USING EVENT BASED SAMPLING TO UNDERSTAND PERFORMANCE ON INTEL® ARCHITECTURE PROCESSORS

Larry Meadows, Intel Corporation
ATPESC, St Charles, IL
August 7, 2018
Outline

What is Event Based Sampling
System-wide Performance
Core Performance

Vtune usage on Theta: https://www.alcf.anl.gov/user-guides/vtune-xc40
WHAT IS EVENT-BASED SAMPLING
Statistical Profiling with Event Based Sampling

The hardware contains programmable counters to count events (e.g., Instructions Retired, HW Thread Clocks, Cache Misses).

One or more counters are programmed to interrupt after some threshold number of a particular event (sample-after value).

At each interrupt the Instruction Pointer (IP) is recorded, along with the HW Context for which the counter overflowed.

Symbolic information in the executable(s) is used to map IP to some meaningful source or object code entity: instruction, line number, loop, function, shared object.

The profiler creates a statistical profile with aggregated counts by bucket item.
Profiler Data Sources

Executable Program

Symbolic Information

Dynamic information

Static information

Sample Database Over Time

Hardware event interrupts after N occurrences (N=14,000,000 here)

Each sample includes:
- instruction pointer
- hardware context (thread)
- currently running thread ID
- Hardware timestamp
## Example Profile

### CPU_CLK_UNHALTED.THREAD

- **Source Code Information**
  - Function: Loop at line 220 in embree::avx512knl::BVHNntsectorKHybrid
  - Clockticks: 4,481,666,000,000
  - Instructions Retired: 493,668,000,000
  - CPI Rate: 9.078
  - Module: libembr...
  - Function (Full): [Loop at line ...] bvh_inters...

- **Source Code Information**
  - Function: Loop at line 72 in embree::avx512knl::BVHNntsectorKHybrid
  - Clockticks: 1,673,630,000,000
  - Instructions Retired: 156,254,000,000
  - CPI Rate: 10.711
  - Module: libembr...
  - Function (Full): [Loop at line ...] bvh_inters...

### INST_RETIRED.ANY

- **Source Code Information**
  - Function: [Loop at line ...] TriangleMesh...
  - Clockticks: 905,128,000,000
  - Instructions Retired: 6,762,000,000
  - CPI Rate: 133.855
  - Module: libospr...
  - Function (Full): [Loop at line ...] TriangleMesh...

### Computed Metric

- **Source Code Information**
  - Function: [Loop at line ...] TriangleMesh...
  - Clockticks: 487,466,000,000
  - Instructions Retired: 770,000,000
  - CPI Rate: 633.073
  - Module: libbb.s...
  - Function (Full): [Loop at line ...] private_ser...

- **Source Code Information**
  - Function: [Loop at line ...] TriangleMesh...
  - Clockticks: 252,112,000,000
  - Instructions Retired: 11,228,000,000
  - CPI Rate: 22.454
  - Module: libospr...
  - Function (Full): [Loop at line ...] TriangleMesh...

- **Source Code Information**
  - Function: lightAlpha_i...
  - Clockticks: 213,360,000,000
  - Instructions Retired: 17,836,000,000
  - CPI Rate: 11.962
  - Module: libospr...
  - Function (Full): [Loop at line ...] lightAlpha_i...

- **Source Code Information**
  - Function: _raw_spin_lock
  - Clockticks: 204,736,000,000
  - Instructions Retired: 27,020,000,000
  - CPI Rate: 7.577
  - Module: vmlinux
  - Function (Full): _raw_spin_lock

- **Source Code Information**
  - Function: [Loop at line ...] Renderer.dual...renderTiler...UM_un_3C_s_5B...
  - Clockticks: 177,968,000,000
  - Instructions Retired: 4,760,000,000
  - CPI Rate: 37.388
  - Module: libospr...
  - Function (Full): [Loop at line ...] Renderer.dual...renderTiler...UM_un_3C_s_5B...

### Advanced Hotspots

- **Hardware Issues viewpoint (change)**
- **Collection Log**
- **Analysis Target**
- **Analysis Type**
- **Summary**
- **Bottom-up**
- **Top-down Tree**
- **Platform**

- **Grouping:** Function / Call Stack

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>Clockticks</th>
<th>Instructions Retired</th>
<th>CPI Rate</th>
<th>Module</th>
<th>Function (Full)</th>
<th>Source File</th>
<th>Start Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop at line 220 ...</td>
<td>4,481,666</td>
<td>493,668,000,000</td>
<td>9.078</td>
<td>libembr...</td>
<td>[Loop at line ...] bvh_inters...</td>
<td>0x1decdd3</td>
<td></td>
</tr>
<tr>
<td>Loop at line 72 ...</td>
<td>1,673,630</td>
<td>156,254,000,000</td>
<td>10.711</td>
<td>libembr...</td>
<td>[Loop at line ...] bvh_inters...</td>
<td>0x1decdb1</td>
<td></td>
</tr>
<tr>
<td>Loop at line 83 ...</td>
<td>905,128</td>
<td>6,762,000,000</td>
<td>133.855</td>
<td>libospr...</td>
<td>[Loop at line ...] TriangleMe...</td>
<td>0x141fa0</td>
<td></td>
</tr>
<tr>
<td>Loop at line 268 ...</td>
<td>487,466</td>
<td>770,000,000</td>
<td>633.073</td>
<td>libbb.s...</td>
<td>[Loop at line ...] private_ser...</td>
<td>0x1c65a</td>
<td></td>
</tr>
<tr>
<td>TriangleMesh_postIntersect___un_3C_s_5B_unGeometry_5D_3E...</td>
<td>252,112,000</td>
<td>11,228,000,000</td>
<td>22.454</td>
<td>libospr...</td>
<td>TriangleMesh...</td>
<td>0x141eb0</td>
<td></td>
</tr>
<tr>
<td>Loop at line 66 ...</td>
<td>213,360</td>
<td>17,836,000,000</td>
<td>11.962</td>
<td>libospr...</td>
<td>lightAlpha_i...</td>
<td>0x1f82d0</td>
<td></td>
</tr>
<tr>
<td>_raw_spin_lock</td>
<td>204,736</td>
<td>27,020,000,000</td>
<td>7.577</td>
<td>vmlinux</td>
<td>_raw_spin_lock</td>
<td>0xffffffff8163...</td>
<td></td>
</tr>
<tr>
<td>Loop at line 0 ...</td>
<td>177,968</td>
<td>4,760,000,000</td>
<td>37.388</td>
<td>libospr...</td>
<td>[Loop at line ...] Renderer.is...</td>
<td>0x1bcd90</td>
<td></td>
</tr>
<tr>
<td>Loop at line 0 ...</td>
<td>122,738</td>
<td>8,218,000,000</td>
<td>14.935</td>
<td>libospr...</td>
<td>[Loop at line ...] surfaceSh...</td>
<td>0x218ec0</td>
<td></td>
</tr>
<tr>
<td>Loop at line 0 ...</td>
<td>117,712</td>
<td>5,698,000,000</td>
<td>20.658</td>
<td>libospr...</td>
<td>[Loop at line ...] SciVisRen...</td>
<td>0x22d590</td>
<td></td>
</tr>
<tr>
<td>embree::avx512knl::BVHNntsectorKHybrid&lt;...</td>
<td>109,802,000</td>
<td>14,490,000,000</td>
<td>7.578</td>
<td>libembr...</td>
<td>embree::avx5... bvh_inters...</td>
<td>0x1dec170</td>
<td></td>
</tr>
<tr>
<td>Loop at line 48 ...</td>
<td>105,168</td>
<td>12,404,000,000</td>
<td>8.479</td>
<td>libbb.s...</td>
<td>[Loop at line ...] gcc_la32...</td>
<td>0x2aa84</td>
<td></td>
</tr>
</tbody>
</table>
Steps to get a Profile

Include –g (and -O) when compiling to get source-code mapping information:

```bash
icpc -O -g myprog.cc -o myprog
```

Collect with the command-line tool:

```bash
amplxe-cl -collect advanced-hotspots -r myresult ./myprog
```

Analyze with the GUI:

```bash
amplxe-gui myresult
```

Or with a command-line report

```bash
amplxe-cl -report hotspots -r myresult
```
Dimensionality of Profiling Analysis

Four dimensions to consider when analyzing an application profile:

• Code Entities
• Software Execution Entities
• Hardware Execution Entities
• Time

The first three have a natural nested structure imposed by the programming model, execution model, and hardware. Time does not have a natural structure, but annotation can impose structure on time.
Code Entities

• Compiler generates instructions organized into functions and source lines exposed by symbolic information in object files.
• Linker generates shared objects (or executables) by combining object files.
• Basic Blocks and Loops are usually determined by static object code analysis.
Software Execution Entities

- A thread is an execution context containing an instruction pointer (IP) and associated context.
- A process contains one or more threads.
- A single system image is a shared address space that contains multiple processes.
Hardware Execution Entities

- A HW Context is the hardware entity that executes instructions. It contains an IP, registers, and associated state.
- A core consists of one or more HW contexts that share resources. Instructions from more than one HW context can be in flight at the same time.
- A socket consists of multiple cores with a cache-coherent interconnect and a memory interface. It may also contain other resources (e.g. L3 cache, PCI-E interface).
- A node is a cache-coherent single address space consisting of one or more sockets and possibly other resources.
Time

Time can be measured in two ways:

• CPU time. This is the time for which a given thread was executing. This is a thread-specific quantity.

• Elapsed time. This is traditional wallclock (e.g., stopwatch) time.

• If CPU time < Elapsed time then the thread was not executing for the entire interval.

Note that we can measure elapsed time on a given thread by reading a wallclock-equivalent timer (RDTSC) on that thread (thread-specific markers).

Time has no natural structure, but in any given application, there will be a natural division into phases (e.g., OpenMP parallel regions). Dynamic instrumentation (tracing) can be used to segment a profile into these phases.
A Word on Quantization

Typical sample-after (overflow) values are $1e6$ to $10e6$ events.

The CPU can do a lot of things in 10 million cycles; sampling implicitly assumes that it was doing the same thing the whole time.

Be very careful about drawing conclusions obtained by zooming in too closely.

Multiplexing events onto HW event counters exacerbates this effect.

Corollary: be very careful about interpreting profile counts for individual instructions and even source lines.
SYSTEM-WIDE PERFORMANCE
Understanding 4D Data

Typical flat profile summarizes data by reducing over Time, HW Execution Entities, and SW Execution Entities and grouping by function.

Vtune Amplifier generalizes this by providing hierarchical groupings, e.g., Thread/Module/Function reduces over Time and groups first by Thread, then by Module, then by Function within the module. You can explore these in the GUI.

We can also generate 2D views using Pivot Tables by selecting a row dimension and a column dimension, deciding on grouping attributes for each, and reducing over the remaining dimensions. For example:

- Pivot by function is a function-level profile
- Pivot function by thread exposes load imbalance
- Pivot process by cpuid exposes multiple processes on same HW thread
- Pivot thread by cpuid exposes thread migration
# Vtune Hierarchical Grouping

![Vtune Hierarchical Grouping](image)

<table>
<thead>
<tr>
<th>Thread / Module / Function / Call Stack</th>
<th>CPU Time</th>
<th>Instructions Retired</th>
<th>CPI Rate</th>
<th>Module</th>
<th>Function (Full)</th>
<th>Source File</th>
<th>Module Path</th>
<th>Start Address</th>
<th>PID</th>
<th>TID</th>
</tr>
</thead>
<tbody>
<tr>
<td>infer (TID: 1737)</td>
<td>7.103s</td>
<td>11,702,500,000</td>
<td>1.445</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>[Unknown]</td>
<td>4.637s</td>
<td>10,172,500,000</td>
<td>1.060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>libbbsd.s0.2</td>
<td>0.984s</td>
<td>200,000,000</td>
<td>11.525</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>infer</td>
<td>0.798s</td>
<td>562,500,000</td>
<td>3.862</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>[Dynamic code]</td>
<td>0.286s</td>
<td>82,500,000</td>
<td>8.758</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>libmkl.so.0</td>
<td>0.191s</td>
<td>580,000,000</td>
<td>0.707</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>vmlinux</td>
<td>0.134s</td>
<td>72,500,000</td>
<td>5.172</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>syux.so</td>
<td>0.064s</td>
<td>22,500,000</td>
<td>5.667</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>libc-2.17.so</td>
<td>0.006s</td>
<td>5,000,000</td>
<td>1.500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>libstdc++-so.0</td>
<td>0.002s</td>
<td>2,500,000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>libpthread-2.17.so</td>
<td>0s</td>
<td>2,500,000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1737</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1786)</td>
<td>6.342s</td>
<td>11,460,000,000</td>
<td>1.390</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1786</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1776)</td>
<td>6.284s</td>
<td>11,460,000,000</td>
<td>1.289</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1776</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1790)</td>
<td>6.239s</td>
<td>11,292,500,000</td>
<td>1.305</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1790</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1767)</td>
<td>6.218s</td>
<td>11,450,000,000</td>
<td>1.299</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1767</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1780)</td>
<td>3.438s</td>
<td>11,432,500,000</td>
<td>1.284</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1737</td>
<td>1780</td>
</tr>
</tbody>
</table>
Generate the report:
```bash
amplxe-cl -report hw-events -r $1 -group-by=cpuid,core,thread,function \ 
   -format=csv -csv-delimiter=tab -inline-mode=off
```

Create a pandas data frame:
```python
import pandas as pd
df = pd.read_csv(file, sep='\t', index_col=False)
```

Generate a pivot table:
```python
p = pd.pivot_table(df,
   values='Hardware Event Count:CPU_CLK_UNHALTED.REF_TSC',
   index='Thread',
   columns='Module',
   aggfunc=np.sum,
   fill_value=0)
```
Custom Grouping of Code Entities

Modules are often too coarse grained, functions too fine-grained. Use pattern matching and add a column to the dataframe to summarize more usefully (e.g., application, MKL, threading overhead, linux):

<table>
<thead>
<tr>
<th>Thread</th>
<th>MKLDNN</th>
<th>App</th>
<th>TBB</th>
<th>OS</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>infer (TID: 1737)</td>
<td>12.28</td>
<td>2.47</td>
<td>2.62</td>
<td>0.34</td>
<td>0.02</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1786)</td>
<td>12.22</td>
<td>2.36</td>
<td>0.93</td>
<td>0.31</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1776)</td>
<td>12.07</td>
<td>2.44</td>
<td>0.84</td>
<td>0.33</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1790)</td>
<td>11.83</td>
<td>2.47</td>
<td>0.95</td>
<td>0.30</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1767)</td>
<td>11.86</td>
<td>2.49</td>
<td>0.88</td>
<td>0.27</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1781)</td>
<td>11.62</td>
<td>2.39</td>
<td>1.01</td>
<td>0.28</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1753)</td>
<td>11.29</td>
<td>2.56</td>
<td>1.09</td>
<td>0.25</td>
<td>0.00</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1756)</td>
<td>10.14</td>
<td>2.37</td>
<td>1.40</td>
<td>0.31</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1739)</td>
<td>9.82</td>
<td>2.60</td>
<td>1.50</td>
<td>0.29</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1759)</td>
<td>10.12</td>
<td>2.45</td>
<td>1.35</td>
<td>0.27</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1768)</td>
<td>9.78</td>
<td>2.68</td>
<td>1.45</td>
<td>0.27</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1746)</td>
<td>10.02</td>
<td>2.53</td>
<td>1.34</td>
<td>0.30</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1744)</td>
<td>10.02</td>
<td>2.54</td>
<td>1.37</td>
<td>0.24</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1779)</td>
<td>10.01</td>
<td>2.44</td>
<td>1.36</td>
<td>0.27</td>
<td>0.00</td>
</tr>
<tr>
<td>TBB Worker Thread (TID: 1742)</td>
<td>9.56</td>
<td>2.61</td>
<td>1.51</td>
<td>0.24</td>
<td>0.00</td>
</tr>
<tr>
<td>vmlinux (TID: 0)</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>8.19</td>
<td>0.00</td>
</tr>
<tr>
<td>Other Threads (78)</td>
<td>0.00</td>
<td>0.00</td>
<td>0.30</td>
<td>0.04</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Adding Structure to the Time Dimension

Add instrumentation to the code to record time stamps for ‘interesting’ intervals, e.g. OpenMP parallel regions (Vtune Amplifier has those built in), user-defined tasks, solver calls, CNN convolutions. Vtune Amplifier can import this data, see:


Example: Use MKLDNN_VERBOSE to print log for each operation with start and end TSC, convert with python script, and import into existing result:

MKLDNN_VERBOSE=2 amplxe-cl -collect advanced-hotspots sh run.sh >Log
python parse.py Log
amplxe-cl -r r000ah -import mkldnn-hostname-ortce-ortce-sk13.csv
### Example: Metrics by MKLDNN call

**amplxe-cl -report hw-events -group-by frame-domain -r r00ah/ -format csv -csv-delimiter comma -q >report.csv**

<table>
<thead>
<tr>
<th>Frame Domain</th>
<th>INST_RETIRED.ANY</th>
<th>CPU_CLK_UNHALTED.THREAD</th>
<th>CPU_CLK_UNHALTED.REF_TSC</th>
<th>CPI</th>
<th>Turbo</th>
<th>Frame Time</th>
<th>Frame Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECF19</td>
<td>178,825,000,000</td>
<td>132,496,100,000</td>
<td>142,772,500,000</td>
<td>0.741</td>
<td>0.93</td>
<td>1.76</td>
<td>100</td>
</tr>
<tr>
<td>ECF20</td>
<td>93,478,900,000</td>
<td>65,727,100,000</td>
<td>71,442,600,000</td>
<td>0.703</td>
<td>0.92</td>
<td>0.88</td>
<td>100</td>
</tr>
<tr>
<td>ECF17</td>
<td>69,250,700,000</td>
<td>53,900,500,000</td>
<td>58,666,100,000</td>
<td>0.778</td>
<td>0.92</td>
<td>0.72</td>
<td>100</td>
</tr>
<tr>
<td>ECF1</td>
<td>52,451,500,000</td>
<td>43,752,900,000</td>
<td>47,913,600,000</td>
<td>0.834</td>
<td>0.91</td>
<td>0.59</td>
<td>100</td>
</tr>
<tr>
<td>ECF13</td>
<td>14,280,700,000</td>
<td>33,037,200,000</td>
<td>32,062,000,000</td>
<td>2.213</td>
<td>1.03</td>
<td>0.39</td>
<td>100</td>
</tr>
<tr>
<td>ECF18</td>
<td>28,975,400,000</td>
<td>28,853,500,000</td>
<td>29,014,500,000</td>
<td>0.85</td>
<td>0.93</td>
<td>0.39</td>
<td>100</td>
</tr>
<tr>
<td>ECF0</td>
<td>32,296,600,000</td>
<td>26,969,800,000</td>
<td>28,267,000,000</td>
<td>0.706</td>
<td>0.91</td>
<td>0.35</td>
<td>100</td>
</tr>
<tr>
<td>ECF14</td>
<td>9,887,700,000</td>
<td>31,700,900,000</td>
<td>31,146,600,000</td>
<td>0.996</td>
<td>0.93</td>
<td>0.39</td>
<td>100</td>
</tr>
<tr>
<td>ECF21</td>
<td>36,406,700,000</td>
<td>25,711,700,000</td>
<td>28,135,900,000</td>
<td>0.36</td>
<td>0.91</td>
<td>0.35</td>
<td>100</td>
</tr>
<tr>
<td>ECF7</td>
<td>6,216,900,000</td>
<td>23,379,500,000</td>
<td>21,318,700,000</td>
<td>1.10</td>
<td>0.26</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ECF16</td>
<td>18,377,000,000</td>
<td>17,994,000,000</td>
<td>19,499,400,000</td>
<td>0.66</td>
<td>0.92</td>
<td>0.24</td>
<td>100</td>
</tr>
<tr>
<td>ECF3</td>
<td>17,381,100,000</td>
<td>16,721,000,000</td>
<td>18,160,800,000</td>
<td>0.692</td>
<td>0.92</td>
<td>0.22</td>
<td>100</td>
</tr>
<tr>
<td>EPF2</td>
<td>1,285,700,000</td>
<td>17,544,400,000</td>
<td>15,957,400,000</td>
<td>3.64</td>
<td>1.10</td>
<td>0.20</td>
<td>100</td>
</tr>
<tr>
<td>ECF11</td>
<td>4,123,900,000</td>
<td>9,832,500,000</td>
<td>10,228,100,000</td>
<td>2.80</td>
<td>0.91</td>
<td>0.13</td>
<td>100</td>
</tr>
<tr>
<td>ECF12</td>
<td>3,454,600,000</td>
<td>8,995,300,000</td>
<td>9,434,600,000</td>
<td>2.68</td>
<td>0.95</td>
<td>0.12</td>
<td>100</td>
</tr>
<tr>
<td>ECF5</td>
<td>7,316,300,000</td>
<td>8,038,500,000</td>
<td>8,574,400,000</td>
<td>1.09</td>
<td>0.94</td>
<td>0.11</td>
<td>100</td>
</tr>
<tr>
<td>ECF9</td>
<td>1,973,400,000</td>
<td>7,659,000,000</td>
<td>7,516,400,000</td>
<td>3.38</td>
<td>1.02</td>
<td>0.09</td>
<td>100</td>
</tr>
<tr>
<td>EPF4</td>
<td>471,500,000</td>
<td>5,556,800,000</td>
<td>5,437,200,000</td>
<td>11.78</td>
<td>1.02</td>
<td>0.07</td>
<td>100</td>
</tr>
<tr>
<td>EPF6</td>
<td>167,900,000</td>
<td>2,279,300,000</td>
<td>2,343,700,000</td>
<td>13.57</td>
<td>0.97</td>
<td>0.03</td>
<td>100</td>
</tr>
<tr>
<td>EPF8</td>
<td>85,100,000</td>
<td>1,087,900,000</td>
<td>975,200,000</td>
<td>12.78</td>
<td>1.12</td>
<td>0.01</td>
<td>100</td>
</tr>
<tr>
<td>EPF10</td>
<td>41,400,000</td>
<td>409,400,000</td>
<td>498,400,000</td>
<td>9.889</td>
<td>1.00</td>
<td>0.01</td>
<td>100</td>
</tr>
</tbody>
</table>

[No frame domain - Outside any frame]

<table>
<thead>
<tr>
<th>INST_RETIRED.ANY</th>
<th>CPU_CLK_UNHALTED.THREAD</th>
<th>CPU_CLK_UNHALTED.REF_TSC</th>
<th>CPI</th>
<th>Turbo</th>
<th>Frame Time</th>
<th>Frame Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>33,846,800,000</td>
<td>66,950,700,000</td>
<td>60,122,000,000</td>
<td>1.978</td>
<td>1.11</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
CORE PERFORMANCE
Microarchitecture Review

Common elements between KNL and Intel® Xeon® processors (SKX/BDW):

- Multithreaded cores with private L1 caches connected by a mesh

Key differences:

- Two KNL cores share one L2, SKX/BDW cores have private L2
- SKX/BDW has an L3 cache, KNL does not
- SKX/BDW systems are multi-socket, KNL is single-socket
- SKX/BDW can retire 4 IPC, KNL can retire 2 IPC
- KNL MCDRAM has much higher bandwidth than SKX/BDW DDR
- BDW does not have AVX-512
Processor Performance

Standard metric is retired Instructions Per Cycle (IPC)
• KNL max is 2 IPC per core. SKX/BDW max is 4 IPC per core.
• Vtune amplifier displays the reciprocal CPI
• Computed as $\frac{\sum_{hwthreads} instructions}{\sum_{hwthreads} cycles}$

• $instructions == INST\_RETIRED\_ANY$ $cycles == CPU\_CLK\_UNHALTED\_THREAD$

Note: IPC per core depends on how many HW threads per core (nHT) are running:

$IPC_{core} = IPC_{thread} \times nHT$

$CPI_{core} = CPI_{thread} / nHT$

Vtune Amplifier always displays $CPI_{thread}$

This is why all scaling graphs should be in terms of cores, not threads, and should show the number of threads per core used.
Common Impediments to Perfect IPC

Bandwidth Bound

• Beware of BW bound code that doesn't need to be (e.g., fails to optimize for reuse in L2 cache)

• Lack of vectorization, lack of streaming stores, missing SW prefetches may limit BW

Other Impediments

• Code may be BW bound from L1 or L2 cache as well as from memory.

• Micro-architectural decisions may limit IPC (e.g., some VPU instructions issue on only one port)

• Real dependences may limit IPC (one instruction needs to wait for another's result)

• Code may be front-end bound (instruction fetch and decode, branch prediction)
Vtune Amplifier Pre-defined Collection Types

- collect advanced-hotspots
  CPI, Turbo frequency
- collect memory-access
  Memory bandwidth, L2 hit/miss information
- collect general-exploration
  Top-down cycle accounting

General exploration multiplexes many events onto the hardware counters. This can lead to inaccurate results especially on short-running program segments. It is more useful on SKX/BDW than on KNL.
Finding Bottlenecks

Two general methods:

Top-down breaks execution into 4 domains: FE Bound, Bad Speculation, BE Bound, and Retiring.

https://www.researchgate.net/publication/269302126_A_Top-Down_method_for_performance_analysis_and_counters_architecture

Bottleneck analysis looks for common bottlenecks, typically some kind of bandwidth, and compares performance against expected peak.
Instruction Roofline Guides BW Analysis

Generalization of roofline model using instructions rather than flops

E5-2699 v4 @ 2.20GHz
2 socket 22 cores/socket
BW = 115GB/Sec
Peak GIPS: IPC*44*2.2 GHz (ignores frequency changes)
Top-Down Analysis Categories

FE Bound: Decoder can’t deliver uops fast enough. Usually I$ or really big codes.

Bad Speculation: uops that are speculatively executed but the result is never used.

Retiring: What you want! Normal retirement of uops

BE Bound: Everything else. Stalls due to data unavailable and no more OOO

Vtune Amplifier general exploration has good documentation on this. The GUI tooltips give the actual names of the events.
## Top-Down Analysis Example

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE Bound</td>
<td>( \frac{\text{IDQ_UOPS_NOT_DELIVERED_CORE}}{4\times(\text{CPU_CLK_UNHALTED_THREAD}/2)} )</td>
</tr>
<tr>
<td>Bad Speculation</td>
<td>( \frac{\text{(UOPS_ISSUED_ANY-} \text{UOPS_RETIRED_RETIRED_SLOTS+4\times\text{INT_MISC_RECOVERY_CYCLES})}}{4\times(\text{CPU_CLK_UNHALTED_THREAD}/2)} )</td>
</tr>
<tr>
<td>Retiring</td>
<td>( \frac{\text{UOPS_RETIRED_RETIRED_SLOTS}}{4\times(\text{CPU_CLK_UNHALTED_THREAD}/2)} )</td>
</tr>
<tr>
<td>BE Bound</td>
<td>( 1 - (\text{FE Bound} + \text{Bad Speculation} + \text{Retiring}) )</td>
</tr>
</tbody>
</table>

**Vtune Command:**
```
amplxe-cl -collect-with runsa \\  \\
-knob event-config=\\  \\
\text{CPU\_CLK\_UNHALTED\_REF\_TSC,_CPU\_CLK\_UNHALTED\_THREAD,}\\  \\
\text{INST\_RETIRED\_ANY,}\\  \\
\text{IDQ\_UOPS\_NOT\_DELIVERED\_CORE,\_INT\_MISC\_RECOVERY\_CYCLES,}\\  \\
\text{UOPS\_RETIRED\_RETIRED\_SLOTS,\_UOPS\_ISSUED\_ANY} \\
<\text{command}>```
# Metrics for KNL

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 input BW</td>
<td>(64 \times (L2_REQUESTS_MISS + L2_PREFETCHER_ALLOC_XQ) / \text{elapsed_time})</td>
<td>380 GB/sec</td>
</tr>
<tr>
<td>L2 Hit Rate</td>
<td>(\frac{\text{MEM_UOPS_RETIRED}_L2_HIT_LOADS}{(\text{MEM_UOPS_RETIRED}_L2_HIT_LOADS + \text{MEM_UOPS_RETIRED}_L2_MISS_LOADS)})</td>
<td>100%</td>
</tr>
<tr>
<td>L1 input lines</td>
<td>(\frac{(2 \times L2_REQUESTS_MISS + 3 \times L2_PREFETCHER_ALLOC_XQ + L2_REQUESTS_REFERENCE)}{\text{CPU_CLK_UNHALTED}_THREAD})</td>
<td>0.46</td>
</tr>
<tr>
<td>L1 Hit Rate</td>
<td>(\frac{(\text{MEM_UOPS_RETIRED}_ALL_LOADS - \text{MEM_UOPS_RETIRED}_L1_MISS_LOADS)}{\text{MEM_UOPS_RETIRED}_ALL_LOADS})</td>
<td>100%</td>
</tr>
<tr>
<td>SIMD operations/clock</td>
<td>(\frac{(\text{UOPS_RETIRED}_SCALAR_SIMD} + \text{UOPS_RETIRED}_PACKED_SIMD)}{\text{CPU_CLK_UNHALTED}_THREAD})</td>
<td>2</td>
</tr>
<tr>
<td>FE Bound</td>
<td>(\frac{\text{NO_ALLOC_CYCLES}_NOT_DELIVERED}{\text{CPU_CLK_UNHALTED}_THREAD})</td>
<td>&lt;5%</td>
</tr>
<tr>
<td>Branch Mispredict Bound</td>
<td>(\frac{\text{NO_ALLOC_CYCLES}_MISPREDS}{\text{CPU_CLK_UNHALTED}_THREAD})</td>
<td>&lt;5%</td>
</tr>
<tr>
<td>Frequency Ratio</td>
<td>(\frac{\text{CPU_CLK_UNHALTED}_THREAD/\text{CPU_CLK_UNHALTED}_REF_TSC}{\text{v}})</td>
<td>varies with SKU</td>
</tr>
</tbody>
</table>
Performance Analysis and Improvement

1. **Measure** the performance of the application running on the system
2. **Estimate** the peak performance of the application
3. If performance is near enough to peak, you are done
4. **Rewrite** the code (and possibly algorithm) and go to step 1

Here we cover #1 and #2
Measuring Performance

Instrumentation

• Modify the code to read timers or counters around interesting code regions
• Summarize the data and print it

Sampling

• Program timers or counters to interrupt the program periodically
• Use the address of the interrupted instruction to build a statistical profile

We primarily discuss sampling but include information on instrumentation techniques
Hardware+OS View of Program Execution

Each HW thread executes instructions from a single OS process at a time.

The process executing on a given thread may change (voluntary or involuntary context switch).

For full understanding of machine performance must see all instructions executed on all HW threads.

`amplxe-cl -analyze-system` or `perf -a`

Otherwise you may miss important events perturbing the execution of your program.
Two Different Ways to Collect and View a Profile Using Hardware Counters

Intel® Vtune™ Amplifier

amplxe-cl -collect advanced-hotspots -r myresult ./a.out
amplxe-gui myresult  # GUI
amplxe-cl -report hotspots -r myresult  # CLI

Linux perf

perf record -e cycles -e ref-cycles -e instructions ./a.out
perf report     # GUI or CLI
perf script     # Raw data
Why use Vtune Amplifier?

Linux perf has higher overhead than the driver used by Vtune Amplifier

- perf -a has lower overhead but requires root permissions or perf_event_paranoid=0

Vtune Amplifier has superior GUI and report generation

- See all events on one grid
- Powerful grouping and filtering capabilities
- Command-line reports can use all grouping and filtering capabilities

We primarily focus on Vtune Amplifier with some discussion of perf
Other Tools

Several third party tools can provide useful information:

- HPCToolkit (Rice); Tau (ParaTools); MAP (ARM); PAPI (UTK); others...

Linux kernel tracing (ftrace) for detailed kernel-level performance analysis.

Linux statistics (via top, ps, sysfs and procfs pseudo-files) for detailed node-level performance analysis like memory usage, overactive daemons, and I/O performance.

perf stat for a quick check on hardware event counts and metrics.

Don’t forget the shell’s time command!
ANALYZING PARALLELISM
Shared Memory Parallelism Review

Multiple hardware threads cooperate to solve a problem
Use a parallel language like OpenMP or OpenCL, or
Use a parallel library like TBB

Underlying implementation uses pthreads on linux:

- Pthreads are essentially linux processes sharing an address space
- Linux scheduling decides which hardware threads run which pthreads
- It may be necessary to affinitize (bind) pthreads to specific hardware threads to avoid linux scheduler issues
- Normally we assume that we own some or all of the cores on the machine
Parallel Scaling

Goal is linear scaling as cores are added:

If one core takes time P, then N cores should take time P/N

Using multiple threads per core can help to hide latency but does not add resources:

2 threads on only 1 core is very different than 2 threads on 2 separate cores

Plot scaling as shown:
Impediments to Parallel Scaling

Algorithmic
• Load Imbalance
• Serial code
• Synchronization

Architectural:
• Cache contention
• Mesh bandwidth
• Memory bandwidth
• NUMA issues
Identifying Load Imbalance

Performance is typically determined by the slowest thread

Serial code is an extreme case of load imbalance (only one thread)

Key insight: look at clocks executed per thread.
Synchronization

On Intel® Architecture Processors synchronization is always through memory. Synchronization implies transfers from cache on one core to cache on another. Implicit synchronization occurs because of the programming model, e.g., starting and ending a TBB task. Explicit synchronization occurs due to a programmer action, e.g., an atomic operation. False synchronization (aka false sharing) occurs when two cores access the same cache line but different parts of that cache line. Note that cache-to-cache transfers can occur even with read-only data; this is not synchronization but still can have performance implications.
Identifying Synchronization

Significant time in runtime shared objects (modules, e.g. libtbb.so)

But that might also be due to load imbalance (waiting for work)

Significant time in or near atomic operations or compare-and-set loops

May be necessary to look at assembly

lock prefix is an atomic operation
Bandwidth (KNL)

Memory bandwidth: on Intel® Xeon Phi™ 7250

- DDR: ~90 GB/sec, can saturate with ~10 cores
- MCDRAM: ~490 GB/sec, can saturate with ~64 cores

MCDRAM is asymmetric, read-only BW is ~380GB/sec, write-only is ~200GB/sec

Mesh bandwidth: includes cache-to-cache transfers and memory transfers

- Rule of thumb: ~380 GB/sec max read BW

We can measure mesh read BW with per-thread hardware events:

\[
64 \times (L2\_REQUESTS.MISS + L2\_PREFETCHER.ALLOC\_XQ) / \text{elapsed\_time}
\]
Bandwidth (Intel® Xeon® processors)

Each socket has its own DDR

Cross-socket traffic is through QPI/UPI and has much less bandwidth than DDR (NUMA effects)

Memory BW is dependent on number of DIMMS and memory frequency. On Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz with 2400MHz DDR4:

- Stream Triad 113GB/sec (56GB/sec/socket)
- SKX gets ~105GB/sec/socket
KNL STREAM Benchmark Scaling

**MCDRAM Stream Triad**

- Graph showing the relationship between #cores and GB/sec for MCDRAM Stream Triad.
- Data points for 1T, 2T, and 4T.

**DDR Stream Triad**

- Graph showing the relationship between #cores and GB/sec for DDR Stream Triad.
- Data points for 1T, 2T, and 4T.
BDW Bandwidth Scaling

One socket, local memory

One socket, remote memory
Vtune Amplifier Data, KNL

amplxe-cl -collect_memory-access

-collect memory-access uses uncore counters at the memory controllers. The data can’t be localized to a particular code segment.

Using core hardware events: amplxe-cl -collect-with runsa -knob event-config=...

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CLK_UNHALTED.THREAD</td>
<td>28,798,043,197</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED.REF_TSC</td>
<td>28,802,043,203</td>
</tr>
<tr>
<td>L2_REQUESTS.MISS</td>
<td>144,602,169</td>
</tr>
<tr>
<td>L2_PREFETCHER.ALLOC_XQ</td>
<td>1,531,807,219</td>
</tr>
<tr>
<td>Lines in</td>
<td>1,676,409,388</td>
</tr>
<tr>
<td>Elapsed Time</td>
<td>0.321451375</td>
</tr>
<tr>
<td>Read BW (GB/sec)</td>
<td>334</td>
</tr>
</tbody>
</table>

Core hardware events can be localized to specific code segments but can measure only read bandwidth and include cache-to-cache transfers. This data is for the STREAM Triad loop only.
Vtune Amplifier Data, BDW

amplxe-cl -collect memory-access

Note two packages (sockets)
Data from uncore counters

Using core hardware events: amplxe-cl -collect-with runsa -knob event-config=...

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CLK_UNHALTED.THREAD</td>
<td>156,602,234,903</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED.REF_TSC</td>
<td>132,510,198,765</td>
</tr>
<tr>
<td>L2_LINES_IN.ALL</td>
<td>1,668,750,061</td>
</tr>
<tr>
<td>Time (s)</td>
<td>1.368907012</td>
</tr>
<tr>
<td>L2 input BW</td>
<td>78.01845046</td>
</tr>
</tbody>
</table>

L2 input bandwidth for stream triad
Generate a report with all useful columns:
```
amplxe-cl -report hw-events -r $1 \ 
  -group-by=process,package,cpuid,core,thread,function \ 
    -format=csv -csv-delimiter=comma -inline-mode=off
```

Pivot, summarizing by addition, to see relationships. Examples:

Pivot by function is a function-level profile

Pivot function by thread exposes load imbalance

Pivot process by cpuid exposes multiple processes on same HW thread

Pivot thread by cpuid exposes thread migration
Pivot Table Example

(part of a) Pivot Table of core by process
Entries are CPU_CLK_UNHALTED.REF_TSC (elapsed time)

<table>
<thead>
<tr>
<th></th>
<th>eu-options-AVX2</th>
<th>md127 RAID10</th>
<th>vmlinux</th>
<th>md127_resync</th>
</tr>
</thead>
<tbody>
<tr>
<td>core_0</td>
<td>2,380,400,000</td>
<td></td>
<td>4,400,000</td>
<td></td>
</tr>
<tr>
<td>core_1</td>
<td>2,292,400,000</td>
<td></td>
<td>6,600,000</td>
<td></td>
</tr>
<tr>
<td>core_10</td>
<td>2,285,800,000</td>
<td></td>
<td>24,200,000</td>
<td></td>
</tr>
<tr>
<td>core_2</td>
<td>2,292,400,000</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>core_20</td>
<td>2,013,000,000</td>
<td>268,400,000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>core_4</td>
<td>2,292,400,000</td>
<td>147,400,000</td>
<td>6,600,000</td>
<td></td>
</tr>
<tr>
<td>Grand Total</td>
<td>99,723,800,000</td>
<td>415,800,000</td>
<td>143,000,000</td>
<td>116,600,000</td>
</tr>
</tbody>
</table>

Note processes md127_* and vmlinux running on same cores as application
Legal Disclaimers

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel’s plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel’s results and plans is included in Intel’s SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo, Atom, Xeon, Xeon Phi, 3D Xpoint, Iris Pro and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

© 2016 Intel Corporation.
Legal Disclaimers

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 SSE3 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