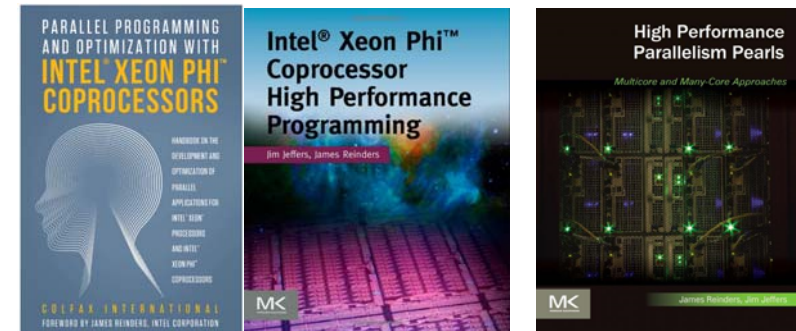


Xeon Phi Architecture #1

KISTI Supercomputing Center
Hong Suk Yi
(hsyi@kisti.re.kr)

1

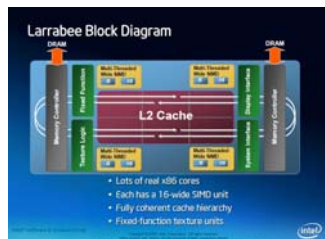
Reference : Book



<http://lotsofcores.com/PearlsExpected20141117>

2

Xeon Phi: What is it?



- Lots of names
 - ✓ Many Integrated Core (MIC) architecture
 - Comparable to Intel 64 on CPU
 - ✓ Knights Corner (code name)
 - ✓ Xeon Phi Coprocessor (product name)
 - ✓ Different models have number designations
 - 5100P, 3120A, 7120
- Co-processor
 - ✓ PCI Express card
 - ✓ Stripped down Linux operating system
- Dense processor
 - ✓ Many power-hungry operations removed
 - ✓ Wider vector unit
 - ✓ Wider hardware thread count

3

MIC (Xeon Phi) History



- Many Integrated Core Architecture
 - ✓ Intel MIC is a multiprocessor computer architecture developed by Intel
 - http://en.wikipedia.org/wiki/Intel_MIC
- Timeline
 - ✓ 2006 Larrabee many core architecture
 - ✓ 2010 05 Knights Ferry (prototype) board.
 - 45nm, 512b, 750 GF SP
 - Initial Developers : CERN, KISTI, LSC
 - ✓ 2011 11 Knights Corner
 - 22nm, 512b
 - ✓ 2012 06 Xeon Phi
 - rebranded at ISC
 - ✓ 2012 11 Xeon Phi
 - ✓ 2015? Knights Landing

4

A Tale of Two Architectures

➤ Each core has a new SIMD 512-bit wide VPU

✓ 8 double (or 16 float/integer) op/cy

- Different from MMX, SSE, AVX, AVX2 which are used on x86 CPUs!

➤ Fused Multiply-Add (FMA)

✓ $A = A + (B * C)$ in a single cycle

5

Vector register

Scalar: 64(80)-bit wide register, 1 double/ instruction
add [a] [b]

Intel SSE: 128-bit wide register(xmm), 2 doubles/ instruction
SSE2, SSE4.2

Intel AVX: 256-bit wide register(ymm), 4 doubles/ instruction
Shipped with Sandy bridge
Later, AMD will implement.

Intel LRB: 512-bit wide register(), 8 doubles/ instruction
Shipped with Knights Corner at 2012

Intrinsic: `_mm{bit}_{operation}_{packed, scalar}{precision}()`
ex. `_mm128_add_pd()`, `_mm128_exp_pd()` on icc

Compiler auto-vectorization, C Extension for Array Notation(Cilk+), ArBB(C++)

OpenCL: easy vectorization combined with multi-threading

6

Xeon Phi 7110



http://www.cpu-world.com/CPUs/Xeon_Phi/Intel-Xeon%20Phi%207120X.html

Type	Floating Point Unit / Co-processor
Family	Intel Xeon Phi 7100 series
Model number	7120X
Frequency	1250 MHz
Package	Card with PCI Express x16 connector
Socket	PCI Express x16
Release date	May 2013
Microarchitecture	Many Integrated Cores
Processor core	Knights Corner
Manufacturing process	22 nano
Data width	64 bit
# number of cores	61
# of threads	244
Level 1 cache	61 x 32 KB 8-way caches; (instruction and data)
Level 2 cache	61 x 512 KB 8-way caches
Cache latency	1 (L1 cache); 11 (L2 cache)
Features	SIMD instructions; Fused Multiply-Add
On-chip peripherals	8 dual-channel 32-bit GDDR5 controllers; PCIe 2.0

7

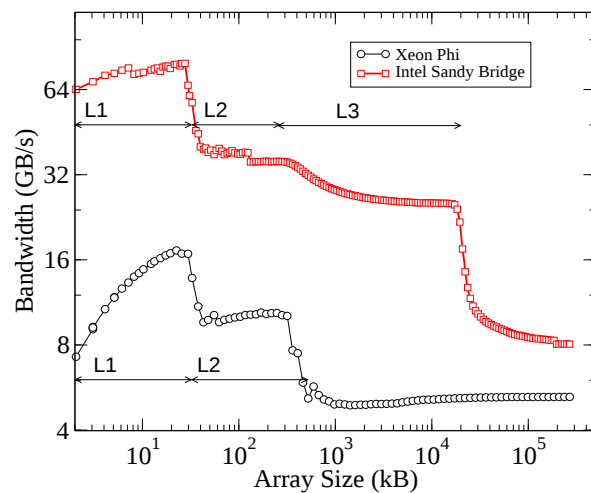
No driving force for performance

$$\text{Perf.} = \text{nCores} * \text{SIMD} * \text{Freq} * \text{FMA}$$

	Xeon Phi	Sandy bridge
nCore	61	16
SIMD (DP)	8	4
Freq	1.09	2.59
FMA	2	2
Execution Style	In-order	Out-of-order
Thread switching	Round Robin	Hyper Threading
Gflops	1063.8	331.5
Bandwidth (GB/s)	350	102

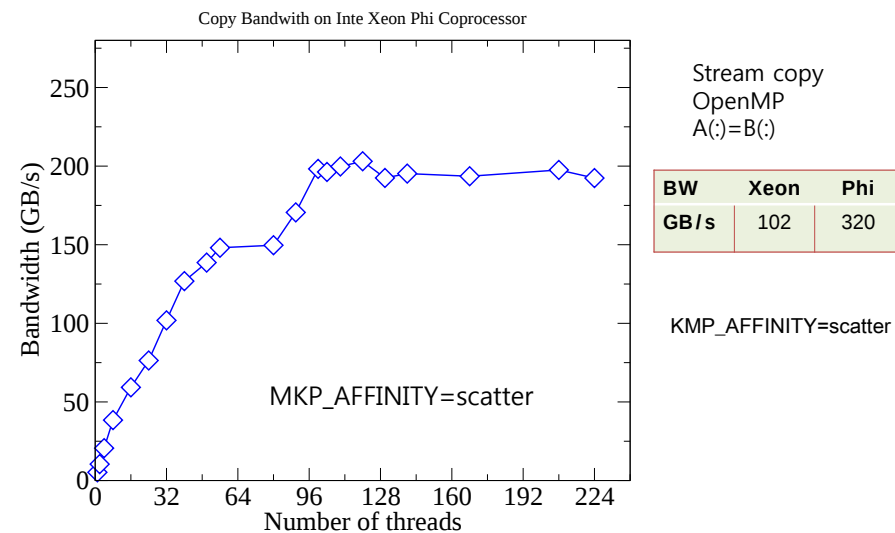
8

Copy bandwidth on Intel Phi



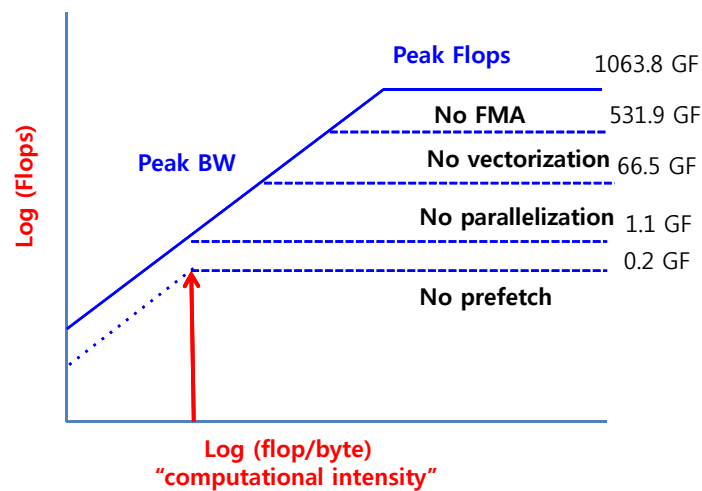
9

Copy Bandwidth on Intel Xeon Phi Coprocessor



10

Roofline Model for Xeon Phi (DP)



11

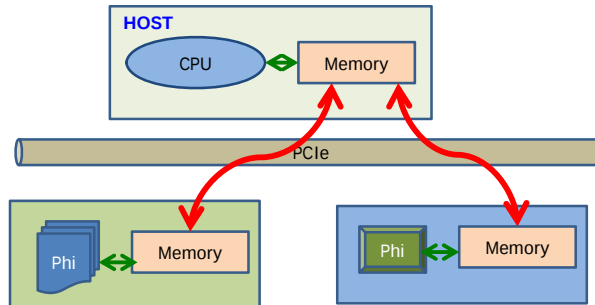
Building a Native Application #2

KISTI Supercomputing Center
Hongsuk Yi
(hsyi@kisti.re.kr)

12

Data transfer for offload

- OpenMP programming model for data transfer
 - Host CPU and MIC do not share memory
 - Add pragmas similar to OpenMP
 - #pragma offload target (mic:0)



13

Native Introduction

➤ Offload 없이 Xeon Phi에서 바로 계산

➤ Basic steps

1. Determine if the application is suitable for native execution.
2. Compile the application for native execution.
3. Build required libraries for native execution.
4. Copy the executable and any dependencies, such as runtime libraries, to the target hardware.
5. Mount file shares to the target hardware for accessing input data sets and saving output data sets.
6. Connect to the target hardware via console, set up the environment, and run the application.

14

1. Suitability for Native Execution

- A modest memory footprint, less than the available physical memory on the device
- Very few serial segments
- Does not perform extensive I/O
- A complex code structure with no well-identified hot kernels that could be offloaded without substantial data transfer overhead

15

2. Compiling a Native Application

➤ Compiling

- ✓ `source /opt/intel/composer_xe_2013/bin/compilervars.sh intel64`
- ✓ Verify that the environment is set correctly by running
 - `icc -V` (C compiler)
 - `icpc -V` (C++ compiler)
 - `ifort -V` (Fortran compiler).
- ✓ Invoke the compiler and include the `-mmic` option in the build line.
 - `$ icc -mmic mycode.c`
 - `$ icpc -mmic mycode.cpp`
 - `$ ifort -mmic mycode.f90`
- ✓ The default optimization level for the Intel compilers is `-O2` when no `-O` option is used.

16

3. Building Libraries

- Compile the library source code
✓\$ `icc -mmic mylib.c`
- Use the compiler -shared option to create the library file from the object file(s).
✓\$ `icc -mmic -shared -o libmylib.so mylib.o`
- Compile and link the native application code with the native shared object.
✓\$ `icc -mmic main.c libmylib.so`

17

Finding Dependencies

need to transfer shared objects that are required by the compiler runtime system
`/opt/intel/composer_xe_2013_sp1.X.XXX/compiler/lib/mic/`

Linux Library	Description
libcilkrts.so.5	Intel® Cilk™ Plus runtime library
libifcoremt.so.5	Thread-safe Intel-specific Fortran run-time library
libifport.so.5	Portability and POSIX support
libimf.so	Math library
libintlc.so.5	Intel support libraries for CPU dispatch, intel_fast_*, and traceback support routines
libiomp5.so	Compatibility OpenMP* dynamic runtime library
libsvml.so	Short vector math library
libirng.so	Random number generator library
libmpi.so.4.0.18; libmpigf.so.4.0; libmpigc4.so.4.0	Intel® MPI runtime libraries
libicaf.so	Coarray Fortran library

18

Using micnativeloadex

- The micnativeloadex utility, when used with option -l, will list shared library dependency information.
 - \$ export
SINK_LD_LIBRARY_PATH=/opt/intel/composer_xe_2013_sp1.X.XXX/compiler/lib/mic:/home/user1/myproject
- ✓Run the utility with the -l option.
 - \$ /opt/intel/mic/bin/micnativeloadex a.out -l

19

Transferring Files

- Connect to the card using ssh and create a /tmp
✓\$ `ssh mic0 'mkdir /tmp/myname'`
- Copy and dependencies
✓\$ `scp ./a.out mic0:/tmp/myname`

20

Running the Application

- Two methods
 - ✓ Manually copy the application and its dependent files to the MIC
 - ✓ login and then invoke the program.
- Manually
 - ✓ \$ ssh mic0
 - ✓ # chmod +x a.mic
 - ✓ # export LD_LIBRARY_PATH=/tmp/home:\$LD_LIBRARY_PATH
 - ✓ # ./a.mic

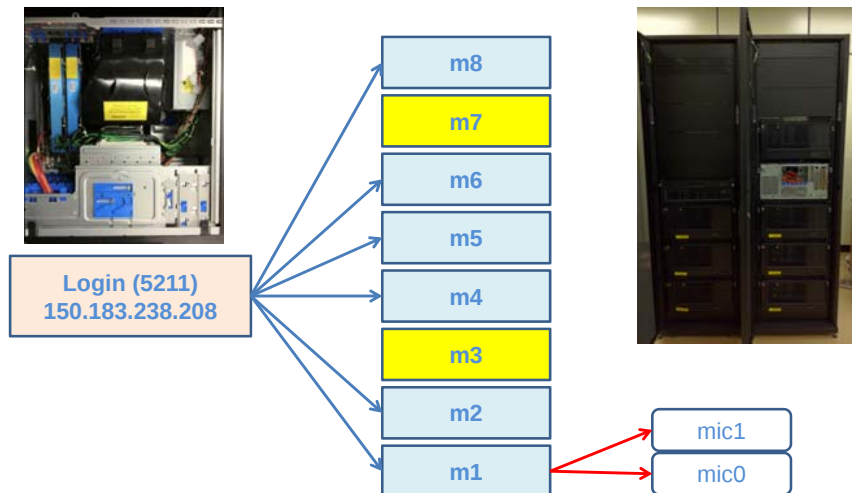
21

Method 2 : using micnativeloadex

- Use micnativeloadex
 - ✓ \$ export SINK_LD_LIBRARY_PATH=
 - ✓ /opt/intel/composer_xe_2013_sp1.0.080/compiler/lib/mic/
 - ✓ \$ /opt/intel/mic/bin/micnativeloadex a.out

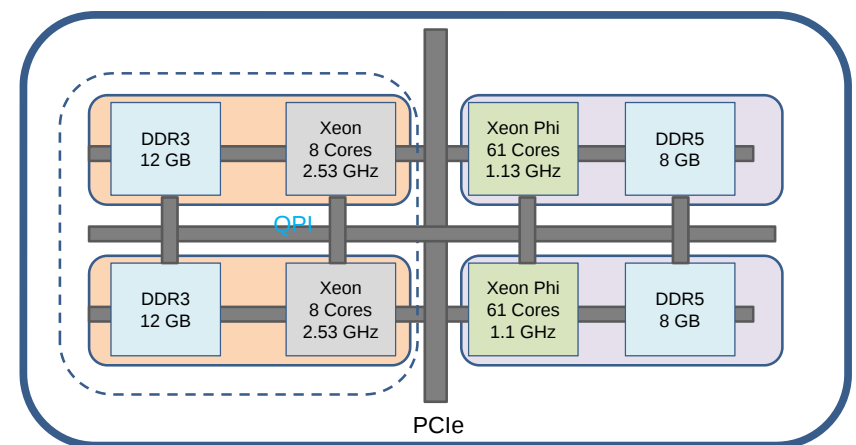
22

KISTI Phi Cluster



23

Computing Node



24

Examples

➤ Library copy

- `scp /opt/intel/composer_xe_2013.0.079/compiler/lib/mic/libiomp5.so mic0:~/.`
- `export LD_LIBRARY_PATH=/home/mic/edun49:$LD_LIBRARY_PATH`
- `source /opt/intel/composer_xe_2013/bin/compilervars.sh intel64`

➤ Example

- ✓ `icc -mmic -openmp -vec-report=3 saxpy-v3.c -o run3.x.phi`
- ✓ `icc -openmp -vec-report=3 saxpy-v3.c -o run3.x.cpu`

➤ Using micnativeloadex

- ✓ `export SINK_LD_LIBRARY_PATH`
- ✓ `micnativeloadex run.x`

25

/proc/cpuinfo

\$ ssh mic0

\$ cat /proc/cpuinfo | tail -26

```
processor       : 243
vendor_id      : GenuineIntel
cpu family     : 11
cpu model      : 1
model name     : 0b/01
stepping       : 1
cpu MHz        : 1090.908
cache size     : 512 KB
physical id    : 0
siblings       : 244
core id        : 60
cpu cores      : 61
apicid         : 243
initial apicid : 243
fpu            : yes
fpu_exception : yes
cpuid level    : 4
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic mtrr mca
                pat fxsr ht syscall lm rep_good nopl lahf_lm
bogomips       : 2190.07
clflush size   : 64
cache_alignment : 64
address sizes  : 40 bits physical, 48 bits virtual
power management:
```

```
~ $ cat /proc/cpuinfo | tail -26
processor       : 227
vendor_id      : GenuineIntel
cpu family     : 11
cpu model      : 1
model name     : 0b/01
stepping       : 3
cpu MHz        : 1100.000
cache size     : 512 KB
physical id    : 0
siblings       : 228
core id        : 56
cpu cores      : 57
apicid         : 227
initial apicid : 227
fpu            : yes
fpu_exception : yes
cpuid level    : 4
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 ap
bogomips       : 2208.15
clflush size   : 64
cache_alignment : 64
address sizes  : 40 bits physical, 48 bits virtual
power management:
```

26

Lab 1 : Saxpy

➤ Step 2 : aligned

- `a = (REAL*)`
`_mm_malloc(sizeof(REAL)*SIZE,64);`
- `b = (REAL*)`
`_mm_malloc(sizeof(REAL)*SIZE,64);`

➤ Native mode

- ✓ **Compile**
 - `icc -mmic a.mic simple.c`
- ✓ **Run 1**
 - `Micnativeloadex ./a.mic`
- ✓ **Run 2**
 - `Scp ./a.mic mic0:~/.`
 - `Ssh mic0`
 - `./a.mic`

SAXPY vs DAXPY (02aligned)			
SIZE	Memory	secs	Gflops
1000			
2000			
10000			
100000			
1000000			
10000000			

27