Introduction to Xeon Phi ("Knights Landing") on Cori

Brandon Cook
Brian Friesen

2016 November 3
Knights Landing is here!

WHEN CAN I GET ON?

• NESAP teams are expected to get on in November
  – planned exclusive period

• After NESAP period users may apply for full access
  – General users limited access to debug, apply for full access
  – More details at 11
Schedule for today

• Morning
  – 9:00 Introduction to Xeon Phi
    • many new features and differences!
  – 10:00 Using Cori
  – 11:00 Getting Access to KNL

• Afternoon
  – 1:00 Tools: Intel Vtune/Advisor
  – 1:30 Tools: Craypat/Reveal
  – 2:00 Case Studies
Knights Landing overview

Many **Trailblazing Improvements** in KNL

<table>
<thead>
<tr>
<th>Improvements</th>
<th>What/Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self Boot Processor</td>
<td>No PCIe bottleneck</td>
</tr>
<tr>
<td>Binary Compatibility with Xeon</td>
<td>Runs all legacy software. No recompilation.</td>
</tr>
<tr>
<td>New Core: Atom™ based</td>
<td>~3x higher ST performance over KNC</td>
</tr>
<tr>
<td>Improved Vector density</td>
<td>3+ TFLOPS (DP) peak per chip</td>
</tr>
<tr>
<td>New AVX 512 ISA</td>
<td>New 512-bit Vector ISA with Masks</td>
</tr>
<tr>
<td>Scatter/Gather Engine</td>
<td>Hardware support for gather and scatter</td>
</tr>
</tbody>
</table>
| New memory technology: MCDRAM + DDR | Large High Bandwidth Memory  \rightarrow  MCDRAM  
                                       | Huge bulk memory  \rightarrow  DDR                                     |
| New on-die interconnect: Mesh | High BW connection between cores and memory                             |
| Integrated Fabric: Omni-Path  | Better scalability to large systems. Lower Cost                         |

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. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
### What is different about Cori?

**Edison (Xeon E7-4680 v2; “Ivy Bridge”):**
- 5576 nodes
- 12 physical cores/socket
- 24 logical cores/socket
- 2.4 - 3.2 GHz
- 8 double precision operations per cycle
- 2.5 GB DRAM/core (64 GB/node)
- ~100 GB/s Memory Bandwidth

**Cori (Xeon Phi 7250; “Knights Landing”):**
- 9304 nodes (+ 1988)
- 68 physical cores/socket
- 272 logical cores/socket
- 1.4 GHz (up to 1.6 GHz in Turbo Mode)
- 32 double precision operations per cycle
- 16 GB of fast memory (MCDRAM)
- 96 GB of DDR4 DRAM
- ~115 GB/s DDR4 bandwidth
- MCDRAM has ~ 5x DDR4 bandwidth
Knights Landing: Next Intel® Xeon Phi™ Processor

First **self-boot** Intel® Xeon Phi™ processor that is **binary compatible** with main line IA. Boots standard OS.

**Significant improvement in scalar** and **vector performance**

Integration of **Memory on package**: innovative memory architecture for high bandwidth and high capacity

Integration of **Fabric on package**

---

**Three products**

- **KNL Self-Boot** (Baseline)
- **KNL Self-Boot w/ Fabric** (Fabric Integrated)
- **KNL Card** (PCIe-Card)

Potential future options subject to change without notice.

All timeframes, features, products and dates are preliminary forecasts and subject to change without further notification.
Knights Landing Overview

Chip: 36 Tiles interconnected by 2D Mesh
Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW
DDR4: 6 channels @ 2400 up to 384GB

IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset

Node: 1-Socket only
Fabric: Omni-Path on-package (not shown)

Vector Peak Perf: 3+TF DP and 6+TF SP Flops
Scalar Perf: ~3x over Knights Corner
Streams Triad (GB/s): MCDRAM: 400+; DDR: 90+

Source: Intel. All products, computer systems, dates and figures specified are preliminary based on current expectations and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX).

Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as the memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system configuration may affect actual performance.

Omni-path not shown
Knights Landing overview

**KNL Tile:** 2 Cores, each with 2 VPU
1M L2 shared between two Cores

**Core:** Changed from Knights Corner (KNC) to KNL. Based on 2-wide OoO Silvermont™ Microarchitecture, but with *many* changes for HPC.
4 thread/core. Deeper OoO. Better RAS. Higher bandwidth. Larger TLBs.

**2 VPU:** 2x AVX512 units. 32SP/16DP per unit. X87, SSE, AVX1, AVX2 and EMU

**L2:** 1MB 16-way. 1 Line Read and ½ Line Write per cycle. Coherent across all Tiles

**CHA:** Caching/Home Agent. Distributed Tag Directory to keep L2s coherent. MESIF protocol. 2D-Mesh connections for Tile
Knights Landing overview

**KNL ISA**

<table>
<thead>
<tr>
<th>E5-2600 (SNB)</th>
<th>E5-2600v3 (HSW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x87/MMX</td>
<td>x87/MMX</td>
</tr>
<tr>
<td>SSE*</td>
<td>SSE*</td>
</tr>
<tr>
<td>AVX</td>
<td>AVX</td>
</tr>
<tr>
<td>AVX2</td>
<td>AVX2</td>
</tr>
<tr>
<td>BMI</td>
<td>BMI</td>
</tr>
<tr>
<td>TSX</td>
<td></td>
</tr>
</tbody>
</table>

**KNL** (Xeon Phi)

- **KNL implements all legacy instructions**
  - Legacy binary runs w/o recompilation
  - KNC binary requires recompilation

- **KNL introduces AVX-512 Extensions**
  - 512-bit FP/Integer Vectors
  - 32 registers, & 8 mask registers
  - Gather/Scatter

  - **Conflict Detection**: Improves Vectorization
  - **Prefetch**: Gather and Scatter Prefetch
  - **Exponential and Reciprocal** Instructions

No TSX. Under separate CPUID bit.

Previous Code name Intel® Xeon® processors
Xeon Phi = Intel® Xeon Phi™ processor
Xeon Phi configurations

• Clustering mode
  – Chip can “look like” a 1, 2 or 4 socket Xeon node

• MCDRAM mode
  – Flat or cache

• These are choices at boot time and each combination has pros/cons
Knights Landing overview

**KNL Mesh Interconnect**

**Mesh of Rings**
- Every row and column is a (half) ring
- YX routing: Go in Y → Turn → Go in X
- Messages arbitrate at injection and on turn

**Cache Coherent Interconnect**
- MESIF protocol (F = Forward)
- Distributed directory to filter snoops

**Three Cluster Modes**
1. All-to-All
2. Quadrant
3. Sub-NUMA Clustering
Knights Landing overview

Cluster Mode: Quadrant

- Chip divided into four virtual Quadrants
- Address hashed to a Directory in the same quadrant as the Memory
- Affinity between the Directory and Memory
- Lower latency and higher BW than all-to-all. SW Transparent.

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return
Knights Landing overview

Cluster Mode: Sub-NUMA Clustering (SNC)

Each Quadrant (Cluster) exposed as a separate NUMA domain to OS.

Looks analogous to 4-Socket Xeon

Affinity between Tile, Directory and Memory

Local communication. Lowest latency of all modes.

SW needs to NUMA optimize to get benefit.

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return
Knights Landing overview

Memory Modes

Three Modes. Selected at boot

**Cache Mode**
- SW-Transparent, Mem-side cache
- Direct mapped. 64B lines.
- Tags part of line
- Covers whole DDR range

**Flat Mode**
- MCDRAM as regular memory
- SW-Managed
- Same address space

**Hybrid Mode**
- Part cache, Part memory
- 25% or 50% cache
- Benefits of both
Flat MCDRAM: SW Architecture

MCDRAM exposed as a separate NUMA node

- Memory allocated in DDR by default → Keeps non-critical data out of MCDRAM.
- Apps explicitly allocate critical data in MCDRAM. Using two methods:
  - "Fast Malloc" functions in High BW library (https://github.com/memkind)
  - Built on top to existing libnuma API
  - "FASTMEM" Compiler Annotation for Intel Fortran

Flat MCDRAM with existing NUMA support in Legacy OS
Flat MCDRAM SW Usage: Code Snippets

### C/C++

(*[https://github.com/memkind](https://github.com/memkind)*)

<table>
<thead>
<tr>
<th>Allocate into DDR</th>
<th>Allocate into MCDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>float *fv;</code>&lt;br&gt;<code>fv = (float *)malloc(sizeof(float)*100);</code></td>
<td><code>REAL, ALLOCATABLE :: A(:)</code>&lt;br&gt;!DEC$ ATTRIBUTES, FASTMEM :: A&lt;br <code>NSIZE=1024</code>&lt;br&gt;<code>allocate array ‘A’ from MCDRAM</code>&lt;br&gt;<code>ALLOCATE (A(1:NSIZE))</code></td>
</tr>
</tbody>
</table>

### Intel Fortran

<table>
<thead>
<tr>
<th>Allocate into MCDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>REAL, ALLOCATABLE :: A(:)</code>&lt;br&gt;!DEC$ ATTRIBUTES, FASTMEM :: A&lt;br <code>NSIZE=1024</code>&lt;br&gt;<code>allocate array ‘A’ from MCDRAM</code>&lt;br&gt;<code>ALLOCATE (A(1:NSIZE))</code></td>
</tr>
</tbody>
</table>
MCDRAM Modes

- **Cache mode**
  - No source changes needed to use
  - Misses are expensive (higher latency)
    - Needs MCDRAM access + DDR access

- **Flat mode**
  - MCDRAM mapped to physical address space
  - Exposed as a NUMA node
    - Use `numactl --hardware`, `lscpu` to display configuration
  - Accessed through memkind library or `numactl`

- **Hybrid**
  - Combination of the above two
    - E.g., 8 GB in cache + 8 GB in Flat Mode
Multi-channel DRAM overview

MCDRAM as Cache

- **Upside**
  - No software modifications required
  - Bandwidth benefit (over DDR)

- **Downside**
  - Higher latency for DDR access
    - i.e., for cache misses
  - Misses limited by DDR BW
  - All memory is transferred as:
    - DDR -> MCDRAM -> L2
  - Less addressable memory

MCDRAM as Flat Mode

- **Upside**
  - Maximum BW
  - Lower latency
    - i.e., no MCDRAM cache misses
  - Maximum addressable memory
  - Isolation of MCDRAM for high-performance application use only

- **Downside**
  - Software modifications (or interposer library) required
  - to use DDR and MCDRAM in the same app
  - Which data structures should go where?
  - MCDRAM is a finite resource and tracking it adds complexity
Accessing MCDRAM in Flat Mode

- **Option A: Using `numactl`**
  - Works best if the whole app can fit in MCDRAM

- **Option B: Using libraries**
  - **Memkind Library**
    - Using library calls or Compiler Directives (Fortran*)
    - Needs source modification
  - **AutoHBW (interposer library based on memkind)**
    - No source modification needed (based on size of allocations)
    - No fine control over *individual* allocations

- **Option C: Direct OS system calls**
  - `mmap(1), mbind(1)`
  - Not the preferred method
    - Page-only granularity, OS serialization, no pool management

*Other names and brands may be claimed as the property of others.*
Option A: Using `numactl` to Access MCDRAM

- MCDRAM is exposed to OS/software as a NUMA node
- Utility `numactl` is standard utility for NUMA system control
  - See “man numactl”
  - Do “numactl --hardware” to see the NUMA configuration of your system
- If the total memory footprint of your app is smaller than the size of MCDRAM
  - Use `numactl` to allocate all of its memory from MCDRAM
  - `numactl --membind=mcdrum_id <your_command>`
    - Where `mcdrum_id` is the ID of MCDRAM “node”
- If the total memory footprint of your app is larger than the size of MCDRAM
  - You can still use `numactl` to allocate part of your app in MCDRAM
    - `numactl --preferred=mcdrum_id <your_command>`
      - Allocations that don’t fit into MCDRAM spills over to DDR
    - `numactl --interleave=nodes <your_command>`
      - Allocations are interleaved across all `nodes`
Multi-channel DRAM overview

Option B.1: Using Memkind Library to Access MCDRAM

Allocate 1000 floats from DDR

```c
float *fv;
fv = (float *)malloc(sizeof(float) * 1000);
```

Allocate 1000 floats from MCDRAM

```c
#include <hbwmalloc.h>
float *fv;
fv = (float *)hbw_malloc(sizeof(float) * 1000);
```

Allocate arrays from MCDRAM and DDR in Intel® Fortran Compiler

```fortran
!DEC$ ATTRIBUTES FASTMEM :: A
REAL, ALLOCATABLE :: A(:), B(:), C(:)
NSIZE=1024
ALLOCATE (A(1:NSIZE))
ALLOCATE (B(NSIZE), C(NSIZE))
```

Allocate array ‘A’ from MCDRAM

```fortran
ALLOCATE (A(1:NSIZE))
```

Allocate arrays that will come from DDR

```fortran
ALLOCATE (B(NSIZE), C(NSIZE))
```
Option B.2: AutoHBW

- AutoHBW: Interposer Library that comes with memkind
  - Automatically allocates memory from MCDRAM
    - If a heap allocation (e.g., malloc/calloc) is larger than a given threshold

- Demo
  - see /examples/autohbw_test.sh
  - Run gpp with AutoHBW

- Environment variables (see autohbw_README)
  - AUTO_HBW_SIZE=x[y]
  - AUTO_HBW_LOG=level
  - AUTO_HBW_MEM_TYPE=memory_type  # useful for interleaving

- Finding source locations of arrays
  - export AUTO_HBW_LOG=2
  - ./app_name > log.txt
  - autohbw_get_src_lines.pl log.txt app_name
Advanced Topic: MCDRAM in SNC4 Mode

- **SNC4**: Sub-NUMA Clustering
  - KNL die is divided into 4 clusters (similar to a 4-Socket Haswell)
  - SNC4 configured at boot time
  - Use `numactl --hardware` to find out nodes and distances
    - There are 4 DDR (+CPU) nodes + 4 MCDRAM (no CPU) nodes, in flat mode

- Running 4-MPI ranks is the easiest way to utilize SNC4
  - Each rank allocates from closest DDR node
    - If a rank allocates MCDRAM, it goes to closest MCDRAM node

- If you run only 1 MPI rank and use `numactl` to allocate on MCDRAM
  - Specify all MCDRAM nodes
    - E.g., `numactl --hardware`
Observing MCDRAM Memory Allocations

- Where is MCDRAM usage printed?
  - `numastat -m`
    - Printed for each NUMA node
    - Includes Huge Pages info
  - `numastat -p <pid>` OR `numastat -p exec_name`
    - Info about process <pid>
    - E.g., `watch -n 1 -p numastat exec_name`
  - `cat /sys/devices/system/node/node*/meminfo`
    - Info about each NUMA node
  - `cat /proc/meminfo`
    - Aggregate info for system

- Utilities that provide MCDRAM node info
  - `<memkind_install_dir>/bin/memkind-hbw-nodes`
  - `numactl --hardware`
  - `lscpu`
MCDRAM Emulation Demo

- Use cores on socket A and DDR on socket B (remote memory)
  - `export MEMKIND_HBW_NODES=0` # HBW allocation go to DDR on Socket A
  - `numactl --membind=1 --cpunodebind=0 <your_command>`

- Any HBW allocations will allocate DDR on socket A
  - Accesses to DDR on socket A (local memory) has higher BW
    - Also has lower latency (which is an inaccuracy)