INTEL® VTUNE™ AMPLIFIER

Jackson Marusarz - Intel
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>

## Optimization Notice

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

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>Basic Hotspots</strong></td>
<td>Advanced Hotspots</td>
</tr>
<tr>
<td>Which functions use the most time?</td>
<td>Which functions use the most time?</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td>General Exploration</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><strong>Locks and Waits</strong></td>
<td>Advanced Analysis</td>
</tr>
<tr>
<td>Tune the #1 cause of slow threaded performance:</td>
<td>Memory-access, HPC Characterization, etc…</td>
</tr>
<tr>
<td>– waiting with idle cores.</td>
<td></td>
</tr>
</tbody>
</table>

Any IA86 processor, any VM, no driver

Higher res., lower overhead, system wide

**No special recompiles - C, C++, C#, Fortran, Java, Assembly**

---

*Other names and brands may be claimed as the property of others.*
Example: Hotspots Analysis

Summary View

**Elapsed Time**: 5.554s
- CPU Time: 10.504s
- Instructions Retired: 21,698,000,000
- CPI Rate: 1.257
- CPU Frequency Ratio: 1.041
- Total Thread Count: 9
- Paused 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</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid_interact</td>
<td>3_techycoomp.exe</td>
<td>5.539s</td>
</tr>
<tr>
<td>sphere_interact</td>
<td>3_techycoomp.exe</td>
<td>3.247s</td>
</tr>
<tr>
<td>func@0x1002a59d</td>
<td>libomp5amd.dll</td>
<td>0.148s</td>
</tr>
<tr>
<td>shader</td>
<td>3_techycoomp.exe</td>
<td>0.117s</td>
</tr>
<tr>
<td>KdDelayExpirationThread</td>
<td>ntok.nlm.exe</td>
<td>0.091s</td>
</tr>
<tr>
<td>Others</td>
<td>N/A*</td>
<td>1.581s</td>
</tr>
</tbody>
</table>

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

**Average Bandwidth**

<table>
<thead>
<tr>
<th>Package</th>
<th>Total GB/s</th>
<th>Read GB/s</th>
<th>Write GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_0</td>
<td>5.715</td>
<td>3.504</td>
<td>2.212</td>
</tr>
</tbody>
</table>

**CPU Usage Histogram**
This histogram displays the 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: Concurrency 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

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier

Transitions
Locks & Waits

CPU Time
Basic Hotspots

Advanced Hotspots

Optional: Use API to mark frames and user tasks
Optional: Add a mark during collection
Command Line Interface

Automate analysis

amplxe-cl is the command line:

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

Design
- Think performance wise (app/sys level)

Prototyping
- Choose performance effective solutions

Implementation
- Apply performance optimization and check results

Testing
- Add performance regressions to test stage

Release
- Collect and analyze performance related issues from users
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

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

**General Exploration**
- General Exploration viewpoint (change)
- Collection Log
- Analysis Target
- Analysis Type
- Summary
- Bottom-up

**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 Clears: 0.1% of Pipeline Slots
- Back-End Bound: 64.1%
- Memory Bound: 33.3%
- Core Bound: 30.6%
- Divider: 0.0%
- Port Utilization: 29.6%
- Rating: 22.7%
- Total Thread Count: 9
- Paused Time: 0s

**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
System-Level Profiling – Disk I/O Analysis

Are You I/O Bound or CPU Bound?
• Explore imbalance between I/O opera (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 disk-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
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 Parallelism Issues**
3. **Address Hardware Issues**
4. **Determine Efficiency**
5. **Rebuild and Compare Results**

The process is cyclical, allowing for iterative improvement and optimization.
Find Hotspots

> amplxe-cl -collect basic-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.
Address Hardware Issues

The X86 Processor Pipeline (simplified)
Address Hardware Issues

For each pipeline slot on each cycle:

1. **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

General Exploration Analysis Shows the Hardware Bottleneck in the Application

> amplxe-cl -collect general-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 = 0.616s

CPU Usage Histogram:
This histogram displays a percentage of the wall time the specific number of CPUs were running.
Summary

• 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
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