Agenda

- Introduction to Performance Tuning
- Introduction to Intel VTune Amplifier
- System-Level Profiling
  - HPC Characterization
  - Disk I/O Analysis
- Application Performance Tuning Process
  - Find Hotspots
  - Determine Efficiency
  - Address Parallelism Issues
  - Address Hardware Issues
  - Rebuild and Compare
- Summary
## Two Great Ways to Collect Data

**Intel® VTune™ Amplifier**

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses OS interrupts</td>
<td>Uses the on chip Performance Monitoring Unit (PMU)</td>
</tr>
<tr>
<td>Collects from a single process tree</td>
<td>Collect system wide or from a single process tree.</td>
</tr>
<tr>
<td>~10ms default resolution</td>
<td>~1ms default resolution (finer granularity - finds small functions)</td>
</tr>
<tr>
<td>Either an Intel® or a compatible processor</td>
<td>Requires a genuine Intel® processor for collection</td>
</tr>
<tr>
<td>Call stacks show calling sequence</td>
<td>Optionally collect call stacks</td>
</tr>
<tr>
<td>Works in virtual environments</td>
<td>Works in a VM only when supported by the VM (e.g., vSphere*, KVM)</td>
</tr>
<tr>
<td>No driver required</td>
<td>Requires a driver</td>
</tr>
<tr>
<td></td>
<td>- Easy to install on Windows</td>
</tr>
<tr>
<td></td>
<td>- Linux requires root (or use default perf driver)</td>
</tr>
</tbody>
</table>

No special recompiles - C, C++, C#, Fortran, Java, Assembly
# A Rich Set of Performance Data

**Intel® VTune™ Amplifier**

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hotspots</strong></td>
<td><strong>Hotspots</strong></td>
</tr>
<tr>
<td>Which functions use the most time?</td>
<td>Which functions use the most time?</td>
</tr>
<tr>
<td><strong>Threading</strong></td>
<td><strong>Microarchitecture Exploration</strong></td>
</tr>
<tr>
<td>Tune parallelism.</td>
<td>Where is the biggest opportunity?</td>
</tr>
<tr>
<td>Colors show number of cores used.</td>
<td>Cache misses? Branch mispredictions?</td>
</tr>
<tr>
<td>Tune the #1 cause of slow threaded performance: – waiting with idle cores.</td>
<td><strong>Advanced Analysis</strong></td>
</tr>
<tr>
<td>Any IA86 processor, any VM, no driver</td>
<td>Memory-access, HPC Characterization, etc...</td>
</tr>
</tbody>
</table>

Higher res., lower overhead, system wide

No special recompiles - C, C++, C#, Fortran, Java, Assembly
Example: Hotspots Analysis

**Summary View**

- **Elapsed Time**: 5.554s
  - **CPU Time**: 10.504s
    - Instructions Retired: 21,099,000,000
    - CPI Rate: 1.257
    - CPU Frequency Ratio: 1.041
    - Total Thread Count: 9
    - Pause Time: 0s

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>CPU Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid_intersect</td>
<td>3_tachyon_omp.exe</td>
<td>5.538s</td>
</tr>
<tr>
<td>sphere_intersect</td>
<td>3_tachyon_omp.exe</td>
<td>3.247s</td>
</tr>
<tr>
<td>func@0x1002a6a9d</td>
<td>libomp5amd.dll</td>
<td>0.148s</td>
</tr>
<tr>
<td>shader</td>
<td>3_tachyon_omp.exe</td>
<td>0.117s</td>
</tr>
<tr>
<td>KeDelayExecutionThread</td>
<td>ntoskm.exe</td>
<td>0.091s</td>
</tr>
<tr>
<td>[Others]</td>
<td>N/A*</td>
<td>1.361s</td>
</tr>
</tbody>
</table>

*Wt is applied to non-summable metrics.

- **Average Bandwidth**
  - Package 0: Total: 5.715 GB/sec, Read: 3.504 GB/sec, Write: 2.212 GB/sec

---

**CPU Usage Histogram**
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.

**Collection and Platform Info**
This section provides information about this collection, including result set size and collection platform data.
Example: Threading Analysis
Bottom-up View
Find Answers Fast

Intel® VTune™ Amplifier

Adjust Data Grouping
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
  ... (Partial list shown)

Double Click Function to View Source

Click [+] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Filter by Process & Other Controls
Tuning Opportunities Shown in Pink. Hover for Tips
See Profile Data On Source / Asm
Double Click from Grid or Timeline

View Source / Asm or both
CPU Time
Right click for instruction reference manual

Quick Asm navigation:
Select source to highlight Asm

Scroll Bar “Heat Map” is an overview of hot spots
Click jump to scroll Asm
Command Line Interface

Automate analysis

amplxe-cl is the command line:

- **Windows:** `C:\Program Files (x86)\IntelSWTools\VTune Amplifier\bin[32|64]\amplxe-cl.exe`
- **Linux:** `/opt/intel/vtune_amplifier/bin[32|64]/amplxe-cl`

Help: `amplxe-cl -help`

Use UI to setup
1) Configure analysis in UI
2) Press “Command Line...” button
3) Copy & paste command

Great for regression analysis – send results file to developer
Command line results can also be opened in the UI
Compare Results Quickly - Sort By Difference

Intel® VTune™ Amplifier

Quickly identify cause of regressions.
- Run a command line analysis daily
- Identify the function responsible so you know who to alert

Compare 2 optimizations – What improved?
Compare 2 systems – What didn’t speed up as much?

![Screen shot of VTune Amplifier interface showing function and call stack comparison]
Introduction to Performance Tuning

System
- H/W tuning:
  - BIOS (TB, HT)
  - Memory
  - Network I/O
  - Disk I/O
- OS tuning:
  - Page size
  - Swap file
  - RAM Disk
  - Power settings
  - Network protocols

Application
- Better application design:
  - Parallelization
  - Fast algorithms / data bases
  - Programming language and RT libs
  - Performance libraries
  - Driver tuning

Processor
- Tuning for Microarchitecture:
  - Compiler settings/Vectorization
  - Memory/Cache usage
  - CPU pitfalls

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Introduction to Intel VTune Amplifier

- **Accurate Data - Low Overhead**
  - CPU, GPU, FPU, threading, bandwidth, and more...
- **Profile applications or systems**
- **Meaningful Analysis**
  - Threading and hardware utilization efficiency
  - Memory and storage device analysis
- **Easy**
  - Data displayed by source code
  - Expert advice built-in
  - Easy set-up, no special compiles

---

*Optimization Notice*

> amplxe-cl -help collect
System-Level Profiling – High-level Overviews

Elapsed Time: 6.306s
Clockticks: 30,669,300,000
Instructions Retired: 25,745,000,000
CPI Rate: 1.199
MUX Reliability: 0.972
Front-End Bound: 7.2% of Pipeline Slots
Bad Speculation: 6.0% of Pipeline Slots
Branch Mispredict: 5.5% of Pipeline Slots
Machine Cuts: 0.1% of Pipeline Slots
Back-End Bound: 44.1%
Memory Bound: 33.3%
Core Bound: 30.6%
Divider: 0.0%
Port Utilization: 29.6%
Buffering: 22.7%
Total Thread Count: 9
Paused Time: 0.8s

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.

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
System-Level Profiling – Process/Module Breakdowns

Processes

Modules

Functions
System-Level Profiling – I/O Analysis

Are You I/O Bound or CPU Bound?
- Explore imbalance between I/O operations (async & sync) and compute
- Storage accesses mapped to the source code

See when CPU is waiting for I/O
- Measure bus bandwidth to storage
- Latency analysis
- Tune storage accesses with latency histogram
- Distribution of I/O over multiple devices

> amplxe-cl -collect io -d 10
System-Level Profiling – HPC Characterization

Three Metric Classes
- CPU Utilization
  - Logical core % usage
  - Includes parallelism and OpenMP information
- Memory Bound
  - Break down each level of the memory hierarchy
- FPU Utilization
  - Floating point GFLOPS and density

> `amplxe-cl -collect hpc-performance -d 10`

**Optimization Notice**

Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.
System-Level Profiling – Memory Bandwidth

Find areas of high and low bandwidth usage. Compare to max system bandwidth based on Stream benchmarks.

-knob collect-memory-bandwidth=true
Application Performance Tuning Process

1. Find Hotspots
2. Address Hardware Issues
3. Address Parallelism Issues
4. Determine Efficiency
5. Rebuild and Compare Results
Find Hotspots

> amplxe-cl -collect hotspots -- ./myapp.out
Find Hotspots

- Drill to source or assembly
- Hottest areas easy to ID
- Is this the expected behavior
- Pay special attention to loops and memory accesses
- Learn how your code behaves
- What did the compiler generate
- What are the expensive statements
Determine Efficiency

Look for Parallelism, Cycles-per-Instruction (CPI), and Retiring %
Address Parallelism Issues

- Use Concurrency Analysis to ensure you’re using all your threads as often as possible.
- Common concurrency problems can often be diagnosed in the timeline.
- Switch to the Locks And Waits viewpoint or run a Locks and Waits analysis to investigate contention.
The X86 Processor Pipeline (simplified)

Address Hardware Issues
Address Hardware Issues

For each pipeline slot on each cycle:

- uop allocated?
  - Yes: uop ever retired?
    - Yes: Retiring
    - No: Bad Speculation
  - No: back end stalled?
    - Yes: Back-End Bound
    - No: Front-End Bound
Address Hardware Issues

Microarchitecture Exploration Analysis Shows the Hardware Bottleneck

```
> amplxe-cl -collect uarch-exploration -- ./myapp.out
```

This data is collected statistically with event multiplexing. Gray data has low confidence levels.
Rebuild and Compare Results

**Elapsed Time**: 7.420s - 5.541s = 1.879s

- Instructions Retired: 24,654,400,000 - 22,868,400,000 = 1,786,000,000
- CPI Rate: 1.326 - 1.363 = -0.037
- CPU Frequency Ratio: 1.040 - 1.042 = -0.003
- Total Thread Count: Not changed, 4
- Paused Time: Not changed, 0s
- CPU Time: 12.603s - 11.967s = 0.516s

**CPU Usage Histogram**

This histogram displays a percentage of the wall time the specific number of CPUs were running.
• Start with the lowest hanging fruit for performance tuning
• Use Intel® VTune™ Amplifier for system and application profiling
• Hotspots, HPC Characterization, and General Exploration are good starting points
• Performance tuning is an iterative process
Agenda

- Motivation
- Threading Advisor
  - Threading Advisor Workflow
  - Advisor Interface
  - Survey Report
  - Annotations
  - Suitability Analysis
  - Dependencies Analysis
- Vectorization Advisor & Roofline
  - Vectorization Advisor recap
  - Roofline
  - Memory Access Patterns Analysis
  - Dependencies Analysis
- Summary
Vectorize & Thread or Performance Dies
Threaded + Vectorized can be much faster than either one alone

"Automatic" Vectorization Not Enough
Explicit pragmas and optimization often required

The Difference Is Growing
With Each New Generation of Hardware

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.


Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
1. Survey Target
   - Collect

1.1 Find Trip Counts and FLOPS
   - Collect
   - Trip Counts
   - FLOPS

2. Annotate Sources
   Add Intel Advisor annotations to identify possible parallel tasks and their enclosing parallel sites.
   - Steps to annotate

3. Check Suitability
   - Collect

4. Check Dependencies
   - Collect

Re-finalize Survey
Serial Modeling Has Multiple Benefits

Intel® Advisor

1) Your application can’t fail due to bugs caused by incorrect parallel execution. (It’s running serially.)

2) You can easily experiment with several different proposals before committing to the expense of implementation.
   a) Measure performance - focus on where it will pay off.
   b) Predict scalability, load balancing and overheads.
   c) Predict (and avoid) data races

3) All of your test suites should still pass. Validate the correctness of your transformations.

4) You can use Advisor on partially or completely parallelized code.

Design, measure and test **before** implementation
Threading Advisor Workflow

• Use the **Survey** to find good potential threading sites.
  • Optionally, follow up with **Trip Counts** to find information about iteration and call counts.

• **Annotate your code.**

• Use **Suitability** to predict how much performance improvement the proposed threading model will create under specific, editable conditions.

• Use **Dependencies** to determine whether the proposed model is safe, and what needs to be done to correct it.
Survey Report
Threading Advisor

Tip:
Survey sorts by Self Time by default. This is good for Vector Advisor, but for Threading Advisor, you may want to sort by Total Time.

- The Survey Report has lots of information, but most of it is more relevant to Vector Advisor.
- Look for outer loops or functions with high Total Time.
- In this example, setQueen has a high Total Time. It’s recursive, but is originally called from a loop in Solve. That makes the loop in Solve a good potential candidate.
Annotating Your Code

- Annotations are notes to Advisor. They are not parallelization commands. They do not affect the way the program itself runs.
- They mark places Advisor should treat as locks or parallel sites.
- To use annotations, you must include the appropriate header/module.

<table>
<thead>
<tr>
<th>C/C++</th>
<th>FORTRAN</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>In source files where annotations are used, add: <code>#include &lt;advisor-annotate.h&gt;</code></td>
<td>In source files where annotations are used, add: <code>use advisor_annotate</code></td>
<td>In source files where annotations are used, add: <code>using AdvisorAnnotate;</code></td>
</tr>
<tr>
<td>Add <code>&lt;install_dir&gt;/include to your include directories.</code></td>
<td>Add <code>&lt;install_dir&gt;/include to your include directories.</code></td>
<td>Add the C# annotations definition file to your project.</td>
</tr>
</tbody>
</table>

- The Advisor User's Guide contains a section on Annotations with full documentation, examples, and instructions on the above if you forget.
Suitability Analysis

- Using your annotations, Advisor models how the program would behave in parallel, and predicts performance in specified hypothetical circumstances.

See how things would change if you altered the duration and/or number of iterations/tasks.

Indicate how many CPUs, what kind of system, and what threading model to make predictions on.

Select a site to view site-specific info in the bottom pane.

Calculate on the assumption you’re using framework constructs that address these issues.
Dependencies Analysis

Threading Advisor

• This is the same analysis as in Vectorization Advisor. It works with annotations as well as selections in the survey report.

• Add lock annotations or reorganize code to resolve reported dependencies, then re-run the analysis to confirm the problem has been resolved.

• Run suitability again to check that you still get good improvement.

• Once you’re happy with Advisor’s predictions, replace the annotations with actual parallelism and locks.
Add Parallel Framework

Here is the list of source locations

Here are templates for popular parallel frameworks

Intel® Advisor
- Contains overhead metrics for popular parallel frameworks
- Quickly prototype and evaluate alternatives
- Detailed help pages for popular parallel frameworks

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Intel® Advisor Workflow

Vectorization

Select loops with potential vector dependencies
Dependences
Force vectorization if appropriate

Select loops with poor vector efficiency
Memory Access Patterns

Improve vectorization

Threading

Annotate Source and rebuild
Suitability
Select best threading candidate
Dependencies
Correct dependencies
Implement Threading

Build in Release
Survey
Trip Count
Examine Results
Examine Roofline

Back to Start
VECTORIZATION ADVISOR & ROOFLINE
Vectorization Advisor Workflow

- **Survey** is the bread and butter of Vectorization Advisor! All else builds on it!

- **Trip Counts** adds onto Survey and enables the **Roofline**.

- **Dependencies** determines whether it's safe to force a scalar loop to vectorize.

- **Memory Access Patterns** diagnoses vectorization inefficiency caused by poor memory striding.
Survey
Vectorization Advisor

**Function/Loop Icons**
- Scalar Function
- Vector Function
- Scalar Loop
- Vector Loop

**Tip:**
For vectorization, you generally only care about loops. Set the type dropdown to “Loops”.

Vectorizing a loop is usually best done on innermost loops. Since it effectively divides duration by vector length, you want to target loops with high self time.

Expand a vectorized loop to see it split into body, peel, and remainder (if applicable).

Advisor advises you on potential vector issues. This is often your cue to run MAP or Dependencies. Click the icon to see an explanation in the bottom pane.

The Intel Compiler embeds extra information that Advisor can report in addition to its sampled data, such as why loops failed to vectorize.

Efficiency is important!
Efficiency = 100%

The black arrow is 1x. Gray means you got less than that. Gold means you got more. You want to get this value as high as possible!

Survey Vectorization Advisor

Function/Loop Icons
Scalar Function
Vector Function
Scalar Loop
Vector Loop

Tip:
For vectorization, you generally only care about loops. Set the type dropdown to “Loops”.

Vectorizing a loop is usually best done on innermost loops. Since it effectively divides duration by vector length, you want to target loops with high self time.

Expand a vectorized loop to see it split into body, peel, and remainder (if applicable).

Advisor advises you on potential vector issues. This is often your cue to run MAP or Dependencies. Click the icon to see an explanation in the bottom pane.

The Intel Compiler embeds extra information that Advisor can report in addition to its sampled data, such as why loops failed to vectorize.

Efficiency is important!
Efficiency = 100%

The black arrow is 1x. Gray means you got less than that. Gold means you got more. You want to get this value as high as possible!
What is a Roofline Chart?

A Roofline Chart plots application performance against hardware limitations.

- Where are the bottlenecks?
- How much performance is being left on the table?
- Which bottlenecks can be addressed, and which should be addressed?
- What’s the most likely cause?
- What are the next steps?

Roofline first proposed by University of California at Berkeley: 

Cache-aware variant proposed by University of Lisbon: 
*Cache-Aware Roofline Model: Upgrading the Loft*, 2013
Roofline Metrics

Roofline is based on Arithmetic Intensity (AI) and FLOPS.

- **Arithmetic Intensity**: FLOP / Byte Accessed
  - This is a characteristic of your algorithm

- **FLOPS**: Floating-Point Operations / Second
  - Is a measure of an implementation (it achieves a certain FLOPS)
  - *And* there is a maximum that a platform can provide
Cache-Aware Roofline Concept

• Prior to collecting data, Advisor runs quick benchmarks to measure hardware limitations.
  • Computational limitations
  • Memory Bandwidth limitations
• These form the performance “roofs”.
• Loops and functions have algorithms and therefore a specific AI.
• Their performance in FLOPS is also measured.
• Optimization changes performance. The goal is to go as far up as possible.

Video Available: Roofline Analysis in Intel® Advisor 2017

Roofline first proposed by University of California at Berkeley:
Rooffline: An Insightful Visual Performance Model for Multicore Architectures, 2009
Cache aware variant proposed by Technical University of Lisbon:
Cache-Aware Roofline Model: Upgrading the Loft, 2013
Cache-Aware Roofline

Next Steps

If under or near a memory roof...

- Try a MAP analysis. Make any appropriate cache optimizations.
- If cache optimization is impossible, try reworking the algorithm to have a higher AI.

If just above the Scalar Add Peak

Check vectorization efficiency in the Survey. Follow the recommendations to improve it if it’s low.

If under the Scalar Add Peak...

Check the Survey Report to see if the loop vectorized. If not, try to get it to vectorize if possible. This may involve running Dependencies to see if it’s safe to force it.

If under the Vector Add Peak

Check “Traits” in the Survey to see if FMAs are used. If not, try altering your code or compiler flags to induce FMA usage.
Memory Access Patterns Analysis
Collecting a MAP

• If you have low vector efficiency, or see that a loop did not vectorize because it was deemed “possible but inefficient”, you may want to run a MAP analysis.

• Advisor will also recommend a MAP analysis if it detects a possible inefficient access pattern.

• Memory access patterns affect vectorization efficiency because they affect how data is loaded into and stored from the vector registers.

• Select the loops you want to run the MAP on using the checkboxes. It may be helpful to reduce the problem size, as MAP only needs to detect patterns, and has high overhead.
  • Note that if changing the problem size requires recompiling, you will need to re-collect the survey before running MAP.
Memory Access Patterns Analysis

Reading a MAP

• MAP is color coded by stride type. From best to worst:
  • **Blue** is unit/uniform (stepping by 1 or 0)
  • **Yellow** is constant (stepping a set distance)
  • **Red** is variable (a changing step distance)

• Click a loop in the top pane to see a detailed report below.
  • The strides that contribute to the loop are broken down in this table.
  • Important information includes the size of the stride, the variable being accessed, and the source.
  • Not all strides will come from your code!
Dependencies Analysis

Vectorization Advisor

• Generally, you don't need to run Dependencies analysis unless Advisor tells you to. It produces recommendations to do so if it detects:
  
  • Loops that remained unvectorized because the compiler was playing it safe with autovectorization.
  
  • Outer loop vectorization opportunities
  
  • Use the survey checkboxes to select which loops to analyze.
  
  • If no dependencies are found, it's safe to force vectorization.
  
  • Otherwise, use the reported variable read/write information to see if you can rework the code to eliminate the dependency.
Summary

**Survey** – Find the most promising sites for threading, see the meat of the vectorization information, and get recommendations from Advisor.

**Trip Counts & FLOPS** – Add to your Survey report to help fine-tune vector efficiency and capability, as well as unlock the powerful **Roofline** to visualize your bottlenecks and help direct your efforts.

**Suitability** – Predict how well your proposed threading model will scale under certain conditions quickly and easily.

**Dependencies** – Prove or disprove the existence of parallel dependencies and learn how to fix them.

**Memory Access Patterns** – See how you traverse your data and how it affects your vector efficiency and cache bandwidth usage.
## Configurations for 2010-2017 Benchmarks

<table>
<thead>
<tr>
<th>Platform</th>
<th>Unscaled Core Frequency</th>
<th>Cores/Socket</th>
<th>Num Sockets</th>
<th>L1 Data Cache</th>
<th>L2 Cache</th>
<th>L3 Cache</th>
<th>Memory</th>
<th>Memory Frequency</th>
<th>Memory Access</th>
<th>H/W Prefetchers Enabled</th>
<th>HT Enabled</th>
<th>Turbo Enabled</th>
<th>C States</th>
<th>O/S Name</th>
<th>Operating System</th>
<th>Compiler Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSM</td>
<td>Intel® Xeon® X5680 Processor</td>
<td>3.33 GHZ</td>
<td>6</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>12 MB</td>
<td>48 MB</td>
<td>1333 MHz NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Enabled</td>
<td>Fedora 20</td>
<td>3.11.10-301.fc20</td>
<td></td>
</tr>
<tr>
<td>SNB</td>
<td>Intel® Xeon® E5 2690 Processor</td>
<td>2.9 GHZ</td>
<td>8</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>20 MB</td>
<td>64 GB</td>
<td>1600 MHz NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Enabled</td>
<td>RHEL 7.1</td>
<td>3.10.0-229.el7.x86_64</td>
<td></td>
</tr>
<tr>
<td>IVB</td>
<td>Intel® Xeon® E5 2697v2 Processor</td>
<td>2.7 GHZ</td>
<td>12</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>30 MB</td>
<td>64 GB</td>
<td>1867 MHz NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Enabled</td>
<td>Fedora 20</td>
<td>3.15.10-200.fc20.x86_64</td>
<td></td>
</tr>
<tr>
<td>HSW</td>
<td>Intel® Xeon® E5 2600v3 Processor</td>
<td>2.2 GHZ</td>
<td>18</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>46 MB</td>
<td>128 GB</td>
<td>2133 MHz NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>RHEL 7.0</td>
<td>3.10.0-123.el7.x86_64</td>
<td></td>
</tr>
<tr>
<td>BDW</td>
<td>Intel® Xeon® E5 2600v4 Processor</td>
<td>2.3 GHZ</td>
<td>18</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>46 MB</td>
<td>256 GB</td>
<td>2400 MHz NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>CentOS 7.2</td>
<td>3.10.0-327.el7.x86_64</td>
<td></td>
</tr>
<tr>
<td>BDW</td>
<td>Intel® Xeon® E5 2600v4 Processor</td>
<td>2.2 GHZ</td>
<td>22</td>
<td>2</td>
<td>32K</td>
<td>256K</td>
<td>56 MB</td>
<td>128 GB</td>
<td>2133 MHz NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>CentOS 7.3</td>
<td>3.10.0-514.10.2.el7.x86_64</td>
<td></td>
</tr>
<tr>
<td>SKX</td>
<td>Intel® Xeon® Platinum 81xx Processor</td>
<td>2.5 GHZ</td>
<td>28</td>
<td>2</td>
<td>32K</td>
<td>1024K</td>
<td>40 MB</td>
<td>192 GB</td>
<td>2666 MHz NUMA</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Disabled</td>
<td>CentOS 7.3</td>
<td>3.10.0-514.10.2.el7.x86_64</td>
<td></td>
</tr>
</tbody>
</table>

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. For more information go to [http://www.intel.com/performance](http://www.intel.com/performance).
Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSSmark 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. For more complete information visit www.intel.com/benchmarks.

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.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Xeon, Xeon Phi, Core, VTune, 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
**Legal Disclaimer & Optimization Notice**

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. For more complete information visit www.intel.com/benchmarks.

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.

Copyright © 2018, 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