KNL PROFILING WITH VTUNE AMPLIFIER XE

Dmitry Prohorov (dmitry.prohorov@intel.com), VTune HPC Lead

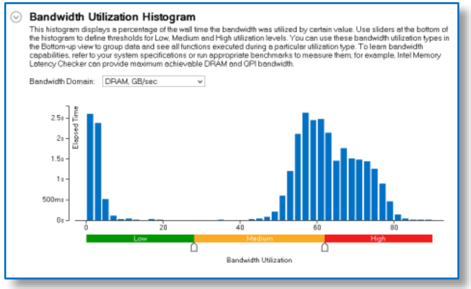
Agenda

- Overview
- System configuration
- Analysis Configuration
- Analysis Workflow
- Memory Access analysis
- Micro-arch analysis with General Exploration
- Performance Overview with HPC Performance Characterization (tech preview)
- Basic Hotspots, Concurrency, Locks and Waits

Overview

Explore Performance on Intel® Xeon Phi™ Processor (KNL Self Boot Linux)

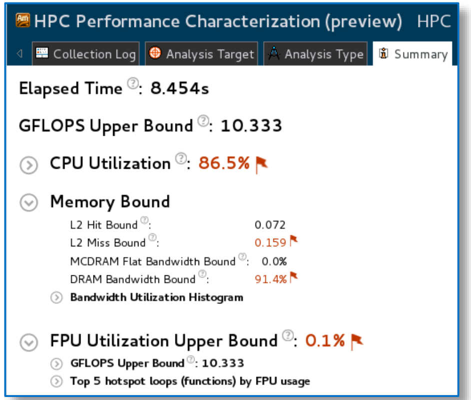
- Use **VTune Amplifier XE 2016 U4 and further** (no NDA package required)
- **Memory Access** analysis
  - Memory access problems by memory hierarchy
  - High Bandwidth Memory analysis
    - Defines if the app is DRAM or MCDRAM bandwidth bound
    - Helps to determine data structures worth to allocate to MCDRAM for DRAM bound apps
- Micro-architectural issues with **General Exploration** analysis
  - Explore how efficiently your code passing through the core pipeline
- Performance overview with **HPC Performance Characterization**
  - Important scalability aspects for OpenMP and hybrid MPI+OpenMP apps
    - CPU utilization
      - Serial vs Parallel time, imbalance, parallel runtime overhead cost, parallel loop parameters
    - Memory access efficiency
    - FPU utilization (upper bound), FLOPS (upper bound), basic loop vectorization info
- Algorithmic tuning opportunities with **Advanced Hotspots**
- Advanced measurements with Custom HW EBS event collection
- And more…



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: 8.454s

GFLOPS Upper Bound: 10.333

CPU Utilization: 86.5%

Memory Bound

L2 Hit Bound: 0.072

L2 Miss Bound: 0.159

MCDRAM Flat Bandwidth Bound: 0.0%

DRAM Bandwidth Bound: 91.4%

Bandwidth Utilization Histogram

FPU Utilization Upper Bound: 0.1%

GFLOPS Upper Bound: 10.333

Top 5 hotspot loops (functions) by FPU usage

System Configuration

Prerequisites for HW EBS event based collections

- VTune on KNL works with SEP driver (recommended) or upon perf
  - Related to: Advanced Hotspots, Memory Access, General Exploration, HPC Performance Characterization, custom event analysis
- Perf-based collection limitations:
  - Memory Access analysis is enabled with perf starting 2017 Gold
  - To enable uncore event collections, system wide collection set under sudo/root:

Highly recommended

```
>echo 0>/proc/sys/kernel/perf_event_paranoid
```

Pre-requisite for memory access analysis upon perf

Switch to system-wide mode with "analyze-system" option for application to launch to avoid perf collection overhead

  - To collect General Exploration or custom event collection with big number of events increase default limit of opened file descriptors:

In /etc/security/limits.conf increase default number to 100 * <number_of_logic_CPU_cores>:

```
<user> hard nofile <100 * number_of_logic_CPU_cores>
<user> soft nofile <100 * number_of_logic_CPU_cores>
```

- Default sampling interval on KNL is 10ms

Analysis Configuration (1/4)

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 **–trace-mpi** –r result_dir ./my_mpi_app
- >mpirun –n 48 -ppn 16 amplxe-cl –collect advanced-hotspots –r result_dir ./my_mpi_app

→ NB: -r <result_dir> is mandatory option for MPI profiling

- 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
- Enabled for perf-based collection from 2017 Gold
  - Highly recommended to set /proc/sys/kernel/perf_event_paranoid=0 to allow system wide collection to capture multiple ranks in profiling per node w/o overhead

Analysis Configuration (2/4)

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:

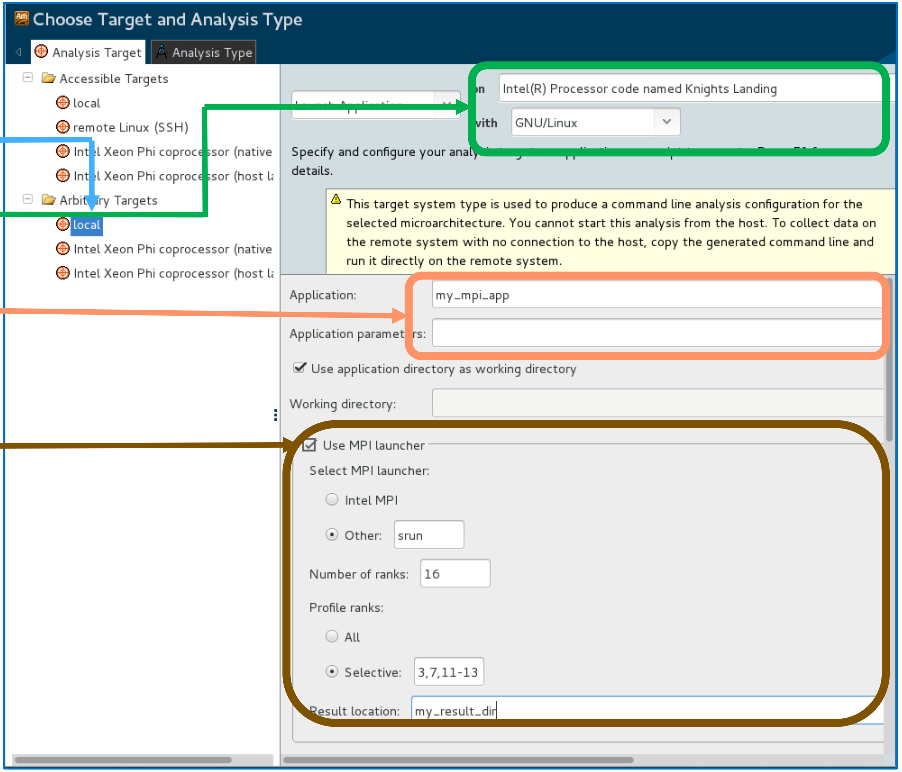
```
>mpirun –gtool “amplxe-cl –collect advanced-hotapots –r result_dir:3,7,11-13” ./my_app
```

Analysis Configuration (3/4)

MPI Profiling Command Line Generation from GUI

1. Create a VTune project
2. Choose “Arbitrary Targets/Local”
3. Set processor arch and OS
4. Set application name and parameters
5. Check “Use MPI Launcher”

   Provide the launcher name, number of ranks, ranks to profile, set result directory


Choose Target and Analysis Type
Analysis Target
Analysis Type
Accessible Targets
local
remote Linux (SSH)
Intel Xeon Phi coprocessor (native
Intel Xeon Phi coprocessor (host l
Arbitrary Targets
local
Intel Xeon Phi coprocessor (native
Intel Xeon Phi coprocessor (host l
Launch Application
Specify and configure your analy
details.

n Intel(R) Processor code named Knights Landing

with GNU/Linux

This target system type is used to produce a command line analysis configuration for the selected microarchitecture. You cannot start this analysis from the host. To collect data on the remote system with no connection to the host, copy the generated command line and run it directly on the remote system.

Application: my_mpi_app

Application parameters:

☑ Use application directory as working directory

Working directory:

☑ Use MPI launcher

Select MPI launcher:

○ Intel MPI

● Other: srun

Number of ranks: 16

Profile ranks:

○ All

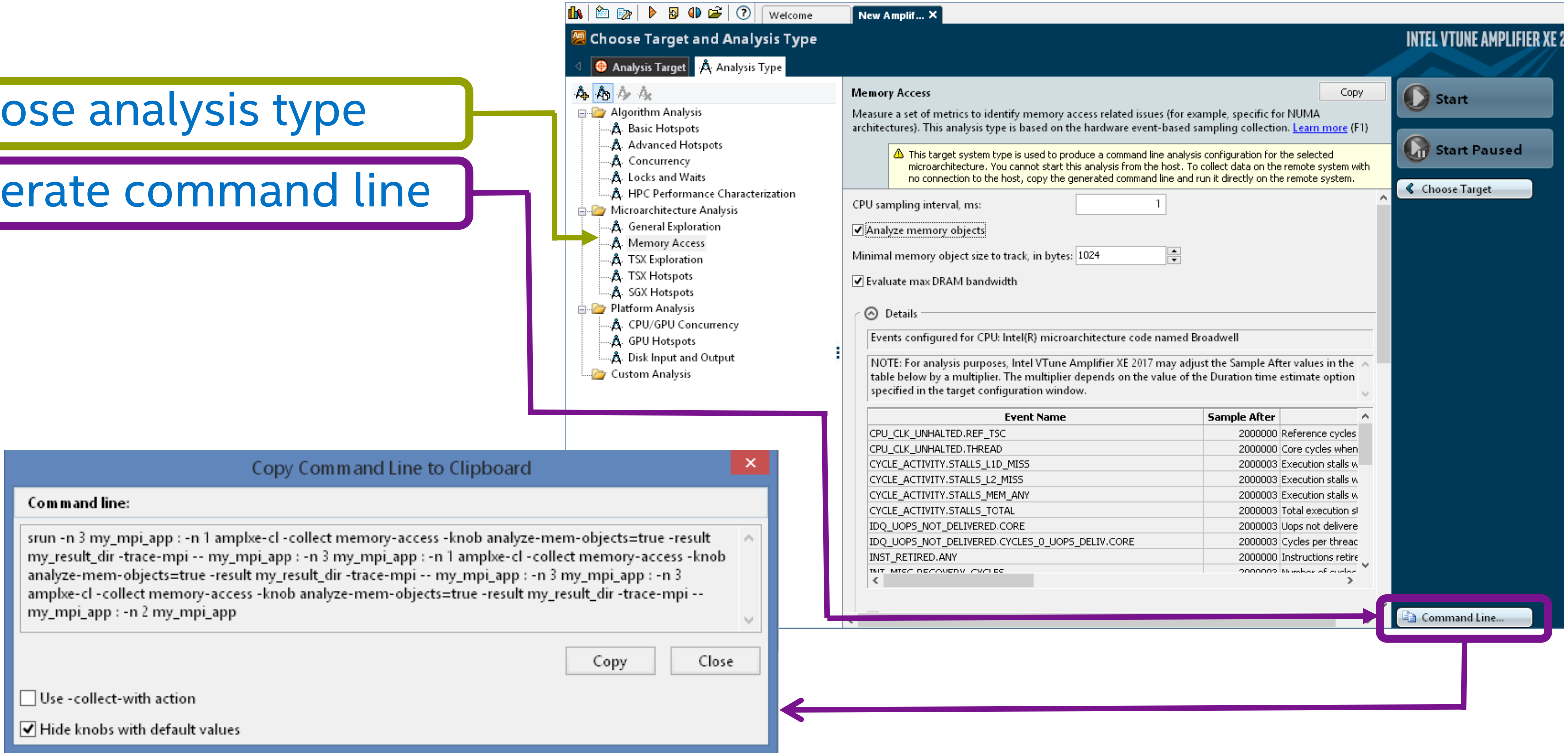
● Selective: 3,7,11-13

Result location: my_result_dir

Analysis Configuration (4/4)

MPI Profiling Command Line Generation from GUI

6. Choose analysis type
7. Generate command line



Memory Access

Measure a set of metrics to identify memory access related issues (for example, specific for NUMA architectures). This analysis type is based on the hardware event-based sampling collection. Learn more (F1)

⚠ This target system type is used to produce a command line analysis configuration for the selected microarchitecture. You cannot start this analysis from the host. To collect data on the remote system with no connection to the host, copy the generated command line and run it directly on the remote system.

CPU sampling interval, ms: 1

☑ Analyze memory objects

Minimal memory object size to track, in bytes: 1024

☑ Evaluate max DRAM bandwidth

Events configured for CPU: Intel(R) microarchitecture code named Broadwell

NOTE: For analysis purposes, Intel VTune Amplifier XE 2017 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the target configuration window.

Event Name	Sample After	
CPU_CLK_UNHALTED.REF_TSC	2000000	Reference cycles
CPU_CLK_UNHALTED.THREAD	2000000	Core cycles when
CYCLE_ACTIVITY.STALLS_L1D_MISS	2000003	Execution stalls w
CYCLE_ACTIVITY.STALLS_L2_MISS	2000003	Execution stalls w
CYCLE_ACTIVITY.STALLS_MEM_ANY	2000003	Execution stalls w
CYCLE_ACTIVITY.STALLS_TOTAL	2000003	Total execution st
IDQ_UOPS_NOT_DELIVERED.CORE	2000003	Uops not delivere
IDQ_UOPS_NOT_DELIVERED.CYCLES_0_UOPS_DELIV.CORE	2000003	Cycles per threac
INST_RETIRED.ANY	2000000	Instructions retire

Command line:

```
srun -n 3 my_mpi_app : -n 1 amplxe-cl -collect memory-access -knob analyze-mem-objects=true -result my_result_dir -trace-mpi -- my_mpi_app : -n 3 my_mpi_app : -n 1 amplxe-cl -collect memory-access -knob analyze-mem-objects=true -result my_result_dir -trace-mpi -- my_mpi_app : -n 3 my_mpi_app : -n 3 amplxe-cl -collect memory-access -knob analyze-mem-objects=true -result my_result_dir -trace-mpi -- my_mpi_app : -n 2 my_mpi_app
```

☐ Use -collect-with action

☑ Hide knobs with default values

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>
```

Memory Access Analysis

Motivation

- **More ranks/threads** to load many-core processor -> **more data** fetching to saturate computations
- Faster reaching DRAM bandwidth limit (~90GB/s) for BW bound apps and it hurts performance
- On-package MCDRAM (400Gb/s+) helps to extend BW limit and increase performance
  - MCDRAM in cache mode is transparent for a user, might bring additional latency if working set does not suit MCDRAM size (16GB)
  - MCDRAM in flat mode is more flexible – user can control what data to allocate to MCDRAM
    - If working set suits 16GB – use “numactl –m 1” to allocate all data to MCDRAM
    - If working set is more than 16GB – use memkind lib to allocate data structures that induce L2 (LLC on KNL) miss traffic

USE VTUNE TO DEFINE PROPER DATA STRUCTURES FOR MCDRAM ALLOCATION

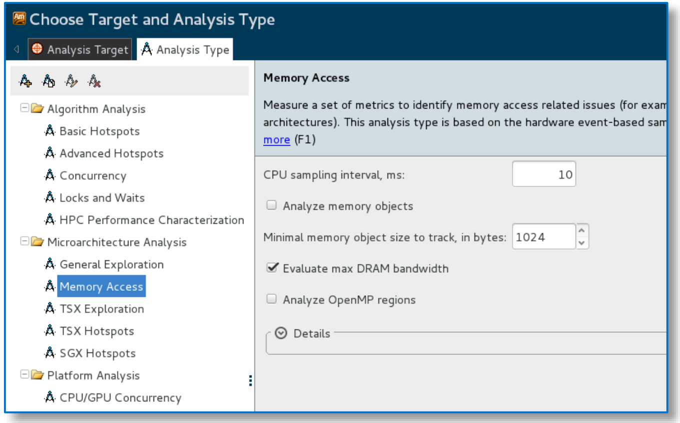
Memory Access Analysis

Configuration

Configuration options:

- **Analyze memory objects:**
  - Enables the instrumentation of memory allocation/de-allocation and mapping hardware events to memory objects
  - May cause additional runtime overhead due to the instrumentation of all system memory allocation/de-allocation API
- **Minimal memory object size to track, in bytes:**
  - Specify a minimal size of memory allocations to analyze. This option helps reduce runtime overhead of the instrumentation
- “Evaluate max DRAM Bandwidth” is not enabled on KNL, limits hardcoded:
  - DRAM: 90GB/s
  - MCDRAM 350GB/s

Can be adjusted on BW histogram per result if needed



Memory Access

Measure a set of metrics to identify memory access related issues (for exam
architectures). This analysis type is based on the hardware event-based sam
more (F1)

CPU sampling interval, ms: 10

☐ Analyze memory objects

Minimal memory object size to track, in bytes: 1024

☑ Evaluate max DRAM bandwidth

☐ Analyze OpenMP regions

```
>amplxe-cl -c memory-access -knob analyze-mem-objects=true -knob mem-object-size-min-thres=1024 -- <app>
```

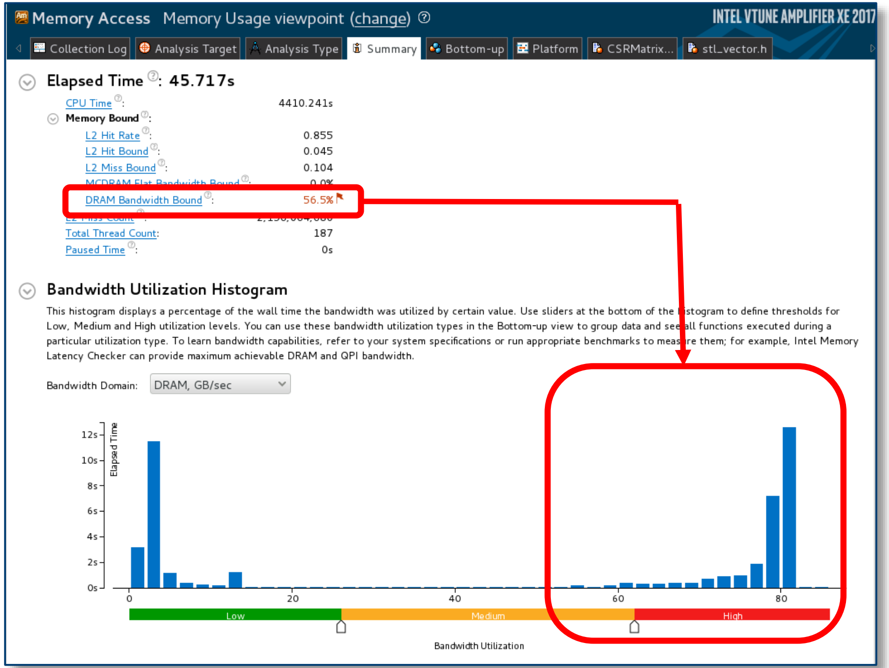
Memory Access Analysis

High Bandwidth Analysis. Step 1

Explore DRAM Bandwidth Bound metric and histogram on summary to see if the app is bandwidth bound

Significant portion of application time spent in high memory bandwidth utilization

The app may benefit from MCDRAM



Memory Access Memory Usage viewpoint (change)

Elapsed Time: 45.717s

CPU Time: 4410.241s

Memory Bound:

L2 Hit Rate: 0.855

L2 Hit Bound: 0.045

L2 Miss Bound: 0.104

MCDRAM Flat Bandwidth Bound: 0.0%

DRAM Bandwidth Bound: 56.5%

Total Thread Count: 187

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.

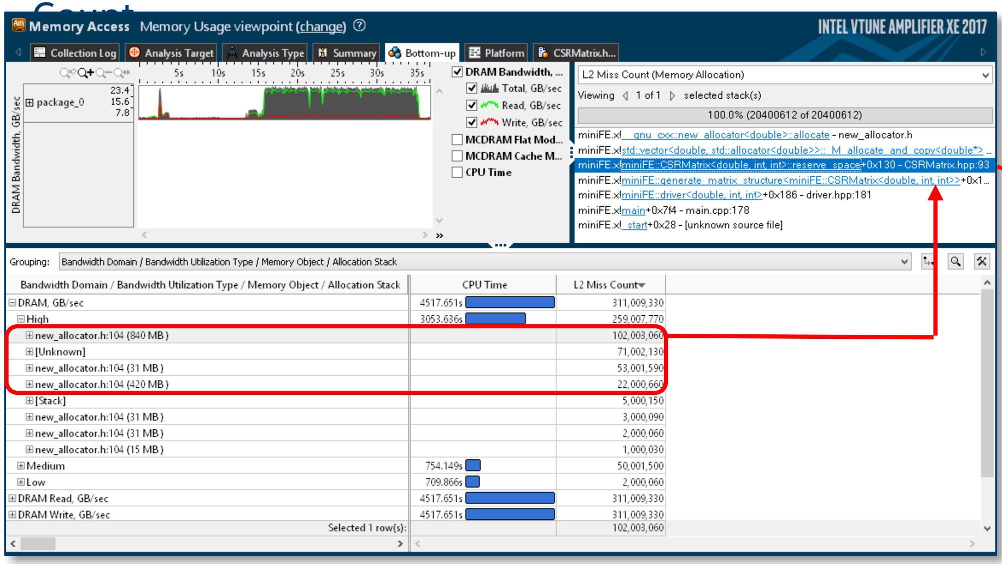
Bandwidth Domain: DRAM, GB/sec

Memory Access Analysis

High Bandwidth Analysis. Step 2

Investigate the memory allocations inducing bandwidth

- “Bandwidth Domain/Bandwidth Utilization Type/Memory Object/Allocation Stack” grouping with expansion by “DRAM/High” and sorting by L2 Miss Count



L2 Miss Count (Memory Allocation)

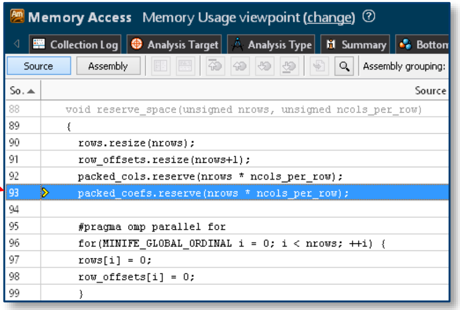
Viewing ◁ 1 of 1 ▷ selected stack(s)

100.0% (20400612 of 20400612)

```
miniFE.x!__gnu_cxx::new_allocator<double>::allocate - new_allocator.h
miniFE.x!std::vector<double, std::allocator<double>>::_M_allocate_and_copy<double*> ...
miniFE.x!miniFE::CSRMatrix<double, int, int>::reserve_space+0x130 - CSRMatrix.hpp:93
miniFE.x!miniFE::generate_matrix_structure<miniFE::CSRMatrix<double, int, int>>+0x1...
miniFE.x!miniFE::driver<double, int, int>+0x186 - driver.hpp:181
miniFE.x!main+0x7f4 - main.cpp:178
miniFE.x!_start+0x28 - [unknown source file]
```

Grouping: Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack

Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack	CPU Time	L2 Miss Count▼
⊟DRAM, GB/sec	4517.651s	311,009,330
⊟High	3053.636s	259,007,770
⊞new_allocator.h:104 (840 MB)		102,003,060
⊞[Unknown]		71,002,130
⊞new_allocator.h:104 (31 MB)		53,001,590
⊞new_allocator.h:104 (420 MB)		22,000,660
⊞[Stack]		5,000,150
⊞new_allocator.h:104 (31 MB)		3,000,090
⊞new_allocator.h:104 (31 MB)		2,000,060
⊞new_allocator.h:104 (15 MB)		1,000,030
⊞Medium	754.149s	50,001,500
⊞Low	709.866s	2,000,060
⊞DRAM Read, GB/sec	4517.651s	311,009,330
⊞DRAM Write, GB/sec	4517.651s	311,009,330
Selected 1 row(s):		102,003,060



```
88  void reserve_space(unsigned nrows, unsigned ncols_per_row)
89  {
90    rows.resize(nrows);
91    row_offsets.resize(nrows+1);
92    packed_cols.reserve(nrows * ncols_per_row);
93    packed_coefs.reserve(nrows * ncols_per_row);
94
95    #pragma omp parallel for
96    for(MINIFE_GLOBAL_ORDINAL i = 0; i < nrows; ++i) {
97    rows[i] = 0;
98    row_offsets[i] = 0;
99    }
```

Focus on allocations inducing L2 misses

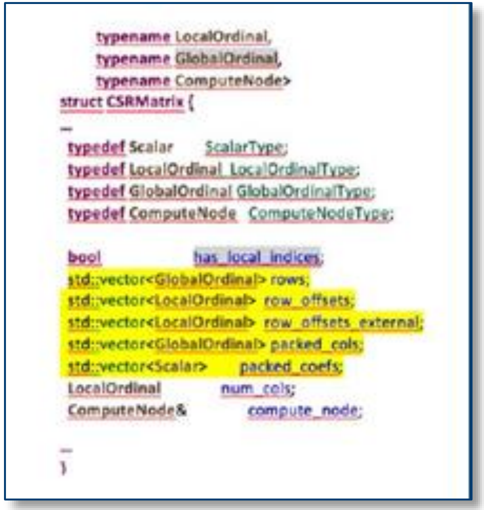
Allocation stack shows the allocation place in user’s code

Memory Access Analysis

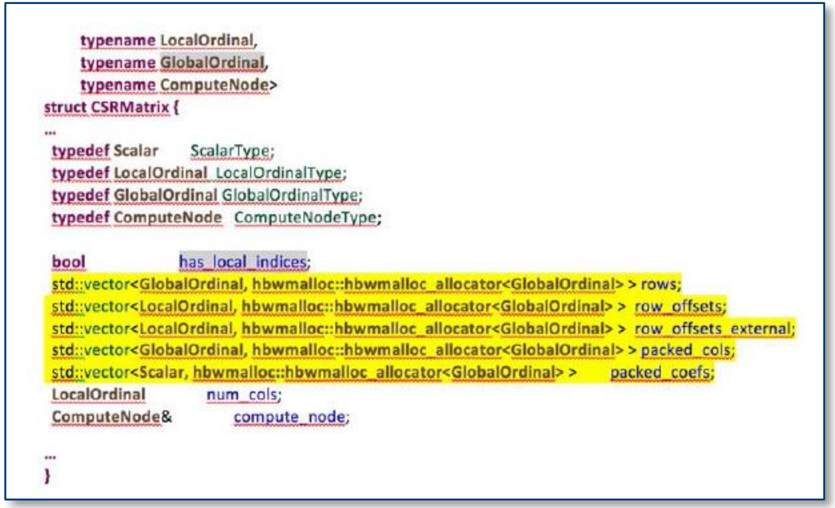
High Bandwidth Analysis. Step 3

Allocate the data structures to MCDRAM using High Bandwidth Memory

E.g.: specifying a custom memory allocator class from memkind lib for stored vector elements



```
        typename LocalOrdinal,
        typename GlobalOrdinal,
        typename ComputeNode>
struct CSRMatrix {
...
 typedef Scalar       ScalarType;
 typedef LocalOrdinal LocalOrdinalType;
 typedef GlobalOrdinal GlobalOrdinalType;
 typedef ComputeNode  ComputeNodeType;

 bool                 has_local_indices;
 std::vector<GlobalOrdinal> rows;
 std::vector<LocalOrdinal>  row_offsets;
 std::vector<LocalOrdinal>  row_offsets_external;
 std::vector<GlobalOrdinal> packed_cols;
 std::vector<Scalar>      packed_coefs;
 LocalOrdinal           num_cols;
 ComputeNode&             compute_node;

...
}
```



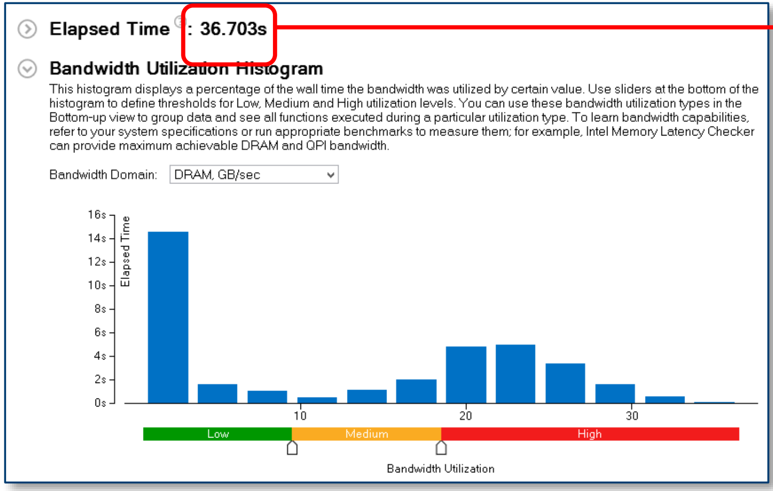
```
        typename LocalOrdinal,
        typename GlobalOrdinal,
        typename ComputeNode>
struct CSRMatrix {
...
 typedef Scalar       ScalarType;
 typedef LocalOrdinal LocalOrdinalType;
 typedef GlobalOrdinal GlobalOrdinalType;
 typedef ComputeNode  ComputeNodeType;

 bool                 has_local_indices;
 std::vector<GlobalOrdinal, hbwmalloc::hbwmalloc_allocator<GlobalOrdinal> > rows;
 std::vector<LocalOrdinal, hbwmalloc::hbwmalloc_allocator<GlobalOrdinal> >  row_offsets;
 std::vector<LocalOrdinal, hbwmalloc::hbwmalloc_allocator<GlobalOrdinal> >  row_offsets_external;
 std::vector<GlobalOrdinal, hbwmalloc::hbwmalloc_allocator<GlobalOrdinal> > packed_cols;
 std::vector<Scalar, hbwmalloc::hbwmalloc_allocator<GlobalOrdinal> >      packed_coefs;
 LocalOrdinal           num_cols;
 ComputeNode&             compute_node;

...
}
```

Or using "numactl –m 1"if the whole working set suits 16 GB

Memory Access Analysis

High Bandwidth Analysis. Step 4

Rerun the benchmark

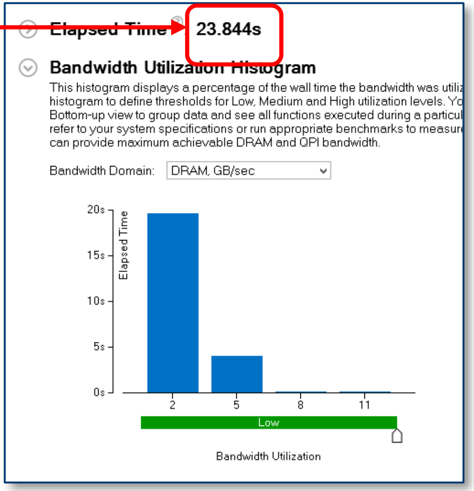


Elapsed Time: 36.703s

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

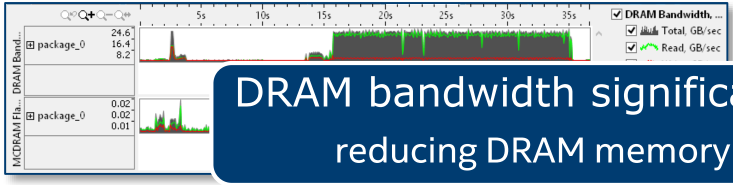


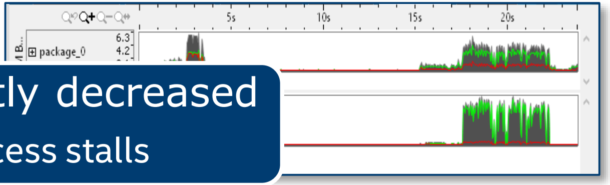
Elapsed Time: 23.844s

Bandwidth Utilization Histogram

This histogram displays a percentage of the wall time the bandwidth was utili
histogram to define thresholds for Low, Medium and High utilization levels. Yo
Bottom-up view to group data and see all functions executed during a particul
refer to your system specifications or run appropriate benchmarks to measur
can provide maximum achievable DRAM and QPI bandwidth.

Bandwidth Domain: DRAM, GB/sec



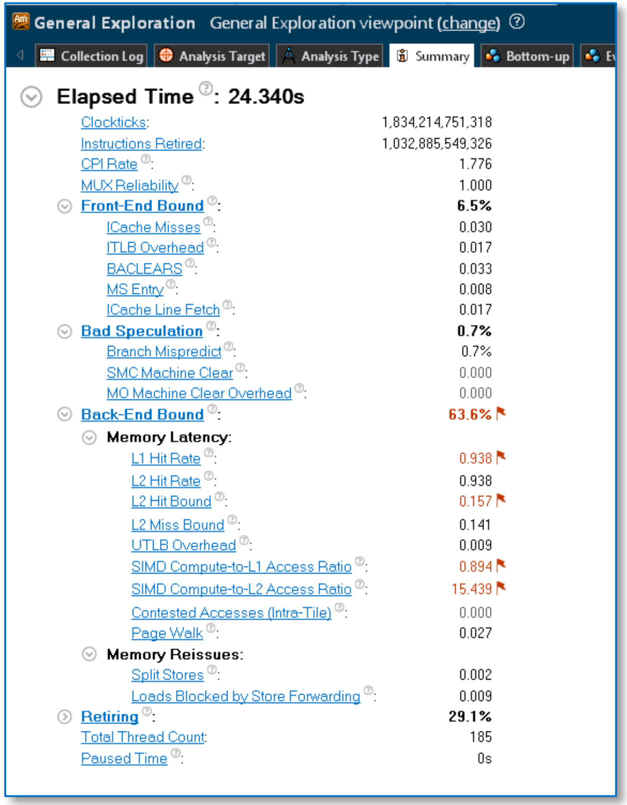


DRAM bandwidth significantly decreased

reducing DRAM memory access stalls

Micro-arch analysis with General Exploration

- 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



```
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
```

Performance Overview with HPC Performance Characterization Motivation

Show important aspects of application performance in one analysis

- Entry point to assess application efficiency on system resources utilization with definition of the next steps to investigate pathologies with significant performance cost
- Monitor how parameters of a run or code changes impact important different performance aspects to better understand their impact on elapsed time

Customers asking

- I eliminated imbalance with dynamic scheduling but elapsed time of my application became worse, why?
- I vectorized the code but don't have much benefit, why?
- I moved from pure MPI to MPI + OpenMP but the results are much worse, why?

CPU Utilization, Memory efficiency and FPU utilization aspects are correlated – let's explore them in one view

- Run HPC Performance Characterization analysis

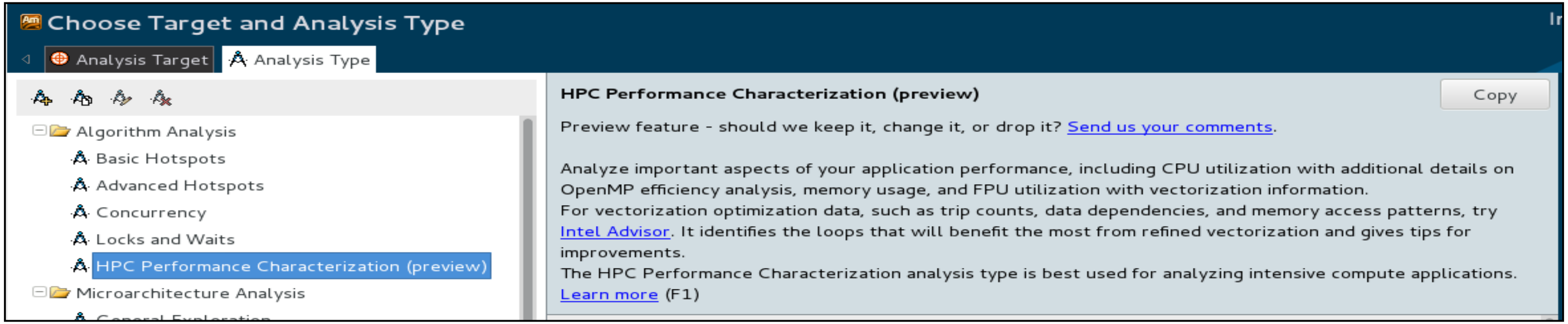
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Performance Overview with HPC Performance Characterization Motivation

CL

```
>amplxe-cl –collect hpc-performance <options> ./my_app
>mpirun –n 16 –ppn 4 –collect hpc-performance –r result_dir ./my_mpi_app
```

GUI



HPC Performance Characterization (preview)

Preview feature - should we keep it, change it, or drop it? Send us your comments.

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

Optimization Notice
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

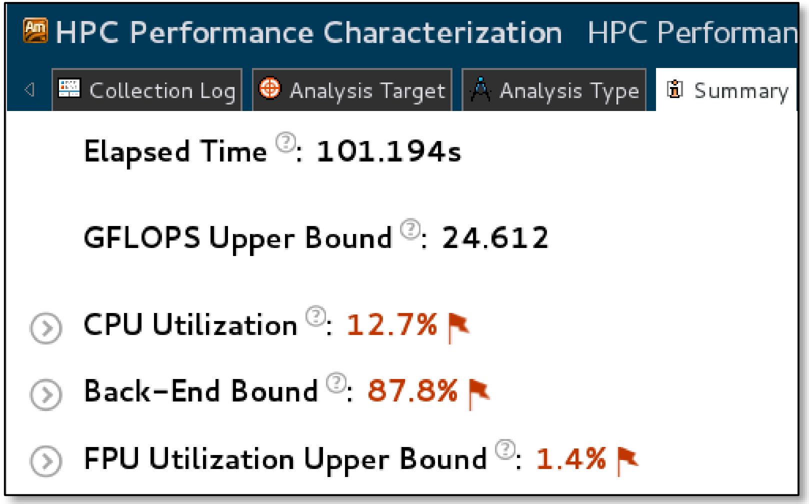
Performance Overview with HPC Performance Characterization Analysis Structure and Metrics

Two characterization metrics

- Elapsed Time
- GFLOPs Upper Bound*

Three performance aspects

- CPU Utilization
- Memory Bound
- FPU Utilization Upper Bound*



Elapsed Time: 101.194s

GFLOPS Upper Bound: 24.612

CPU Utilization: 12.7%

Back-End Bound: 87.8%

FPU Utilization Upper Bound: 1.4%

*Calculated based on FLOP HW counters assuming full vector utilization

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

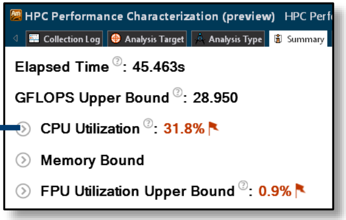
Performance Overview with HPC Performance Characterization
CPU Utilization

CPU Utilization

- % of “Effective” CPU usage by the application under profiling (threshold 90%)
  - Under assumption that the app should use all available logical cores on a node
  - Subtracting spin/overhead time spent in MPI and threading runtimes

Metrics in CPU utilization section

- Average CPU usage
- Additional MPI and OpenMP scalability metrics impacting effective CPU utilization
- CPU utilization histogram



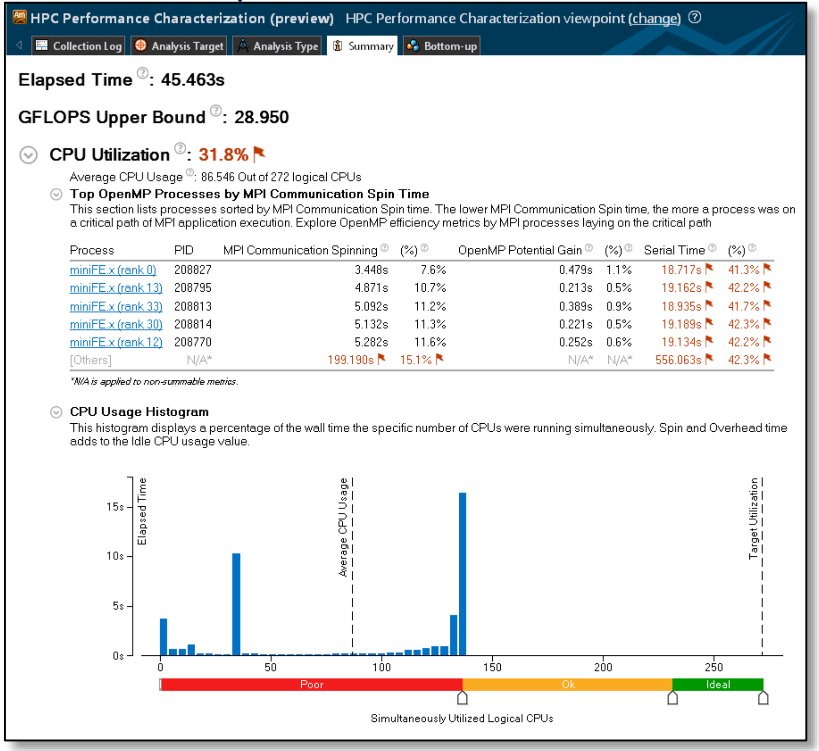
Elapsed Time: 45.463s

GFLOPS Upper Bound: 28.950

CPU Utilization: 31.8% ⚑

Memory Bound

FPU Utilization Upper Bound: 0.9% ⚑


HPC Performance Characterization (preview) HPC Performance Characterization viewpoint (change)
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up

Elapsed Time: 45.463s

GFLOPS Upper Bound: 28.950

CPU Utilization: 31.8% ⚑

Average CPU Usage: 86.546 Out of 272 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 a process was on a critical path of MPI application execution. Explore OpenMP efficiency metrics by MPI processes laying on the critical path

Process	PID	MPI Communication Spinning	(%)	OpenMP Potential Gain	(%)	Serial Time	(%)
miniFE.x (rank 0)	208827	3.448s	7.6%	0.479s	1.1%	18.717s ⚑	41.3% ⚑
miniFE.x (rank 13)	208795	4.871s	10.7%	0.213s	0.5%	19.162s ⚑	42.2% ⚑
miniFE.x (rank 33)	208813	5.092s	11.2%	0.389s	0.9%	18.935s ⚑	41.7% ⚑
miniFE.x (rank 30)	208814	5.132s	11.3%	0.221s	0.5%	19.189s ⚑	42.3% ⚑
miniFE.x (rank 12)	208770	5.282s	11.6%	0.252s	0.6%	19.134s ⚑	42.2% ⚑
[Others]	N/A*	199.190s ⚑	15.1% ⚑	N/A*	N/A*	556.063s ⚑	42.3% ⚑

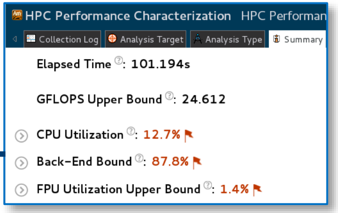
**N/A is applied to non-summable metrics.*

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.

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Performance Overview with HPC Performance Characterization Memory Bound



Elapsed Time: 101.194s

GFLOPS Upper Bound: 24.612

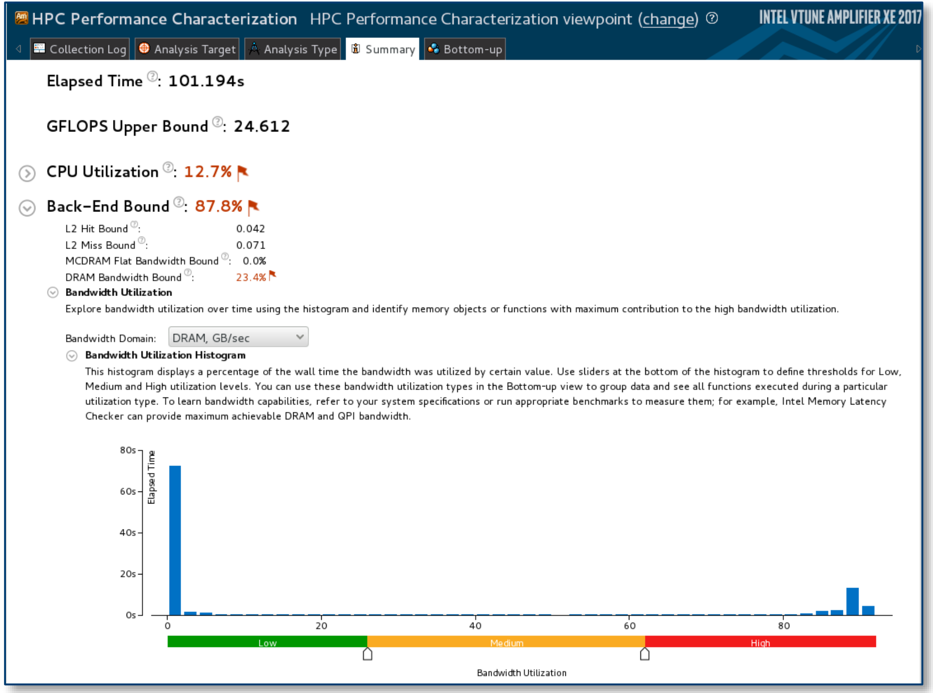
CPU Utilization: 12.7%

Back-End Bound: 87.8%

FPU Utilization Upper Bound: 1.4%

Since no memory stall measurement on KNL "Memory Bound" high level metric replaced with Backed-Bound with second level based on misses and bandwidth measurement from uncore events:

- L2Hit Bound
  - Cost of L1 misses served in L2
- L2 Miss Bound
  - Cost of L2 misses
- MCDRAM Bandwidth Bound
  - % of app elapsed time consuming high MCDRAM Bandwidth
- MCDRAM Bandwidth Bound
  - % of app elapsed time consuming high MCDRAM Bandwidth
- Bandwidth utilization histogram



Elapsed Time: 101.194s

GFLOPS Upper Bound: 24.612

CPU Utilization: 12.7%

Back-End Bound: 87.8%

L2 Hit Bound:	0.042
L2 Miss Bound:	0.071
MCDRAM Flat Bandwidth Bound:	0.0%
DRAM Bandwidth Bound:	23.4%

Bandwidth Utilization

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

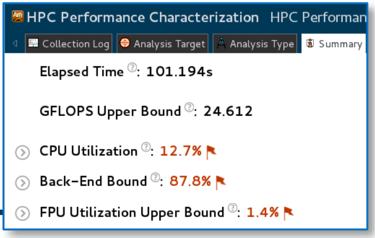
Bandwidth Domain: DRAM, GB/sec

Bandwidth Utilization Histogram

This histogram displays 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.

Optimization Notice
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Performance Overview with HPC Performance Characterization FPU Utilization



Elapsed Time: 101.194s

GFLOPS Upper Bound: 24.612

CPU Utilization: 12.7%

Back-End Bound: 87.8%

FPU Utilization Upper Bound: 1.4%

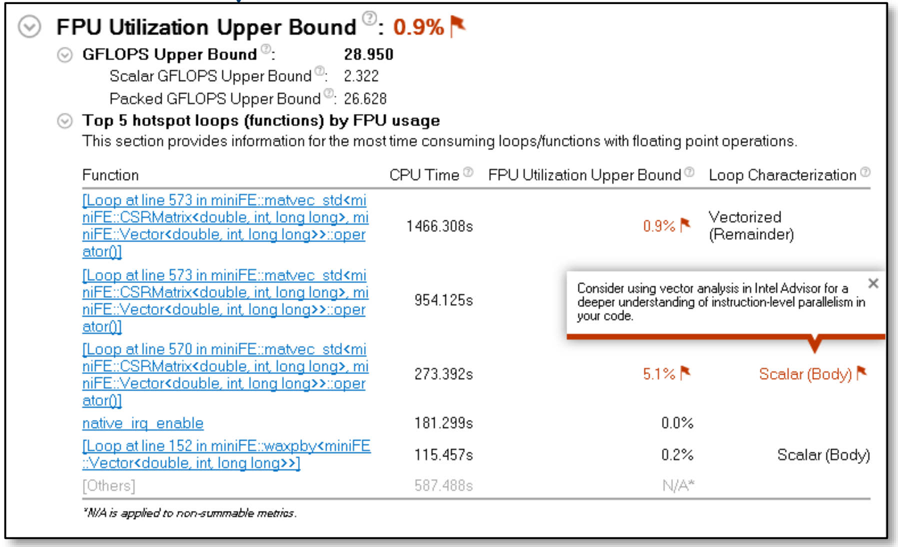
FPU utilization Upper Bound

- % of FPU load (100% - FPU is fully loaded, threshold 50%)

Metrics in FPU utilization section

- GLOPs broken down by scalar and packed
- Top 5 loops/functions by FPU usage
  - Dynamically generated issue descriptions on low FPU usage help to define the reason and next steps:

    Non-vectorized, vectorized with legacy instruction set, memory bound limited loops not benefiting from vectorization etc.
- Vector instruction set, FP ratio and FLOPs per cycle are available in Grid per rank/loop/region



FPU Utilization Upper Bound: 0.9%

GFLOPS Upper Bound: 28.950

Scalar GFLOPS Upper Bound: 2.322

Packed GFLOPS Upper Bound: 26.628

Top 5 hotspot loops (functions) by FPU usage

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

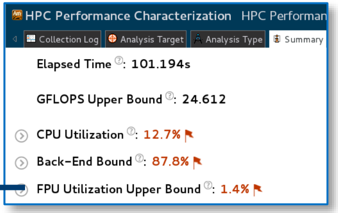
Function	CPU Time	FPU Utilization Upper Bound	Loop Characterization
[Loop at line 573 in miniFE::matvec_std<miniFE::CSRMatrix<double, int, long long>, miniFE::Vector<double, int, long long>>::operator()]	1466.308s	0.9%	Vectorized (Remainder)
[Loop at line 573 in miniFE::matvec_std<miniFE::CSRMatrix<double, int, long long>, miniFE::Vector<double, int, long long>>::operator()]	954.125s		
[Loop at line 570 in miniFE::matvec_std<miniFE::CSRMatrix<double, int, long long>, miniFE::Vector<double, int, long long>>::operator()]	273.392s	5.1%	Scalar (Body)
native_irq_enable	181.299s	0.0%	
[Loop at line 152 in miniFE::waxpby<miniFE::Vector<double, int, long long>>]	115.457s	0.2%	Scalar (Body)
[Others]	587.488s	N/A*	

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

Consider using vector analysis in Intel Advisor for a deeper understanding of instruction-level parallelism in your code.

Optimization Notice
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Performance Overview with HPC Performance Characterization FPU Utilization



Elapsed Time: 101.194s

GFLOPS Upper Bound: 24.612

CPU Utilization: 12.7% ⚑

Back-End Bound: 87.8% ⚑

FPU Utilization Upper Bound: 1.4% ⚑

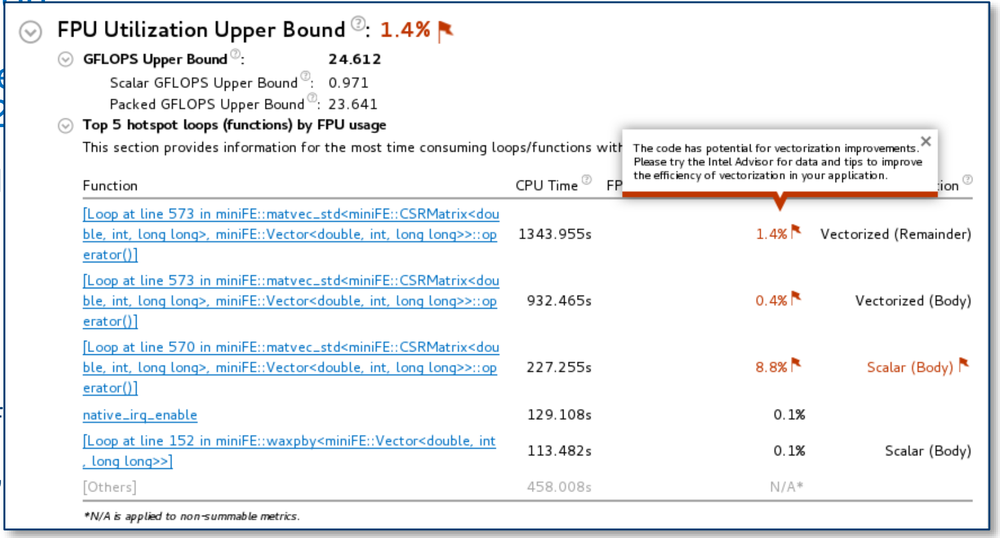
Since on KNL only SIMD scalar and vector instruction count enabled w/o vector utilizatio info we show FLOP Upper Bound metrics assuming full vector utilization. FMA is counte as single instruction so FLOPs multiplied by 2

- % of FPU load (100% - FPU is fully loaded, threshold 50%)

Metrics in FPU utilization section

- GLOPs broken down by scalar and packed
- Top 5 loops/functions by FPU usage
  - Dynamically generated issue descriptions on low F usage help to define the reason and next steps:

    Non-vectorized, vectorized with legacy instruction set, memory bound limited loops not benefiting from vectorization etc.
- Vector instruction set and FLOPs , SIMD ratio, Upper Bound per cycle are available in Grid per rank/loop/region



FPU Utilization Upper Bound: 1.4% ⚑

GFLOPS Upper Bound: 24.612

Scalar GFLOPS Upper Bound: 0.971

Packed GFLOPS Upper Bound: 23.641

Top 5 hotspot loops (functions) by FPU usage

This section provides information for the most time consuming loops/functions with

The code has potential for vectorization improvements. Please try the Intel Advisor for data and tips to improve the efficiency of vectorization in your application.

Function	CPU Time	FP	tion
[Loop at line 573 in miniFE::matvec_std<miniFE::CSRMatrix<double, int, long long>, miniFE::Vector<double, int, long long>>::operator()]	1343.955s	1.4% ⚑	Vectorized (Remainder)
[Loop at line 573 in miniFE::matvec_std<miniFE::CSRMatrix<double, int, long long>, miniFE::Vector<double, int, long long>>::operator()]	932.465s	0.4% ⚑	Vectorized (Body)
[Loop at line 570 in miniFE::matvec_std<miniFE::CSRMatrix<double, int, long long>, miniFE::Vector<double, int, long long>>::operator()]	227.255s	8.8% ⚑	Scalar (Body) ⚑
native_irq_enable	129.108s	0.1%	
[Loop at line 152 in miniFE::waxpby<miniFE::Vector<double, int, long long>>]	113.482s	0.1%	Scalar (Body)
[Others]	458.008s	N/A*	

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

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Basic Hotspots ,Concurrency, Locks and Waits

- Basic Hotspots, Concurrency, Locks and Waits use instrumentation and might bring significant overhead on runs with big number of threads
- Recommendation:
  - Use Advanced Hotspots instead of Basic Hotspots
    - Switch on stacks in Advanced Hotspots configuration if absolutely needed

```
>amplxe-cl –collect advanced-hotspots –knob collection-detail=stack-sampling ./my_app
```

  - Use HPC Performance Characterization instead of Concurrency and Locks and Waits to assess CPU utilization of throughput applications
      - Only if you have different kinds of locks in one lexical OpenMP region and want to understand each lock impact – use Locks and Waits and be ready to significant collection overhead

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

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.

Copyright © 2015, 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