HPC Performance Profiling using Intel VTune Amplifier XE

Thanh Phung, SSG/DPD/TCAR, thanh.phung@intel.com
Dmitry Prohorov, VTune HPC Lead, dmitry.prohorov@intel.com

Agenda

- Intel Parallel Studio XE – An Introduction
- VTune Amplifier XE: 2016 U4, 2017 U1 and U2
  - Analysis Configuration and Workflow
  - VTune Performance Metrics:
    - Memory Access analysis
    - Micro-arch analysis with General Exploration
    - Advanced Hotspots
    - Performance Overview with HPC Performance Characterization

Intel Parallel Studio XE: An Introduction

Intel® Parallel Studio XE (Linux, Window, MacOS)

Profiling, Analysis & Architecture

Intel® Inspector
Memory & Threading Checking

Intel® Advisor
Threading & Vectorization Architecture

Intel® VTune™ Amplifier
Performance Profiler

Cluster Tools

Intel® Trace Analyzer & Collector
MPI Profiler

Intel® MPI Library

Performance Libraries

Intel® Data Analytics Acceleration Library
Optimized for Data Analytics & Machine Learning

Intel® Integrated Performance Primitives
Image, Signal & Compression Routines

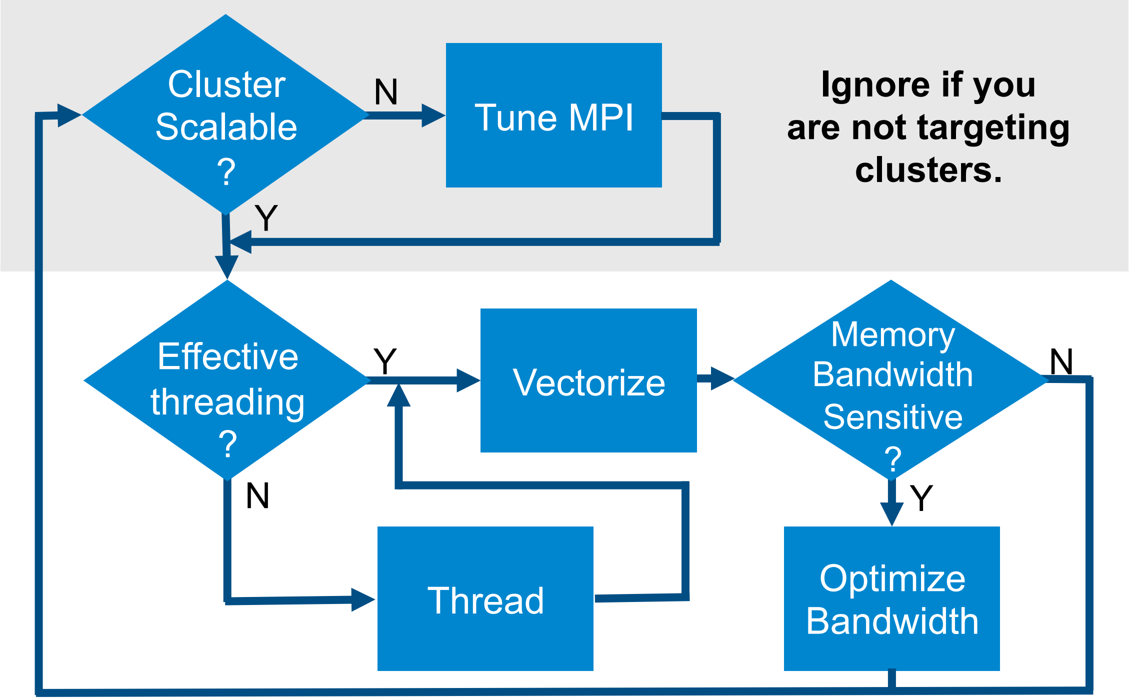
Intel® Math Kernel Library
Optimized Routines for Science, Engineering & Financial

Intel® Threading Building Blocks
Task Based Parallel C++ Template Library

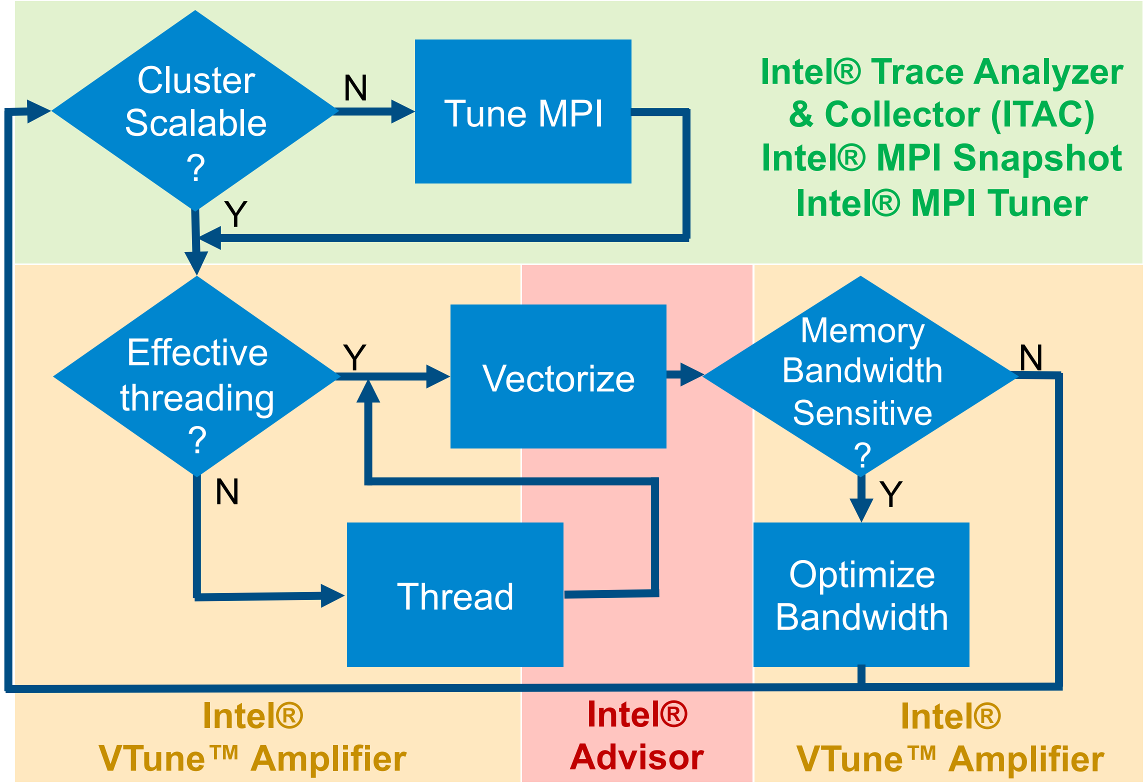
Intel® C/C++ & Fortran Compilers

Intel® Distribution for Python
Performance Scripting - Coming Soon – Q3'16

Optimizing Workload Performance - It's an iterative process…



Intel Parallel Studio XE: A complete tool suit for code and HW performance characterization



VTune: System Configuration- Prerequisites for HW EBS event based collections

- VTune on KNL works with SEP driver (recommended) + PIN 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 not enabled with perf**
  - To collect General Exploration 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*>

  - To enabled system wide collections, uncore event collections set:

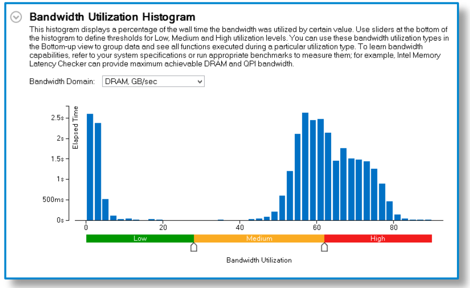
    \> echo 0 > **/proc/sys/kernel/perf_event_paranoid**

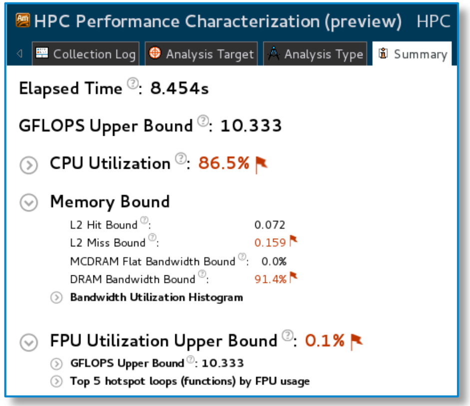
- Default sampling interval on KNL is 10ms
- EMON driver for counter mode

VTune Amplifier XE: Performance Analyzer

Overview: Explore Performance on Intel® Xeon and Xeon Phi™ Processor

- Use **VTune Amplifier XE 2017 U1** (2017 U2 will be available in WW12)
- **Memory Access - BW traffic and memory accesses**
  - Memory hierarchy and high BW usage (MCDRAM Vs DDR4)
- **General Exploration - Micro-architectural issues**
  - Explore how efficiently your code passing through the core pipeline
- **Advanced Hotspots - Algorithmic tuning opportunities**
- **HPC Performance Characterization**
  - 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





Analysis Configuration - How to Run VTune CLI on MPI Applications

<mpi_launcher> – n N <vtune_command_line> ./app_to_run

srun –n 48 -N 16 amplxe-cl –collect advanced-hotspots **–trace-mpi** –r result_dir ./my_mpi_app

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

- Add **-trace-mpi** option for VTune CLI to enable per-node result directories for non-Intel MPIs
- Works for software and Intel driver-based collectors
- Superposition of application to launch and VTune command line for selective ranks to reduce trace size

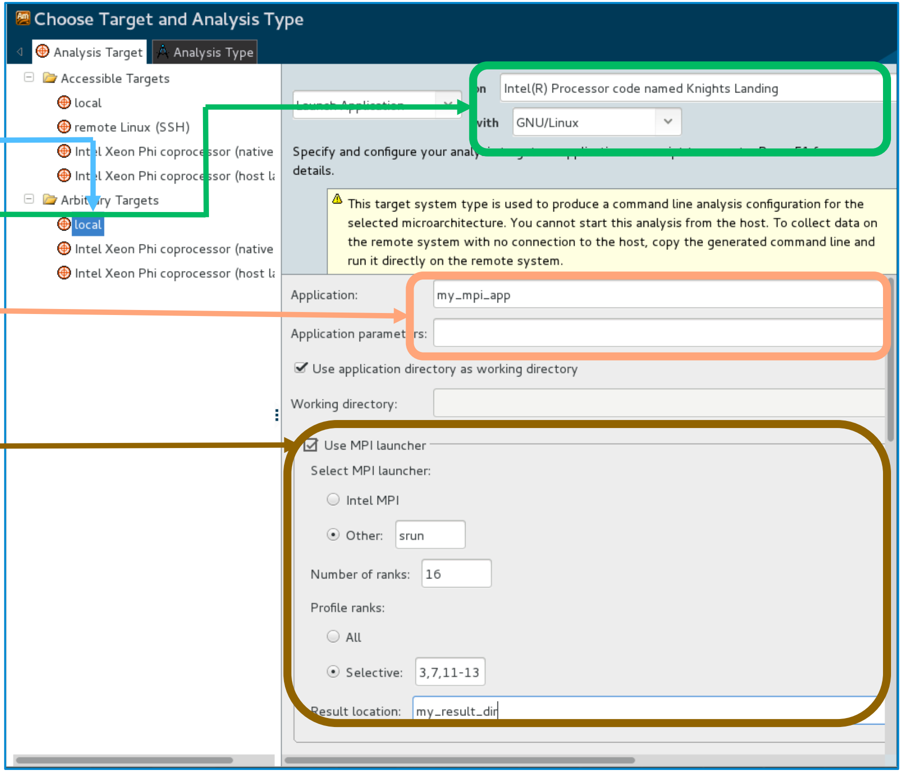
Example: profile rank 1 from 0-15:

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

Analysis Configuration - MPI Profiling Command Line Generation from GUI

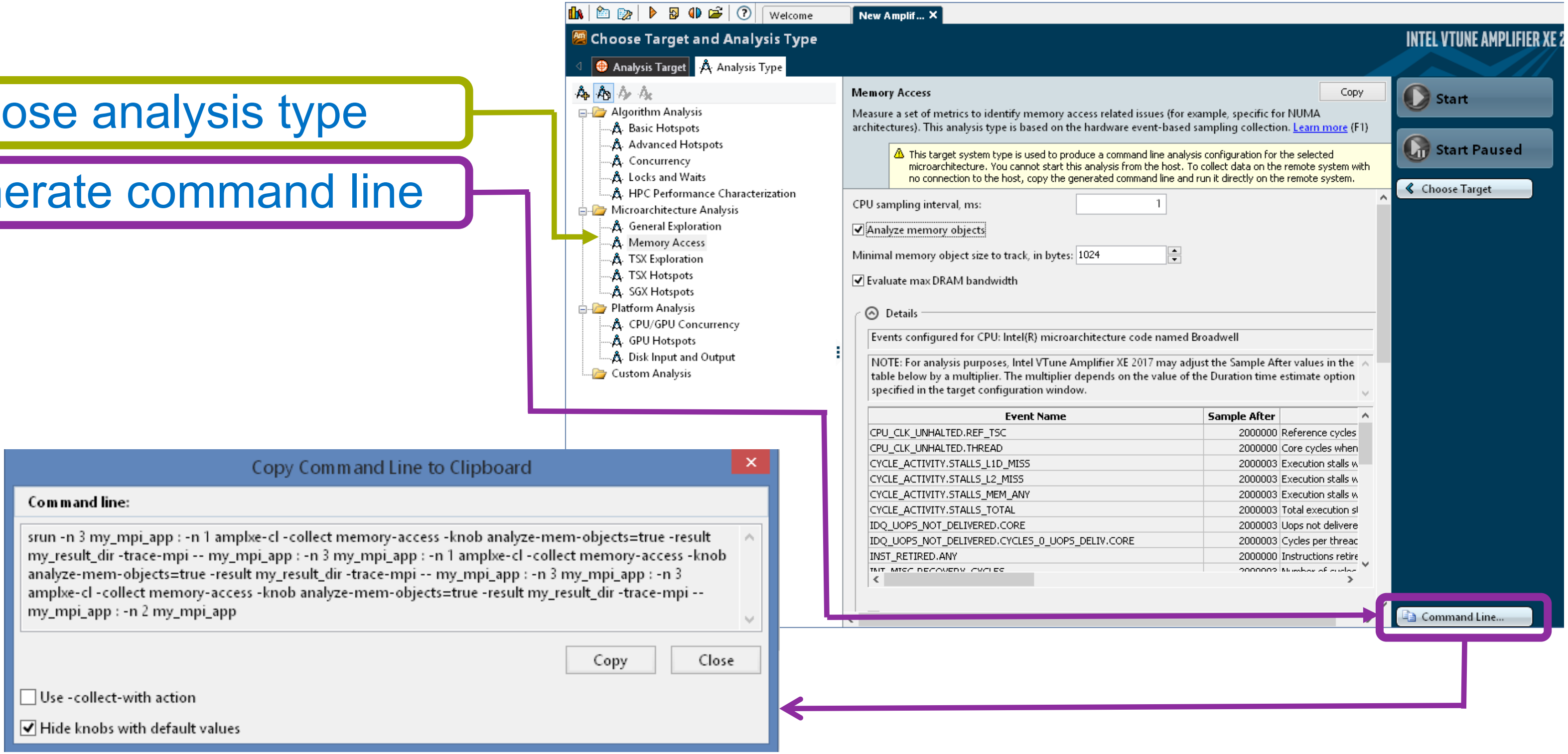
1. Create a VTune project
2. Choose "Arbitrary Targets/Local"
3. Set processor arch and OS
4. Set application name and parameters
5. Check "Use MPI Launcher"

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



Analysis Configuration - MPI Profiling Command Line Generation from GUI

6. Choose analysis type
7. Generate command line



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

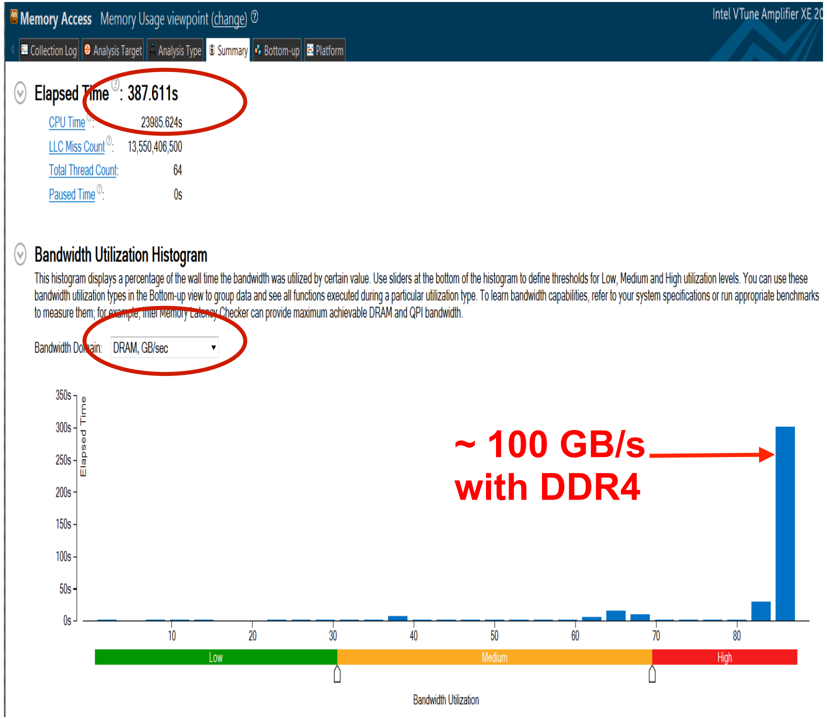
    amplxe-cl –finalize –r <my_result_dir> –search-dir <my_binary_dir>

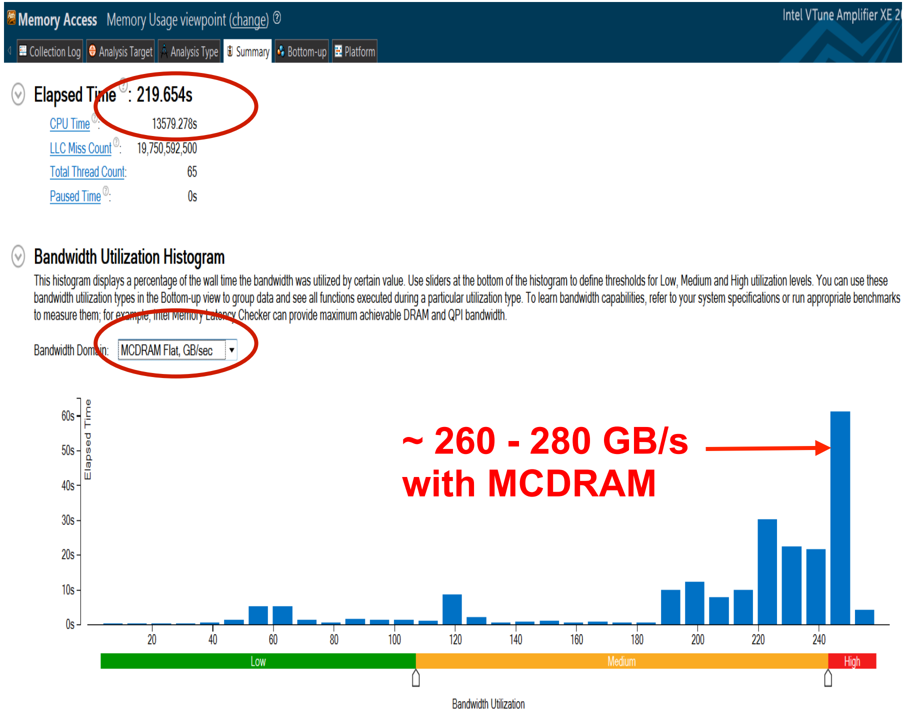
3. Generate reports, work with GUI

    amplxe-cl –report hotspots –r <my_result_dir>

VTune Amplifier XE: Performance Analyzer – Memory Access

BT Class D with 4 MPI ranks and 16 OMP threads/rank: memory bandwidth ~100 GB/s with DDR4 (left) and ~280 GB/s with MCDRAM (right)





BT Class D with 4 MPI ranks and 16 OMP threads/rank: hotspots from run on DDR4 (left) Versus on MCDRAM (right)

Memory Access Hotspots viewpoint (change)

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

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

Overhead Time:	38.890s
Instructions Retired:	9,704,520,000,000
CPI Rate:	3.642

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. E hardware-related metrics to identify what is causing high CPI.

CPU Frequency Ratio:	1.055
Total Thread Count:	64
Paused Time:	0s

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 ap processes laying on the critical path

Process	PID	MPI Communication Spinning	(%)	OpenMP Potential Gain	(%)	Serial Time	(%)
bt-mz.D.4 (rank 3)	21451	0.100s	0.0%	27.256s	7.0%	3.917s	1.0%
bt-mz.D.4 (rank 2)	21450	11.928s	3.1%	53.647s	13.8%	15.657s	4.0%
bt-mz.D.4 (rank 1)	21449	15.035s	3.9%	20.660s	5.3%	19.725s	5.1%
bt-mz.D.4 (rank 0)	21448	45.305s	11.7%	15.009s	3.9%	50.299s	13.0%

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
y_solve_ompparallel@43	bt-mz.D.4	4626.791s
x_solve_ompparallel@46	bt-mz.D.4	4594.517s
z_solve_ompparallel@43	bt-mz.D.4	4449.782s
binvcrhs	bt-mz.D.4	2374.487s
_kmp_wait_template<kmp_flag_64>	libiomp5.so	2165.303s

Memory Access Hotspots viewpoint (change)

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

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. E hardware-related metrics to identify what is causing high CPI.

CPU Frequency Ratio:	1.071
Total Thread Count:	65
Paused Time:	0s

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 ap processes laying on the critical path

Process	PID	MPI Communication Spinning	(%)	OpenMP Potential Gain	(%)	Serial Time	(%)
bt-mz.D.4 (rank 3)	15315	0.100s	0.0%	19.780s	9.0%	4.138s	1.9%
bt-mz.D.4 (rank 2)	15314	14.233s	6.5%	36.126s	16.5%	18.043s	8.2%
bt-mz.D.4 (rank 1)	15313	28.566s	13.0%	18.336s	8.4%	33.747s	15.4%
bt-mz.D.4 (rank 0)	15312	31.372s	14.3%	12.209s	5.6%	36.424s	16.6%

Top Hotspots

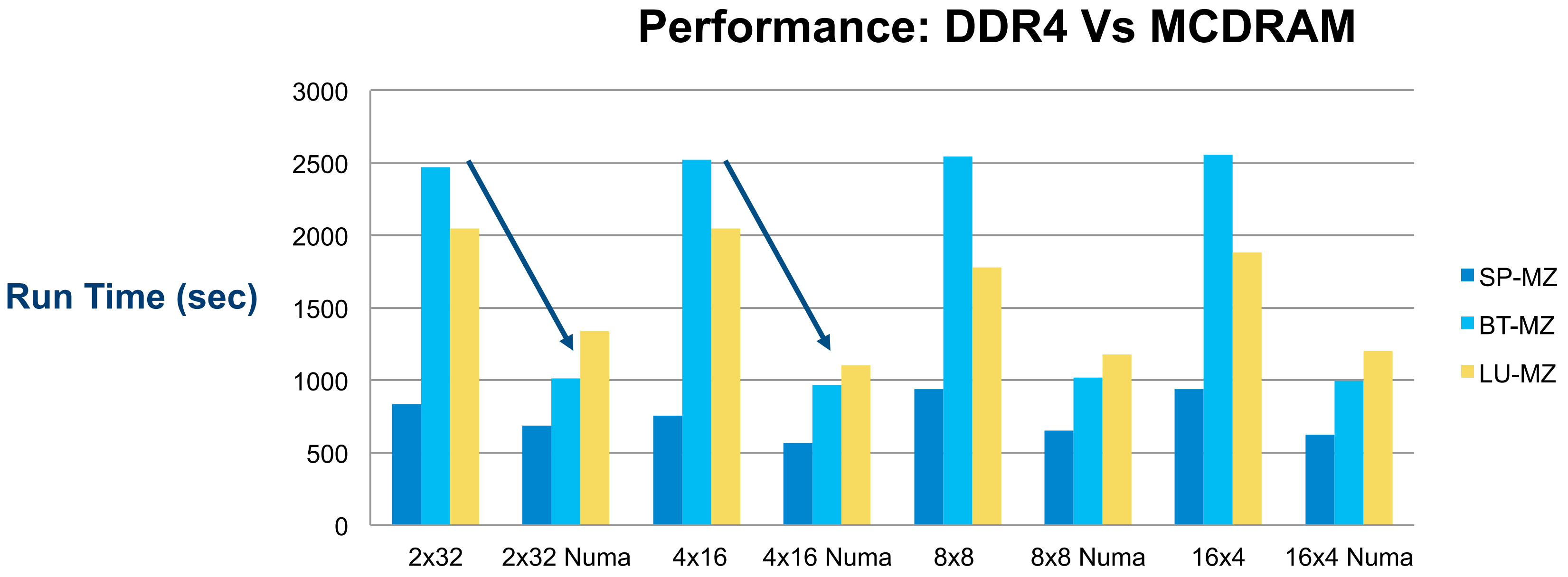
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
binvcrhs	bt-mz.D.4	2532.652s
_kmp_wait_template<kmp_flag_64>	libiomp5.so	2024.378s
matmul_sub	bt-mz.D.4	1836.544s
y_solve_ompparallel@43	bt-mz.D.4	1640.691s
x_solve_ompparallel@46	bt-mz.D.4	1620.946s
[Others]	N/A*	3924.067s

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

intel

NPB-MZ Class D run time (sec) comparison on DDR4 Vs. MCDRAM with various MPI ranks X OMP threads → MCDRAM speed up as high as 2.5 X



Allocate HBW memory with Intel compiler directive fastmem and compile with –lmemkind that can be download from http://memkind.github.io/memkind/ (for C codes: int hbw_malloc (size_t size)

Intel Fortran compiler directive

```
!DIR$ ATTRIBUTES FASTMEM :: u, rhs, forcing
!DIR$ ATTRIBUTES FASTMEM :: us, vs, ws, qs
!DIR$ ATTRIBUTES FASTMEM :: rho_i, square
!DIR$ ATTRIBUTES FASTMEM :: qbc_ou, qbc_in

       write (*,101)
 101   format ('FASTMEM DIRECTIVE TO ALLOCATE ARRAYS')

       allocate (u(proc_max_size5))
       allocate (rhs(proc_max_size5))
       allocate (forcing(proc_max_size5))

       allocate (us(proc_max_size))
       allocate (vs(proc_max_size))
       allocate (ws(proc_max_size))
       allocate (qs(proc_max_size))

       allocate (rho_i(proc_max_size))
       allocate (square(proc_max_size))

       allocate (qbc_ou(proc_max_bcsize))
       allocate (qbc_in(proc_max_bcsize))
```

Example of run script with VTune command line amplxe-cl

```
#!/bin/bash -l

AMESROOTDIR=/repo/KNLWORK/APPSDIR/NPB3.3.1-MZ/NPB3.3-MZ-MPI
echo 'AMESROOTDIR: ' $AMESROOTDIR

BinDIR=$AMESROOTDIR/bin_withMinusg
echo 'BinDIR: ' $BinDIR

export OMP_NUM_THREADS=8
NRANKS=8
###############################################################################################
###############################################################################################
###############                  RUN VTUNE TO COLLECT PERFORMANCE DATE                 ############
###############################################################################################
###############################################################################################
VTDir=$AMESROOTDIR/MyTestDIR/VTuneResultDir
echo "VTDir $VTDir"

###vtuneresultdir=$VTDir/AdvancedHotSpots
vtuneresultdir=$VTDir/BT_MemoryAccess
vtuneresultdir=$VTDir/GeneralExploration

echo "vtuneresultdir $vtuneresultdir"

vtunecollection=hpc-performance
vtunecollection=memory-access
vtunecollection=general-exploration

echo "VTune Collection Metric: $vtunecollection"

SrcDIR1=$AMESROOTDIR/LU-MZ
echo "SOURCE SEARCH DIR: $SrcDIR1"

###amplxe-cl -collect memory-access -data-limit=500 \
###      -knob sampling-interval=100 \
###         -knob analyze-mem-objects=false -knob dram-bandwidth-limits=true \
###      -r $vtuneresultdir -source-search-dir $SrcDIR1 \
###      -- numactl -m 4,5,6,7 mpiexec -n $NRANKS -ppn 1 $BinDIR/bt-mz.D.4 >& bt.mz.D.4.16.mem.nonuma.vtune.out

amplxe-cl -collect general-exploration \
     -knob collect-memory-bandwidth=false -knob dram-bandwidth-limits=false \
     -knob enable-stack-collection=false -knob enable-user-tasks=false \
     -r $vtuneresultdir -source-search-dir $SrcDIR1 \
        -- numactl -m 4,5,6,7 mpiexec -n $NRANKS -ppn 1 $BinDIR/lu-mz.D.4 >& lu.mz.D.4.16.gen.numa.vtune.out
```

numactl to allocate all memory to 4 MCDRAM memory nodes 4-7

“watch –n 1 numstat –m” shows NUMA nodes with DDR4 (0-3) and MCDRAM (4-7) showing only MCDRAM memory being allocated for LU Class D benchmark on KNL

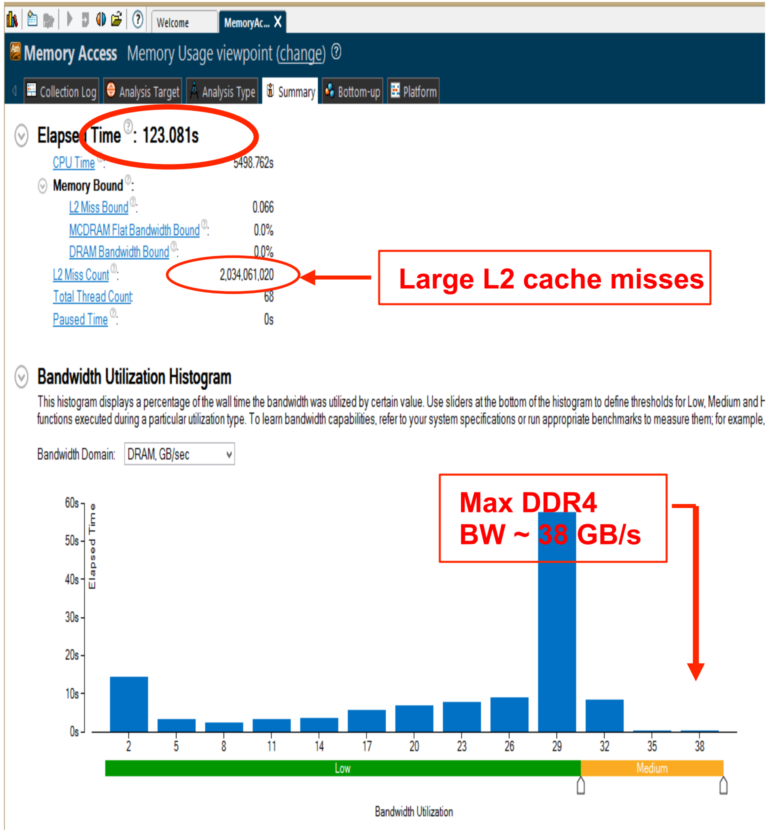
Every 1.0s: numastat -m   Thu Feb 25 09:12:05 20

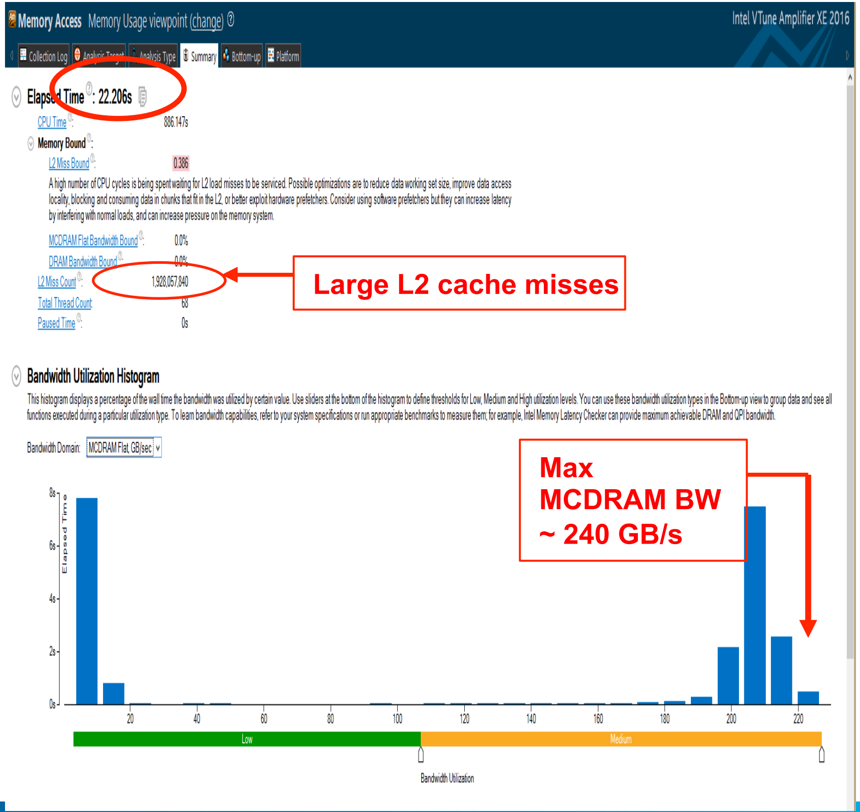
DDR4   MCDRAM

Per-node system memory usage (in MBs):

	Node 0	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Total
MemTotal	24231.73	24232.57	24232.57	23922.40	4039.33	4039.33	4039.33	4036.62	112773.88
MemFree	21727.73	23667.77	23692.06	21601.48	871.25	1293.75	1295.81	1744.19	95894.05
MemUsed	2503.99	564.79	540.51	2320.91	3168.08	2745.58	2743.52	2292.43	16879.82
Active	1775.60	226.73	235.02	1794.54	2818.23	2341.84	2442.16	1900.74	13534.87
Inactive	116.96	89.65	61.32	147.84	133.79	136.31	133.58	132.17	951.61
Active(anon)	11.93	19.41	207.36	30.38	2716.22	2237.90	2347.10	1805.76	9376.07
Inactive(anon)	0.43	0.06	1.49	8.05	1.02	1.03	0.00	0.00	12.08
Active(file)	1763.67	207.32	27.66	1764.16	102.01	103.94	95.06	94.98	4158.80
Inactive(file)	116.53	89.59	59.83	139.79	132.76	135.28	133.58	132.17	939.53
Unevictable	0.00	0.02	0.02	0.05	0.00	0.00	0.00	0.00	0.08
Mlocked	0.00	0.02	0.02	0.05	0.00	0.00	0.00	0.00	0.08
Dirty	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Writeback	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FilePages	1881.07	296.98	89.30	1912.12	235.82	240.36	228.68	227.17	5111.50
Mapped	10.00	2.86	0.05	28.04	3.14	3.10	0.00	0.00	47.20
AnonPages	12.31	19.50	207.26	30.74	2864.48	2435.95	2449.36	1999.78	10019.38
Shmem	0.64	0.12	1.63	8.20	1.02	1.03	0.00	0.00	12.64
KernelStack	4.12	3.92	4.87	6.95	0.00	0.00	0.00	0.00	19.85
PageTables	1.04	0.54	1.79	1.43	6.09	5.21	4.97	4.10	25.19
NFS_Unstable	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Bounce	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
WritebackTmp	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Slab	127.64	52.75	50.51	132.03	16.93	17.50	15.64	15.99	429.00
SReclaimable	83.20	13.30	8.50	67.41	8.36	8.52	8.21	8.04	205.55
SUnreclaim	44.45	39.45	42.00	64.62	8.57	8.98	7.42	7.95	223.45
AnonHugePages	0.00	0.00	128.00	6.00	2848.00	2420.00	2444.00	1994.00	9840.00
HugePages_Total	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
HugePages_Free	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

BW usage on 64 threads (cores) (Animation code) - max 38 GB/s with DDR4 (left) and 240 GBs with MCDRAM (right)





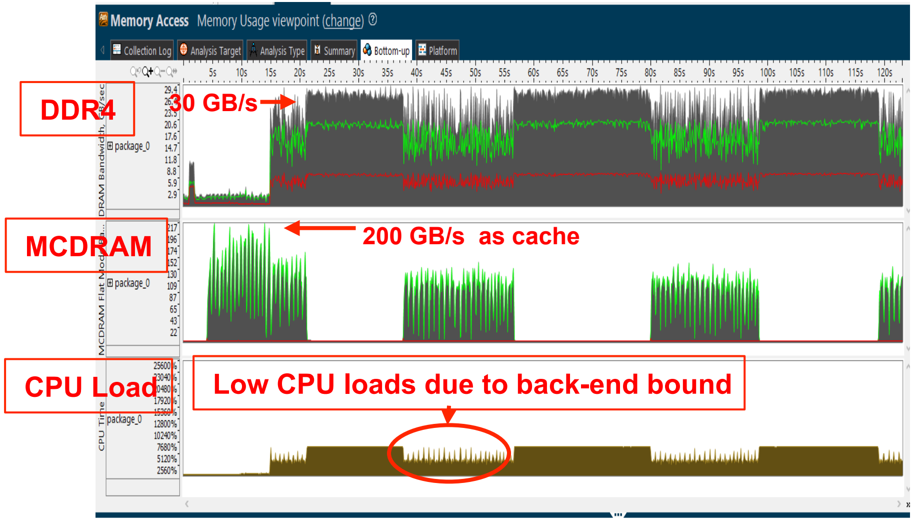
Top memory objects and large L2 cache misses with MCDRAM as HBM

Top Memory Objects

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

Memory Object	L2 Miss Count
Add_Stress_Force_Differential_tester.cpp:244 (732 MB)	921,027,630
Add_Stress_Force_Differential_tester.cpp:245 (732 MB)	534,016,020
SIMD_POLICY.h:204 (30 MB)	106,003,180
SIMD_POLICY.h:204 (30 MB)	89,002,670
SIMD_POLICY.h:204 (30 MB)	82,002,460
[Others]	196,005,880

Performance of animation code with DDR4 BW (left) compared to MCDRAM BW (right)



Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time	Memory Bound / L2 Miss Bound	L2 Miss Count	Start Address
Number<_m256>::Load_Aligned	277.030s	0.525	809,024,270	0x402c8e
Number<_m256>::Load_Aligned	893.825s	0.125	604,018,120	0x4033a1
Number<_m256>::Load_Aligned	103.339s	0.132	77,002,310	0x402f8f
Number<_m256>::operator*	0.140s	1.000	69,002,070	0x4031ce
Number<_m256>::Load_Aligned	0.692s	1.000	57,001,710	0x402fbe
Number<_m256>::operator+	2.847s	1.000	31,000,930	0x40321d
Number<_m256>::operator+	0.862s	1.000	30,000,900	0x40320f
rcu_check_callbacks	0s		30,000,900	0xffffffff811263a0
trigger_load_balance	0s		16,000,480	0xffffffff810c6230

Running on DDR4: numctl –m 0

Grouping: Function / Memory Object / Allocation Stack

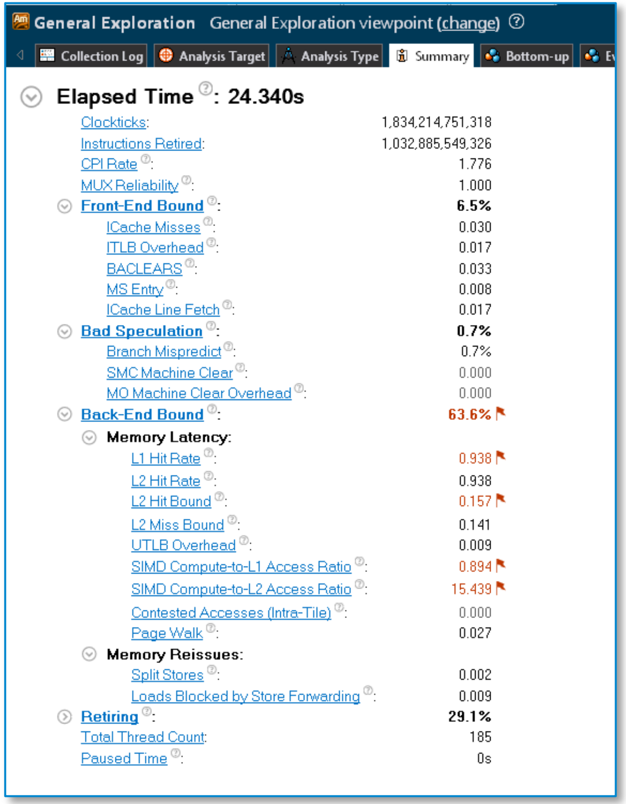
Function / Memory Object / Allocation Stack	CPU Time	Memory Bound / L2 Miss Bound	L2 Miss Count	Start Address
Number<_m256>::Load_Aligned	38.699s	1.000	921,027,630	0x402c8e
Add_Stress_Force_Differential_tester.cpp:244 (732 MB)			921,027,630	0x402c8e
[Unknown]			0	0x402c8e
Number<_m256>::Load_Aligned	72.097s	1.000	519,015,570	0x4033a1
Add_Stress_Force_Differential_tester.cpp:245 (732 MB)			519,015,570	0x4033a1
[Unknown]			0	0x4033a1
Number<_m256>::Load_Aligned	14.143s	1.000	108,003,240	0x402f8f
Number<_m256>::operator*	0.070s	1.000	91,002,730	0x4031ce
Number<_m256>::Load_Aligned	0.571s	1.000	82,002,460	0x402fbe
Selected 1 row(s):	38.699s	1.000	921,027,630	

Running on MCDRAM: numctl –m 1

VTune Amplifier XE: Performance Analyzer – General Exploration

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



Performance summary and top OMP regions for BT-MZ

General Exploration Hotspots viewpoint (change)

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

Elapsed Time: 43.525s

- CPU Time: 2404.587s
  - Effective Time: 1950.254s
  - Spin Time: 445.908s

    A significant portion of CPU time is spent waiting. Use this metric to discover which synchronizations are spinning. Consider adjusting spin wait p lock implementation (for example, by backing off then descheduling), or adjusting the synchronization granularity.
  - Overhead Time: 8.425s
- Instructions Retired: 2,288,493,432,735
- CPI Rate: 1.572

  The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instru hardware-related metrics to identify what is causing high CPI.
- CPU Frequency Ratio: 1.071
- Total Thread Count: 69
- Paused Time: 0s

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 c processes laying on the critical path

Process	PID	MPI Communication Spinning	(%)	OpenMP Potential Gain	(%)	Serial Time	(%)
bt-mz.D.4 (rank 3)	16101	0.029s	0.1%	4.681s	10.8%	0.913s	2.1%
bt-mz.D.4 (rank 2)	16100	3.474s	8.0%	7.089s	16.3%	4.485s	10.3%
bt-mz.D.4 (rank 1)	16099	5.875s	13.5%	3.541s	8.2%	7.651s	17.6%
bt-mz.D.4 (rank 0)	16098	6.680s	15.4%	2.705s	6.2%	7.815s	18.0%

Hot functions and OMP hot spots with most run time and CPU usage profile

General Exploration Hotspots viewpoint (change) Intel VTune Amplifier

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

Top Hotspots

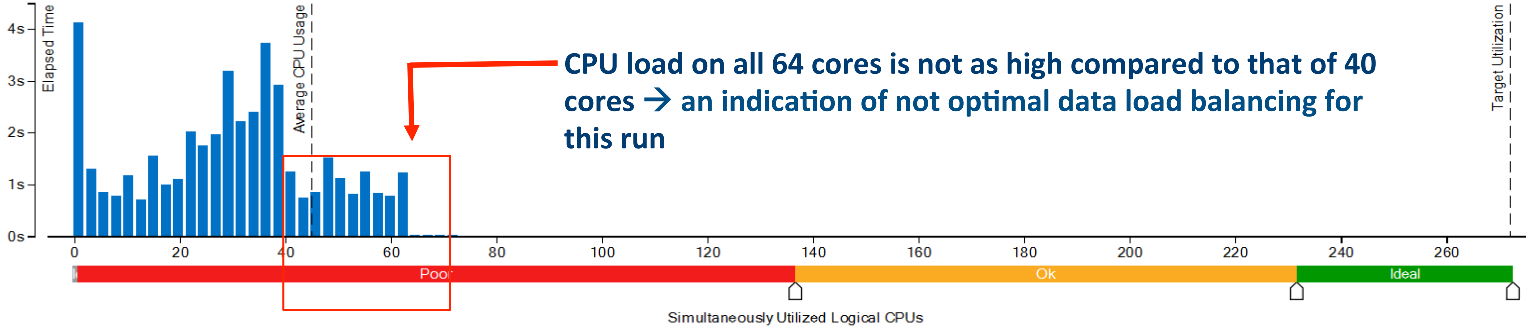
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
binvcrhs	bt-mz.D.4	430.059s
__kmp_wait_template<kmp_flag_64>	libiomp5.so	372.729s
matmul_sub	bt-mz.D.4	309.055s
z_solve_ompparallel@43	bt-mz.D.4	276.360s
y_solve_ompparallel@43	bt-mz.D.4	275.411s
[Others]	N/A*	740.973s

**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.



CPU load on all 64 cores is not as high compared to that of 40 cores → an indication of not optimal data load balancing for this run

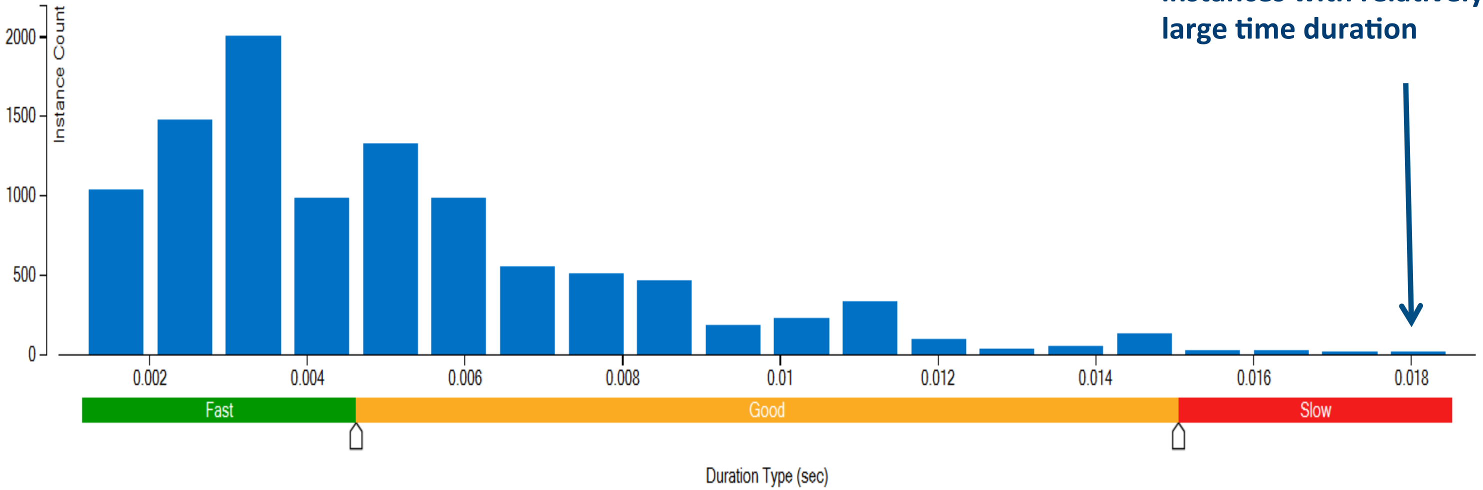
CPU performance statistics of different OMP regions for BT-MZ

OpenMP Region Duration Histogram

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

OpenMP Region: z_solve_ompparallel:16@unknown:43:454

Very small number of instances with relatively large time duration



Summary of all HW events collected using general-exploration for BT-MZ on KNL: AVX-512 instructions are included in UOPS_RETIRED_PACK.SIMD and UOPS_RETIRED_SCALAR_SIMD or ~ 60 %+ off all UOPS_RETIRED_ALL

Welcome | GeneralExpl ...

General Exploration Hardware Events viewpoint (change)

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

Hardware Events

Hardware Event Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample
CPU_CLK_UNHALTED.REF_TSC	3,358,643,037,957	1,679,319	2000003
CPU_CLK_UNHALTED.THREAD	3,598,587,397,873	1,799,291	2000003
CPU_CLK_UNHALTED.THREAD_P	3,591,485,387,220	179,574	2000003
CYCLES_DIV_BUSY.ALL	0	0	2000003
FETCH_STALL.ICACHE_FILL_PENDING_CYCLES	95,497,432,440	47,748	200003
INST_RETIRED.ANY	2,288,493,432,735	1,144,245	2000003
MACHINE_CLEARS.FP_ASSIST	0	0	200003
MEM_UOPS_RETIRED.ALL_LOADS	968,036,520,330	484,011	200003
MEM_UOPS_RETIRED.ALL_STORES	465,076,976,050	232,535	200003
MEM_UOPS_RETIRED.HITM_PS	0	0	200003
MEM_UOPS_RETIRED.L2_HIT_LOADS_PS	18,024,270,360	9,012	200003
MEM_UOPS_RETIRED.L2_MISS_LOADS_PS	3,390,237,300	3,390	100007
NO_ALLOC_CYCLES.MISPREDICTS	31,452,471,780	15,726	200003
NO_ALLOC_CYCLES.NOT_DELIVERED	531,339,969,980	265,666	200003
PAGE_WALKS.CYCLES	9,476,142,140	4,738	200003
PAGE_WALKS.I_SIDE_CYCLES	5,764,086,460	2,882	200003
REHABQ.LD_BLOCK_ST_FORWARD_PS	23,240,348,600	11,620	200003
REHABQ.LD_SPLITS_PS	78,655,179,810	39,327	200003
REHABQ.ST_SPLITS	14,064,210,960	7,032	200003
UOPS_RETIRED.ALL	2,368,423,552,630	118,421	2000003
UOPS_RETIRED.MS	97,660,146,490	4,883	2000003
UOPS_RETIRED.PACKED_SIMD	251,385,770,730	125,691	200003
UOPS_RETIRED.SCALAR_SIMD	1,217,346,259,920	608,664	200003

Memory Uops/ Insts retired

SIMD/all UOPS

VTune Amplifier XE: Performance Analyzer – Advanced Hotspots

Advanced-hotspot performance analysis- summary view

General Exploration Hotspots viewpoint (change) Intel VTune Amplifier XE

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

Elapsed Time: 43.525s

CPU Time:	2404.587s
Effective Time:	1950.254s
Spin Time:	445.908s

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

Overhead Time:	8.425s
Instructions Retired:	2,288,493,432,735
CPI Rate:	1.572

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI.

CPU Frequency Ratio:	1.071
Total Thread Count:	69
Paused Time:	0s

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

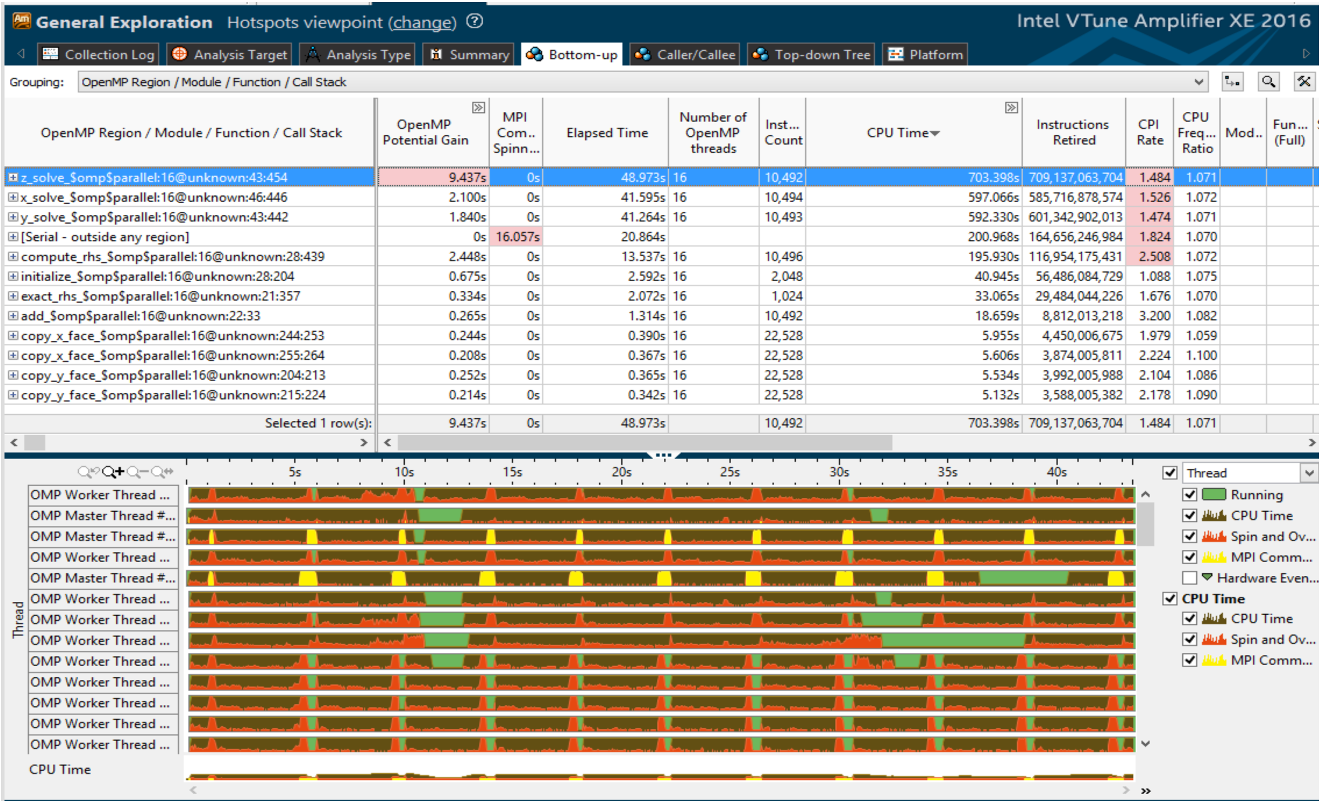
Process	PID	MPI Communication Spinning	(%)	OpenMP Potential Gain	(%)	Serial Time	(%)
bt-mz.D.4 (rank 3)	16101	0.029s	0.1%	4.681s	10.8%	0.913s	2.1%
bt-mz.D.4 (rank 2)	16100	3.474s	8.0%	7.089s	16.3%	4.485s	10.3%
bt-mz.D.4 (rank 1)	16099	5.875s	13.5%	3.541s	8.2%	7.651s	17.6%
bt-mz.D.4 (rank 0)	16098	6.680s	15.4%	2.705s	6.2%	7.815s	18.0%

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
binvcrhs	bt-mz.D.4	430.059s
__kmp_wait_template<kmp_flag_64>	libiomp5.so	372.729s
matmul_sub	bt-mz.D.4	309.055s
z_solve_ompparallel@43	bt-mz.D.4	276.360s
y_solve_ompparallel@43	bt-mz.D.4	275.411s

Advanced-hotspot performance analysis – bottom up view



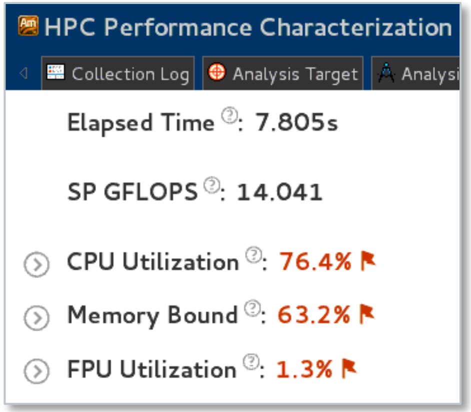
OpenMP Region / Module / Function / Call Stack	OpenMP Potential Gain	MPI Com.. Spinn...	Elapsed Time	Number of OpenMP threads	Inst... Count	CPU Time	Instructions Retired	CPI Rate	CPU Freq... Ratio	Mod..	Fun... (Full)
z_solve_ompparallel:16@unknown:43:454	9.437s	0s	48.973s	16	10,492	703.398s	709,137,063,704	1.484	1.071		
x_solve_ompparallel:16@unknown:46:446	2.100s	0s	41.595s	16	10,494	597.066s	585,716,878,574	1.526	1.072		
y_solve_ompparallel:16@unknown:43:442	1.840s	0s	41.264s	16	10,493	592.330s	601,342,902,013	1.474	1.071		
[Serial - outside any region]	0s	16.057s	20.864s			200.968s	164,656,246,984	1.824	1.070		
compute_rhs_ompparallel:16@unknown:28:439	2.448s	0s	13.537s	16	10,496	195.930s	116,954,175,431	2.508	1.072		
initialize_ompparallel:16@unknown:28:204	0.675s	0s	2.592s	16	2,048	40.945s	56,486,084,729	1.088	1.075		
exact_rhs_ompparallel:16@unknown:21:357	0.334s	0s	2.072s	16	1,024	33.065s	29,484,044,226	1.676	1.070		
add_ompparallel:16@unknown:22:33	0.265s	0s	1.314s	16	10,492	18.659s	8,812,013,218	3.200	1.082		
copy_x_face_ompparallel:16@unknown:244:253	0.244s	0s	0.390s	16	22,528	5.955s	4,450,006,675	1.979	1.059		
copy_x_face_ompparallel:16@unknown:255:264	0.208s	0s	0.367s	16	22,528	5.606s	3,874,005,811	2.224	1.100		
copy_y_face_ompparallel:16@unknown:204:213	0.252s	0s	0.365s	16	22,528	5.534s	3,992,005,988	2.104	1.086		
copy_y_face_ompparallel:16@unknown:215:224	0.214s	0s	0.342s	16	22,528	5.132s	3,588,005,382	2.178	1.090		
Selected 1 row(s):	9.437s	0s	48.973s		10,492	703.398s	709,137,063,704	1.484	1.071		

VTune Amplifier XE: Performance Analyzer – HPC Performance

HPC Performance Characterization Analysis

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 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'm moving from pure MPI to MPI + OpenMP but the results are worse, why?

CPU utilization, memory efficiency and FPU utilization aspects are important for performance study and correlated – let's explore them in one view

> *amplxe-cl* ***–collect hpc-performance*** *–data-limit=0 –r result_dir ./my_app*

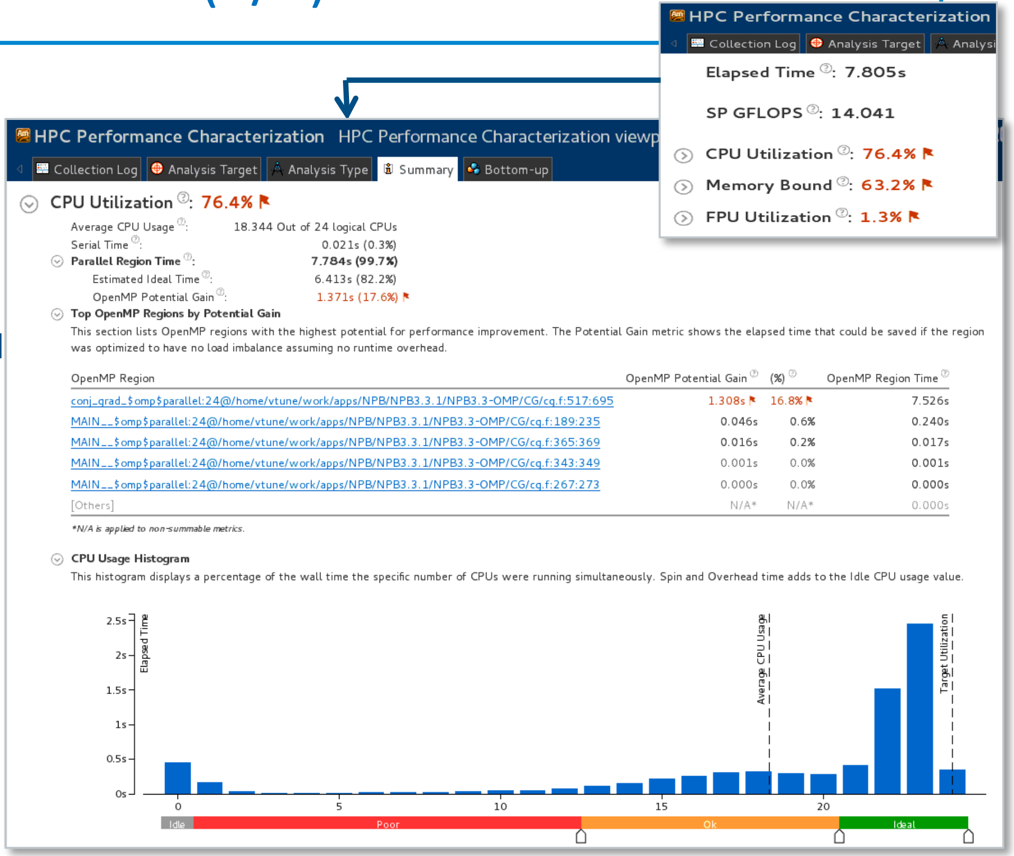
Performance Aspects: CPU Utilization (1/2)

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

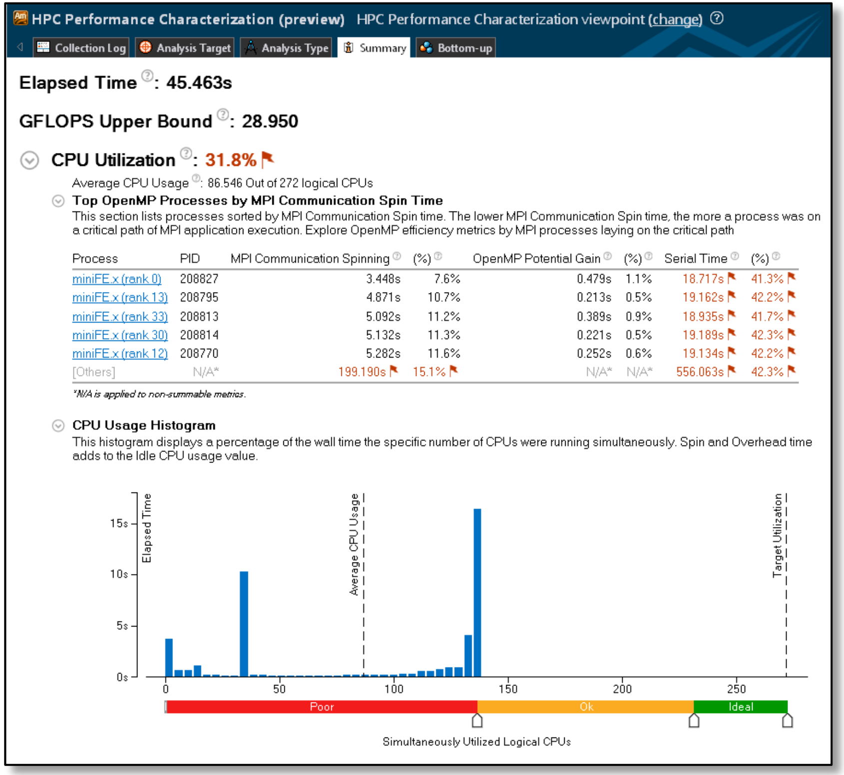
Metrics in CPU utilization section

- Average CPU usage
- Intel OpenMP scalability metrics impacting effective CPU utilization
- CPU utilization histogram



Performance Aspects: CPU Utilization (2/2) - Specifics for hybrid MPI + OpenMP apps

- **MPI communication spinning metric for MPICH-based MPIs (Intel MPI, CRAY MPI, .._)**
  - Difference in MPI communication spinning between ranks can signal MPI imbalance
- Showing OpenMP metrics and serial time per process sorting by processes laying on critical path of MPI execution



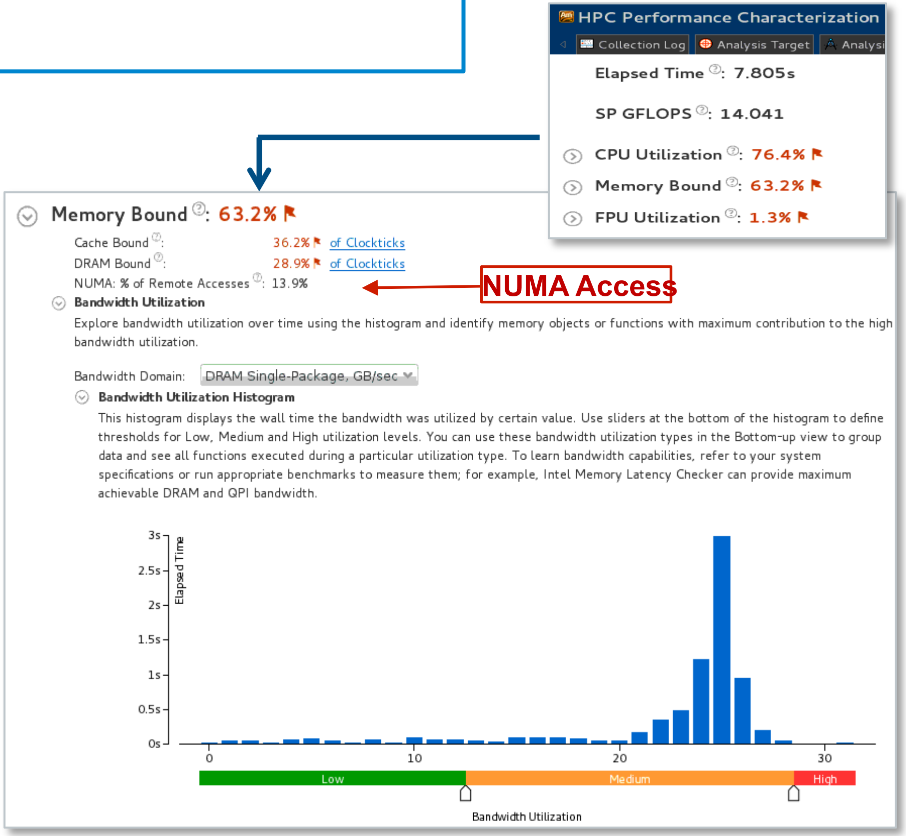
Performance Aspects: Memory Bound

Memory Bound

- **% of potential execution pipeline slots lost because of fetching memory from different levels of hierarchy (threshold 20%)**

Metrics in Memory Bound section

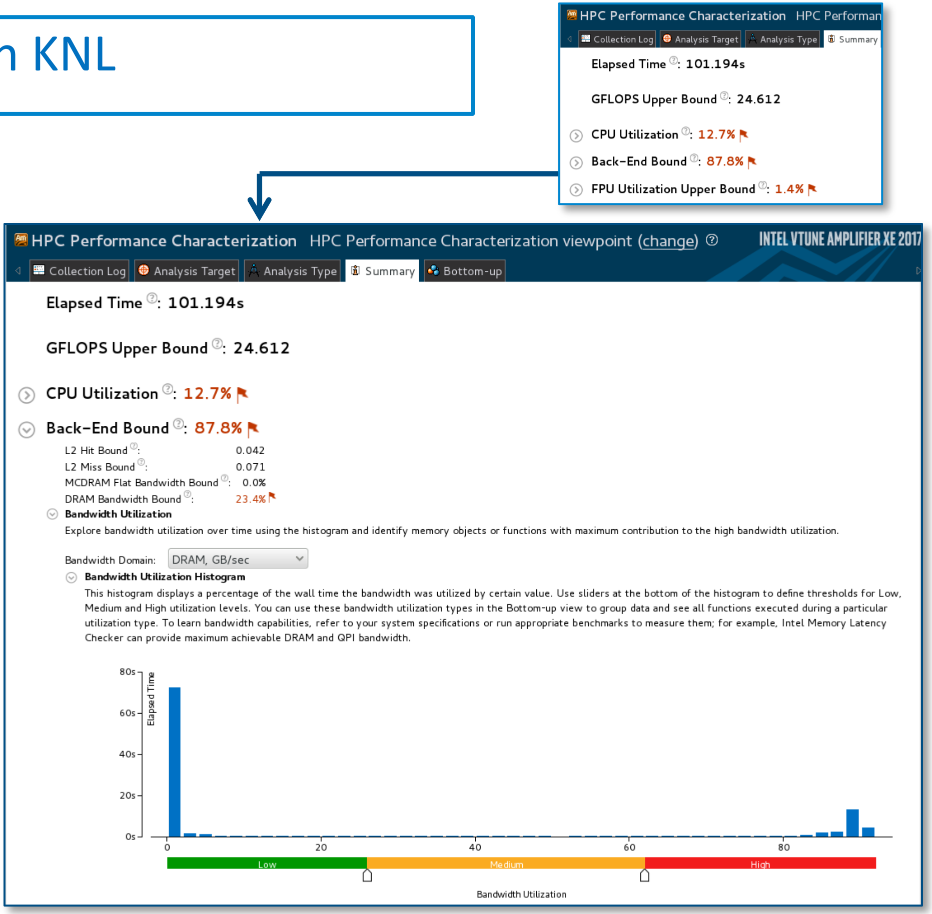
- Cache bound
- DRAM bound
    - Issue description specifies if the code is bandwidth or latency bound with proper advice of how to fix
  - NUMA: % of remote accesses
    - Important to explore if the code is bandwidth bound
  - Bandwidth utilization histogram



Performance Aspects: Memory Bound on KNL

Since no memory stall measurement on KNL "Memory Bound" high level metric replaced with Backend-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
- DRAM Bandwidth Bound
  - % of app elapsed time consuming high DRAM Bandwidth
- MCDRAM Bandwidth Bound
  - % of app elapsed time consuming high MCDRAM Bandwidth
- Bandwidth utilization histogram



Performance aspects: FPU Utilization

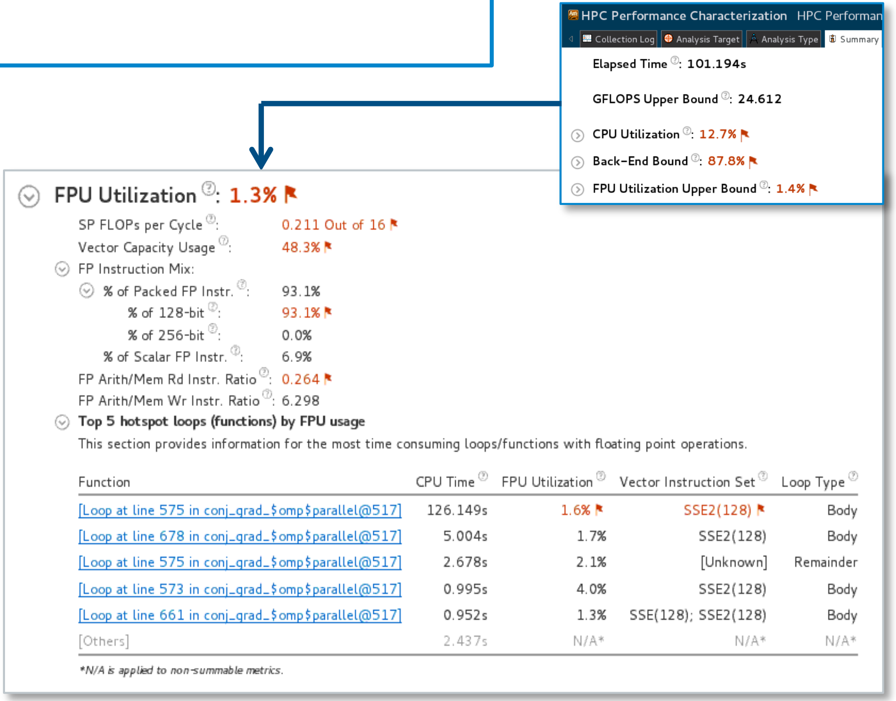
FPU utilization

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

Metrics in FPU utilization section

- SP FLOPs per Cycle (vector code generation and execution efficiency)
- Vector Capacity Usage and FP Instruction Mix, FPArith/Mem ratios (vector code generation efficiency)
- 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.



These renewed FPU Utilization metrics will be available in 2017 Update 2

Performance aspects: FPU utilization on KNL

No FLOP counters on KNL to calculate FLOPS and FPU Utilization

Showing SIMD Instructions per cycle and SIMD Packed vs SIMD Scalar based on available HW counters + Vector instruction set per loop based on static analysis

SIMD Instructions per Cycle: 0.038

FP Instruction Mix:

% of Packed SIMD Instr.: 100.0%

% of Scalar SIMD Instr.: 0.0%

Top 5 hotspot loops (functions) by FPU usage

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

A significant fraction of SIMD instructions executed with partial vector load. A possible reason is compilation with legacy instruction set. Check the compiler options. Another possible reason is compiler code generation specifics. Use Intel Advisor to learn more.

Function	CPU Time	SIMD Instructions per Cycle	Vector Instruction Set	Loop Type
[Loop at line 211 in multiply3ompparallel_for@207]	446.632s	0.049	AVX(256); FMA(256)	Body
native_irq_enable	36.123s	0.001	[Unknown]	[Unknown]
[Loop at line 208 in multiply3ompparallel_for@207]	2.165s	0.025	AVX(256); FMA(256)	Body
intel_pstate_timer_func	0.170s	0.057	[Unknown]	[Unknown]
init_arr	0.030s	1.257	AVX(128); AVX(256); AVX2(256); FMA(256)	[Unknow

These renewed FPU Utilization metrics will be available in 2017 Update 2

HPC Performance Characterization – Command Line Reporting

- Generated after collection is done or with "-R summary" option of amplxe-cl
- With issue descriptions that can be suppressed by "–report-knob show-issues=false" option

```
Elapsed Time: 7.805s
SP GFLOPS: 14.041
CPU Utilization: 76.4%
 | The metric value is low, which may signal a poor logical CPU cores
 | utilization caused by load imbalance, threading runtime overhead, contended
 | synchronization, or thread/process underutilization. Explore CPU Utilization
 | sub-metrics to estimate the efficiency of MPI and OpenMP parallelism or run
 | the Locks and Waits analysis to identify parallel bottlenecks for other
 | parallel runtimes.
 |
    Average CPU Usage: 18.344 Out of 24 logical CPUs
    Serial Time: 0.021s (0.3%)
    Parallel Region Time: 7.784s (99.7%)
        Estimated Ideal Time: 6.413s (82.2%)
        OpenMP Potential Gain: 1.371s (17.6%)
         | The time wasted on load imbalance or parallel work arrangement is
         | significant and negatively impacts the application performance and
         | scalability. Explore OpenMP regions with the highest metric values.
         | Make sure the workload of the regions is enough and the loop schedule
         | is optimal.
         |
Memory Bound: 63.2% of Pipeline Slots
 | The metric value is high. This can indicate that the significant fraction of
 | execution pipeline slots could be stalled due to demand memory load and
 | stores. Use Memory Access analysis to have the metric breakdown by memory
 | hierarchy, memory bandwidth information, correlation by memory objects.
 |
    Cache Bound: 36.2% of Clockticks
     | A significant proportion of cycles are being spent on data fetches from
     | caches. Check Memory Access analysis to see if accesses to L2 or L3
     | caches are problematic and consider applying the same performance tuning
     | as you would for a cache-missing workload. This may include reducing the
     | data working set size, improving data access locality, blocking or
     | partitioning the working set to fit in the lower cache levels, or
     | exploiting hardware prefetchers. Consider using software prefetchers, but
     | note that they can interfere with normal loads, increase latency, and
     | increase pressure on the memory system. This metric includes coherence
     | penalties for shared data. Check General Exploration analysis to see if
     | contested accesses or data sharing are indicated as likely issues.
     |
    DRAM Bound: 28.9% of Clockticks
     | The metric value is high. This indicates that a significant fraction of
     | cycles could be stalled on the main memory (DRAM) because of demand loads
     | or stores.
     |
     | The code is memory bandwidth bound, which means that there are a
     | significant fraction of cycles during which the bandwidth limits of the
     | main memory are being reached and the code could stall. Review the
     | Bandwidth Utilization Histogram to estimate the scale of the issue.
     | Consider improving data locality on NUMA multi-socket systems, which will
     | reduce code memory bandwidth consumption.
     |
    NUMA: % of Remote Accesses: 13.9%
```

```
Elapsed Time: 7.805s
SP GFLOPS: 14.041
CPU Utilization: 76.4%
    Average CPU Usage: 18.344 Out of 24 logical CPUs
    Serial Time: 0.021s (0.3%)
    Parallel Region Time: 7.784s (99.7%)
        Estimated Ideal Time: 6.413s (82.2%)
        OpenMP Potential Gain: 1.371s (17.6%)
Memory Bound: 63.2% of Pipeline Slots
    Cache Bound: 36.2% of Clockticks
    DRAM Bound: 28.9% of Clockticks
    NUMA: % of Remote Accesses: 13.9%
FPU Utilization: 1.3%
    SP FLOPs per Cycle: 0.211 Out of 16
    Vector Capacity Usage: 48.3%
    FP Instruction Mix
        % of Packed FP Instr.: 93.1%
            % of 128-bit: 93.1%
            % of 256-bit: 0.0%
        % of Scalar FP Instr.: 6.9%
    FP Arith/Mem Rd Instr. Ratio: 0.264
    FP Arith/Mem Wr Instr. Ratio: 6.298
Collection and Platform Info
    Application Command Line: ./cg.B.x
    User Name: vtune
```

Legal Disclaimer & Optimization Notice

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

Back-up Slides

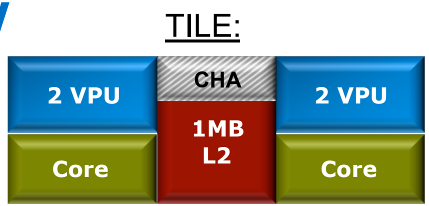
Code tuning requirements: know your code, know the compiler and know the platform microarchitecture

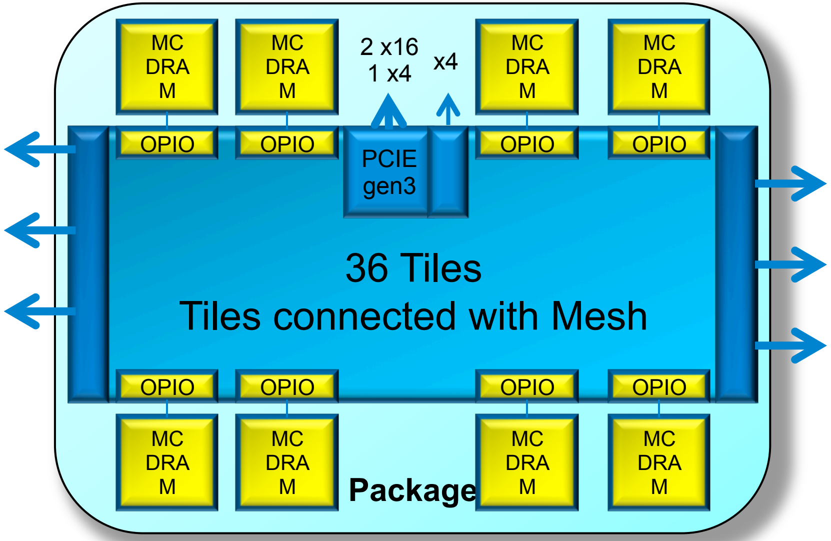
- **Core tuning:**
  - **Cache or vector friendly or both:**
    - **AVX-2 and AVX-512**
    - **Use best compiler options and *check compiler report***

mpiifort **–g** –O3 –xMIC-AVX512 –align array64byte … –qopt-report=5 –qopt-report-phase=loop, vec, openmp…

  - **Compilers directives and pragmas: SIMD, Alignment, …**
  - **OpenMP 4.0 with OMP SIMD directives/pragmas**
  - **NUMA: MCDRAM Vs DDR – Allocate memory for active arrays or use NUMA command to use MCDRAM for better performance**

Knights Landing (KML) Overview





36 Tiles w/ 72 new Silvermont-based cores

4 Threads per core

<u>2 Vector Processing Units per core</u>

6 channels of DDR4 2400 up to 384GB

8 to16 GB of on-package MCDRAM memory

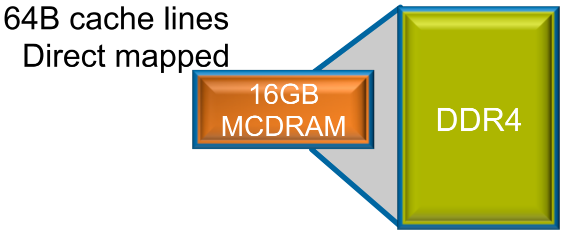
36 lanes PCIE Gen 3. 4 lanes of DMI

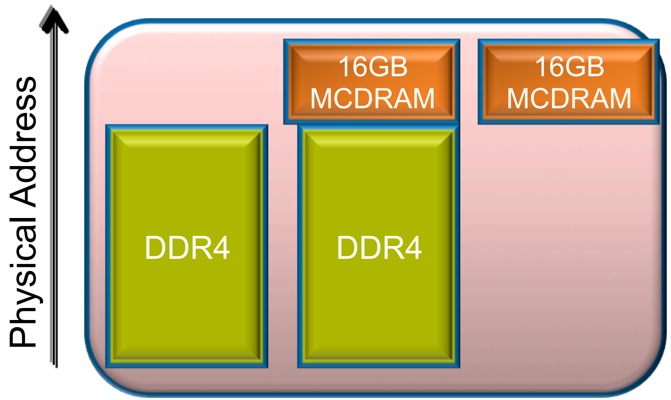
45

3 Memory Modes

- Mode selected at boot time
- MCDRAM-Cache covers all HBM

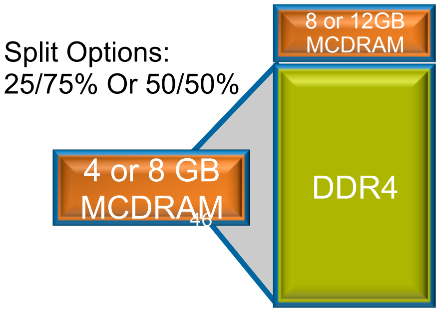
Cache Model





Flat Models

Hybrid Model



Code tuning requirements: Parallel Scalability with MPI, OMP, Hybrid MPI+OMP

- **Scalability:**
  - **OMP**
    - **Load balance over all threads**
    - **Private Vs shared data**
    - **Synchronization**
    - **Lock, wait and spinning Vs doing work**
    - **SIMD directives**
  - **MPI**
    - **Timing cost due to communication Vs computing**
    - **Block Vs non-blocking message types**
    - **Global synchronizations**
    - **All-to-all communication**

intel®