Performance Tuning for Intel® Xeon Phi™ Coprocessors

Robert Reed
Intel Technical Consulting Engineer
Agenda

Start tuning on host

Overview of Intel® VTune™ Amplifier XE

Efficiency metrics

Problem areas
Performance Analysis Methodology
Optimization: A Top-down Approach

• Use top down approach
• Understand application and system characteristics
  – Use appropriate tools at each level

- System Config, BIOS, OS, Network I/O, Disk I/O, Database Tuning, etc.
- Application Design, Algorithmic Tuning, Driver Tuning, Parallelization
- Cache/Memory Instructions, SIMD others
Performance Analysis Methodology
Optimization: A Top-down Approach

- Use top down approach
- Understand application and system characteristics
  - Use appropriate tools at each level

VTune™ Amplifier XE can help here
Start with **host-based profiling** to identify vectorization/parallelism/offload candidates

Start with representative/reasonable workloads!

Use Intel® VTune™ Amplifier XE to gather hot spot data

- Tells what functions account for most of the run time
- Often, this is enough
  - But it does not tell you much about program structure
Start with **host-based profiling** to identify vectorization/parallelism/offload candidates

Start with representative/reasonable workloads!

Use Intel® VTune™ Amplifier XE to gather hot spot data

- Tells what functions account for most of the run time
- Often, this is enough
  - But it does not tell you much about program structure

**Alternately, profile functions & loops using Intel® Composer XE**

- Build with options
  - `-profile-functions -profile-loops=all -profile-loops-report=2`
- Run the code (which may run slower) to collect profile data
- Look at the resulting `dump` files, or open the `xml` file with the data viewer `loopprofileviewer.sh` located in the compiler `./bin` directory

- Tells you which loops and functions account for the most run time
  - how many times each loop executes (min, max and average)
Correctness/Performance Analysis of Parallel code

Intel® Inspector XE and thread-reports in VTune™ Amplifier XE are not available on the Intel® Xeon Phi™ coprocessor

So...
Correctness/Performance Analysis of Parallel code

Intel® Inspector XE and thread-reports in VTune™ Amplifier XE are not available on the Intel® Xeon Phi™ coprocessor

So...

• Use Intel Inspector XE on your code with **offload disabled** (on host) to identify correctness errors (e.g., deadlocks, races)
  - Once fixed, then enable offload and continue debugging on the coprocessor
Correctness/Performance Analysis of Parallel code

Intel® Inspector XE and thread-reports in VTune™ Amplifier XE are not available on the Intel® Xeon Phi™ coprocessor

So...

• Use Intel Inspector XE on your code with **offload disabled** (on host) to identify correctness errors (e.g., deadlocks, races)
  - Once fixed, then enable offload and continue debugging on the coprocessor

• 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
  - Then study scaling on the coprocessor using lessons from host tuning to further optimize parallel performance
    - Be wary of synchronization across more than a handful of threads
    - Pay attention to load balance.
Agenda

Start tuning on host

Overview of Intel® VTune™ Amplifier XE

Efficiency metrics

Problem areas
Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

- Fast, Accurate Performance Profiles
  - Hotspot (Statistical call tree)
  - Hardware-Event Based Sampling
- Thread Profiling
  - Visualize thread interactions on timeline
  - Balance workloads
- Easy set-up
  - Pre-defined performance profiles
  - Use a normal production build
- Compatible
  - Microsoft*, GCC*, Intel compilers
  - C/C++, Fortran, Assembly, .NET*
  - Latest Intel processors and compatible processors¹
- Find Answers Fast
  - Filter out extraneous data
  - View results tied to source/assembly lines
  - Event multiplexing
- Windows* or Linux*
  - Visual Studio* Integration (Windows)
  - Standalone user interface and command line
  - 32 and 64-bit

¹ IA-32 and Intel® 64 architectures. Many features work with compatible processors. Event based sampling requires a genuine Intel Processor.
VTune™ Amplifier XE visualizes performance
VTune™ Amplifier XE visualizes performance
VTune™ Amplifier XE visualizes performance

Project Navigator
VTune<sup>™</sup> Amplifier XE visualizes performance

Result Display Tabs
VTune™ Amplifier XE visualizes performance

**Result Analysis Type**
VTune™ Amplifier XE visualizes performance

Result Viewpoint
VTune™ Amplifier XE visualizes performance
VTune™ Amplifier XE visualizes performance

Result Components
VTune™ Amplifier XE visualizes performance

Grid Pane
VTune™ Amplifier XE visualizes performance
VTune™ Amplifier XE visualizes performance
VTune™ Amplifier XE visualizes performance
VTune™ Amplifier XE visualizes performance

Filter/Options Bar

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time by Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
</tr>
<tr>
<td>multiply3$omp$parallel</td>
<td>11.238s</td>
</tr>
<tr>
<td>[OpenMP worker]</td>
<td>2.872s</td>
</tr>
<tr>
<td>[OpenMP fork]</td>
<td>0.150s</td>
</tr>
<tr>
<td>int_ar</td>
<td>0.010s</td>
</tr>
<tr>
<td>int_ar</td>
<td>0.010s</td>
</tr>
</tbody>
</table>

Selected 1 row(s): 11.238s
VTune™ Amplifier XE visualizes performance

Source View / Per line localization

```
for (k = k0; k < k0 + mblock; k++) {
  #pragma unroll(8)
  #pragma ivdep

  for (j = j0; j < j0 + mblock; j++) {
    c[i][j] = c[i][j] + a[i][k] + b[k][j];
  }
}

# ifdef defined (USE_OMP) // }
  #pragma omp parallel for collapse (2)
for (i=0; i<msize; i++) {
  for (k=0; k<msize; k++) {
    #pragma unroll(8)
    #pragma ivdep

    for (j=0; j<msize; j++) {
      c[i][j] = c[i][j] + a[i][k] + b[k][j];
    }
  }
  #endif // }
```

Selected 1 row(s): 528.349s
VTune™ Amplifier XE visualizes performance

Source View / View / Hot spot Navigation controls
VTune™ Amplifier XE visualizes performance

Assembly View / View / Hot spot Navigation controls
VTune™ Amplifier XE visualizes performance

Assembly View / Assembly groupings
For event collection the coprocessor is treated as a special HW architecture
Project properties provides the means to invoke data collection by target type.
Launch Application serves many uses, from host/offload to native execution.
Search directories have been reorganized to speed symbol resolution during finalization

- Enumerate source directories under this tab

Notable coprocessor library paths:

- /lib/firmware/mic
- /usr/linux-k1om-4.7/linux-k1om/lib64
- /opt/intel/composerxe/lib/mic
- /opt/intel/composerxe/tbb/lib/mic
- /opt/intel/composerxe/mkl/lib/mic
- /opt/intel/mpi-rt/4.1.0/mic
General Exploration runs a set of events to drive top-down analysis.
Agenda

Start tuning on host

Overview of Intel® VTune™ Amplifier XE

Efficiency metrics

Problem areas
Cycles Per Instruction (CPI), a standard measure, has some special kinks

- Threads on each Intel® Xeon™ Phi core share a clock
- If all 4 HW threads are active, each gets $\frac{1}{4}$ total cycles
- Multi-stage instruction decode requires two threads to utilize the whole core – one thread only gets half
- With two ops/per cycle (U-V-pipe dual issue):

<table>
<thead>
<tr>
<th>Threads per Core</th>
<th>Best CPI per Core</th>
<th>Best CPI per Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x</td>
<td>1.0</td>
<td>= 1.0</td>
</tr>
<tr>
<td>2 x</td>
<td>0.5</td>
<td>= 1.0</td>
</tr>
<tr>
<td>3 x</td>
<td>0.5</td>
<td>= 1.5</td>
</tr>
<tr>
<td>4 x</td>
<td>0.5</td>
<td>= 2.0</td>
</tr>
</tbody>
</table>

- To get thread CPI, multiply by the active threads
As an efficiency metric, CPI must be considered carefully: it IS a ratio

- Changes in CPI absent major code changes can indicate general latency gains/losses

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI per Thread</td>
<td>CPU_CLK_UNHALTED/INSTRUCTIONS_EXECUTED</td>
<td>&gt; 4.0, or increasing</td>
</tr>
<tr>
<td>CPI per Core</td>
<td>(CPI per Thread) / Number of hardware threads used</td>
<td>&gt; 1.0, or increasing</td>
</tr>
</tbody>
</table>

- Note the effect on CPI from applied optimizations
- Reduce high CPI through optimizations that target latency
  - Better prefetch
  - Increase data reuse through better blocking
Two more examples why absolute CPI value is less important than changes

- Scaling data from a typical lab workload:

<table>
<thead>
<tr>
<th>Metric</th>
<th>1 hardware thread / core</th>
<th>2 hardware threads / core</th>
<th>3 hardware threads / core</th>
<th>4 hardware threads / core</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI per Thread</td>
<td>5.24</td>
<td>8.80</td>
<td>11.18</td>
<td>13.74</td>
</tr>
<tr>
<td>CPI per Core</td>
<td>5.24</td>
<td>4.40</td>
<td>3.73</td>
<td>3.43</td>
</tr>
</tbody>
</table>

- Observed CPIs from several tuned workloads:
Efficiency Metric: Compute to Data Access Ratio

- Measures an application’s computational density, and suitability for Intel® Xeon Phi™ coprocessors

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorization Intensity</td>
<td>VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED</td>
<td></td>
</tr>
<tr>
<td>L1 Compute to Data Access Ratio</td>
<td>VPU_ELEMENTS_ACTIVE / DATA_READ_OR_WRITE</td>
<td>&lt; Vectorization Intensity</td>
</tr>
<tr>
<td>L2 Compute to Data Access Ratio</td>
<td>VPU_ELEMENTS_ACTIVE / DATA_READ_MISS_OR_WRITE_MISS</td>
<td>&lt; 100x L1 Compute to Data Access Ratio</td>
</tr>
</tbody>
</table>

- Increase computational density through vectorization and reducing data access (see cache issues, also, DATA ALIGNMENT!)
Agenda

Start tuning on host

Overview of Intel® VTune™ Amplifier XE

Efficiency metrics

Problem areas*

*tuning suggestions requiring deeper understanding of architectural tradeoffs and application data handling details are highlighted with this “ninja” notation
Problem Area: L1 Cache Usage

• Significantly affects data access latency and therefore application performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Misses</td>
<td>DATA_READ_MISS_OR_WRITE_MISS + L1_DATA_HIT_INFLIGHT_PF1</td>
<td></td>
</tr>
<tr>
<td>L1 Hit Rate</td>
<td>(DATA_READ_OR_WRITE - L1 Misses) / DATA_READ_OR_WRITE</td>
<td>&lt; 95%</td>
</tr>
</tbody>
</table>

• Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores

  ![Image](image.png)

  If using 4K access stride, may be experiencing conflict misses

  ![Image](image.png)

  Examine Compiler prefetching (Compiler-generated L1 prefetches should not miss)
Problem Area: Data Access Latency

- Significantly affects application performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated Latency Impact</td>
<td>(CPU_CLK_UNHALTED - EXEC_STAGE_CYCLES - DATA_READ_OR_WRITE) / DATA_READ_OR_WRITE_MISS</td>
<td>&gt;145</td>
</tr>
</tbody>
</table>

- Tuning Suggestions:
  - Software prefetching
  - Tile/block data access for cache size
  - Use streaming stores

Check cache locality – turn off prefetching and use CACHE_FILL events - reduce sharing if needed/possible

If using 64K access stride, may be experiencing conflict misses
**Problem Area: TLB Usage**

- Also affects data access latency and therefore application performance

### Metric | Formula | Investigate if
---|---|---
L1 TLB miss ratio | DATA\_PAGE\_WALK/\(DATA\_READ\_OR\_WRITE\) | > 1%
L2 TLB miss ratio | \(LONG\_DATA\_PAGE\_WALK\) / DATA\_READ\_OR\_WRITE | > .1%
L1 TLB misses per L2 TLB miss | DATA\_PAGE\_WALK / LONG\_DATA\_PAGE\_WALK | > 100x

**Tuning Suggestions:**

- Improve cache usage & data access latency
- If L1 TLB miss/L2 TLB miss is high, try using large pages
- For loops with multiple streams, try splitting into multiple loops
- If data access stride is a large power of 2, consider padding between arrays by one 4 KB page
### Problem Area: VPU Usage

- Indicates whether an application is vectorized successfully and efficiently

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorization Intensity</td>
<td>$\frac{\text{VPU_ELEMENTS_ACTIVE}}{\text{VPU_INSTRUCTIONS_EXECUTED}}$</td>
<td>$&lt;8$ (DP), $&lt;16$ (SP)</td>
</tr>
</tbody>
</table>

- **Tuning Suggestions:**
  - Use the Compiler vectorization report!
  - For data dependencies preventing vectorization, try using Intel® Cilk™ Plus `#pragma SIMD` (if safe!)
  - Align data and tell the Compiler!
  - Restructure code if possible: Array notations, AOS->SOA
Problem Area: Memory Bandwidth

- Can increase data latency in the system or become a performance bottleneck

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Investigate if</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth</td>
<td>(UNC_F_CH0_NORMAL_READ + UNC_F_CH0_NORMAL_WRITE + UNC_F_CH1_NORMAL_READ + UNC_F_CH1_NORMAL_WRITE) X 64/time</td>
<td>&lt; 80GB/sec (practical peak 140GB/sec) (with 8 memory controllers)</td>
</tr>
</tbody>
</table>

- Tuning Suggestions:
  - Improve locality in caches
  - Use streaming stores
  - Improve software prefetching
Final caution: coprocessor collections can generate dense volumes of data

Example: DGEMM on 60+ cores

Tip: Use a CPU Mask to reduce data volume while maintaining equivalent accuracy.
Summary

• Vectorization, Parallelism, and Data locality are critical to good performance for the Intel® Xeon Phi™ Coprocessor

• Event names can be misleading – we recommend using the metrics given in this presentation or our tuning guide at http://software.intel.com/en-us/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coprocessors-part-2-understanding

• Intel® VTune™ Amplifier XE supports collecting all of the above metrics, as well as providing special analysis types like General Exploration and Memory Bandwidth
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR
IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY
PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on
Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using
specific computer systems, components, software, operations and functions. Any change to any of
those factors may cause the results to vary. You should consult other information and performance
tests to assist you in fully evaluating your contemplated purchases, including the performance of that
product when combined with other products.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk 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