Intel® Xeon Phi™ Coprocessor

Beilun Wang
Lin Gong
Chunkun Bo
Motivation

- High demand for High performance computing especially for parallel computing
- Competing with graph chips
  - Such as AMD, NVIDIA
- Advantage
  - Low cost in developing and maintaining software
Roadmap

- Architecture Overview
- Pipeline
- VPU and ISA
- Memory Hierarchy
- Performance Comparison
The Intel® Xeon Phi™ is a coprocessor.

It has 61 dual-issue in-order cores.

Each core includes:
- 512 bits wide Vector Processor Unit (VPU)
- Core Ring Interface (CRI)
- Interfaces to the Core and the Ring Interconnect
- L2 cache
- Tag Directory (TD)
Why only 60 cores for pure computation tasks?

The Intel® Xeon Phi™ runs an OS inside.

One core service hardware/software requests like interrupts.
The core pipeline:

- seven stages for integer instructions
- six extra stages for vector pipeline.

Compare to generic pipeline

Fetch → Decode → Execute → Write Back

PPF → PF → D0 → D1 → D2 → E → WB
Fetch stage

- Traditional instruction fetch stage (IF) is broken down into two stages.
- They select the thread to execute.
The PPF stage prefetches instructions for a thread context into the prefetch buffers.
There are two streams per thread.

Once one of the stream is stalled (due to branch mispredict), a second stream is switched in while the branched target stream is being prefetched.
The PF stage uses picker function (PF) selects the thread to execute by sending the instruction pairs to decode stages.

Each core is able to issue two instructions per clock (U-pipe + V-pipe).

Refill(PPF) a prefetch buffer
If prefetch buffer not refilled, a core stall will happen.

The refill of the instruction prefetch buffer takes 4-5 cycles.

If PF and PPF are not synchronized, a one clock bubble may be inserted.
D0 and D1 decode at the rate of two instruction per clock
Two pipelines

Pipes & Instructions

- V-Pipe
  - Scalar Register - ALU
    -> New Scalar Instructions

- U-pipe
  - X87 Register - X87 FP Execution Unit
    -> X87 FP Instructions
  - Vector Register - Vector Execution Unit
    -> Vector Instructions
    -> Vector Memory Instructions
    -> Vector Mask Instructions
Vector Processing Unit

The Key Component

Source:
New Vector Instructions

- A high performance 64 bit execution environment.
- 32 new vector registers.
- Ternary instructions.
- Vector mask support.
- Coherent memory model.
- Gather/Scatter support.

Extended Math Unit (EMU)
32 New Vector Registers

The 512-bit wide vector SIMD registers.

- 16 single-precision floating-point instructions.
- 16 32-bit integer instructions.
- 8 double-precision floating-point instructions.
- 8 64-bit integer vector instructions.
Vector mask support

- Protect elements from updates during the executions.
- Control the vector length of the operations being performed.
- 8 vector mask registers.
Vector Mask Instructions

- **Set, Copy, or Operate** on the contents of a given vector mask.

- **Three types:**
  - Mask read/write instructions.
  - Flag instructions.
  - Mask logical instructions.
### Vector Mask Example

**MSB**

<table>
<thead>
<tr>
<th>ZMM0</th>
<th>ZMM1</th>
<th>ZMM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000003</td>
<td>0x00000007</td>
<td>0x000000BB</td>
</tr>
<tr>
<td>0x00000002</td>
<td>0x00000006</td>
<td>0x0000000A</td>
</tr>
<tr>
<td>0x00000001</td>
<td>0x00000005</td>
<td>0x00000009</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x00000004</td>
<td>0x00000008</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

**LSB**

<table>
<thead>
<tr>
<th>ZMM0</th>
<th>ZMM1</th>
<th>ZMM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

(\(\text{bytes 15 through 0}\))

(\(\text{bytes 31 through 16}\))

(\(\text{bytes 47 through 32}\))

(\(\text{bytes 63 through 48}\))

**k3 = 0x8F03**

\[
\text{vpadd zmm2 \{k3\}, zmm0, zmm1}
\]

\[
\begin{array}{cccccc}
\text{bytes 15 through 0} & \text{bytes 31 through 16} & \text{bytes 47 through 32} & \text{bytes 63 through 48} \\
\hline
\text{0x00000010} & \text{0x00000019} & \text{0x0000001A} & \text{0x0000001E} \\
\text{0x00000000} & \text{0x00000000} & \text{0x00000000} & \text{0x00000000} \\
\end{array}
\]
Gather/Scatter Support

- A special case of memory instructions.
- Allow manipulation of irregular data patterns of memory.
- Enable vectorization of algorithms with complex data structures.
Gathers/ Scatters Inets

- **Vector Loads/ Broadcasts**
  
  \[ zmm0 \leq U(m) \]

  A special sub-case:

  **Vector Gathers.**

  \[ zmm0 \leq U(mv) \]

- **Vector Stores**
  
  \[ m \leq D(zmm0) \]

  A special sub-case:

  **Vector Scatters.**

  \[ mv \leq D(zmm0) \]
Extended Math Unit

- Execute complex operations.
  - Reciprocal, square root, log.
  - Execute in a vector fashion with high bandwidth.

- Operate by calculating polynomial approximation of these functions.
Memory Hierarchy

Framework

## Memory Hierarchy

### Cache Hierarchy

<table>
<thead>
<tr>
<th>Parameter</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coherence</td>
<td>MESI</td>
<td>MESI</td>
</tr>
<tr>
<td>Size</td>
<td>32KB + 32KB</td>
<td>512KB</td>
</tr>
<tr>
<td>Associativity</td>
<td>8-way</td>
<td>8-way</td>
</tr>
<tr>
<td>Line Size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>Banks</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Access Time</td>
<td>1 cycle</td>
<td>11 cycle</td>
</tr>
<tr>
<td>Policy</td>
<td>Pseudo LRU</td>
<td>Pseudo LRU</td>
</tr>
<tr>
<td>Duty Cycle</td>
<td>1 per clock</td>
<td>1 per clock</td>
</tr>
<tr>
<td>Ports</td>
<td>Read or Write</td>
<td>Read or Write</td>
</tr>
</tbody>
</table>
Memory Hierarchy

- Streaming hardware prefetcher for L2.

- L2 cache supports ECC, and power states such as C1 (shuts off clocks to the core and the VPU), C6 (shuts off clocks and power to the core and the VPU), and the package C3 (all core clocks are stopped).

- L2 cache is inclusive of the L1 data and instruction caches.

- Actual size of L2 storage is a function of the degree of code and data sharing among cores.
Memory Hierarchy

- A miss in L1/L2 cache does not stall the entire core.
- Prefetch data aggressively

L1 and L2 cache can support up to about 38 outstanding requests per core.

System agent (PCI Express* agent and DMA controller) can generate 128 outstanding requests.

In total, $38 \times (\text{number of cores}) + 128$. 
Memory Hierarchy

- CRI (Core Ring Interface)

Hosts the L2 cache and connects each core to a Ring Stop.

Comprises private L2 cache itself plus all of the off-core transaction.

Two other major blocks: the R-Unit (APIC) and the Tag Directory (TD).
Memory Hierarchy

- TD (Tag Directory)

The tag directory is not centralized but broken up into 64 distributed tag directories.

L2 caches are kept fully coherent with each other by the TDs. Each DTD is responsible for its assigned cache lines.

A TD tag contains the address, state (GOLS, GS, GE/GM, GI), and an ID for the owner of the cache line.
Memory Hierarchy

- Page Table

Supports 32-bit physical addresses in 32-bit mode, 36-bit physical address extension (PAE) in 32-bit mode, and 40-bit physical address in 64-bit mode.

Supports 4-GB, 64-GB and 2-GB page sizes.

Supports the Execute Disable (NX) bit.

Each L1 dTLB has 64 entries for 4 KB pages, 32 entries for 64-KB pages and 8 entries for 2-MB pages.

Each core has one iTLB, owning 32 entries for 4-KB pages.
Memory Hierarchy

Discrepancy?

1. Another new feature is the 512 KB unified Level Two (L2) cache unit. The L2 organization comprises 64 bytes per way with 8-way associativity, 1024 sets, 2 banks, 32GB (35 bits) of cacheable address range and a raw latency of 11 clocks.

2. are summarized in Table 2-4. Two unusual fields in this table are the Duty Cycle and Ports designations, which are specific only to the Intel® Xeon Phi™ coprocessor design. The L1 cache can be accessed each clock, whereas the L2 can only be accessed every other clock. Additionally, on any given clock software can either read or write the L1 or L2, but it

3. The details of the L1 instruction and data cache structure are shown in Table 1. The data cache allows simultaneous read and write allowing cache line replacement to happen in a single cycle. The L1 cache consists of 8 ways set associative 32
Performance Comparison (Intel)

Source:
FACT: A GPU is significantly faster than Intel's Xeon Phi on real HPC applications.

Speeding time to result for key science applications by 2x over Xeon Phi.

MiniMD
Although Xeon Phi can be optimized to outperform a CPU, GPU consistently outperforms Xeon Phi on a wide range of supercomputing applications. System and configuration details\(^1\) (Data from August 2013)

Source: http://www.nvidia.com/object/justthefacts.html
FACT: "Recompile & Run" on Xeon Phi actually slows down your application.

The notion that developers can simply "recompile and run" applications on Intel's Xeon Phi, without any change to their CPU code, is attractive but misleading. The resulting performance is usually much slower than CPU performance, literally the opposite of acceleration.

Simple recompile and run on Xeon Phi can work, but codes run much slower than on the CPU. System and configuration details² (Data from August 2013)
Conclusion?

- Hard to say which one is better. Comparison may not be totally fair.

- If only from performance perspective, most data support that Nvidia is better.
Thank you!
Backup for MESI

Backup for MESI

Backup Page Table

http://en.wikipedia.org/wiki/Physical_Address_Extension
Backup for Page Table

Linear address:

page-directory-pointer table
- Dir. Pointer entry
- Dir. Pointer entry
- Dir. Pointer entry
- Dir. Pointer entry

page directory
- 64 bit PD entry

page table
- 64 bit PT entry

4K memory page

*) 32 bits aligned to a 32-Byte boundary

http://en.wikipedia.org/wiki/Physical_Address_Extension
Backup for Page Table