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)

**Performance Libraries**
- Intel® Data Analytics Acceleration Library
  Optimized for Data Analytics & Machine Learning
- Intel® Math Kernel Library
  Optimized Routines for Science, Engineering & Financial

**Profiling, Analysis, & Architecture**
- Intel® Inspector
  Memory & Threading Checking
- Intel® VTune™ Amplifier
  Performance Profiler
- Intel® Trace Analyzer & Collector
  MPI Profiler
- Intel® Integrated Performance Primitives
  Image, Signal & Compression Routines
- Intel® Threading Building Blocks
  Task Based Parallel C++ Template Library
- Intel® Trace Analyzer & Collector
  MPI Profiler

**Cluster Tools**
- Intel® MPMI Library
- Intel® MPI Library
- 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...

- [ ] Cluster Scalable
  - [ ] Effective threading
    - [ ] Thread
    - [ ] Vectorize
      - [ ] Memory Bandwidth Sensitive
        - [ ] Optimize Bandwidth
        - [ ] Tune MPI
  - [ ] Ignore if you are not targeting clusters.

- [ ] Ignore if you are not targeting clusters.
Intel Parallel Studio XE: A complete tool suit for code and HW performance characterization

Cluster Scalable?

N → Tune MPI

Y → Intel® Trace Analyzer & Collector (ITAC)
Intel® MPI Snapshot
Intel® MPI Tuner

Effective threading?

N → Tune MPI

Y → Vectorize

Memory Bandwidth Sensitive?

N → Thread

Y → Optimize Bandwidth

Intel® VTune™ Amplifier

Intel® Advisor

Intel® VTune™ Amplifier
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
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
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)
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

Performance: DDR4 Vs MCDRAM

Run Time (sec)
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))

```
!DIR$ ATTRIBUTES FASTMEM :: u, rhs, forcing
!DIR$ ATTRIBUTES FASTMEM :: us, vs, ws, gs
!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_size))
allocate (qbc_in(proc_max_size))
```
Example of run script with VTune command line amplxe-cl

```bash
#!/bin/bash -l
AMESROOTDIR=/repo/KNLWORK/APPSDIR/NPB3.3-MZ/NPB3.3-MZ-MPI
echo 'AMESROOTDIR': $AMESROOTDIR
BinDIR=$AMESROOTDIR/bin_withMinsg
echo 'BinDIR': $BinDIR
export OMP_NUM_THREADS=8
NRANKS=8

##########################
# RUN VTUNE TO COLLECT PERFORMANCE DATA
##########################
VTDir=$AMESROOTDIR/MyTestDIR/V TuneResultDir
echo "VTDir $VTDir"

src=vtune RESULTDIR=$VTDir/AdvancedHotspots
tuneresultdir=$VTDir/FT MemoryAccess
vtune RESULTDIR=$VTDir/General Exploration
echo "vtuneresultdir $tuneresultdir"
vtune collection=hpc-performance
vtune collection=memory-access
vtune collection=general-exploration
echo "Tune Collection Metric: $vtune collection"

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

### amplxe-cl -collect memory-access -data-limit=500 
### -knob sampling-interval=100 
### -knob analyze-mem-objects=false -knob dram-bandwidth-limits-true 
### -r $tuneresultdir -source-search-dir $SrcDIR 
### -n 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

ampixe-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 $tuneresultdir -source-search-dir $SrcDIR 
-- 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

<table>
<thead>
<tr>
<th>MemTotal</th>
<th>Node 0</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
<th>Node 7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>24323.73</td>
<td>24322.57</td>
<td>24322.57</td>
<td>23922.40</td>
<td></td>
<td>4039.33</td>
<td>4039.33</td>
<td>4039.33</td>
<td>4036.62</td>
<td>117273.8</td>
</tr>
<tr>
<td>MemFree</td>
<td>21772.73</td>
<td>23667.77</td>
<td>23663.06</td>
<td>21604.9</td>
<td>871.25</td>
<td>1295.75</td>
<td>1290.01</td>
<td>1744.19</td>
<td>55894.05</td>
</tr>
<tr>
<td>MemUsed</td>
<td>2503.99</td>
<td>564.79</td>
<td>540.51</td>
<td>2329.91</td>
<td>3168.08</td>
<td>2745.58</td>
<td>2743.52</td>
<td>2292.43</td>
<td>16879.82</td>
</tr>
<tr>
<td>Active</td>
<td>1775.60</td>
<td>226.73</td>
<td>235.02</td>
<td>1794.54</td>
<td>2818.23</td>
<td>2341.84</td>
<td>2442.16</td>
<td>1900.74</td>
<td>13534.87</td>
</tr>
<tr>
<td>Inactive</td>
<td>116.96</td>
<td>89.65</td>
<td>61.32</td>
<td>147.84</td>
<td>132.79</td>
<td>133.32</td>
<td>133.38</td>
<td>132.17</td>
<td>951.61</td>
</tr>
<tr>
<td>Active(anon)</td>
<td>11.93</td>
<td>19.41</td>
<td>207.36</td>
<td>30.38</td>
<td>2716.22</td>
<td>2237.90</td>
<td>2437.10</td>
<td>1805.76</td>
<td>9376.07</td>
</tr>
<tr>
<td>Inactive(anon)</td>
<td>0.43</td>
<td>0.06</td>
<td>1.49</td>
<td>8.05</td>
<td>1.02</td>
<td>1.03</td>
<td>0.00</td>
<td>0.00</td>
<td>12.08</td>
</tr>
<tr>
<td>Active(file)</td>
<td>1763.67</td>
<td>207.32</td>
<td>27.66</td>
<td>1764.16</td>
<td>102.01</td>
<td>103.94</td>
<td>95.06</td>
<td>94.98</td>
<td>4158.80</td>
</tr>
<tr>
<td>Inactive(file)</td>
<td>116.53</td>
<td>89.59</td>
<td>59.83</td>
<td>139.79</td>
<td>132.76</td>
<td>135.20</td>
<td>132.00</td>
<td>132.17</td>
<td>939.52</td>
</tr>
<tr>
<td>Dirty</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Writeback</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>FilePages</td>
<td>1883.07</td>
<td>296.98</td>
<td>89.30</td>
<td>1912.12</td>
<td>235.82</td>
<td>240.36</td>
<td>228.68</td>
<td>227.17</td>
<td>5111.50</td>
</tr>
<tr>
<td>Mapped</td>
<td>10.00</td>
<td>2.96</td>
<td>0.05</td>
<td>28.04</td>
<td>3.14</td>
<td>3.10</td>
<td>0.00</td>
<td>0.00</td>
<td>47.20</td>
</tr>
<tr>
<td>AnonPages</td>
<td>12.31</td>
<td>19.50</td>
<td>207.26</td>
<td>30.74</td>
<td>2864.48</td>
<td>2435.95</td>
<td>2448.56</td>
<td>1999.78</td>
<td>10019.38</td>
</tr>
<tr>
<td>Shmem</td>
<td>0.64</td>
<td>0.52</td>
<td>1.63</td>
<td>2.50</td>
<td>1.02</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>10.54</td>
</tr>
<tr>
<td>KernelStack</td>
<td>4.12</td>
<td>3.92</td>
<td>4.87</td>
<td>6.95</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>19.85</td>
</tr>
<tr>
<td>PageTables</td>
<td>1.64</td>
<td>0.54</td>
<td>1.79</td>
<td>1.43</td>
<td>6.09</td>
<td>5.21</td>
<td>4.97</td>
<td>4.10</td>
<td>25.19</td>
</tr>
<tr>
<td>NFS_Unstable</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Bounce</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>WritebackTmp</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Slab</td>
<td>127.64</td>
<td>52.75</td>
<td>50.51</td>
<td>132.03</td>
<td>16.93</td>
<td>17.50</td>
<td>15.64</td>
<td>15.99</td>
<td>425.00</td>
</tr>
<tr>
<td>SReclaimable</td>
<td>83.20</td>
<td>15.00</td>
<td>9.56</td>
<td>67.41</td>
<td>8.36</td>
<td>8.58</td>
<td>8.21</td>
<td>8.04</td>
<td>203.35</td>
</tr>
<tr>
<td>SFreeclaim</td>
<td>464.50</td>
<td>30.45</td>
<td>42.75</td>
<td>64.02</td>
<td>8.75</td>
<td>8.98</td>
<td>8.90</td>
<td>8.85</td>
<td>228.45</td>
</tr>
<tr>
<td>AnonHugePages</td>
<td>0.00</td>
<td>0.00</td>
<td>128.00</td>
<td>6.00</td>
<td>2848.00</td>
<td>2420.00</td>
<td>2444.00</td>
<td>1994.00</td>
<td>9840.00</td>
</tr>
<tr>
<td>HugePages_Total</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>HugePages_Free</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**Notes:**
- **DDR4** is highlighted in red.
- **MCDRAM** is highlighted in yellow.

**Legend:**
- **MemTotal** represents the total memory used by all processes.
- **MemFree** shows the free memory available.
- **MemUsed** indicates the memory currently in use.
- **Active** and **Inactive** show the memory usage in active and inactive states respectively.
- **Dirty** indicates dirty pages that need to be written back to disk.
- **FilePages** are pages assigned to files.
- **Mapped** are pages that are assigned to mapped memory.
- **AnonPages** are anonymous pages.
- **Shmem** represents shared memory.
- **KernelStack** is used to store kernel stack frames.
- **PageTables** are page table pages.
- **NFS_Unstable** are unstable NFS pages.
- **Bounce** is used for bounce pages.
- **WritebackTmp** are temporary writeback pages.
- **Slab** is used for slab allocator.
- **SReclaimable** and **SFreeclaim** are reclaimable and free reclaimable memory respectively.
- **AnonHugePages** indicate the number of anonymous huge pages.
- **HugePages_Total** and **HugePages_Free** show the total and free huge pages respectively.
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

<table>
<thead>
<tr>
<th>Memory Object</th>
<th>L2 Miss Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Stress Force Differential tester.cpp:244 (732 MB)</td>
<td>921,027,630</td>
</tr>
<tr>
<td>SIMD POLICY h:204 (30 MB)</td>
<td>106,003,180</td>
</tr>
<tr>
<td>SIMD POLICY h:204 (30 MB)</td>
<td>89,002,670</td>
</tr>
<tr>
<td>SIMD POLICY h:204 (30 MB)</td>
<td>82,002,460</td>
</tr>
<tr>
<td>[Others]</td>
<td>196,005,880</td>
</tr>
</tbody>
</table>
Performance of animation code with DDR4 BW (left) compared to MCDRAM BW (right)

- **DDR4**
  - 30 GB/s

- **MCDRAM**
  - 200 GB/s as cache
  - 220 GB/s

- **CPU Load**
  - Low CPU loads due to back-end bound

**Running on DDR4:** `numctl --m 0`

**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
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 lock implementation (for example, by backing off then rescheduling), 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. The processes laying on the critical path:

<table>
<thead>
<tr>
<th>Process</th>
<th>PID</th>
<th>MPI Communication Spinning (%)</th>
<th>OpenMP Potential Gain (%)</th>
<th>Serial Time (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bl-mz.D.4 (rank 3)</td>
<td>16101</td>
<td>0.020s 0.1%</td>
<td>4.681s 10.8%</td>
<td>0.913s 2.1%</td>
</tr>
<tr>
<td>bl-mz.D.4 (rank 2)</td>
<td>16100</td>
<td>3.474s 8.0%</td>
<td>7.089s 16.3%</td>
<td>4.485s 10.3%</td>
</tr>
<tr>
<td>bl-mz.D.4 (rank 1)</td>
<td>16099</td>
<td>5.875s 13.5%</td>
<td>3.541s 8.2%</td>
<td>7.651s 17.6%</td>
</tr>
<tr>
<td>bl-mz.D.4 (rank 0)</td>
<td>16098</td>
<td>6.680s 15.4%</td>
<td>2.705s 6.2%</td>
<td>7.815s 18.0%</td>
</tr>
</tbody>
</table>
Hot functions and OMP hot spots with most run time and CPU usage profile

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

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.
VTune Amplifier XE: Performance Analyzer – Advanced Hotspots
Advanced-hotspot performance analysis- summary view

**Elapsed Time**: 43.525s
- **CPU Time**: 2404.587s
- **Effective Time**: 1950.254s
- **Spin Time**: 445.008s

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**: 2280,493,432,735
- **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.

<table>
<thead>
<tr>
<th>Process</th>
<th>PID</th>
<th>MPI Communication Spin (%)</th>
<th>OpenMP Potential Gain (%)</th>
<th>Serial Time (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>btmz.D.4 (rank 3)</td>
<td>16101</td>
<td>0.029s 0.1%</td>
<td>4.651s 10.8%</td>
<td>0.913s 2.1%</td>
</tr>
<tr>
<td>btmz.D.4 (rank 2)</td>
<td>16100</td>
<td>3.474s 8.0%</td>
<td>7.089s 15.3%</td>
<td>4.485s 10.3%</td>
</tr>
<tr>
<td>btmz.D.4 (rank 1)</td>
<td>16099</td>
<td>9.875s 13.5%</td>
<td>3.541s 8.2%</td>
<td>7.651s 17.6%</td>
</tr>
<tr>
<td>btmz.D.4 (rank 0)</td>
<td>16098</td>
<td>6.660s 15.4%</td>
<td>2.709s 6.2%</td>
<td>7.319s 18.0%</td>
</tr>
</tbody>
</table>

**Top Hotspots**

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmvortha</td>
<td>btmz.D.4</td>
<td>430.065s</td>
</tr>
<tr>
<td>_kmp_wait_template-kmp_flag_54</td>
<td>libomp.so</td>
<td>352.722s</td>
</tr>
<tr>
<td>matmul_sub</td>
<td>btmz.D.4</td>
<td>309.065s</td>
</tr>
<tr>
<td>z_solve_omp$parallel@43</td>
<td>btmz.D.4</td>
<td>276.360s</td>
</tr>
<tr>
<td>y_solve_omp$parallel@43</td>
<td>btmz.D.4</td>
<td>275.411s</td>
</tr>
</tbody>
</table>
Advanced-hotspot performance analysis – bottom up view
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

```bash
> ampltxe-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

<table>
<thead>
<tr>
<th>SIMD Instructions per Cycle 🕰️: 0.038</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP Instruction Mix:</td>
</tr>
<tr>
<td>% of Packed SIMD Instr.: 100.0%</td>
</tr>
<tr>
<td>% of Scalar SIMD Instr.: 0.0%</td>
</tr>
<tr>
<td>Top 5 hotspot loops (functions) by FPU usage:</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time 🕒</th>
<th>SIMD Instructions per Cycle 🕰️</th>
<th>Vector Instruction Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop at line 211 in multiply32omp5parallel_for@207</td>
<td>446.632s</td>
<td>0.049</td>
<td>AVX(256); FMA(256)</td>
</tr>
<tr>
<td>native_tpl_enable</td>
<td>36.123s</td>
<td>0.001</td>
<td>[Unknown] [Unknown]</td>
</tr>
<tr>
<td>Loop at line 208 in multiply32 omp5parallel_for@207</td>
<td>2.165s</td>
<td>0.025</td>
<td>AVX(256); FMA(256)</td>
</tr>
<tr>
<td>intel_private_timer_func</td>
<td>0.170s</td>
<td>0.057</td>
<td>[Unknown] [Unknown]</td>
</tr>
<tr>
<td>initant</td>
<td>0.030s</td>
<td>1.257</td>
<td>AVX(128); AVX(256); AVX2(256); FMA(256)</td>
</tr>
</tbody>
</table>

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
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
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
2 Vector Processing Units per core
6 channels of DDR4 2400 up to 384GB
8 to 16 GB of on-package MCDRAM memory
36 lanes PCIe Gen 3. 4 lanes of DMI
3 Memory Modes

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

Flat Models

- DDR4
- 16GB MCDRAM
- 16GB MCDRAM

Cache Model

- 64B cache lines
- Direct mapped
- 16GB MCDRAM

Hybrid Model

- Split Options: 25/75% or 50/50%
- 8 or 12GB MCDRAM
- 4 or 8 GB MCDRAM

Physical Address
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