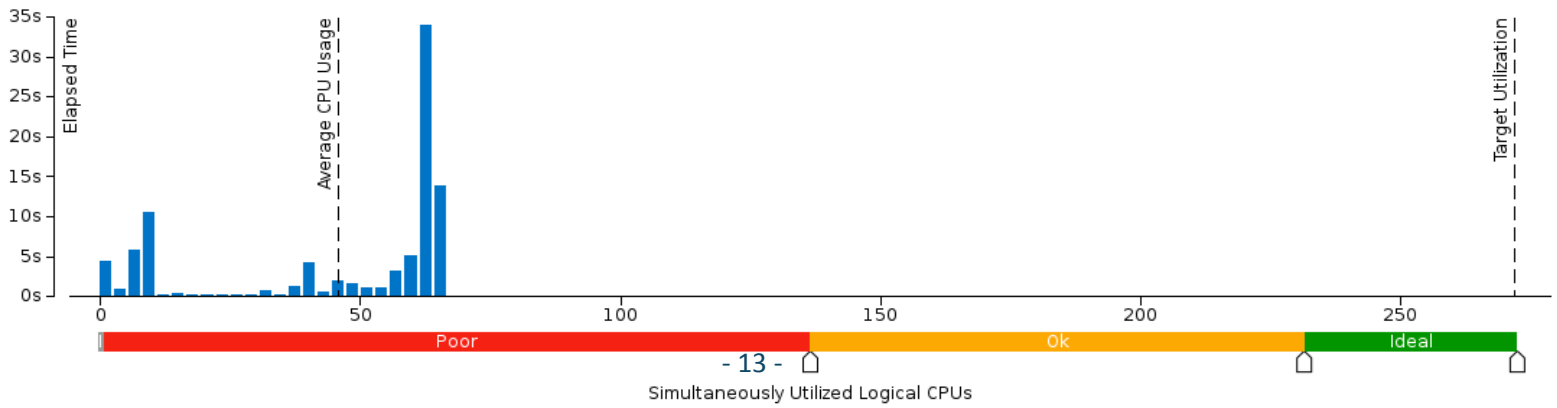
Top Hotspots

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
[libiomp5.so]	libiomp5.so	1955.410s
[vasp_std]	vasp_std	1821.137s
[vmlinux]	vmlinux	139.823s
[libmpich_intel.so.3.0.1]	libmpich_intel.so.3.0.1	117.726s
[libimf.so]	libimf.so	115.019s
[Others]		118.799s

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.



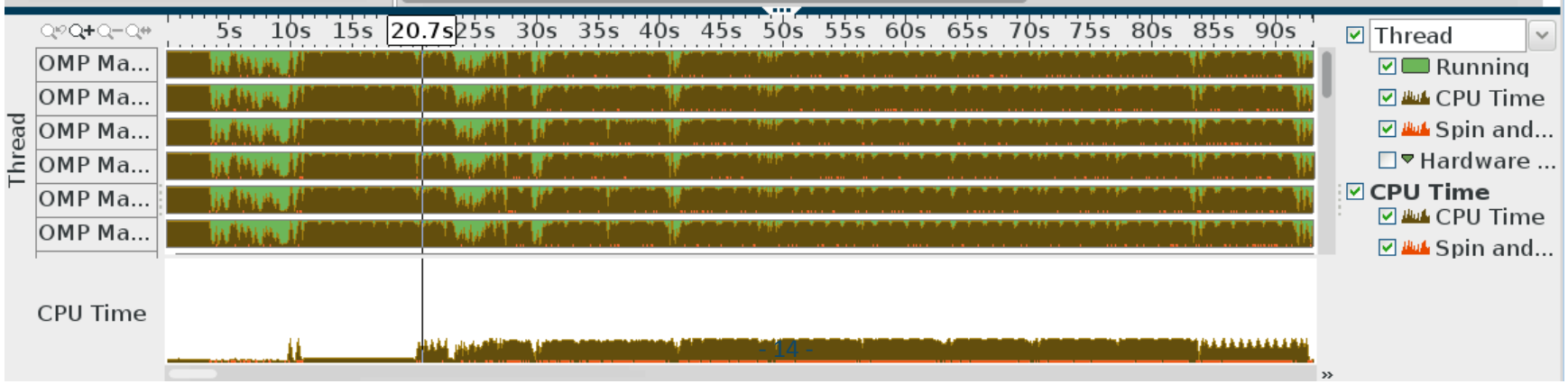
General Exploration – Hotspots bottom-up viewpoint

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

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

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: Spin Time	CPU Time: Overhead Time	Instructions Retired	CPI Rate	CPU Frequency Ratio
[libiomp5.so]	1955.410s	0s	0s	1,388,909,200,000	2.111	1.071
[vasp_std]	1821.137s	0s	0s	2,048,333,000,000	1.333	1.071
[vmlinux]	139.823s	0s	0s	71,926,400,000	2.949	1.084
[libmpich_intel.so.3.0.1]	117.726s	0s	0s	116,454,800,000	1.514	1.070
[libimf.so]	115.019s	0s	0s	80,430,000,000	2.145	1.071
[libugni.so.0.6.0]	34.402s	0s	0s	52,456,600,000	0.985	1.073
[libpmi.so.0.5.0]	24.414s	0s	0s	43,323,000,000	0.845	1.071
[libintlc.so.5]	21.547s	0s	0s	19,944,400,000	1.611	1.065
[libc-2.19.so]	20.592s	0s	0s	17,732,400,000	1.736	1.068
[ld-2.19.so]	4.491s	0s	0s	2,144,800,000	3.097	1.056
[libpthread-2.19.so]	0s	4.231s	0s	4,870,600,000	1.360	1.119
[obdclass]	1.595s	0s	0s	371,000,000	6.626	1.101
[xpmem]	1.356s	0s	0s	935,200,000	2.289	1.128
[ptlrpc]	1.142s	0s	0s	105,000,000	15.427	1.013
[vdso]	0.986s	0s	0s	460,600,000	3.392	1.132
[lov]	0.722s	0s	0s	159,600,000	6.851	1.082
[libudreg.so.0.2.3]	0.679s	0s	0s	158,200,000	6.274	1.044
[libcfs]	0.609s	0s	0s	389,200,000	2.385	1.089



HPC Performance Characterization viewpoint

HPC Performance Characterization HPC Performance Characterization viewpoint (change)

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

Elapsed Time: 98.250s

CPU Utilization: 17.2%

Average CPU Usage: 46.831 Out of 272 logical CPUs

MPI Rank on the Critical Path:

MPI Busy Wait Time:

Serial Time (outside any parallel region):

OpenMP Potential Gain:

CPU Usage Histogram

Back-End Bound: 28.1%

L2 Hit Bound:	8.8%	of Clockticks
L2 Miss Bound:	23.1%	of Clockticks
MCDRAM Cache Bandwidth Bound:	0.1%	
MCDRAM Flat Bandwidth Bound:	0.1%	
DRAM Bandwidth Bound:	0.0%	

Bandwidth Utilization

SIMD Instructions per Cycle: 0.138

FP Instruction Mix:

% of Packed SIMD Instr.: 86.2%

% of Scalar SIMD Instr.: 13.8%

Top Loops/Functions with FPU Usage by CPU Time

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

Function	CPU Time	SIMD Instructions per Cycle	Vector Instruction Set	Loop Type
[libiomp5.so]	2228.553s	0.000		
[vasp_std]	1825.883s	0.329		
[vmlinux]	170.380s	0.005		
[libmpich_intel.so.3.0.1]	135.620s	0.048		
[libimf.so]	115.210s	0.206		
[Others]	94.890s	N/A*		

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

HPC Performance Characterization

This view displays performance statistics for applications using OpenMP and MPI runtimes.

Use this view to:

- Explore OpenMP code optimization Potential Gain by parallelization inefficiencies representing their wall time cost.
- Explore precise Imbalance metrics to identify imbalance of working threads on barriers.
- Analyze barrier-to-barrier region segments to explore performance of OpenMP work-sharing constructs and barrier cost inside a region.
- For hybrid MPI and OpenMP applications using Intel MPI library, analyze per-rank MPI communication busy wait time.

Press **F1** for help on each window.

Show the analysis description when result opens

HPC Performance Characterization viewpoint

HPC Performance Characterization HPC Performance Characterization viewpoint (change)

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

Elapsed Time: 98.250s

CPU Utilization: 17.2%

Average CPU Usage: 46.831 Out of 272 logical CPUs

MPI Rank on the Critical Path:

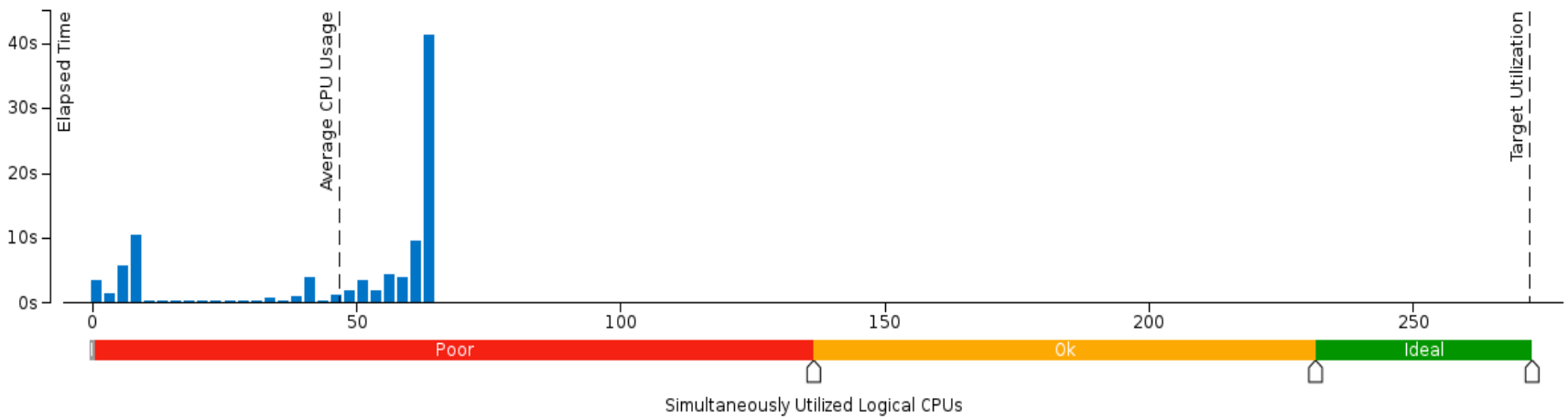
MPI Busy Wait Time:

Serial Time (outside any parallel region):

OpenMP Potential Gain:

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.



HPC Performance Characterization viewpoint

HPC Performance Characterization HPC Performance Characterization viewpoint (change)

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

Back-End Bound: 28.1%

L2 Hit Bound:	8.8%	of Clockticks
L2 Miss Bound:	23.1%	of Clockticks
MCDRAM Cache Bandwidth Bound:	0.1%	
MCDRAM Flat Bandwidth Bound:	0.1%	
DRAM Bandwidth Bound:	0.0%	

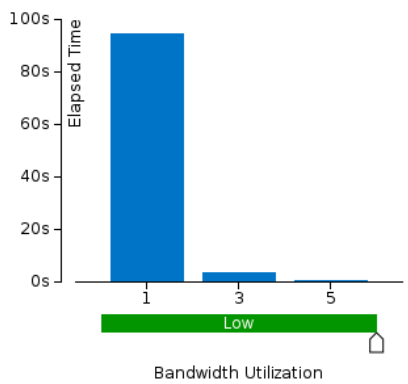
Bandwidth Utilization

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

Bandwidth Domain: DRAM, GB/sec

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 High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and QPI bandwidth.



Top Functions with High Bandwidth Utilization

This section shows top functions, sorted by LLC Misses that were executing when bandwidth utilization was high for the domain selected in the histogram area.

No data to show. The collected data is not sufficient.

HPC Performance Characterization viewpoint

HPC Performance Characterization HPC Performance Characterization viewpoint (change) INTEL VTUNE AMPLIFIER XE 2017

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

Back-End Bound: 28.1%

L2 Hit Bound:	8.8%	of Clockticks
L2 Miss Bound:	23.1%	of Clockticks
MCDRAM Cache Bandwidth Bound:	0.1%	
MCDRAM Flat Bandwidth Bound:	0.1%	
DRAM Bandwidth Bound:	0.0%	

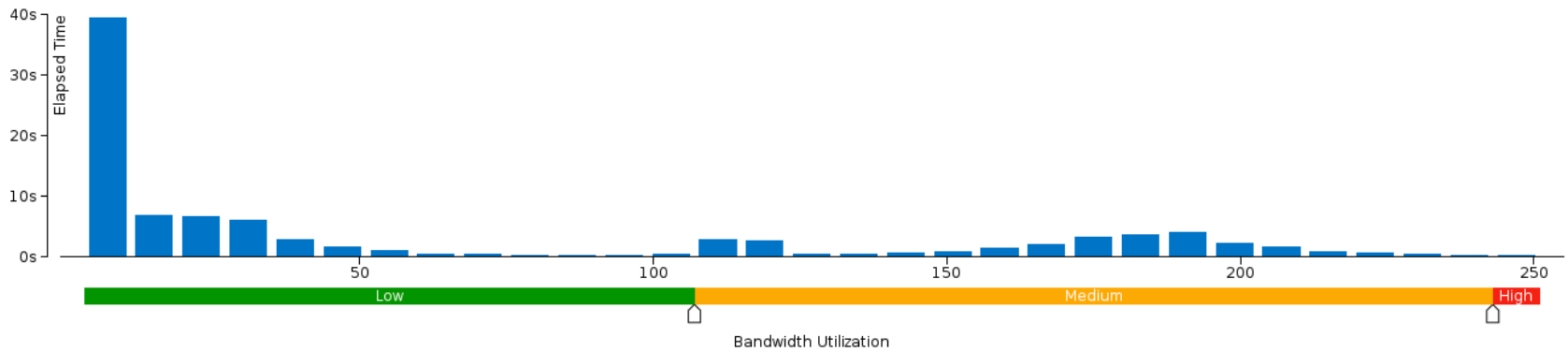
Bandwidth Utilization

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

Bandwidth Domain: MCDRAM Flat, GB/sec

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 High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and QPI bandwidth.

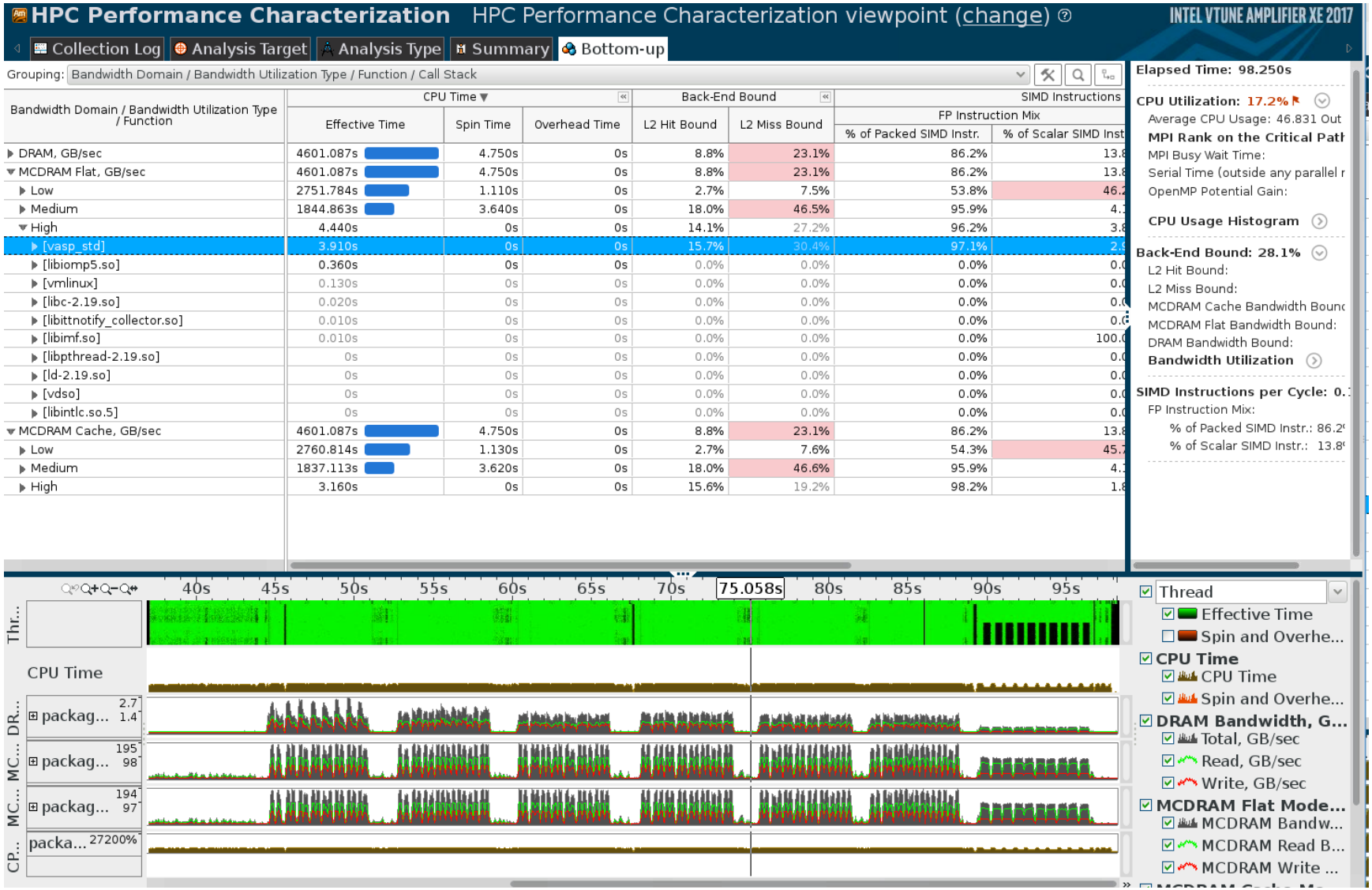


Top Functions with High Bandwidth Utilization

This section shows top functions, sorted by LLC Misses that were executing when bandwidth utilization was high for the domain selected in the histogram area.

Function	LLC Miss Count
[vasp_std]	0.1%

HPC Performance Characterization viewpoint



HPC Performance Characterization viewpoint

HPC Performance Characterization HPC Performance Characterization viewpoint (change)

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

SIMD Instructions per Cycle: 0.138

FP Instruction Mix:

% of Packed SIMD Instr.: 86.2%

% of Scalar SIMD Instr.: 13.8%

Top Loops/Functions with FPU Usage by CPU Time

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

Function	CPU Time	SIMD Instructions per Cycle	Vector Instruction Set	Loop Type
[libiomp5.so]	2228.553s	0.000		
[vasp_std]	1825.883s	0.329		
[vmlinux]	170.380s	0.005		
[libmpich_intel.so.3.0.1]	135.620s	0.048		
[libimf.so]	115.210s	0.206		
[Others]	94.890s	N/A*		

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

Advanced Hotspots – hotspots viewpoint

Advanced Hotspots Hotspots viewpoint (change) INTEL VTUNE AMPLIFIER XE 2017

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

Elapsed Time: 142.760s

CPU Time:	6564.929s
Instructions Retired:	6,561,240,000,000
CPI Rate:	1.498
CPU Frequency Ratio:	1.070
Total Thread Count:	67
Paused Time:	0s

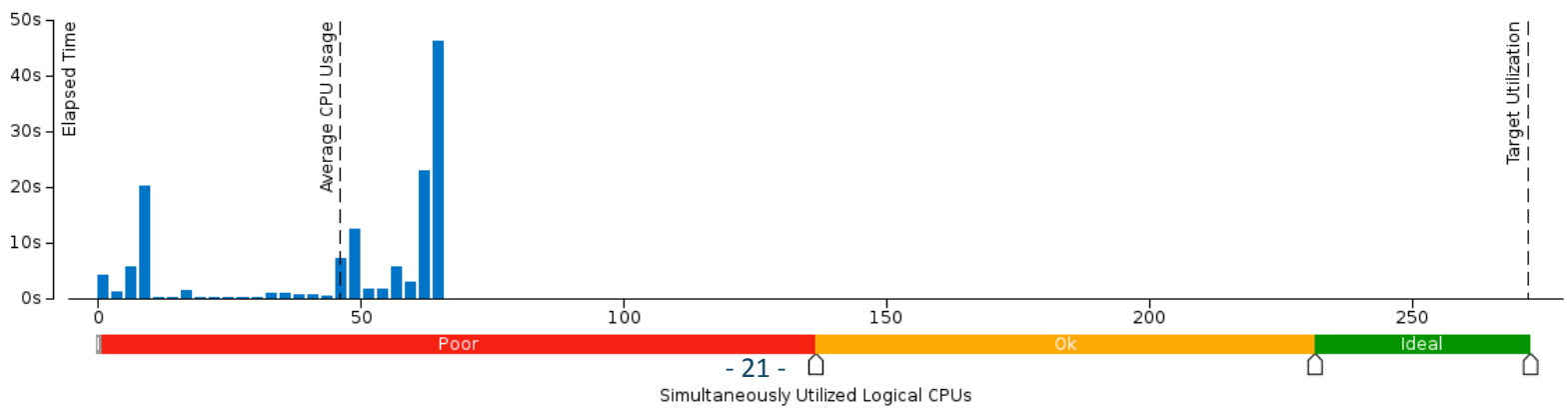
Top Hotspots

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
[libiomp5.so]	libiomp5.so	2981.554s
[vasp_std]	vasp_std	2532.104s
[vmlinux]	vmlinux	456.501s
[libmpich_intel.so.3.0.1]	libmpich_intel.so.3.0.1	302.210s
[libugni.so.0.6.0]	libugni.so.0.6.0	157.450s
[Others]		135.110s

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.



More resources

- **At NERSC**
  - NERSC Vtune page, more details about how to run Vtune http://www.nersc.gov/users/software/performance-and-debugging-tools/vtune/
  - Various training slides by NERSC and Intel staff, available at NERSC user training web site
- **At Intel**
  - https://software.intel.com/en-us/intel-vtune-amplifier-xe-support/documentation

Acknowledgement

Thank you!

Application Performance Snapshot

- **Intel profiling tool is recently made available on Cori (under beta besting)**
  - Application Performance Snapshot provides a quick view into a shared memory or MPI application's use of available hardware (CPU, FPU, and Memory). It analyzes your application's time spent in MPI, MPI and OpenMP imbalance, memory access efficiency, FPU usage, and I/O and memory footprint, and displays basic performance enhancement opportunities for systems using Intel® platformin the standard output and also stores the analysis result in the html format to displays.
  - Reference, https://software.intel.com/sites/products/snapshots/application-snapshot/

APS report

Application Performance Snapshot

221.32s
Elapsed Time

2.41
CPI
(MAX 2.43, MIN 2.35)

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. Use MPI profiling tools like Intel® Trace Analyzer and Collector to explore performance bottlenecks.

	Current run	Target	Delta
MPI Time	22.64%	<15%	
OpenMP Imbalance	1.77%	<10%	
Back-End Stalls	70.14%	<20%	
SIMD Instr. per Cycle	0.16	>1	
I/O Bound	0.01%	<10%	

MPI Time
22.64% of Elapsed Time
(50.10s)

MPI Imbalance
0.00% of Elapsed Time
(0.00s)

TOP 5 MPI Functions	%
Allreduce	5.86
Waitall	5.57
Barrier	4.93
Ireduce	2.62
Bcast	2.23

OpenMP Imbalance
1.77% of Elapsed Time
(3.91s)

Memory Footprint
AVG 10739.15 MB, PEAK 11728.88 MB

Back-End Stalls
70.14% of pipeline slots

L2 Hit Bound
18.46%

L2 Miss Bound
41.26%

SIMD Instr. per Cycle
0.16

FP Instruction Mix
% of Packed SIMD Instr.: 94.21%
% of Scalar SIMD Instr.: 5.79%

I/O Bound
0.01%
(AVG 4.07, PEAK 0.07)

Read
AVG 7.0 MB, MAX 240.4 MB

Write
AVG 27.9 KB, MAX 453.8 KB

To use on Cori

- **Link dynamically, and run with the following script**

```
#!/bin/bash
#SBATCH −J aps_test
#SBATCH -N 2
#SBATCH -C knl,quad,cache  #running in quad,cashe mode
#SBATCH −p debug
#SBATCH -t 00:30:00
#SBATCH --perf=vtune
#SBATCH -o myjob.o%j

module load aps
. $APS_DIR/apsvars.sh

export OMP_NUM_THREADS=8

srun -n32 -c16 --cpu_bind=cores aps.sh jax-dynamic.x
aps.sh --report stat_$rundate  # $rundate looks like this 20170609
```