Intel® Xeon Phi™ Coprocessor Performance Analysis
Legal Disclaimer

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards 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.
- Sandy Bridge and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.
- 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
- Intel, Core, Xeon, VTune, Cilk, Intel and Intel Sponsors of Tomorrow. and Intel Sponsors of Tomorrow. logo, and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- Copyright ©2011 Intel Corporation.
- Hyper-Threading Technology: Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on all Intel® Core™ processors. For more information including details on which processors support HT Technology, visit http://www.intel.com/info/hyperthreading
- Intel® 64 architecture: Requires a system with a 64-bit enabled processor, chipset, BIOS and software. Performance will vary depending on the specific hardware and software you use. Consult your PC manufacturer for more information. For more information, visit http://www.intel.com/info/em64t
- Intel® Turbo Boost Technology: Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit http://www.intel.com/technology/turboboost
Module Goals

• Topics Covered
  – Basic use of Intel® VTune™ Amplifier XE and other tools
  – Identifying code that are good candidates for offload
  – How to debug correctness or performance issues in parallel code intended for the Intel® MIC Architecture

• Goals
  – You can describe two techniques for finding parts of your code that might be a good target for the Intel MIC Architecture
Performance Analysis Topics

• Performance Analysis: Procedure

• VTune™ Analyzer
  – Features
  – Application Configuration
  – Performance Analysis

• Performance Counters / Events
  – Useful events and metrics
  – Cache metrics
  – Prefetcher metrics

• Performance Analysis: Triage
Module Outline

• Performance Analysis: Procedure
  • VTune™ Amplifier XE
  • Performance Counters / Events
  • Performance Analysis: Triage
Performance Analysis Procedure

This is just one possible technique
- Use what works best for you

1. Use VTune™ Amplifier XE to gather hotspot data
   - Alternately, use the profiling capabilities of the Intel® Compiler
2. Use Intel® Inspector XE on your code (on the host) with `offload disabled`
3. Use VTune Amplifier XE’s parallel performance analysis tools to find issues (on the host) by running your program with `offload disabled`
4. Analyze event data using VTune Amplifier XE
5. Perform initial analysis / triage procedure
Use **host-based** profiling to identify vectorization/parallelism/offload candidates

- Start with representative/reasonable workloads!
- Use VTune™ Amplifier XE to gather hotspot data
  - Tells you what functions account for most of the runtime
  - This is often enough
    - But it does not tell you much about program structure
- Alternately, use the profiling capabilities of the Intel® Compiler
  - Build with `-profile-functions -profile-loops=all -profile-loops-report=2 options`
  - Run your code
  - Look at the resulting dump files, or open the xml file with the data viewer `loopprofileviewer.sh` located in the compiler’s `/bin` directory
  - Tells you which loops and functions account for the most runtime, and how many times each loop executes
Correctness/Performance Analysis of Parallel code

- Intel® Inspector XE and the thread-checking features of VTune™ Amplifier XE are not available for the Intel® MIC Architecture
- So...
  - Use Intel Inspector XE on your code with offload disabled to identify correctness errors (deadlocks, races) on the host system
    - Once those are fixed, then enable offload and continue debugging on the card
  - Use VTune Amplifier XE’s parallel performance analysis tools to find issues on the host by running your program with offload disabled
    - Fix everything you can
    - Now conduct scaling studies on the card and use what you learned on the host to further optimize parallel performance
      - Be wary of any sort of synchronization when the number of threads becomes more than a handful
      - Also pay attention to load-balancing.
Module Outline

• Performance Analysis: Procedure
• VTune™ Amplifier XE
• Performance Counters / Events
• Performance Analysis: Triage
VTune™ Amplifier XE Video Tutorials

• Tutorial 1: Collecting Hotspots for Native Applications
• Tutorial 2: Collecting Hotspots for Offload Applications
• Tutorial 3: Running the General Exploration Analysis
• Tutorial 4: Running Memory Bandwidth and Custom Analysis
• Tutorial 5: Using the command line
VTune™ Amplifier XE

Menu and Toolbars
- Analysis Type
- Viewpoint currently being used
- Tabs within each result
- Grid area
- Stack Pane
- Filter area
- Timeline area

Current grouping

Analysis Type
- Function
- Call Stack

Viewpoint
- Summary
- Bottom-up
- Top-down Tree

Timeline area
- Selected 1 row(s)
- Grid area
- Stack Pane
- Filter area
- Timeline area

Analysis Type
- Function
- Call Stack

Viewpoint
- Summary
- Bottom-up
- Top-down Tree

Timeline area
- Selected 1 row(s)
- Grid area
- Stack Pane
- Filter area
- Timeline area

Analysis Type
- Function
- Call Stack

Viewpoint
- Summary
- Bottom-up
- Top-down Tree

Timeline area
- Selected 1 row(s)
- Grid area
- Stack Pane
- Filter area
- Timeline area
VTune™ Amplifier XE

Adjust Data Grouping
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack

... (Partial list shown)

Click [+ ] for Call Stack

Double Click Function to View Source

Filter by Timeline Selection (or by Grid Selection)
- Zoom In And Filter On Selection
- Filter In by Selection
- Remove All Filters

Filter by Module & Other Controls

Hotspots - View hotspots colored by CPU usage

CPU Time

Module

Function - Call Stack

Module

Filter by Timeline Selection (or by Grid Selection)

Optimization Notice
VTune™ Amplifier XE

- Time on Source / Asm
- Quick Asm navigation: Select source to highlight Asm
- Quickly scroll to hot spots. Scroll Bar “Heat Map” is an overview of hot spots
- Right click for instruction reference manual
- Click jump to scroll Asm
VTune™ Amplifier XE features

- Event browsing features are available in the grid:

<table>
<thead>
<tr>
<th>Function</th>
<th>Hardware Event Count by Hardware...</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INSTRUCTI... by Package</td>
<td>CPU_CLK_U... by Package</td>
</tr>
<tr>
<td>diffuse3d</td>
<td>340,740,000,000</td>
<td>755,485,500,000</td>
</tr>
<tr>
<td>__kmp_wait_sleep</td>
<td>196,721,000,000</td>
<td>262,001,500,000</td>
</tr>
<tr>
<td>__kmp_execute_tasks</td>
<td>83,836,000,000</td>
<td>77,340,500,000</td>
</tr>
<tr>
<td>spinlock_exit</td>
<td>21,199,500,000</td>
<td>46,172,000,000</td>
</tr>
<tr>
<td>__kmp_yield</td>
<td>15,322,000,000</td>
<td>19,670,500,000</td>
</tr>
<tr>
<td>[userret]</td>
<td>6,514,500,000</td>
<td>706,000,000</td>
</tr>
</tbody>
</table>

- Imported symbol data allows drilling down to source - basic block assembly annotated by line number is also available

Optimization Notice
Intel® Xeon Phi™ Coprocessor Workshop
Pawsey Centre & CSIRO, Aug 2013
VTune™ Amplifier XE features

- Collector provides events and some thread info for the timeline
  - Interval thread is active
  - Black is thread activity
  - Black is total activity
  - "activity" event

- However, thread info is limited—no locks and waits or concurrency analysis—those require a different collector
Application Configuration

- Application settings:
  - Application: ssh
  - Parameters: mic0 "<app startup>"
  - Working directory: Does not matter 😊, use cd to get there

![Application Configuration Interface](image-url)
Application Configuration

Choose Analysis Type

- Access Contention
- Branch Analysis
- Client Analysis
- Core Port Saturation
- Cycles and uOps
- Loop Analysis
- Memory Access
- Port Saturation
- Intel Atom Processor Analysis
  - General Exploration
- Knights Corner Platform Analysis
  - Lightweight Hotspots
- Power Analysis
  - CPU Sleep States
  - CPU Frequency
- Custom Analysis
  - Branch Miss predicts
  - Cache Misses
  - Execution Stalls
  - False Sharing

Knights Corner Platform - Lightweight Hotspots

Identify your most time-consuming source code. Unlike Hotspots, Lightweight Hotspots has lower overhead because it does not collect stack information. It can also be used to sample all processes on a system. This analysis type uses hardware event-based sampling collection. Press F1 for more details.

List of MIC cards (e.g. 0.1.2.3): 0

Details

- Events configured for CPU: Intel(R) Xeon(R) / Core i7 980X Processor

NOTE: For analysis purposes, Intel VTune Amplifier XE 2013 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the Project Properties dialog.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Sample After</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CLK_UNHALTED</td>
<td>100000000</td>
<td>Number of cycles during which</td>
</tr>
<tr>
<td>INSTRUCTIONS_EXECUTED</td>
<td>10000000</td>
<td>Number of instructions executed</td>
</tr>
</tbody>
</table>
Performance Analysis

- VTune™ Amplifier XE at the moment only supports “Lightweight Hotspots” for Intel® Xeon Phi™ coprocessor
  - Based on Event-based Sampling, uses the Performance Monitoring Unit
  - No instrumentation ("Locks&Waits", "Concurrency", etc.)
  - More to come!

- Other analysis types need to be configured
  - Use “Lightweight Hotspots” and create a copy of it
  - Add the desired counters
  - Add some useful name and documentation
Configuring a User-defined Analysis

![Graphical interface for configuring a custom analysis in Intel VTune Amplifier XE 2013. The interface includes options for selecting events and properties for the analysis, with specific events like CPU_CLK_UNHALTED, INSTRUCTIONS_EXECUTED, and DATA_READ_MISS highlighted.](image-url)
Module Outline

- Performance Analysis: Procedure
- VTune™ Amplifier XE
- **Performance Counters / Events**
- Performance Analysis: Triage
Collecting Hardware Performance Data

• Hardware counters and events
  – 2 counters in core, most are thread specific
  – 4 in uncore, that are not thread or core specific
  – See PMU documentation for a full list of events

• Collection
  – Invoke from VTune Analyzer’s GUI (or command line interface)
  – For core events, as many runs as necessary are spawned to collect all event counts in a single database
  – For uncore events, there’s a limit of 4 events and a single run
  – Uncore event sampling needs a source of PMU interrupts, e.g. programming cores to CPU_CLK_UNHALTED

• Output files
  – VTune Analyzer’s performance database
### Some useful events and metrics

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Event name(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall-clock profiling</td>
<td><code>CPU_CLK_UNHALTED, INSTRUCTIONS_EXECUTED</code> (or <code>EXEC_STAGE_CYCLES</code>)</td>
</tr>
<tr>
<td>Main memory bandwidth</td>
<td><code>L2_DATA_READ_MISS_MEM_FILL, L2_DATA_WRITE_MISS_MEM_FILL</code></td>
</tr>
<tr>
<td>L1 Cache misses</td>
<td><code>DATA_READ_MISS_OR_WRITE_MISS</code></td>
</tr>
<tr>
<td>TLB misses and page faults</td>
<td><code>DATA_PAGE_WALK, DATA_PAGEFAULT</code></td>
</tr>
<tr>
<td>Vectorized code execution (sort of)</td>
<td><code>VPU_INSTRUCTIONS_EXECUTED</code></td>
</tr>
<tr>
<td>Various hazards</td>
<td><code>BRANCHES_MISPREDICTED</code></td>
</tr>
<tr>
<td>Cycles per instruction</td>
<td><code>CPU_CLK_UNHALTED / INSTRUCTIONS_EXECUTED</code></td>
</tr>
</tbody>
</table>
| Memory Bandwidth (used by all cores at once)  | \[
|                                              | \((L2\_DATA\_READ\_MISS\_MEM\_FILL + L2\_DATA\_WRITE\_MISS\_MEM\_FILL) \times 64\) / \(CPU\_CLK\_UNHALTED / \) Frequency \] |
### Some useful events and metrics

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Event name(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall-clock profiling</td>
<td><code>CPU_CLK_UNHALTED, INSTRUCTIONS_EXECUTED</code> <em>(or EXEC_STAGE_CYCLES)</em></td>
</tr>
<tr>
<td>Main memory bandwidth</td>
<td><code>L2_DATA_READ_MISS_MEM_FILL, L2_DATA_WRITE_MISS_MEM_FILL</code></td>
</tr>
<tr>
<td>L1 Cache misses</td>
<td><code>DATA_READ_MISS_OR_WRITE_MISS</code></td>
</tr>
<tr>
<td>TLB misses</td>
<td><code>DATA_PAGE_WALK</code></td>
</tr>
<tr>
<td>Vectorized code execution (sort of)</td>
<td><code>VPU_INSTRUCTIONS_EXECUTED</code></td>
</tr>
<tr>
<td>Various hazards</td>
<td><code>BRANCHES_MISPREDICTED</code></td>
</tr>
<tr>
<td>Cycles per instruction</td>
<td><code>CPU_CLK_UNHALTED / INSTRUCTIONS_EXECUTED</code></td>
</tr>
<tr>
<td>Memory Bandwidth (used by all cores at once)</td>
<td><code>(L2_DATA_READ_MISS_MEM_FILL + L2_DATA_WRITE_MISS_MEM_FILL) * 64 / CPU_CLK_UNHALTED / Frequency</code></td>
</tr>
</tbody>
</table>
## Interesting Cache Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Demand Reads</td>
<td>DATA_READ - L1_DATA_PF1 - L1_DATA_PF2</td>
</tr>
<tr>
<td>L1 Demand Misses</td>
<td>DATA_READ_MISS - L1_DATA_PF1_MISS - L2_DATA_PF2_MISS - L2_DATA_PF1_MISS</td>
</tr>
<tr>
<td>L1 Miss Ratio</td>
<td>L1 Demand Misses / L1 Demand Reads</td>
</tr>
<tr>
<td>L2 Prefetch Hits</td>
<td>L2_DATA_PF2 - L2_DATA_PF2_MISS</td>
</tr>
<tr>
<td>L2 Demand Reads</td>
<td>L2_READ_HIT_E + L2_READ_HIT_M + L2_READ_HIT_S) - L2 Prefetch Hits</td>
</tr>
</tbody>
</table>
Interesting Cache Metrics cont.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 Demand Misses</td>
<td>$L2_READ_MISS - L2_DATA_PF2_MISS$</td>
</tr>
<tr>
<td>L2 Miss Ratio</td>
<td>$\frac{L2_Demand\ Misses}{L2_Demand\ Reads}$</td>
</tr>
<tr>
<td>L1 Cache Hit Ratio</td>
<td>$(DATA_READ_OR_WRITE - DATA_READ_MISS_OR_WRITE_MISS) / DATA_READ_OR_WRITE$</td>
</tr>
<tr>
<td>L2 Cache Hit Ratio</td>
<td>$(DATA_READ_OR_WRITE - L2_READ_MISS) / DATA_READ_OR_WRITE$</td>
</tr>
</tbody>
</table>
## Interesting Prefetcher Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 PF Miss Ratio</td>
<td>L1_DATA_PF1_MISS / L1_DATA_PF1</td>
</tr>
<tr>
<td>L2 PF Miss Ratio</td>
<td>L2_DATA_PF2_MISS / L2_DATA_PF2</td>
</tr>
<tr>
<td>L1 PF Inflight Ratio</td>
<td>L1_DATA_HIT_INFLIGHT_PF1 / L1_DATA_PF1</td>
</tr>
<tr>
<td>L2 PF Inflight Ratio</td>
<td>L2_DATA_HIT_INFLIGHT_PF2 / L2_DATA_PF2</td>
</tr>
<tr>
<td>L1 PF Drop Ratio</td>
<td>L1_DATA_PF1_DROP / L1_DATA_PF1</td>
</tr>
<tr>
<td>L2 PF Drop Ratio</td>
<td>L2_DATA_PF1_DROP / L2_DATA_PF2</td>
</tr>
</tbody>
</table>
Module Outline

• Performance Analysis: Procedure
• VTune™ Amplifier XE
• Performance Counters / Events

• Performance Analysis: Triage
Initial Analysis/Triage Process (1)

- Start by collecting `CPU_CLK_UNHALTED`, `INSTRUCTIONS_EXECUTED`, `DATA_READ_OR_WRITE`, and `L2_READ_MISS`
- Look at the timeline in Intel® VTune™ Amplifier XE, viewing `CPU_CLK_UNHALTED`
  - Are the cores busy most of the time, or is there a lot of serial work?
- Sort the functions by `CPU_CLK_UNHALTED` and look for functions where the most CPU time was spent
Initial Analysis/Triage Process (2)

• Sort the functions by `CPU_CLK_UNHALTED` and look for functions where the most CPU time was spent
  - If `CPU_CLK_UNHALTED/INSTRUCTIONS_EXECUTED` > 1.5 for these hotspots, something is stalling execution
    o Dig into the hotspot and find/understand the slow source lines – beware skid
    o Check for large numbers of `DATA_READ_OR_WRITE` and `L2_READ_MISS` events to see if memory access is the cause
    o Collecting data with `DATA_PAGE_WALK`, `DATA_PAGE_FAULT`, and `BRANCHES_MISPREDICTED` might also explain this hotspot
  - If the ratio is less than 1.5, look to the algorithm – the processor is efficiently spending time where you told it to
    o But you might want to check `VPU_INSTRUCTIONS_EXECUTED`
  - Do not worry about functions where very little CPU time is spent
Performance Analysis Summary

• Performance Analysis: Procedure
• VTune™ Analyzer
  – Features
  – Application Configuration
  – Performance Analysis
• Performance Counters / Events
  – Useful events and metrics
  – Cache metrics
  – Prefetcher metrics
• Performance Analysis: Triage
Labs

- Performance Analysis Lab (Lesson 9)
Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101
Backup