Intel® Xeon Phi™ Coprocessor

Introduction
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. logo, and Intel Sponsors of Tomorrow. trademark 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

Statements and Assumptions

- This workshop is not comprehensive training
  - Looking at breath and not depth
  - We will limit the depth of questions
    - Contact your Intel representative for in-depth training
- All attendees are experienced software developers
  - Single-machine parallel programming
  - Production environment
- Attendees will be writing code for the Intel® Xeon Phi™ Coprocessor
Topics Covered During this Workshop

• Introduction (this presentation)
• Offload Compilation
• Advanced Offload Compilation Topics
• Optimization
• Debugging
• Performance Analysis
• Programming Models
• SW Ecosystem and what is to come
Background

- High-level overview of the Intel® Xeon Phi™ Coprocessor
  - Hardware
  - Software
- Intel® Xeon Phi™ Coprocessor programming considerations
- Terminology
Intel® Xeon Phi™ Coprocessor Overview – Features of an Individual Core

- Up to 61 in-order cores
  - Ring interconnect
- 64-bit addressing
- Two pipelines
  - Intel® Pentium® processor family-based scalar units
    - Dual issue with scalar instructions
  - Pipelined one-per-clock scalar throughput
    - 4 clock latency, hidden by round-robin scheduling of threads
- 4 hardware threads per core
  - Cannot issue back to back inst in same thread
Intel® Xeon Phi™ Coprocessor Overview – Features of an Individual Core

- Up to 61 in-order cores
  - Ring interconnect
- 64-bit addressing
- Two pipelines
  - Intel® Pentium® processor family-based scalar units
    - Dual issue with scalar instructions
  - Pipelined one-per-clock scalar throughput
    - 4 clock latency, hidden by round-robin scheduling of threads
- 4 hardware threads per core
  - Cannot issue back to back inst in same thread
Intel® Xeon Phi™ Coprocessor Overview – Features of an Individual Core

• Up to 61 in-order cores
  – Ring interconnect
• 64-bit addressing
• Two pipelines
  – Intel® Pentium® processor family-based scalar units
    o Dual issue with scalar instructions
  – Pipelined one-per-clock scalar throughput
    o 4 clock latency, hidden by round-robin scheduling of threads
• 4 hardware threads per core
  – Cannot issue back to back inst in same thread
Intel® Xeon Phi™ Coprocessor Overview – Features of an Individual Core

- Up to 61 in-order cores
  - Ring interconnect
- 64-bit addressing
- Two pipelines
  - Intel® Pentium® processor family-based scalar units
    - Dual issue with scalar instructions
  - Pipelined one-per-clock scalar throughput
    - 4 clock latency, hidden by round-robin scheduling of threads
- 4 hardware threads per core
  - Cannot issue back to back inst in same thread
Intel® Xeon Phi™ Coprocessor Overview – Features of an Individual Core

- Up to 61 in-order cores
  - Ring interconnect
- 64-bit addressing
- Two pipelines
  - Intel® Pentium® processor family-based scalar units
    - Dual issue with scalar instructions
  - Pipelined one-per-clock scalar throughput
    - 4 clock latency, hidden by round-robin scheduling of threads
- 4 hardware threads per core
  - Cannot issue back to back inst in same thread
Optimization Notice

Beginning Intel® Xeon Phi™ Coprocessor Workshop, Sept 2012

Intel® Xeon Phi™ Coprocessor Overview – Features of an Individual Core (2)

- Optimized
  - Single and Double precision
- All new vector unit
  - 512-bit SIMD Instructions – not Intel® SSE, MMX™, or Intel® AVX
  - 32 512-bit wide vector registers
    - Hold 16 singles or 8 doubles per register
- Fully-coherent L1 and L2 caches
Intel® Xeon Phi™ Coprocessor Overview – Cache

- **L1 cache**
  - 32K I-cache per core
  - 32K D-cache per core
  - 3 cycle access
  - Up to 8 outstanding requests
  - Fully coherent

- **L2 cache**
  - 512K Unified per core
  - Sum across all cores
  - 11 cycle best access
  - Up to 32 outstanding requests
  - Fully coherent
**Intel® Xeon Phi™ Coprocessor Overview – Cache**

- **L1 cache**
  - 32K I-cache per core
  - 32K D-cache per core
  - 3 cycle access
  - Up to 8 outstanding requests
  - Fully coherent

- **L2 cache**
  - 512K Unified per core
  - Sum across all cores
  - 11 cycle best access
  - Up to 32 outstanding requests
  - Fully coherent
**Intel® Xeon Phi™ Coprocessor Overview – Cache**

- **L1 cache**
  - 32K I-cache per core
  - 32K D-cache per core
  - 3 cycle access
  - Up to 8 outstanding requests
  - Fully coherent

- **L2 cache**
  - 512K Unified per core
  - Sum across all cores
  - 11 cycle best access
  - Up to 32 outstanding requests
  - Fully coherent
Intel® Xeon Phi™ Coprocessor Overview – Memory and Alignment

- **Alignment**
  - Based upon the number of elements, the element size and the vector load and store instruction
  - 64B alignment for 4B (float) data elements for a 16 to 16 vector load

- **Memory**
  - GDDR5
  - 16 Memory channels
    - 5.5 Gbit each
  - 8 GB
  - 300 ns access
Intel® Xeon Phi™ Coprocessor Overview – Software Architecture: Two Modes

Linux* Host

- Host-side offload application
  - User code
- Offload libraries, user-level driver, user-accessible APIs and libraries
- User-level code
- System-level code

Intel® Xeon Phi™ Coprocessor

Intel® Xeon Phi™ Coprocessor Architecture support libraries, tools, and drivers

PCI-E Bus

Linux* OS

Intel® Xeon Phi™ Coprocessor

Intel® Xeon Phi™ Coprocessor communication and application-launching support

PCI-E Bus

Linux* OS

Intel® Xeon Phi™ Coprocessor

Offload libraries, user-accessible APIs and libraries

User code

User-level code

System-level code
Intel® Xeon Phi™ Coprocessor Overview – Software Architecture: Two Modes

**Linux* Host**
- ssh or telnet connection to /dev/mic*

**Intel® Xeon Phi™ Coprocessor**
- Target-side "native" application
  - User code
  - Standard OS libraries plus any 3rd-party or Intel libraries
  - Virtual terminal session
- Intel® Xeon Phi™ Coprocessor communication and application-launching support

**Intel® Xeon Phi™ Coprocessor Architecture support libraries, tools, and drivers**

**PCI-E Bus**

**System-level code**
- User-level code

**Linux* OS**

**PCI-E Bus**

**Intel® Xeon Phi™ Coprocessor Linux* Host**
- System-level code
- User-level code
Intel® Xeon Phi™ Coprocessor Overview – Programming Models

The familiar Intel development environment is available:

- Intel® C, C++ and Fortran Compilers
- OpenMP*
- Intel® MPI Library support for the Intel® Xeon Phi™ Coprocessor as an MPI node
- Intel® Parallel Building Blocks
  - Intel® Threading Building Blocks (Intel® TBB)
  - Intel® Cilk™ Plus
- Intel® Debugger
- Intel® Performance Libraries (e.g. Intel® MKL)
  - Three versions: host-only, coprocessor-only, heterogeneous
- Intel® VTune™ Amplifier XE 2013
- Standard runtime libraries, even pthreads*
Background

• High-level overview of the Intel® Xeon Phi™ Coprocessor
  – Hardware
  – Software
• Intel® Xeon Phi™ Coprocessor programming considerations
• Terminology
• Getting full performance from the Intel® Xeon Phi™ Coprocessor requires both a high degree of parallelism and vectorization
  – Not all code can be written this way
  – Not all programs make sense on this architecture

• Intel® Xeon Phi™ Coprocessor is not an Intel® Xeon® processor
  – It specializes in running highly parallel and vectorized code.
  – New vector instruction set and 512-bit wide registers
  – Not optimized for processing serial code
Intel® Xeon Phi™ Coprocessor Programming Considerations - Setting Expectations (2)

• Very short (low-latency) tasks not optimal for offload to the coprocessor
  – Costs that you need to amortize to make it worthwhile:
    o Cost of code and data transfer
    o Cost of process/thread creation
  – Fastest data transfers currently require careful data alignment

• This architecture is not optimized for serial performance
Intel® Xeon Phi™ Coprocessor Programming Considerations – This is not a GPU

- Very different memory architectures
  - The Intel® Xeon Phi™ Coprocessor is not optimized for concurrent out-of-cache random memory accesses by large numbers of threads (GPUs are)
  - The Intel® Xeon Phi™ Coprocessor has a “traditional” coherent-cache architecture
  - GPUs have a memory architecture specialized for localized “shared memory” processing
- “Threads” and “cores” mean something very different - GPU versions of these are limited and lighter-weight
- Each architecture (host CPU, Intel® Xeon Phi™ Coprocessor, or GPU) is really good at some things, and not optimal for other things
  - Because the Intel® Xeon Phi™ Coprocessor has an architecture similar to the Intel® Xeon®, it might be your best choice for further accelerating highly parallel and vectorized code developed on Intel® Xeon®
Background

• High-level overview of the Intel® Xeon Phi™ Coprocessor platform
  – Hardware
  – Software
• Intel® Xeon Phi™ Coprocessor programming considerations
• Terminology
SIMD Width/Hardware Abstraction – What is Vectorization and SIMD Processing?

- SISD: Single Instruction, Single Data
- SIMD: Single Instruction, Multiple Data
- SPMD: Single Program, Multiple Data
- Vectorization: Is the process of transforming a scalar operation that acts on single data elements at a time (SISD), to an operation that acts on multiple data elements at once (SIMD).
  - For loops in “unvectorized” (i.e. “normal”) code, each assembly instruction processes the data from only a single loop iteration
  - SIMD can result in more compact and efficient generated code
    - Assuming that setup code does not is not too costly
- The compiler can also transform appropriately constructed functions or blocks of code into a SPMD implementation that can be called/inlined within SIMD code
SIMD Width/Hardware Abstraction – Vectorization/SIMD Example

for (i = 0; i < 15; i++)
    if (v5[i] < v6[i])
        v1[i] += v3[i];

Note the lack of jumps or conditional code branches

4 inst x 16 = 64 min
**SIMD Width/Hardware Abstraction – Vectorization/SIMD Example**

```c
for (i = 0; i < 15; i++)
    if (v5[i] < v6[i])
        v1[i] += v3[i];
```

- **Note the lack of jumps or conditional code branches**

- **Only 2 vector inst!**

  ```c
  vcmppi_lt k7, v5, v6
  vaddpi v1{k7}, v1, v3
  ```
SIMD Width/Hardware Abstraction – Vectorization/SIMD Example

for (i = 0; i < 15; i++)
  if (v5[i] < v6[i])
    v1[i] += v3[i];

Note the lack of jumps or conditional code branches

v5 = 0 4 7 8 3 9 2 0 6 3 8 9 4 5 0 1
v6 = 9 4 8 2 0 9 4 5 5 3 4 6 9 1 3 0
vcmppi_lt k7, v5, v6
k7 = 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0
v3 = 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8
v1 = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
vaddpi v1{k7}, v1, v3
v1 = 6 1 8 1 1 1 8 9 1 1 1 1 6 1 8 1

512-bits
SIMD Width/Hardware Abstraction – Vectorization/SIMD Example

for (i = 0; i < 15; i++)
    if (v5[i] < v6[i])
        v1[i] += v3[i];

Note the lack of jumps or conditional code branches

v5 = 0 4 7 8 3 9 2 0 6 3 8 9 4 5 0 1
v6 = 9 4 8 2 0 9 4 5 5 3 4 6 9 1 3 0
vcmppi_lt k7, v5, v6
k7 = 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0
v3 = 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8
v1 = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
vaddpi v1{k7}, v1, v3
v1 = 6 1 8 1 1 1 8 9 1 1 1 1 6 1 8 1

512-bits
SIMD Width/Hardware Abstraction – Vectorization/SIMD Example

```plaintext
for (i = 0; i < 15; i++)
    if (v5[i] < v6[i])
        v1[i] += v3[i];
```

Note the lack of jumps or conditional code branches.

```
v5 = 0 4 7 8 3 9 2 0 6 3 8 9 4 5 0 1
v6 = 9 4 8 2 0 9 4 5 5 3 4 6 9 1 3 0
vcmpipi_lt k7, v5, v6
k7 = 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0
v3 = 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8
v1 = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
vaddpi v1{k7}, v1, v3
v1 = 6 1 8 1 1 1 8 9 1 1 1 1 6 1 8 1
```

Unrolled v5 & v6

Mask (compare)

Result

512 bits
Most Code Contains Some Parallelism

- **Functional parallelism:**
  - Entirely different operations that can safely execute in parallel

- **Data parallelism**
  - Same operation performed on different data
  - Requires data domains that can be computed concurrently
  - Normally found in loops
  - Applies to both vectorization and higher-level parallelism

```c
for (y = 0; y < nLines; y++) {
    genLine (model, im[y]);
}
```

```c
fluxA(v, a);
fluxB(v, b);
fluxC(v, c);
```
Thank you for coming!!!
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