INTRODUCTION TO XEON PHI
Naming Conventions

- It is a **coprocessor**, not a GPU or accelerator
- **MIC** is the name of the architecture
  - Comparable to Intel64 on CPU side
- **Xeon Phi** is the brand name of the product
- Architecture generation named as **Knight’s …**
  - Comparable to ”Nehalem”, ”Sandy Bridge” etc. on CPU side
- Different models have number designations
  - i.e. 5110P, SE10
Timeline

2008 - **Larrabee** GPGPU announced
  - Was not productized

2010 - **MIC** and the **Knight’s** series announced
  - Re-architected for pure computing
  - **Knight’s Ferry (KNF)** development kit

2011 - **Knight’s Corner (KNC)** development kit
  - Alpha/beta versions of the final products

2012 - **Intel Xeon Phi** brand introduced
  - First products based on the KNC architecture

???? - **Knight’s Landing (KNL)**
  - Future architecture
Intel Many Integrated Core (MIC)

- PCI Express card, a "PC in a PC"
- 10s of x86-based cores
  - Hardware multithreading
  - Instruction set extensions for HPC
- Very high-bandwidth local GDDR5 memory
- Runs a stripped-down version of Linux
Intel MIC Philosophy

- Design the hardware for HPC
  - Strip out "general purpose" CPU features
    - Elaborate branch prediction, out-of-order execution etc.

- Leverage existing x86 architecture and programming models
  - Enable development using common code base
    - Same tools and libraries as on CPU
    - Same parallel paradigms (OpenMP, Cilk etc.)
  - Optimization strategies similar to CPU ones
    - Optimizations for Phi tend to improve CPU performance
# Current Xeon Phi Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Cores</th>
<th>Clock Rate (GHz)</th>
<th>Raw perf (Gflops)</th>
<th>Memory Size (GB)</th>
<th>L2 Size (MB)</th>
<th>Memory BW (GB/s)</th>
<th>TDP Power (W)</th>
<th>Cooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>5110P</td>
<td>60</td>
<td>1.053</td>
<td>1011</td>
<td>8</td>
<td>30</td>
<td>320</td>
<td>225</td>
<td>Passive</td>
</tr>
<tr>
<td>SE10 (special edition)</td>
<td>61</td>
<td>1.1</td>
<td>1073</td>
<td>8</td>
<td>30.5</td>
<td>352</td>
<td>300</td>
<td>Passive (SE10P)</td>
</tr>
<tr>
<td>3100</td>
<td>TBA (1H13)</td>
<td>TBA (1H13)</td>
<td>&gt;1000</td>
<td>6</td>
<td>28.5</td>
<td>240</td>
<td>240</td>
<td>Passive, Active</td>
</tr>
</tbody>
</table>

What we will be using
Xeon Phi Core Architecture

- 22nm process using 3D TriGate transistors
- Based on the old Pentium (P65C) core
  - Added 64-bit addressing, SIMD, distributed L2 etc.
  - In-order execution
  - Designed for 4 hardware threads per core
  - Ring bus similar to current Sandy Bridge CPUs
- Fast GDDR5 GPU memory on card
  - Discrete from host memory
  - Highly efficient HW prefetching
  - 4KB default page size
    - 2MB huge pages
Xeon Phi Instruction Set Architecture

Aka. LRBNI (Larrabee New Instructions)

- 512-bit wide SIMD vector ops
  - 8 double (or 16 float or integer) operations per cycle
- Fused Multiply-Add (FMA)
  - $A \leftarrow A + (B \times C)$ in a single cycle (with 1 rounding error)
    - 16 DP or 32 SP FLOPS/cycle
- Vector masks
  - Toggles which vector elements are affected by an instruction
- Gather / scatter operations
  - Used to access memory (in cache) in an irregular pattern
- Extended Math Unit (EMU)
  - Transcendental operations (sqrt, log etc.)

Different from MMX, SSE, AVX, AVX2 which are used on x86 CPUs!
MIC vs. CPU

Larger amount of simpler cores at a lower clock rate
  - Much less memory per core
  - Similar memory bandwidth per core

In-order execution
  - Thread waits (stalls) on L1 cache miss
  - Sophisticated HW prefetching helps

2x wider vectors than AVX/AVX2
  - Also richer set of vector instructions (FMA, mask etc.)
  - Exploiting this fully is essential for performance on MIC

Using 2-4 threads per core is essential
  - Using just 1 thread gives 50% performance
  - Can hide some of the latency caused by stalls

<table>
<thead>
<tr>
<th></th>
<th>L1 (Data + Instr)</th>
<th>Shared L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>32 KB + 32 KB</td>
<td>512 KB</td>
</tr>
<tr>
<td>Miss Latency</td>
<td>15-30 cyc</td>
<td>500-1000 cycles</td>
</tr>
<tr>
<td>CUDA</td>
<td>MIC</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>CUDA core (1 FP op / cycle)</td>
<td>~= MIC SIMD lane</td>
<td></td>
</tr>
<tr>
<td>CUDA symmetric multiprocessor</td>
<td>~= MIC core</td>
<td></td>
</tr>
<tr>
<td>CUDA thread block</td>
<td>~= MIC threads in a core</td>
<td></td>
</tr>
<tr>
<td>One operation on a CUDA Warp</td>
<td>~= One MIC SIMD operation</td>
<td></td>
</tr>
</tbody>
</table>
| Large oversubscription of work (>
  4 x resident warps per SM optimal)       | Moderate oversubscription of work (2-4x
  threads per core optimal)               |
| Automatic and manual local caching        | Coherent, automatic L2 cache and
  hardware prefetching                     |
| CUDA, OpenCL, OpenACC offloads, Libraries (CUBLAS, CUFFT etc.) | Legacy programming models
  (OpenMP etc.), LEO offloads, OpenCL, Libraries (MKL etc.) |
| Host CPU needed for execution             | Independent native execution of code     |
Operating System & Environment

- Standard Linux kernel with patches
- Lightweight "Busybox" shell environment
  - A subset of the standard shell commands & tools
    - Usage differs from their full counterparts
- Only core set of libraries (glibc etc.) by default
- Binary architecture: **k1om** (not x86_64)
- Standard set of "de facto" HPC languages
  - C/C++/Fortran
- MIC’s local disk is mapped into it’s memory

Copying files onto the Phi’s local "disk" takes up precious GDDR5 memory!
<table>
<thead>
<tr>
<th></th>
<th>Host Native</th>
<th>Offload</th>
<th>Symmetric</th>
<th>Reverse Offload</th>
<th>MIC Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>prog() {</td>
<td>prog() {</td>
<td>prog() {</td>
<td>foo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>foo</td>
<td>#pragma offload foo</td>
<td>foo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phi</td>
<td></td>
<td>foo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Programming Models**

- **Host-centric**
- **MIC-centric**

**Host**
- Native
- Offload
- Symmetric
- Reverse Offload
- MIC Native

**MIC**
- Native
- Offload
- Symmetric
- Reverse Offload
- MIC Native
Useful Links

PRACE Xeon Phi Best Practices Guide
  – http://www.prace-ri.eu/Best-Practice-Guides

Xeon Phi Developer’s Quick Start Guide

Dr. Dobb’s Xeon Phi Guide

Phi programming for CUDA developers