

Introduction to Manycore Tools: Intel VTune & Advisor

Todd Evans

Feb 3rd, 2017

Intel VTune

Line-level Performance Profiling Tool

- ▶ Supports serial, threaded, MPI, and hybrid profiling
- ▶ Shows users where runtime is spent in code
- ▶ Samples hardware counters at regular intervals and correlates values to location on code
- ▶ Specialized support for KNL including memory access details

Why Use VTune?

- ▶ Provides insight into performance, threading performance, scalability, bandwidth, caching and more
- ▶ Analysis is simple and convenient (Good GUI)
 - ▶ Sort, filter, and visualize results on the timeline and on your source
- ▶ Useful to look at memory access on KNL
 - ▶ DDR4 vs MCDRAM
- ▶ Useful for fine grained control of memory allocation using libmemkind (hbw_malloc)

How-to Use VTune

1. Compile your code with:

- ▶ -g flag: provides symbol table information
- ▶ Link to libmemkind -lmemkind: provides memory access information
- ▶ `icc -g -O2 -xMIC-AVX512 -qopenmp -lmemkind omp_mm_knl.c`
- ▶ `mpicc -g -O2 -xMIC-AVX512 -qopenmp -lmemkind
omp_mm_knl.c`

2. Run the program with VTune Amplifier:

- ▶ Graphical User Interface: `amplxe-gui`
- ▶ Command Line Interface: `amplxe-cl`

3. Analyze the results with VTune Amplifier

- ▶ Graphical User Interface: `amplxe-gui`
- ▶ Command Line Interface: `amplxe-cl`

Run your job with VTune **amplxe-cl**

Job submission script

```
#!/bin/bash
#SBATCH -J knl-run.%j.err      # job name
#SBATCH -o knl-run.%j.out      # output and error file name
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p normal             # queue (partition)
#SBATCH -t 00:30:00            # run time (hh:mm:ss)
#SBATCH -A Your_Project

export OMP_NUM_THREADS=16
#Launch amplxe-cl with proper options
amplxe-cl -collect memory-access -knob analyze-mem-objects=true \\
           -r output_dir ./a.out
```

Run your job with VTune **amplxe-cl**

```
amplxe-cl <-action> [-action-option]
[-global-option] [[--] <target> [target-options]]
```

<-action>: The action to perform, such as collect or report collect, report, help, etc.

[-action-option]: Action-options modify behavior specific to the action.

[-global-option]: Global-options modify behavior in the same manner for all actions. For example: -q, -quiet to suppress non-essential messages.

<target>: The target application to analyze.

[target-options]: Options for the application.

Complete **amplxe-cl** command syntax from Intel:

<https://software.intel.com/en-us/node/544220>

Run your job with VTune **amplxe-cl**

Tips

- ▶ VTune is easiest to use by employing pre-defined *collections*
- ▶ Collections have *knobs* that allow you to capture additional information
- ▶ Examples
 - ▶ `amplxe-cl -collect hotspots -r output_dir ./a.out`
 - ▶ `amplxe-cl -collect memory-access -knob analyze-mem-objects=true -r output_dir ./a.out`
 - ▶ `amplxe-cl -collect hotspots -r output_dir ibrun -np 16 ./a.out -other-options`
- ▶ For help: `amplxe-cl -help` or `amplxe-cl -help collect`

Pre-defined Collections

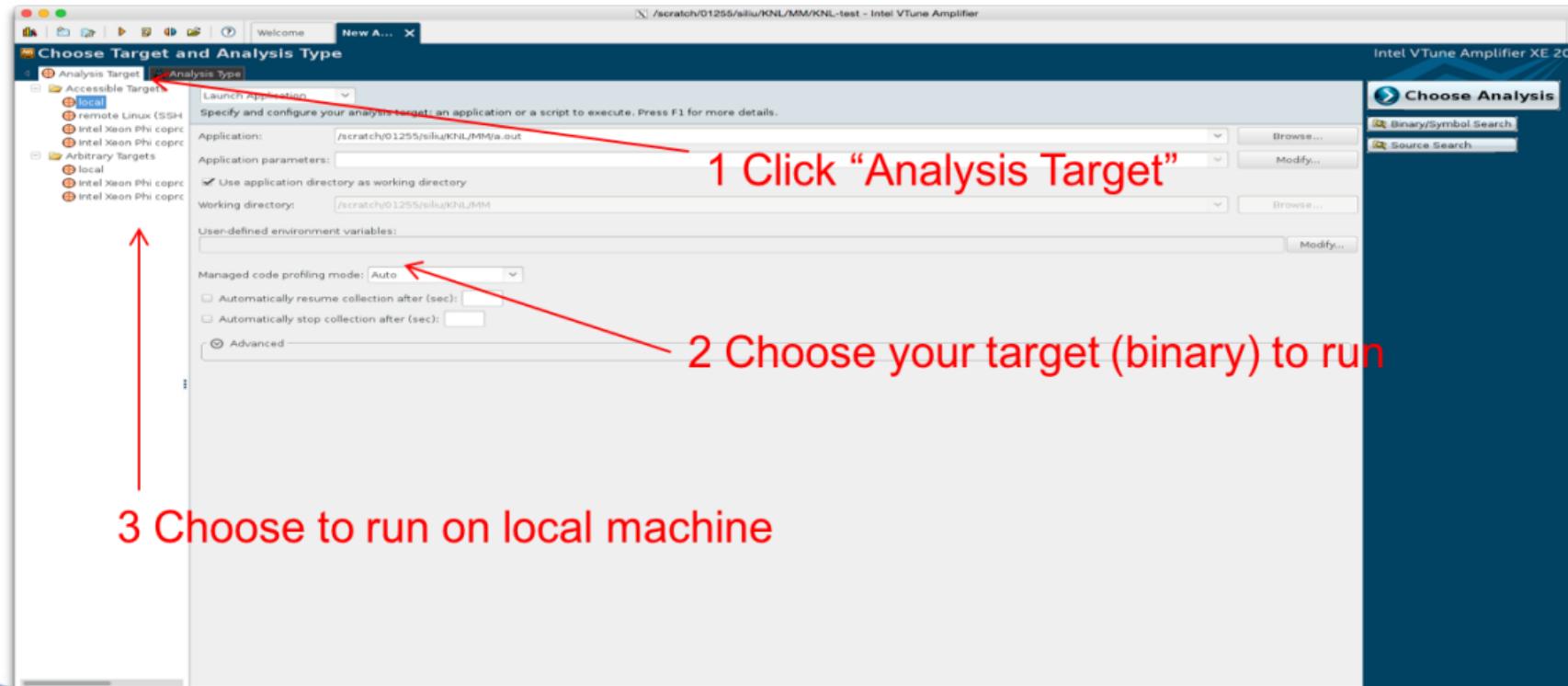
Collection	Information Provided
hotspots	Identify most time-consuming code sections (user mode)
advanced-hotspots	Adds CPI, higher frequency low overhead sampling
concurrency	CPU utilization, threading synchronization overhead (user mode)
disk-io	Disk IO preview, not working in Stampede (requires root)
memory-access	Memory access details and memory bandwidth utilization. May include specific arrays that are heavy BW users (useful for fine-grained MCDRAM use)
hpc-performance	Performance characterization, including floating point unit and memory bandwidth utilization

Useful knobs and options

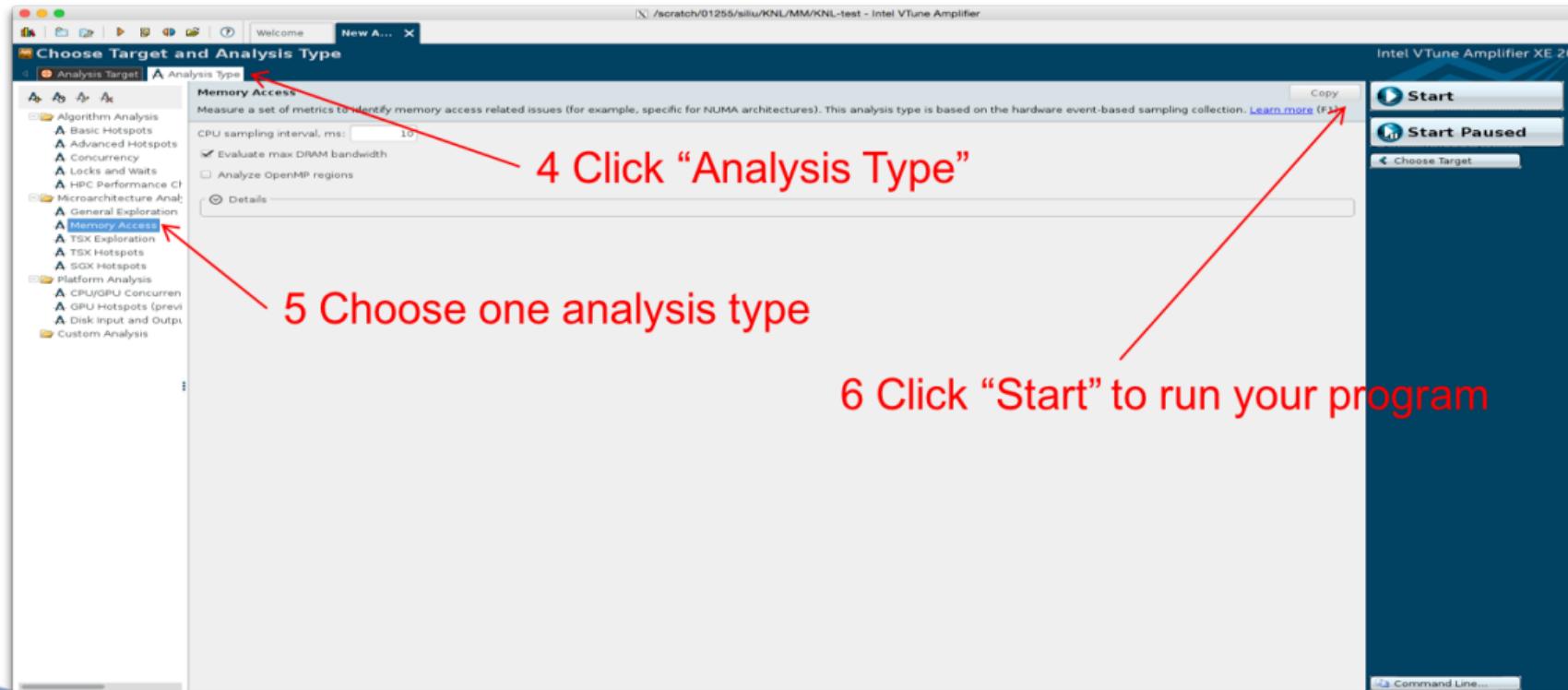
Option	Description
-data-limit=0	Override default maximum data collection size
-no-summary	Do not produce text summary
-no-auto-finalize	Do not finalize data analysis after collection
-finalize	Carry out data analysis after collection
-start-paused	Start application without profiling
-resume-after=X	Resume profiling after X seconds
-duration=Y	Profile only for Y seconds

Knob	Description
analyze-memory-objects=true.	Determine arrays using most memory bandwidth (highest L2 miss rates)
analyze-openmp=true.	Determine inefficiencies in OpenMP regions

Run your job with VTune ampxe-gui



Run your job with VTune ampxe-gui



amplxe-cl to analyze performance

It is possible to use the VTune command line (`amplxe-cl`) to analyze performance. However, using the Graphic User Interface (`amplxe-gui`) is strongly recommended!

Example

```
amplxe-cl -report hotspots -result-dir r000hs -group-by  
function
```

amplxe-gui: Summary

Elapsed Time: 16.919s

CPU Time: 173.818s

Total Thread Count: 18

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
mainomp_parallel@:43	a.out	53.151s
mainomp_parallel@:95	a.out	53.089s
mainomp_parallel@:134	a.out	52.817s
_kmpc_barrier	libiomp5.so	10.840s
_kmpc_fork_barrier	libiomp5.so	3.771s
[Others]	N/A*	0.101s

*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.

Collection and Platform Info

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

Application Command Line: ./a.out

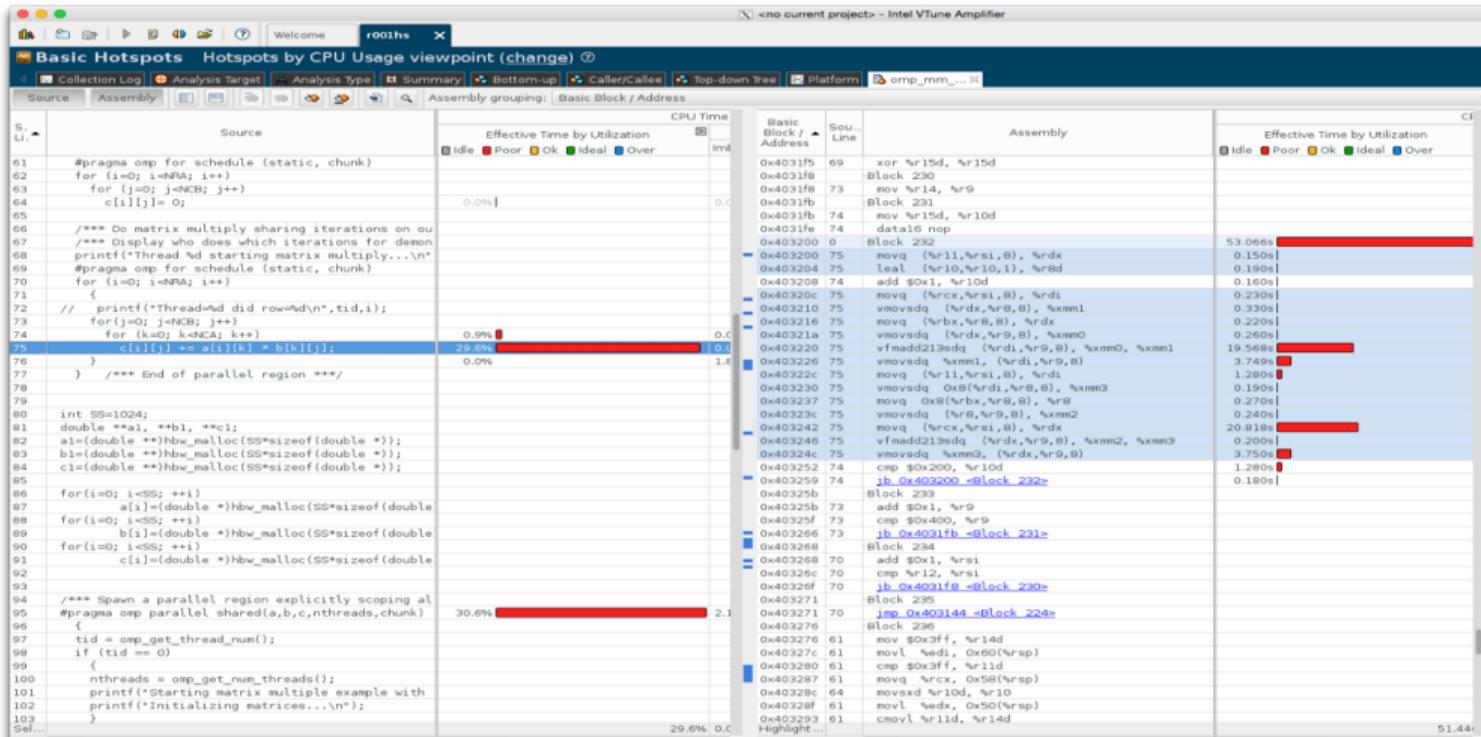
Operating System: 3.10.0-327.13.1.el7.xpsl_1.3.3.151>#6_64 NAME="CentOS Linux" VERSION="7 (Core)" ID="centos" ID_LIKE="rhel fedora" VERSION_ID="7" PRETTY_NAME="CentOS Linux 7 (Core)" ANSI_COLOR="0;31" CPE_NAME="cpe:/o:centos:centos:7"

Computer Name: c564-021.stampede.tacc.utexas.edu

Result Size: 5 MB

Wall Time: 16.919s

amplxe-gui: Analyze Hot Subtrees



amplxe-gui: Memory Access

Intel VTune Amplifier XE 2017

Memory Access Memory Usage viewpoint (change) ②

Collection Log Analysis Target Analysis Type Summary Bottom-up Platform

Elapsed Time: 17.759s

CPU Time: 177.089s

Memory Bound:

- L2 Hit Rate: 0.102
- L2 Hit Bound: 0.023
- L2 Miss Bound: 1.000

A high number of CPU cycles is being spent waiting for L2 load misses to be serviced. Possible optimizations are to reduce data working set size, improve data access locality, blocking and consuming data in chunks that fit in the L2, or better exploit hardware prefetchers. Consider using software prefetchers but they can increase latency by interfering with normal loads, and can increase pressure on the memory system.

Memory Cache Bandwidth Bound:

- DRAM Bandwidth Bound: 0.0%
- L2 Miss Bandwidth Bound: 0.4%
- MCDRAM Hit Rate: 3.114.093.420 100.0%
- MCDRAM Hit/Miss Ratio: 99.6%
- Total Thread Count: 25
- Paused Time: 0s

Bandwidth Utilization Histogram

This histogram displays a percentage of 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.

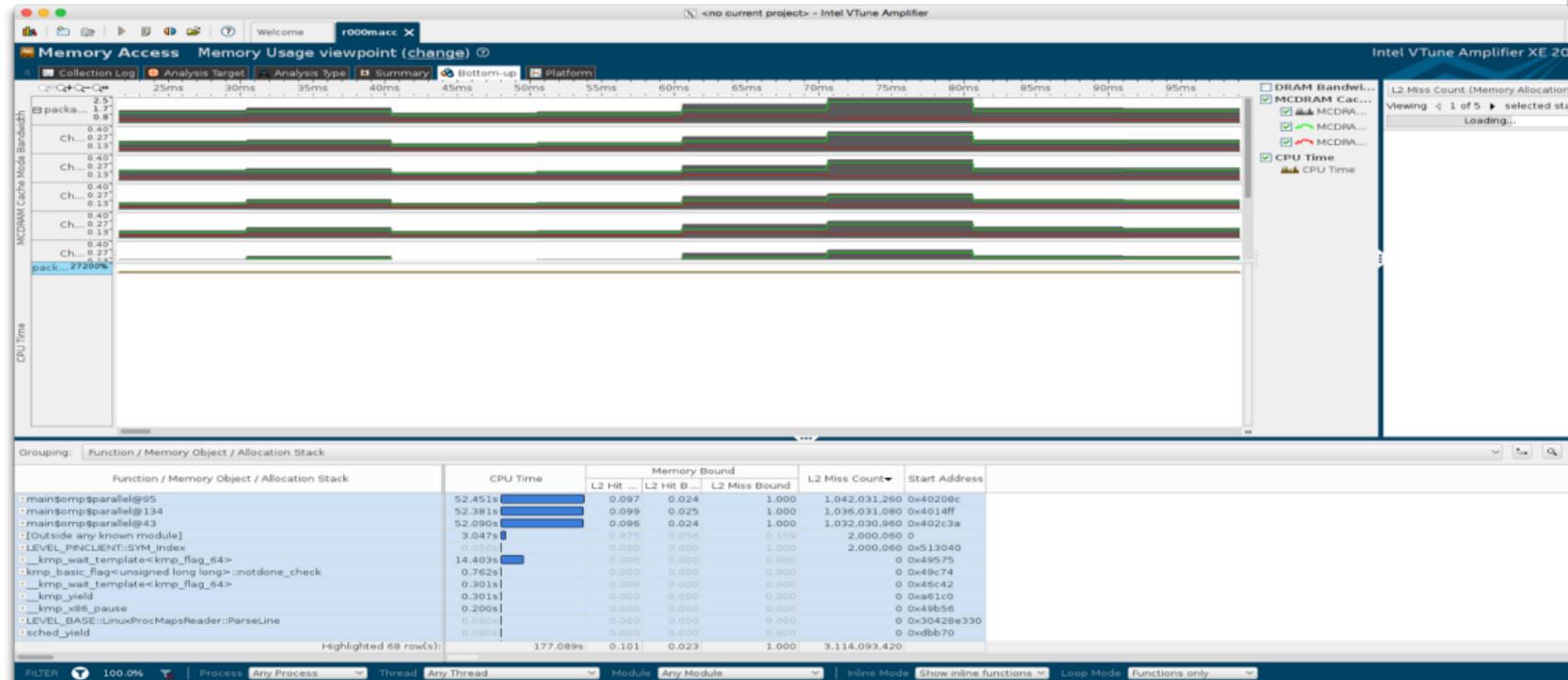
Bandwidth Domain: MCDRAM Cache, GB/sec

Top Memory Objects

This section lists the most actively used memory objects in your application.

Memory Object	L2 Miss Count
omp_mm_knl_c_89 (8 kB)	10,000,300
omp_mm_knl_c_89 (8 kB)	10,000,300
omp_mm_knl_c_89 (8 kB)	10,000,300

amplxe-gui: Memory Access



hotspots

getting the basics right

hotspots Summary

<no current project> - Intel VTune Amplifier

Welcome r000hs X

Basic Hotspots Hotspots by CPU Usage viewpoint (change) Intel VTune Amplifier XE 2016

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

Elapsed Time : 19.160s

CPU Time : 1767.714s
Effective Time : 1495.563s
Spin Time : 267.933s

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.

Imbalance or Serial Spinning (OpenMP) : 251.729s

CPU time spent waiting on an OpenMP barrier inside of a parallel region can be a result of load imbalance. Where relevant, try dynamic work scheduling to reduce the imbalance. High metric value on serial execution (Serial - outside any region) may signal significant serial application time that is limiting efficient processor utilization. Explore options for parallelization, algorithm or microarchitecture tuning of the serial part of the application.

Lock Contention (OpenMP) : 0.010s
Other : 16.194s
Overhead Time : 4.218s

Total Thread Count: 137
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.

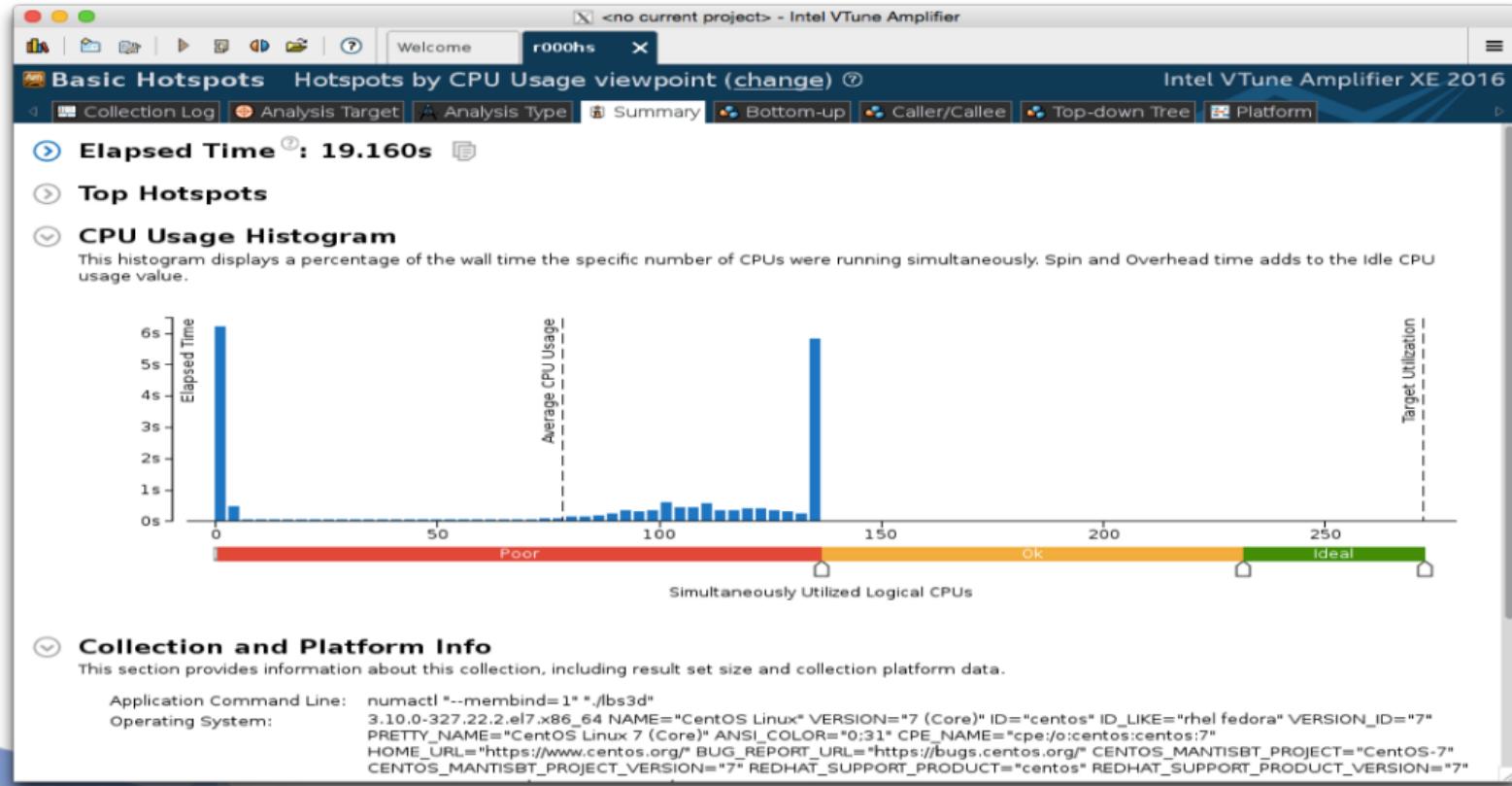
Function	Module	CPU Time
collision_#omp\$parallel@51	lbs3d	1254.026s
stream_#omp\$parallel_for@39	lbs3d	184.591s
_kmp_fork_barrier	libiomp5.so	150.065s
_kmpc_barrier	libiomp5.so	115.491s
poststream_#omp\$parallel@37	lbs3d	36.135s
[Others]	N/A*	27.407s

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

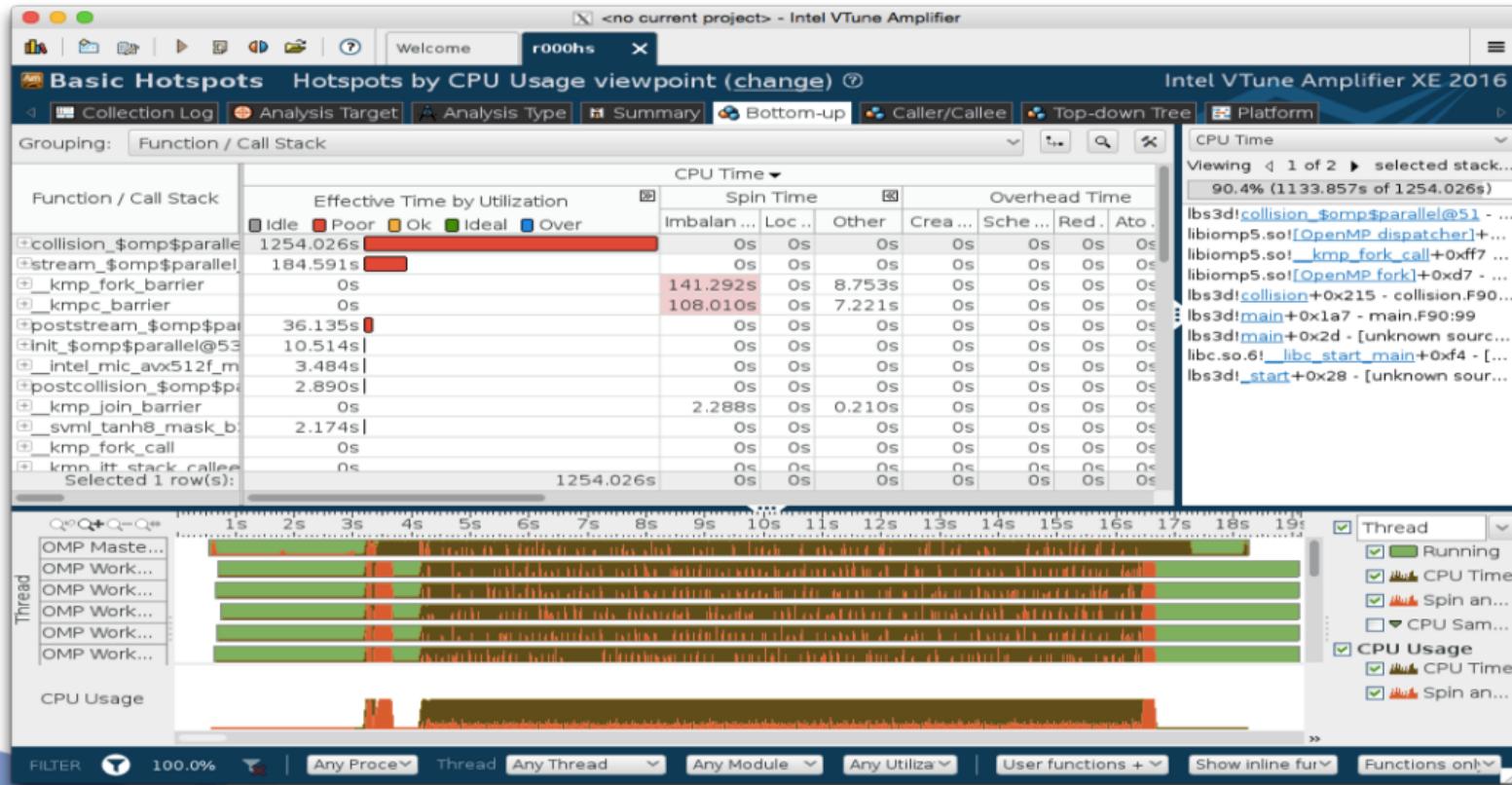
CPU Usage Histogram

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

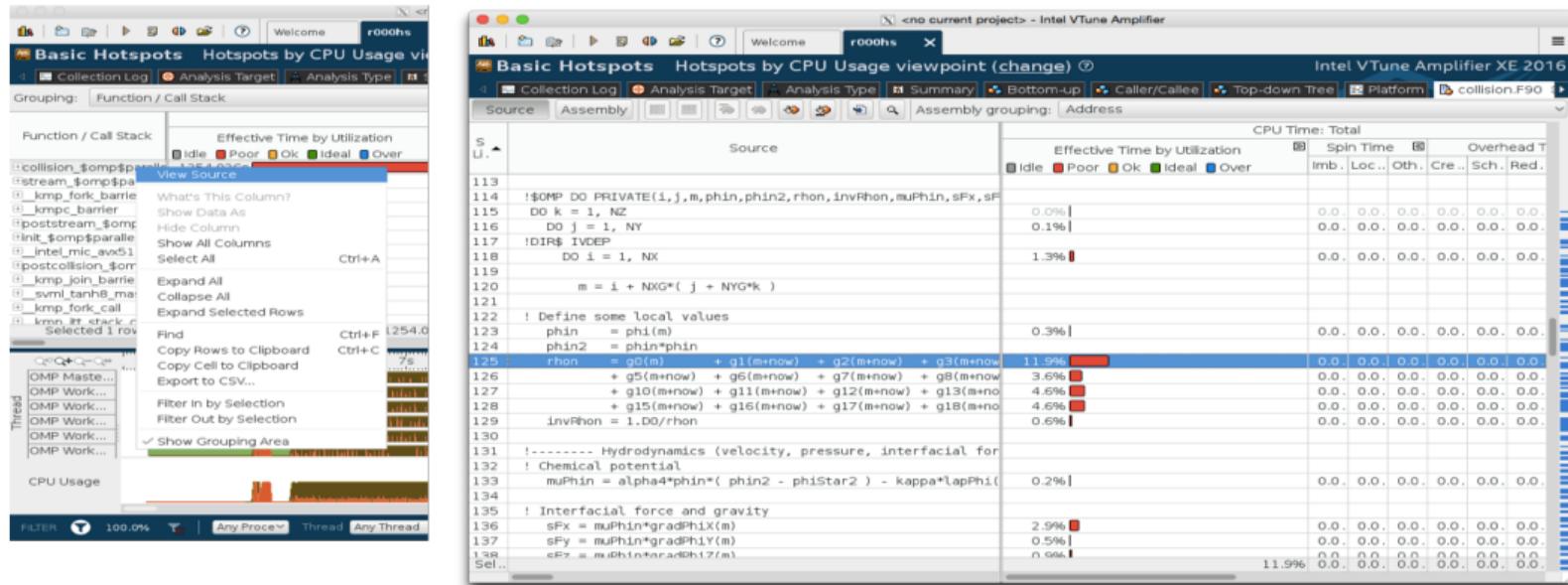
hotspots CPU Histogram



hotspots Bottom-up View



hotspots Getting to the Source



hotspots Digging into the Assembly

The screenshot shows the Intel VTune Amplifier XE 2016 interface with the "Basic Hotspots" view selected. The left pane displays the source code for a Fortran-like program, and the right pane shows the corresponding assembly code.

Source:

```
S 113
114 !$OMP DO PRIVATE(i,j,m,phin,phin2,rhon,invRhon,muPhin,s
115 DO k = 1, NZ
116   DO j = 1, NY
117     !DIR$ IVDEP
118     DO i = 1, NX
119       m = i + NXG*( j + NYG*k )
120
121     ! Define some local values
122     phin = phi(m)
123     phin2 = phin*phin
124     rhon = g0(m) + g1(m+now) + g2(m+now) + g3(
125       + g5(m+now) + g6(m+now) + g7(m+now) + g8(
126       + g10(m+now) + g11(m+now) + g12(m+now) + g13
127       + g15(m+now) + g16(m+now) + g17(m+now) + g18
128     invRhon = 1.0D/rhon
129
130 !----- Hydrodynamics (velocity, pressure, interfacia
131 ! Chemical potential
132   muPhin = alpha4*phin*( phin2 - phiStar2 ) - kappa*la
133
134 ! Interfacial force and gravity
135   sFx = muPhin*gradPhiX(m)
136   sFy = muPhin*gradPhiY(m)
137   sFz = muPhin*gradPhiZ(m)
138 Sel..
```

Assembly:

Address	Line	Assembly
0x40817a	125	vmovupsz %zmm11, -0x930(%rbp)
0x408184	123	vmovupsz 0x8(%r15,%r13,8), %zmm3
0x40818f	125	movq -0x248(%rbp), %r15
0x40819e	125	vmovupsz 0x8(%r15,%r13,8), %zmm23
0x4081a1	125	movq -0x240(%rbp), %r15
0x4081a8	125	vmovupsz 0x8(%r15,%r13,8), %zmm13
0x4081b3	125	vmovupsz %zmm13, -0x8f0(%rbp)
0x4081bd	125	movq -0x238(%rbp), %r15
0x4081c4	125	vaddpd %zmm23, %zmm11, %zmm15
0x4081ca	125	vmovupsz 0x8(%r15,%r13,8), %zmm24
0x4081d5	125	movq -0x230(%rbp), %r15
0x4081dc	125	vmovupsz 0x8(%r15,%r13,8), %zmm20
0x4081e7	125	vmovupsz %zmm20, -0x830(%rbp)
0x4081f1	126	movq -0x228(%rbp), %r15
0x4081f8	125	vaddpd %zmm24, %zmm13, %zmm17
0x4081fe	126	vmovupsz 0x8(%r15,%r13,8), %zmm21
0x408209	126	movq -0x220(%rbp), %r15
0x408210	125	vaddpd %zmm17, %zmm15, %zmm28
0x408216	126	vmovupsz 0x8(%r15,%r13,8), %zmm22
0x408221	126	vmovupsz %zmm22, -0x870(%rbp)
0x40822b	126	movq -0x218(%rbp), %r15
0x408232	125	vaddpd %zmm21, %zmm20, %zmm25
0x408238	126	vmovupsz 0x8(%r15,%r13,8), %zmm18
0x408243	126	movq -0x1e8(%rbp), %r15
0x40824a	126	vmovupsz 0x8(%r15,%r13,8), %zmm30
0x408255	126	vmovupsz %zmm30, -0x8f0(%rbp)

memory-access

It's all about data movement...

memory-access Summary

<no current project> - Intel VTune Amplifier

Welcome r003macc X

Memory Access Memory Usage viewpoint (change) Intel VTune Amplifier XE 2016

Collection Log Analysis Target Analysis Type Summary Bottom-up Platform

Elapsed Time: 14.838s

CPU Time: 1703.506s

Memory Bound:

- L2 Hit Rate: 0.617
- L2 Hit Bound: 0.153
- L2 Miss Bound: 1.000

A high number of CPU cycles is being spent waiting for L2 load misses to be serviced. Possible optimizations are to reduce data working set size, improve data access locality, blocking and consuming data in chunks that fit in the L2, or better exploit hardware prefetchers. Consider using software prefetchers but they can increase latency by interfering with normal loads, and can increase pressure on the memory system.

MCDRAM Flat Bandwidth Bound: 78.6%

The system spent a significant percentage of elapsed time with high MCDRAM Flat bandwidth utilization. Review the Bandwidth Utilization Histogram to estimate the scale of the issue. Consider improving data locality and/or merging compute-intensive code with bandwidth-intensive code.

DRAM Bandwidth Bound: 0.0%

L2 Miss Count: 14,036,421,080

Total Thread Count: 136

Paused Time: 0s

Bandwidth Utilization Histogram

This histogram displays a percentage of 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.

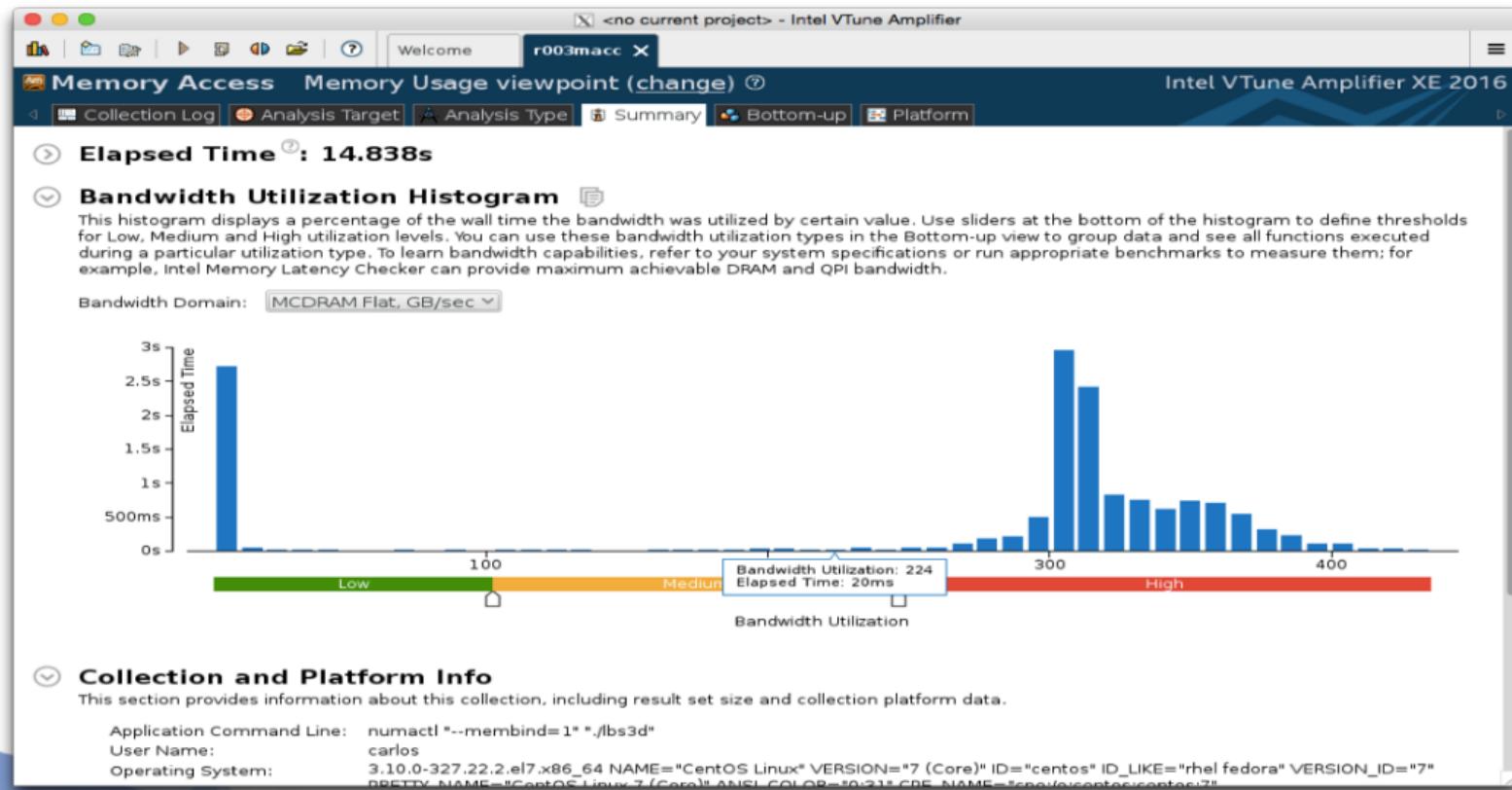
Bandwidth Domain: DRAM, GB/sec

Elapsed Time

12s
10s
8s
6s

0.0%
25.0%
50.0%
75.0%
100.0%

memory-access Bandwidth Histogram



memory-access With Objects Knob

The screenshot shows the Intel VTune Amplifier XE 2016 interface with the following details:

- Viewpoint:** Memory Access - Memory Usage viewpoint (change)
- Analysis Type:** Bottom-up
- Elapsed Time:** 18.536s (CPU Time: 1797.894s)
- Memory Bound:**
 - L2 Hit Rate: 0.617
 - L2 Hit Bound: 0.145
 - L2 Miss Bound: 1.000

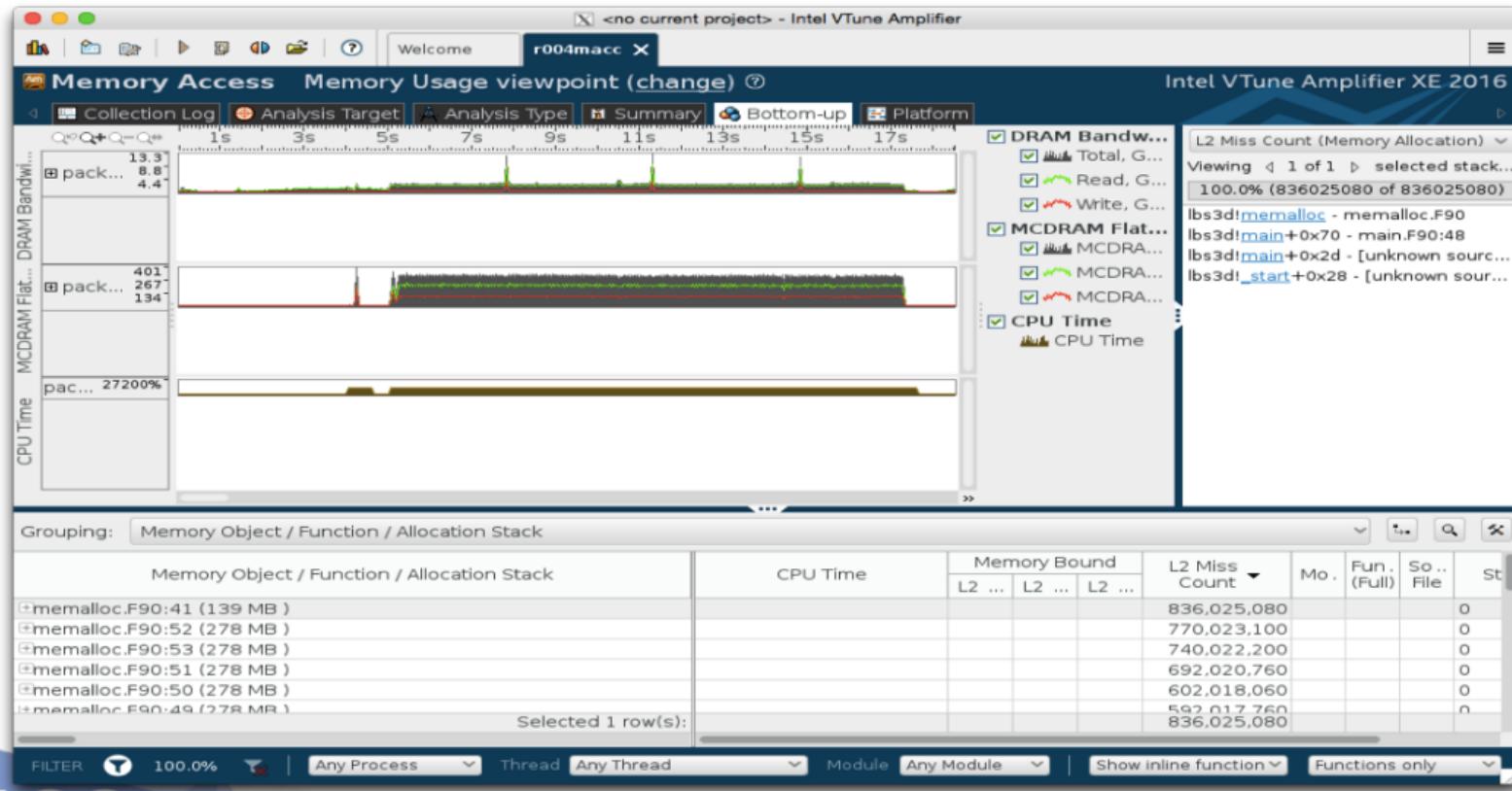
A note states: "A high number of CPU cycles is being spent waiting for L2 load misses to be serviced. Possible optimizations are to reduce data working set size, improve data access locality, blocking and consuming data in chunks that fit in the L2, or better exploit hardware prefetchers. Consider using software prefetchers when they can increase latency by interfering with normal loads, and can increase pressure on the memory system."
- MCDRAM Flat Bandwidth Bound:** 65.3%

The system spent a significant percentage of elapsed time with high MCDRAM Flat bandwidth utilization. Review the Bandwidth Utilization Histogram to estimate the scale of the issue. Consider improving data locality and/or merging compute-intensive code with bandwidth-intensive code.
- DRAM Bandwidth Bound:**
 - L2 Miss Count: 14,092,422,760
 - Total Thread Count: 140
 - Paused Time: 0s
- Bandwidth Utilization Histogram**
- Top Memory Objects**

This section lists the most actively used memory objects in your application.

Memory Object	L2 Miss Count
memalloc.F90:41 (139 MB.)	836,025,080
memalloc.F90:52 (278 MB.)	770,023,100
memalloc.F90:53 (278 MB.)	740,022,200
memalloc.F90:51 (278 MB.)	692,020,760
memalloc.F90:50 (278 MB.)	602,018,060
[others]	10,452,313,560

memory-access Object Details



hpc-performance

It's all about performance...

hpc-performance Summary

<no current project> - Intel VTune Amplifier

Welcome r006hpc X

HPC Performance Characterization (preview) HPC Performance Characterization viewpoint (change) Ⓜ

Collection Log Analysis Target Analysis Type Summary Bottom-up

Elapsed Time Ⓜ: 14.854s

GFLOPS Upper Bound Ⓜ: 92.504

CPU Utilization Ⓜ: 40.2%

- Average CPU Usage Ⓜ: 109.260 Out of 272 logical CPUs
- Serial Time Ⓜ: 2.085s (14.0%)
- Parallel Region Time Ⓜ: 12.769s (86.0%)
- Top OpenMP Regions by Potential Gain
- CPU Usage Histogram

Memory Bound

- L2 Hit Bound Ⓜ: 0.143
- L2 Miss Bound Ⓜ: 1.000
- A high number of CPU cycles is being spent waiting for L2 load misses to be serviced. Possible optimizations are to reduce data working set size, improve data access locality, blocking and consuming data in chunks that fit in the L2, or better exploit hardware prefetchers. Consider using software prefetchers but they can increase latency by interfering with normal loads, and can increase pressure on the memory system.
- MCDRAM Flat Bandwidth Bound Ⓜ: 84.0%
- The system spent a significant percentage of elapsed time with high MCDRAM Flat bandwidth utilization. Review the Bandwidth Utilization Histogram to estimate the scale of the issue. Consider improving data locality and/or merging compute-intensive code with bandwidth-intensive code.
- DRAM Bandwidth Bound Ⓜ: 0.0%
- Bandwidth Utilization Histogram

FPU Utilization Upper Bound Ⓜ: 1.6%

- GFLOPS Upper Bound Ⓜ: 92.504
 - Scalar GFLOPS Upper Bound Ⓜ: 0.000
 - Packed GFLOPS Upper Bound Ⓜ: 92.504
- 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 Ⓜ
fLoop at line 118 in collision_parallel@511	1101.094s	2.0%	Vectorized (Body)
fLoop at line 43 in stream_parallel_for@391	180.998s	0.7%	Vectorized (Body)
Manycore	Feb 3rd, 2017	29	

hpc-performance OpenMP Analysis

<no current project> - Intel VTune Amplifier

Welcome r006hpc X

HPC Performance Characterization (preview) HPC Performance Characterization viewpoint (change) Ⓜ

Collection Log Analysis Target Analysis Type Summary Bottom-up

Elapsed Time: 14.854s

GFLOPS Upper Bound: 92.504

CPU Utilization: 40.2% ⓘ

Average CPU Usage: 109.260 Out of 272 logical CPUs
Serial Time: 2.085s (14.0%)

Parallel Region Time: 12.769s (86.0%) ⓘ
Estimated Ideal Time: 12.090s (81.4%)
OpenMP Potential Gain: 0.679s (4.6%)

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 assuming no runtime overhead.

OpenMP Region	OpenMP Potential Gain (%)	OpenMP Region Time
collision_omp\$parallel:136@unknown:51:257	0.435s 2.9%	10.544s
stream_omp\$parallel:136@unknown:39:57	0.126s 0.9%	1.577s
poststream_omp\$parallel:136@unknown:37:362	0.072s 0.5%	0.389s
init_omp\$parallel:136@unknown:53:145	0.023s 0.2%	0.197s
postcollision_omp\$parallel:136@unknown:37:76	0.015s 0.1%	0.058s
[Others]	N/A*	0.005s

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

CPU Usage Histogram ⓘ

Memory Bound ⓘ

L2 Hit Bound: 0.143
L2 Miss Bound: 1.000

A high number of CPU cycles is being spent waiting for L2 load misses to be serviced. Possible optimizations are to reduce data working set size, improve data access locality, blocking and consuming data in chunks that fit in the L2, or better exploit hardware prefetchers. Consider using software prefetchers but they can increase latency by interfering with normal loads, and can increase pressure on the memory system.

MCDRAM Flat Bandwidth Bound: 84.0% ⓘ
The system spent a significant percentage of elapsed time with high MCDRAM Flat bandwidth utilization. Review the Bandwidth Utilization Histogram to estimate the scale of the issue. Consider improving data locality and/or merging compute-intensive code with bandwidth-intensive code.

Manycore | Feb 3rd, 2017 | 30

hpc-performance Memory Information

<no current project> - Intel VTune Amplifier

Welcome r006hpc X

HPC Performance Characterization (preview) HPC Performance Characterization viewpoint (change) ⓧ

Collection Log Analysis Target Analysis Type Summary Bottom-up

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

CPU Usage Histogram

Memory Bound

L2 Hit Bound: 0.143
L2 Miss Bound: 1.000

A high number of CPU cycles is being spent waiting for L2 load misses to be serviced. Possible optimizations are to reduce data working set size, improve data access locality, blocking and consuming data in chunks that fit in the L2, or better exploit hardware prefetchers. Consider using software prefetchers but they can increase latency by interfering with normal loads, and can increase pressure on the memory system.

MCDRAM Flat Bandwidth Bound: 84.0%

The system spent a significant percentage of elapsed time with high MCDRAM Flat bandwidth utilization. Review the Bandwidth Utilization Histogram to estimate the scale of the issue. Consider improving data locality and/or merging compute-intensive code with bandwidth-intensive code.

DRAM Bandwidth Bound: 0.0%

Bandwidth Utilization Histogram

This histogram displays a percentage of 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.

Bandwidth Domain: MCDRAM Flat, GB/sec

The histogram shows the distribution of bandwidth utilization over time. The Y-axis represents Elapsed Time from 0s to 2.5s. The X-axis represents Bandwidth Utilization from 0 to 400. The distribution is highly skewed, with most time spent at low utilization levels (below 100). A secondary distribution is visible between 250 and 350, peaking around 2.5s. A legend at the bottom indicates utilization levels: Low (green), Medium (orange), and High (red).

Bandwidth Utilization Range	Elapsed Time Range
Low (0-100)	0s to ~1.8s
Medium (100-250)	~1.8s to ~2.5s
High (250-400)	~2.5s to ~0.5s

Manycore | Feb 3rd, 2017 | 31

hpc-performance FPU Information

<no current project> - Intel VTune Amplifier

Welcome r006hpc X

HPC Performance Characterization (preview) HPC Performance Characterization viewpoint (change) ⓘ

Collection Log Analysis Target Analysis Type Summary Bottom-up

Elapsed Time ⓘ: 14.854s

GFLOPS Upper Bound ⓘ: 92.504

CPU Utilization ⓘ: 40.2%

Memory Bound

FPU Utilization Upper Bound ⓘ: 1.6% ⓘ

GFLOPS Upper Bound ⓘ: 92.504

Scalar GFLOPS Upper Bound ⓘ: 0.000

Packed GFLOPS Upper Bound ⓘ: 92.504

Top 5 hotspot loops (functions) by FPU usage

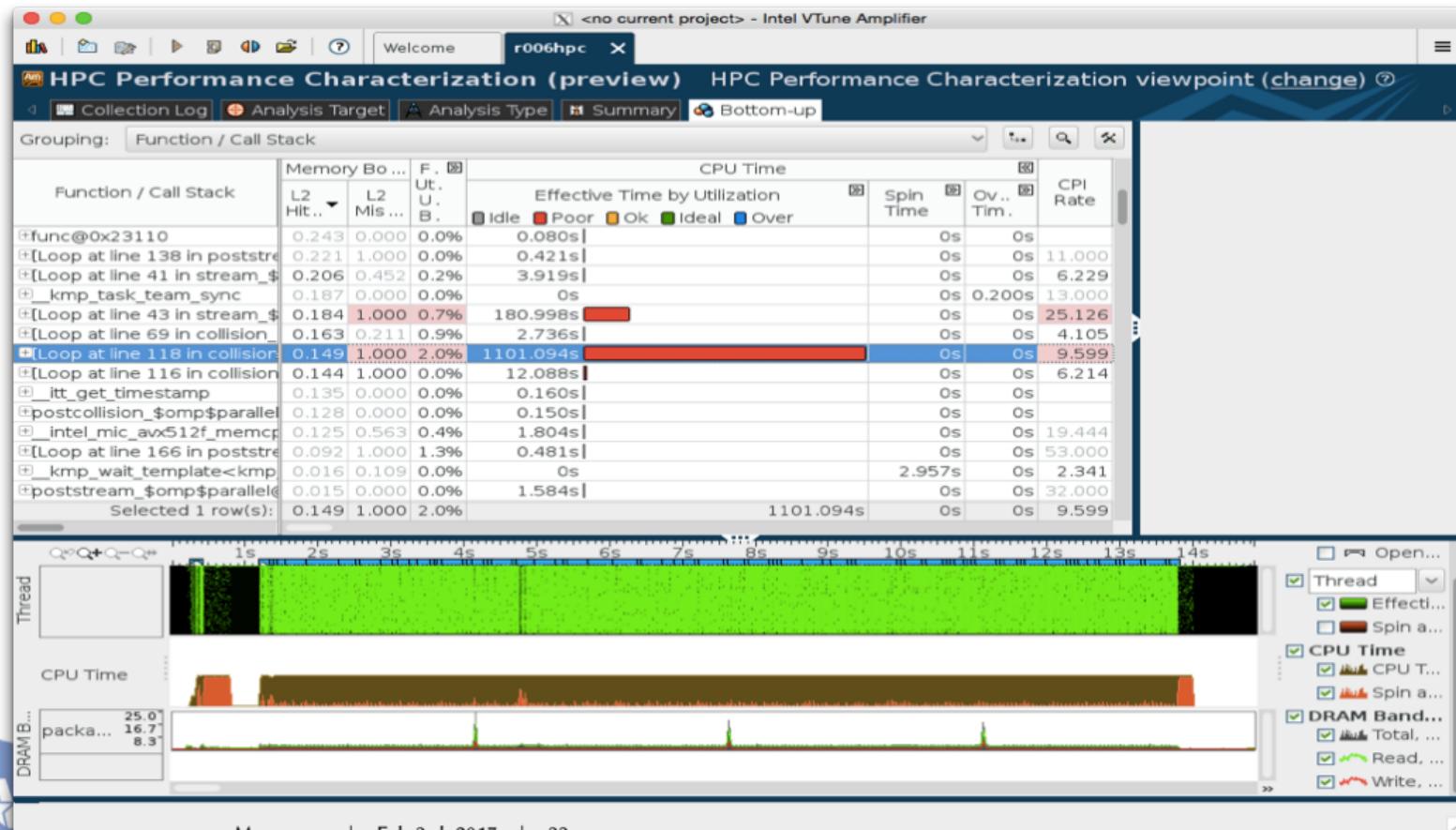
This section provides information for the most time consuming loops/functions with floating point operations.

Function	CPU Time ⓘ	FPU Utilization Upper Bound ⓘ	Loop Characterization ⓘ
[Loop at line 118 in collision_omp\$parallel@51]	1101.094s	2.0%	Vectorized (Body)
[Loop at line 43 in stream_omp\$parallel_for@39]	180.998s	0.7%	Vectorized (Body)
[Outside any known module]	99.089s	0.0%	
[Loop at line 71 in collision_omp\$parallel@51]	95.671s	4.2%	Vectorized (Body)
[Loop at line 58 in collision_omp\$parallel@51]	87.723s	0.6%	Vectorized (Body)
[Others]	21.179s	N/A*	

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

Collection and Platform Info

hpc-performance Bottom-up View



Basic VTune Workflow

Result finalization and viewing on KNL might be slow

- ▶ Compile with debug symbols
 - ▶ `$ mpicc -g -O3 -xMIC-AVX512 -qopenmp ./code.c`
- ▶ Run collection on KNL
 - ▶ `$ amplxe-cl -c hotspots ./app`
- ▶ Generate reports, work with GUI
 - ▶ `$ amplxe-gui`

About Advisor

- ▶ Intel Advisor is a vectorization optimization and shared memory threading assistance tool for C, C++, C#, and Fortran code
- ▶ Advisor supports both serial, threaded, and MPI applications
- ▶ Current version is 2017.1.0

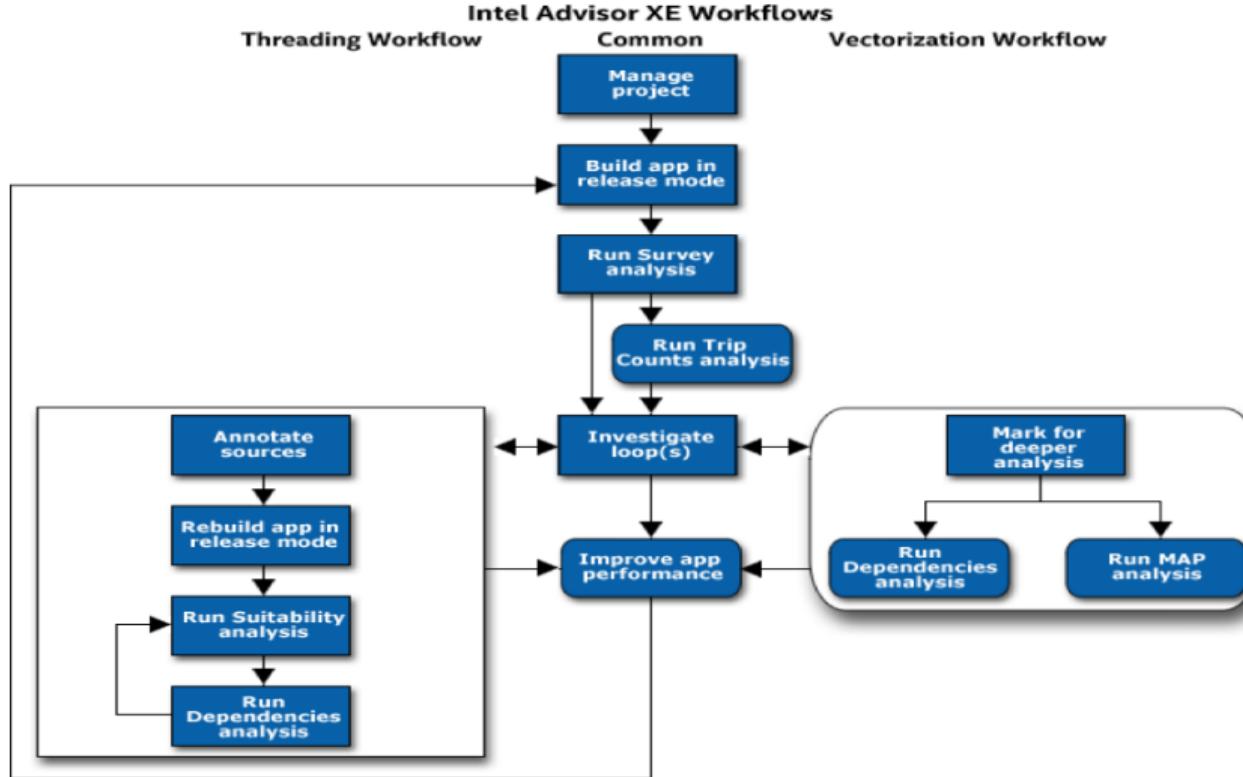
Advisor Capabilities

- ▶ Vectorization and Roofline Advisor
 - ▶ Evaluate the efficiency of vectorized code and key SIMD bottlenecks
 - ▶ Provide precise Trip/Call Counts, FLOPS and AVX-512 mask utilization
 - ▶ Check for loop-carried dependencies dynamically
 - ▶ Identify Memory vs. Compute balance via automated Roofline and Memory Access Pattern tools
- ▶ Threading Advisor
 - ▶ Discover where to add parallelism to your code by identifying where your code spends its time
 - ▶ You propose parallel code regions when you annotate the parallel sites and tasks
 - ▶ Predict the performance you might achieve with the proposed parallel code regions
 - ▶ Predict the data sharing problems that could occur in the proposed parallel code regions

Basic Advisor Tools

Tool	Description	Target Program Requirements
Survey	Helps you discover and select the best places to add parallelism in your program.	Moderate optimization Limited function inlining
Trip Counts	Helps you to collect loop iteration statistics.	Moderate optimization Run survey analysis first
Suitability	Helps you predict the likely performance impact of adding parallelism to the selected places.	Moderate optimization Limited function inlining
Dependencies	Helps you predict and eliminate data sharing problems before you add parallelism. 50-500 times slower.	Optimization disabled Minimize execution time
Memory Access Patterns	Helps you to collect data on memory access strides. 3-20 times slower.	Optimization disabled Minimize execution time

How-to use Advisor: Workflow



Intel Advisor Reference:

<https://software.intel.com/en-us/node/608339>

Using Advisor

- ▶ Setup environment
 - ▶ \$ module load advisor
 - ▶ \$ export OMP_NUM_THREADS=1 #for your OpenMP code
- ▶ Compile with optimization and debug symbols
 - ▶ \$icc -g -xMIC-AVX512 -O2 -qopt-report=5 demo.c
- ▶ Collect data
 - ▶ \$ advixe-cl -c survey --search-dir src:=./ ./a.out
- ▶ Analyze the data:
 - ▶ \$ advixe-gui

Advisor Summary

4410/huang/training/lab_advisor/advi - Intel Advisor

File View Help

Welcome e000

Vectorization Workflow Threading Workflow

Summary Survey Report Refinement Reports Annotation Report

INTEL ADVISOR XE 2016

Summary of predicted parallel behavior

Elapsed time: 5.41s Vectorized Not Vectorized FILTER: All Module All Source Loop All Threads

Vectorization Advisor

Vectorization Advisor is a vectorization analysis tool that lets you identify loops that will benefit most from vectorization.

Program metrics

Elapsed Time: 5.41s
Vector Instruction Set: AVX512
Number of CPU Threads: 1

Loop metrics

Total CPU time	5.40s	100.0%
Time in 1 vectorized loop	4.24s	78.5%
Time in scalar code	1.16s	21.5%

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	14.67x	~92%
Program Theoretical Gain	11.73x	

Top time-consuming loops

Loop	Source Location	Self Time [®]	Total Time [®]	Trip Counts [®]
main	org.c.35	4.2400s	4.2400s	320000
main	org.c.28	0.2102s	0.3900s	5120000
main	org.c.27	0.2000s	0.3800s	5120000
main	org.c.26	0.1600s	0.3898s	5120000
main	org.c.33	0s	4.2400s	500

Collection details

Platform information

Frequency:	1.40 GHz
Logical CPU Count:	272
Operating System:	Linux
Computer Name:	c569-062.stampede.tacc.utexas.edu

Survey: loop analytics

2410/huang/training/lab_advisor/advi - Intel Advisor

File View Help

Welcome e000 X

Vectorization Workflow Threading Workflow

Where should I add vectorization and/or threading parallelism?

Elapsed time: 5.41s Vectorized Not Vectorized FILTER: All Modules All Sources Loops All Threads

INTEL ADVISOR XE 2016

Batch mode

1. Survey Target

1.1 Find Trip Counts

Mark Loops for Deeper Analysis

Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.

-- There are no marked loops.

2.1 Check Dependencies

2.2 Check Memory Access

Source Top Down Loop Analytics Loop Assembly Recommendations Compiler Diagnostic Details

This is an early version of the Loop Analytics tab; it will be enhanced with more information over time.

4.24s

Vectorized (Body) Total time

AVX512F_512 4.24s

Instruction Set Self time

Memory 44% (4)
Compute 33% (3)
Other 22% (2)

Instruction Mix Summary

14.67x ~92%

Vectorization Gain
Achieved Vectorization Efficiency = (Estimated Gain/Vector Length) * 100%
Estimated Gain = 14.67x
Vector Length = 16
Orange color = Achieved vectorization efficiency is higher than reference efficiency for original scalar loop
Efficiency is approximately ~92%, which means actual efficiency may be lower

Traits

NT-stores

Instruction Mix

Memory: 4 Compute: 3 Other: 2 Number of Vector Registers: 4

Memory: 44.44% Compute: 33.33% Other: 22.22%

Vector: 44.44% Vector: 22.22% Scalar: 11

AVX-512: 44.44% AVX-512: 22.22% SIMD: 11

Survey: loop assembly

The screenshot shows the Intel Advisor XE 2016 interface. The main window title is "0410/huang/training/lab_advisor/advi - Intel Advisor". The top menu bar includes File, View, Help, and various tool icons. The left sidebar contains sections for Vectorization Workflow (Batch mode), Threading Workflow, Survey Target (Collect, Find Trip Counts), and Memory Access Patterns analysis (Check Dependencies, Check Memory Access). The central area displays a table titled "Where should I add vectorization and/or threading parallelism?" showing function call sites and loops. A detailed assembly view for module org!0x400e4b is shown, highlighting code blocks like "body", "Block 1", "Block 2", and "Block 3". The assembly code uses SIMD instructions such as vaddpdz, vpaddpd, and vpsubd. The bottom right corner shows a performance summary: Selected (Total Time): 0s.

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops	Th... Medi...	Instruction Set Analysis
						Vecto... AVX5... -92%	Efficiency 14.67x 16	Traits NT-stor... Exponent extractions; ... Exponent extractions; ... Exponent extractions; ... Mask Manipulations Data Ty... Float64 Float64 Float64 Float64 Int32; U...
+ [loop in main at org.c:35]	4.240s	4.240s	4.240s	Vectorized (Body)	function call cannot ...			
+ [loop in main at org.c:28]	0.210s	0.210s	0.210s	Scalar	function call cannot ...			NT-stores Exponent extractions; ...
+ [loop in main at org.c:27]	0.200s	0.200s	0.200s	Scalar	function call cannot ...			Exponent extractions; ...
+ [loop in main at org.c:26]	0.160s	0.160s	0.160s	Scalar	function call cannot ...			Exponent extractions; ...
+ [loop in main at org.c:33]	0.000s	4.240s	4.240s	Scalar	inner loop was already ...			Mask Manipulations Int32; U...

Source Top Down Loop Analytics Loop Assembly Recommendations Compiler Diagnostic Details

Module: org!0x400e4b

Address	Line	Assembly	Total Time	%	Self Time	%	Traits
0x400e80	35	cmp %r11, %r10	0.050s	0.050s			
0x400e83	35	jb 0x400e4b <Block 1>	0.560s	0.560s			
0x400e99		body:					
0x400e99	37	vmovupsz (%r14,%r11,8), %k0, %zmm4	1.100s	1.100s			
0x400ea0	37	vmovupsz 0x40(%r14,%r11,8), %k0, %zmm5	1.040s	1.040s			
0x400ea8	37	vaddpdz (%rbx,%r11,8), %zmm4, %k0, %zmm6	1.050s	1.050s			
0x400ea9	37	vaddpdz 0x40(%rbx,%r11,8), %zmm5, %k0, %zmm7	0.120s	0.120s			NT-store...
0x400eb7	37	vmovntpdz %zmm6, (%r15,%r11,8)					
0x400eb8	37	vmovntpdz %zmm7, 0x40(%r15,%r11,8)					
0x400ec6	35	add \$0x10, %r11	0.190s	0.190s			
0x400eca	35	cmp %r9, %r11	0.060s	0.060s			
0x400ecd	35	jb 0x400e99 <Block 2>	0.070s	0.070s			
0x400f0a		Block 2:					
0x400f0a	35	vpsubd %ymm0, %ymm5, %ymm6					
0x400f0e	37	lea (%rax,%r10,1), %r11d					
0x400f12	35	vpaddl %ymm2, %ymm5, %ymm5					
0x400f16	37	movsd %r11d, %r11					
0x400f19	35	add \$0x8, %r10d					
0x400f1d	35	vpcmpgtd %zmm4, %zmm6, %k1, %k0					
0x400f23	35	cmp %r9d, %r10d					
0x400f26	35	knotw %k0, %k2					
0x400f2a	37	vmovupdz (%r14,%r11,8), %k2{z}, %zmm7					Mask ...
0x400f31	37	vmovupdz (%rbx,%r11,8), %k2{z}, %zmm8					

Example of multi-run analysis

- ▶ Survey
 - ▶ \$ advixe-cl -c survey --search-dir src:=./ ./a.out
- ▶ Tripcounts
 - ▶ \$ advixe-cl -c tripcounts --search-dir src:=./ ./a.out
- ▶ Suitability
 - ▶ Add site annotations in source code, recompile, and run suitability analysis

```
$icc -g -xMIC-AVX512 -O2 -qopt-report=5 demo.c  
-I$ADVISOR_2017_DIR/include
```
 - ▶ \$ advixe-cl -c suitability --search-dir src:=./ ./a.out

Annotating Source Code

C/C++ Applications

```
#include <advisor-annotate.h>
...
ANNOTATE_SITE_BEGIN();
...
ANNOTATE_ITERATION_TASK (task1);
...
ANNOTATE_SITE_END();
```

Fortan Applications

```
use advisor_annotate
...
CALL ANNOTATE_SITE_BEGIN()
...
CALL ANNOTATE_ITERATION_TASK(task1)
...
CALL ANNOTATE_SITE_END()
```

From serial to annotated to OMP code

```
// matrix multiply routine
void multiply_d(int Size, double **aMatrix,double **bMatrix,
                double **product) {
    for(int i=0;i<Size;i++) {
        for(int j=0;j<Size;j++) {
            double sum = 0;
            for(int k=0;k<Size;k++) {
                sum += aMatrix[i][k] * bMatrix[k][j];
            }
            product[i][j] = sum;
        }
    }
}
```

From serial to annotated to OMP code

```
// matrix multiply routine
void multiply_d(int Size, double **aMatrix,double **bMatrix,
                double **product) {
    ANNOTATE_SITE_BEGIN(matrix_multiply);
    for(int i=0;i<Size;i++) {
        ANNOTATE_ITERATION_TASK(multiply_task);
        for(int j=0;j<Size;j++) {
            double sum = 0;
            for(int k=0;k<Size;k++) {
                sum += aMatrix[i][k] * bMatrix[k][j];
            }
            product[i][j] = sum;
        }
    }
    ANNOTATE_SITE_END();
}
```

Intel Advisor Reference:

<https://software.intel.com/en-us/node/608619>

Advisor Predict mult-thread performance for selected place

410/huang/training/lab_advisor/suit - Intel Advisor

File View Help

Welcome e000

INTEL ADVISOR XE 2016

What are the performance implications of the annotated sites?

Maximum Program Gain For All Sites: 1.18x

Target System: CPU Threading Model: OpenMP CPU Count: 8

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances	Site Instance Metrics, Parallel Time
"site_test"	demo.c:32	1.18x	Total Serial Time: 0.0142s	Total Parallel Time: 0.0026s Site Gain: 5.35x

Site Performance Scalability

Scalability of Maximum Site Gain

32x
16x
8x
4x
2x
1x

CPU Count: 2, 4, 8, 16, 32, 64

30.6% Load Imbalance: 0.0008s
19.4% Runtime Overhead: 0.0005s
0.0% Lock Contention: 0s

Total Parallel Time: 0.0026s

Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks):	Avg. Iteration (Task) Duration:
512	< 0.0001s 0.008x 0.040x 0.200x 1x (< 0.0001s)
5x	0.008x 0.040x 0.200x
25x	5x 25x
125x	125x

Runtime Modeling

Type of Change	Gain Benefit if Enabled
<input type="checkbox"/> Reduce Site Overhead	+1.12x
<input type="checkbox"/> Reduce Task Overhead	+1.12x
<input type="checkbox"/> Reduce Lock Overhead	
<input type="checkbox"/> Reduce Lock Contention	
<input type="checkbox"/> Enable Task Chunking	+1.15x

Apply

From serial to annotated to OMP code

```
// matrix multiply routine
void multiply_d(int Size, double **aMatrix,double **bMatrix,
                double **product) {
#pragma omp parallel for schedule(dynamic)
    for(int i=0;i<Size;i++) {
        for(int j=0;j<Size;j++) {
            double sum = 0;
            for(int k=0;k<Size;k++) {
                sum += aMatrix[i][k] * bMatrix[k][j];
            }
            product[i][j] = sum;
        }
    }
}
```

What Advisor Cannot Do

Intel Advisor is designed to analyze serial code. It does not catch all problems, and it cannot ensure that you have correctly implemented the parallelism! Before deploying your parallel program, you need to test it for dependencies and verify its performance.

Combine with other available tools

Intel Inspector includes a data race and deadlock detection tool that works on the parallel code. It can find more errors because it operates on the parallel code instead of working on the annotated serial code analyzed by the Dependencies tool. Intel Inspector also can find problems with memory: memory leaks, references to freed storage, references to uninitialized memory, and so forth. The memory-checking tool works on serial or parallel code.

VTune for hot spot and memory access analysis

References

Intel VTune Amplifier:

<https://software.intel.com/en-us/intel-VTune-amplifier-xe>

Intel Advisor:

<https://software.intel.com/en-us/get-started-with-advisor>

<https://software.intel.com/en-us/articles/advisorxe-tutorials>

Intel Software Documentation Library

<https://software.intel.com/en-us/intel-software-technical-documentation>