Intel® Xeon Phi™ Coprocessor

Optimization for the Intel® Xeon Phi™ Coprocessor
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 PROPERTY 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.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

- 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, Phi and 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 ©2012 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

Performance Notice
Optimization for the Intel® Xeon Phi™ Coprocessor

- Optimization Methodology
- Using Loop Profiler to Determine Where to Optimize
- Using Vector Reports to Learn What Code Vectorized
- Vectorization Guidelines for the Intel® Xeon Phi™ Coprocessor
- Affinity & Load Balancing
- Other Optimization Topics
General Performance Guidelines

• Three things you must do to attain maximum performance on the Intel® Xeon Phi™ Coprocessor
  – Optimize for memory access
    o Minimize gathers and scatters by converting arrays of structures (AOS) to structures of arrays (SOA)
    o “Block” algorithms to maximize time data spends in cache
  – Exploit thread and task parallelism
  – Optimize for SIMD
    o Choose SIMD-friendly algorithms
      ▪ Remove back-to-back dependencies between loop iterations
      ▪ Minimize scatter/gather
      ▪ Minimize branch misprediction
      ▪ Etc.
    o Vectorize inner loops with #pragma simd or the like
    o Vectorize outer loops using Intel® Cilk™ Plus array notation

• Balance performance and accuracy
  – Ex: use fast sqrt, rsqrt, sin, exp, etc. when possible
Improving the Odds That Code Will Vectorize – 1 of 4

- Remove conditionals and function calls from inner loops
- Increase the odds of vectorization if you need function calls in inner loops by:
  - Use `#pragma simd`, elemental functions marked with `__attribute__((vector))`, or Intel® Cilk™ Plus Array Notation
- If you compile with `-g`, be sure to specify the optimization level (`-Ox`)
  - When you wish to do debugging or performance analysis on optimized code use “-g”
  - Otherwise, the default optimization level switches from `-O2` to `-O0` and no vectorization occurs
- Example:
  ```
  icc -O2 -g -vec-report3 -offload-build foo.c
  ```

Note: Some outer loops will vectorize in 12.1 and beyond
Improving the Odds That Code Will Vectorize – 2 of 4

• Use `-vec-report1` through `-vec-report3` to figure out what does not vectorize and why
  – But only in hot code

• Avoid use/construction of `int64` vectors if possible
  – `int64` types on Intel® Xeon Phi™ Coprocessor: `size_t`, `pointer`, `long`
  – Performance: `int64` does not mix well with `int32`
  – It is much harder to vectorize
    o Avoid arithmetic, loop counters, array indexing, scatter/gather, etc. with `int64` types
    o “unsupported data type” or “dereference/subscript too complex” result
  – Avoid converting between `int64` and `float`
    o Another cause of “unsupported data type” `-vec-report` output
  – If possible use a signed 32-bit integer
  – `int64` operations in the vector unit are (at best) half the bandwidth of `int32` operations in vector unit!
Improving the Odds That Code Will Vectorize—3 of 4

• Limited 8-bit and 16-bit data support
  – Look for “unsupported data type” in -vec-report
  – Try manually converting to/from int to prevent the compiler from attempting automatic integer type conversions

• Help the compiler with loop trip counts
  – If you see “low trip count” in -vec-report, help the compiler by using #pragma loop_count
  – Make sure loops operating on doubles have at least 8 iterations, operations on floats have at least 16 iterations

• For best performance, try marking any functions called by the loop you are trying to vectorize with combinations of
  – #pragma inline, forceinline, or noinline
  – __attribute__((vector))


Improving the Odds That Code Will Vectorize– 4 of 4

- You will get best performance using signed integers as loop counters
  - Not unsigned integers or floats
- For best performance, branches that do conditional assignments should use a loop-local variable
  - Then store the value of the local to the desired variable outside the branch
- Avoid expensive, unintended type conversions where possible
  - E.g: use: float x; x = sin(x + 1.0) + 1.0
- Vectorize inner loops and collapse outer loops to allow more work for parallelism
- When porting code from Fortran, make sure to account for the switch from column-major to row-major array layout (swap inner and outer loop indices when traversing array)
- File a bug if you see vectorization differences between Intel® Xeon Phi™ Coprocessor and Intel® Xeon®
Optimization for the Intel® Xeon Phi™ Coprocessor

- Optimization Methodology
- Using Loop Profiler to Determine Where to Optimize
- Using Vector Reports to Learn What Code Vectorized
- Vectorization Guidelines for the Intel® Xeon Phi™ Coprocessor
- Affinity & Load Balancing
- Other Optimization Topics
Affinity & Load Balancing – How Many Threads per Core?

• While Intel® Xeon Phi™ Coprocessor allows 4 threads per core, there can be good reasons to use fewer than that
  – Minimize cache/TLB thrashing when too many threads compete for L1 or L2
  – Lower competition for the core’s single vector unit
  – Minimize requests to main memory by lowering total thread count

• There are also good reasons to have all 4 threads in use
  – To take advantage of data locality between threads and data fits in cache
  – To keep per-core thread working set small enough

• What is best depends...
  – 1 thread/core for workloads with high DRAM bandwidth requirements
  – 4 threads/core for many of the GEMM workloads
  – 3 threads/core is the sweet spot for most assembly code we wrote
  – >=2 threads/core gives you >90% of instruction issue bandwidth

• Best case, all threads on a core should collaborate
• OpenMP will avoid core 0 (OS & services) for performance reasons
• This topic is vastly more complicated than we can discuss in one slide
Affinity & Load Balancing – Controlling Threads per Core Using OpenMP* (1)

- It can be important to tell OpenMP* how you want threads distributed in the system using KMP_AFFINITY (default = none)
  - Especially when OMP_NUM_THREADS is less than the number of available threads
  - KMP_AFFINITY="compact" assigns the OpenMP thread \( <n> + 1 \) to a free thread context as close as possible to the thread context where the \( <n> \) OpenMP thread was placed
    - Good when data locality matters
    - Can result in a load imbalance if not used carefully
  - KMP_AFFINITY="scatter" distributes the threads as evenly as possible across the entire system
    - Good when you want to make maximum use of system resources
  - Additional tuning is possible using granularity=
  - You can also explicitly assign threads to specific thread contexts
  - But be careful – setting affinity or thread count incorrectly can negatively affect performance
Affinity & Load Balancing – Controlling Threads per Core Using OpenMP* (2)

- KMP_AFFINITY="balanced"
  - Distributes threads but keeps adjacent thread numbers next to each other
  - Essentially “scatter” when # threads <= # available cores
  - Differs when # of threads > # available cores
  - Provides locality between adjacent threads that can be exploited

- For detailed information on controlling affinity in OpenMP*, look up “Thread Affinity Interface (Linux* and Windows*)” in the Intel® Compiler XE 2013 User and Reference Guides

<table>
<thead>
<tr>
<th>7 (u)ser threads</th>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical 0</td>
<td>u0</td>
<td>u3</td>
<td>u5</td>
</tr>
<tr>
<td>Logical 1</td>
<td>u1</td>
<td>u4</td>
<td>u6</td>
</tr>
<tr>
<td>Logical 2</td>
<td>u2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: 7 user threads over a 3 core co-processor
Optimization for the Intel® Xeon Phi™ Coprocessor

- Optimization Methodology
- Using Loop Profiler to Determine Where to Optimize
- Using Vector Reports to Learn What Code Vectorized
- Vectorization Guidelines for the Intel® Xeon Phi™ Coprocessor
- Affinity & Load Balancing
- Other Optimization Topics
Other Optimization Suggestions –

Performance is **All About Memory**

- **Prefetching**
  - Intel® Xeon Phi™ Coprocessor has hardware prefetcher
  - The compiler generates SW prefetches automatically
  - If you have to do it yourself, prefetch to L2 about 1000 cycles in advance of demand usage. Then prefetch to L1 about 50 cycles in advance. (Do NOT prefetch directly to L1!)

- **Pick data pitches to minimize number of cache lines accessed**
  - In both L1 and L2
  - Also pay attention to how many memory addresses alias to the same cache line – working on multiples will thrash the cache

- **Use 2M pages for large, heavily-accessed structures**
  - Minimizes TLB misses, but fewer TLB entries available

- **Each output cache line should only be written to by only one core.**

- **Latency to main memory can rise >3X when GDDR is heavily burdened**
  - Thereby lowering per-core maximum bandwidth

- **Gather/scatter a good thing if used carefully**
Ensuring Memory Alignment

- Memory alignment is an essential part of high speed vector processing
- Unaligned memory will incur performance penalties
- Allocating aligned memory
  - Static memory
    - Allocated by compiler/linker
    - Add __attribute__((aligned(n))) in front variable declaration
    - Applies to global/local static variables as well as stack/auto variables
  - Dynamic memory
    - Allocated by language runtime
    - Use __mm_aligned_malloc(size, alignment_bytes)
    - Pair it with __mm_aligned_free()
  - Using Intel® Threading Building Blocks
    - Dynamic memory allocation API that supports the Intel® Xeon Phi™ Coprocessor
    - Use scalable_aligned_malloc()/scalable_aligned_free()
    - Include <tbb/scalable_allocator.h>, and link with -ltbbmalloc
Other Optimization Suggestions

• Make sure vectorized code does array accesses to contiguous memory elements
  – Improves data locality and minimizes masking

• See the *Intel® Xeon Phi™ Optimization Guide* and *Intel® Xeon Phi™ Profiling Guide* posted on http://software.intel.com/mic-developer
Measuring Offload Performance (1)

Watch out

- What you transfer
- What you time

• Be careful what you transfer
  - All C/C++ pointer data is transferred via DMA in the explicit data copy model
  - Non-pointer data and Fortran data is transferred by a slower mechanism
  - Transferring 4K (page) aligned addresses gives best performance
  - Transfer time will be dominated by DMA setup time when transferring only small amounts of pointer data
Measuring Offload Performance (2)

• Be careful what you time
  – In the explicit data copy model, the first #pragma offload statement executed is special
    o This is when the Intel® Xeon Phi™ binaries are transferred in addition to any user data
    o This is also when thread startup happens on the Intel® Xeon Phi™ Coprocessor
    o So this transfer is not a good one measure
  – Separate buffer memory management from user data transfer when timing with three offload statements
    o One offload with RETAIN to allocate the buffer
    o Another with REUSE and RETAIN for the real transfer - Time this one
    o Then a final offload with REUSE to release the buffer
  – Debugging output will interfere with your performance measurements
Timing APIs for Performance Measurement

• Accurate timing is essential for performance-critical software
• From Linux host use `gettimeofday(&tv, NULL)`
• On coprocessor, roll your own...
  – You can use `__rdtsc()` and divide the result by the frequency
  – Division creates systematic errors
  – Querying the frequency requires access to the system call interface
  – See next slide
• ...Or take advantage of timing code in Intel® Thread Building Blocks
  – Include `<tbb/tick_count.h>`
  – Link with `-ltbb`
  – Available on Host and Intel® Xeon Phi™ coprocessor

```cpp
using namespace tbb;
tick_count start = tick_count::now();
// your high performance
// workload to measure

tick_count stop = tick_count::now();
double duration = (stop - start).seconds();
```
A Timing API that does a poor job

```c
#include <sys/time.h>
#ifdef __MIC__
#include <sys/types.h>
#include <sys/sysctl.h>
#endif

// Note - most accurate results will be obtained when math is done after start/stop
// timing data are collected, rather than here
__attribute__((target(mic))) double second()
{
    #ifdef __MIC__
        static int freq;
        if (!freq) {
            size_t len=sizeof(freq);
            sysctlbname("hw.clockrate", &freq, &len, 0L, 0);
        }
        long clocks = __rdtsc();
        if (freq)
            return ((double)clocks)/((double)freq*1e6);
    #else
        struct timeval tv;
        gettimeofday(&tv, NULL);
        return (double)tv.tv_sec + (double)tv.tv_usec / 1000000.0;
    #endif
    return 0.0;
}
```

Unnecessary computation done within a timing loop.
Module Summary

- Optimization Methodology
- Using Loop Profiler to Determine Where to Optimize
- Using Vector Reports to Learn What Code Vectorized
- Vectorization Guidelines for the Intel® Xeon Phi™ Coprocessor
- Affinity & Load Balancing
- Other Optimization Topics
Module Discussion

Can you:

• List at least one tool that will help you determine where to optimize code?
• Tell if a loop was vectorized?
• Explain at least one way to improve the performance of code on the Intel® Xeon Phi™ Coprocessor?
Labs

• Getting Code to Vectorize (Lesson 6)
• Finding Good Offload Candidates (Lesson 7)
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