

MIC Programming with Intel® Xeon Phi™ Coprocessors

- Overview on Its Validity for Realistic Scientific Computing -

Hoon Ryu, Ph.D.

Principal Investigator, (E: elec1020@kisti.re.kr)

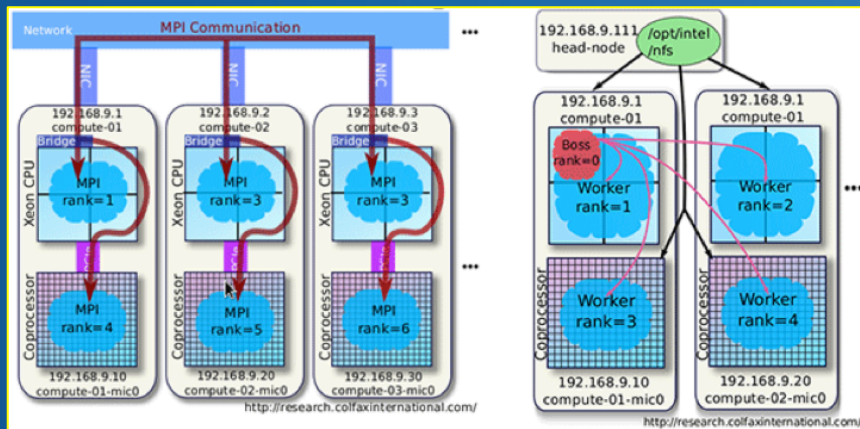
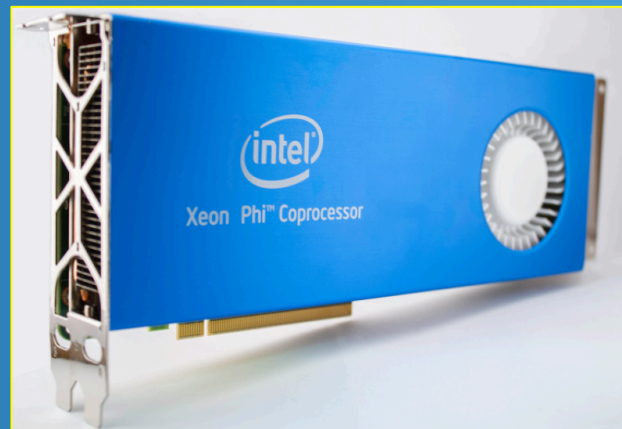
Intel® Parallel Computing Center @

Korea Institute of Science and Technology Information

Environment for MIC Programming

Intel® Xeon Phi Coprocessors and the MIC Architecture

- PCIe end-point device (KNC: card-type)
- High power efficiency
- ~1 TFlops/sec in double-precision ops.
- Heterogeneous clustering



- (May be) beneficial for highly parallel applications reaching the scaling limit on multicores
- KNL? Later in this talk

From white paper in [coalfaxinternational.com](http://research.coalfaxinternational.com/)

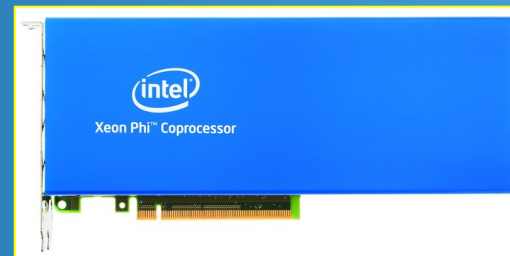
Environment for MIC Programming

Intel® Xeon Phi Coprocessors and the MIC Architecture (cont'd)



Intel® Xeon™ CPUs

- C/C++/Fortran; OpenMP/MPI
- Standard Linux OS
- Up to 768GB of DDR3 (D)RAM
- ≤ 12 cores per socket ~ 3 GHz
- 2-way hyper-threading



Intel® Xeon Phi™ Coprocessors

- C/C++/Fortran; OpenMP/MPI
- Special Linux μ OS distribution
- 6-16GB cached GDDR5 (SD)RAM
- 57~61 cores per card ~ 1 GHz
- 4 hardware threads per core

Environment for MIC Programming

Linux μ OS on Intel® Xeon Phi™ Coprocessors (part of MPSS)

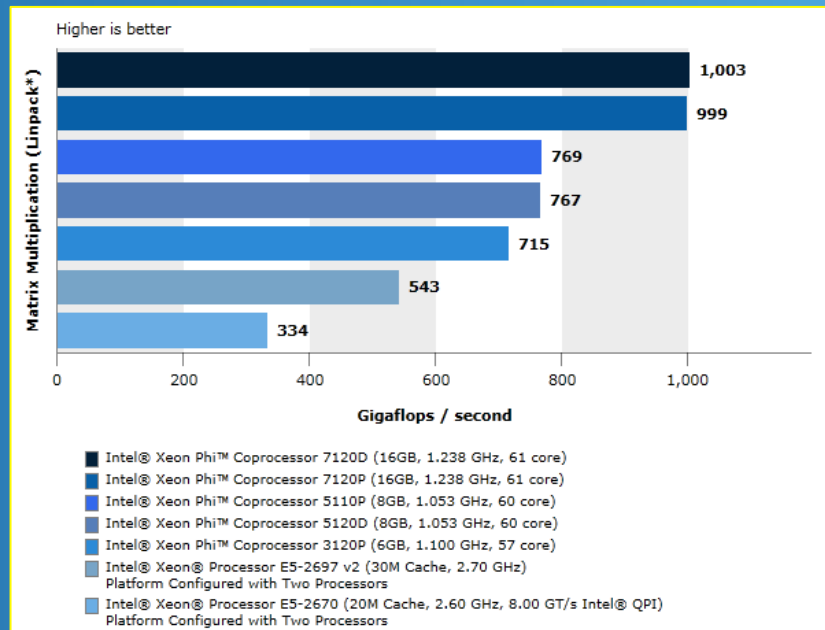
- Figuring out Xeon Phi™ coprocessors in host servers

```
user@host% lspci | grep -i "co-processor"
06:00.0 Co-processor: Intel Corporation Xeon Phi coprocessor 3120 series (rev 20)
82:00.0 Co-processor: Intel Corporation Xeon Phi coprocessor 3120 series (rev 20)
user@host% sudo service mpss status
mpss is running
user@host% cat /etc/hosts | grep mic
172.31.1.1 host-mic0 mic0
172.31.2.1 host-mic1 mic1
user@host% ssh mic0
user@mic0% cat /proc/cpuinfo | grep proc | tail -n 3
processor : 237
processor : 238
processor : 239
user@mic0% ls /
amplxe dev home lib64 oldroot proc sbin sys usr
bin etc lib linuxrc opt root sep3.10 tmp var
```


A Glance at Performance in Dense Arithmetic

Linpack in Xeon™ and Xeon Phi™ Family Product

- (Dense) Matrix Multiplication BMT with Linpack
- Performance comparison
 - CPU only vs. Single coprocessor card
 - Up to ~3x speed-up
- Single code for two platforms
 - Easy Porting
 - Incremental optimization
- Performance may be worse without careful consideration in numerical analysis; i.e.; sparse-matrix ops.



<https://www-ssl.intel.com/content/dam/www/public/us/en/images/charts/chart-id-635.png>

Vectorization: A Rough Concept

SIMD Operations

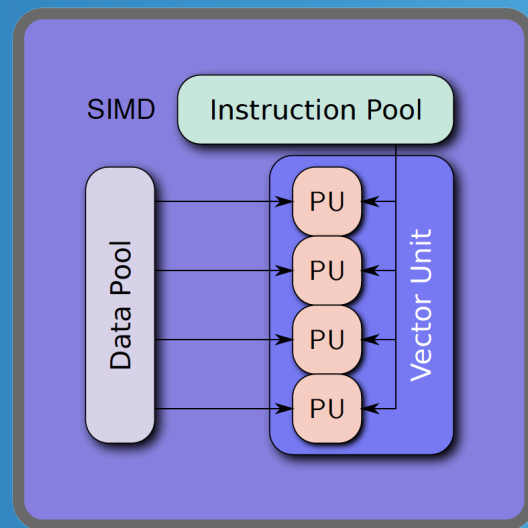
- SIMD: **S**ingle **I**nstruction **M**ultiple **D**ata
- Each SIMD (arithmetic) operator acts on 16 (SP) numbers at a time
 - DP (Double Precision): 8 numbers at a time

Scalar loop

```
01 for (i = 0; i < n; i++)
02     A[i] = A[i] + B[i];
```

SIMD loop (for SP ops.)

```
01 for (i = 0; i < n; i += 16 )
02     A[i:(i+16)] = A[i:(i+16)] + B[i:(i+16)];
```



Parallel access to data with
a single instruction

Vectorization: SIMD Instruction Sets

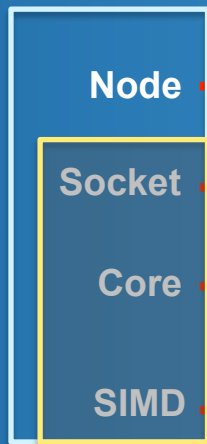
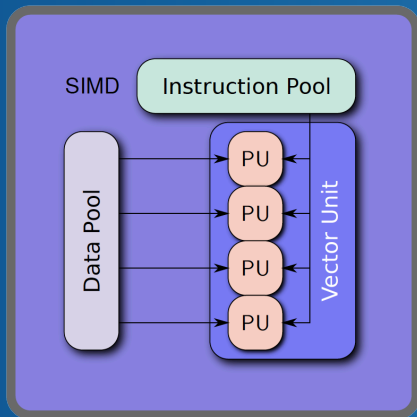
Instruction Sets in Intel® Architectures

Instruction Set	Year / Intel® Proc.	Vector Register	Packed Data Types
MMX	1997 / Pentium	64-bit	8-, 16- and 32-bit INT
SSE	1999 / Pentium III	128-bit	32-bit SP
SSE2	2001 / Pentium IV	128-bit	8 to 64-bit INT; SP & DP
SSE3-SSE4.2	2004 - 2009	128-bit	Additional instructions
AVX	2011 / Sandy Bridge	256-bit	SP & DP
AVX2	2013 / Haswell	256-bit	INT, Additional instructions
IMCI	2012 / KNC Xeon Phi	512-bit	32- and 64-bit INT; SP & DP
AVX-512	(2016) / KNL Xeon Phi	512-bit	32- and 64-bit INT; SP & DP

How to Parallelize?

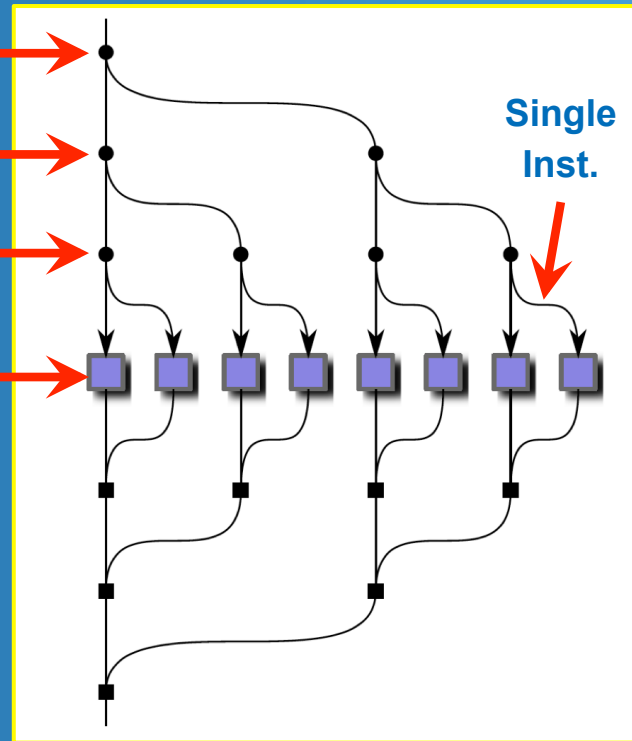
Conceptual Flow of Parallelization of Large-scale Computation

SIMD Unit



MPI (openMP)

Conceptual Flow for Parallelization

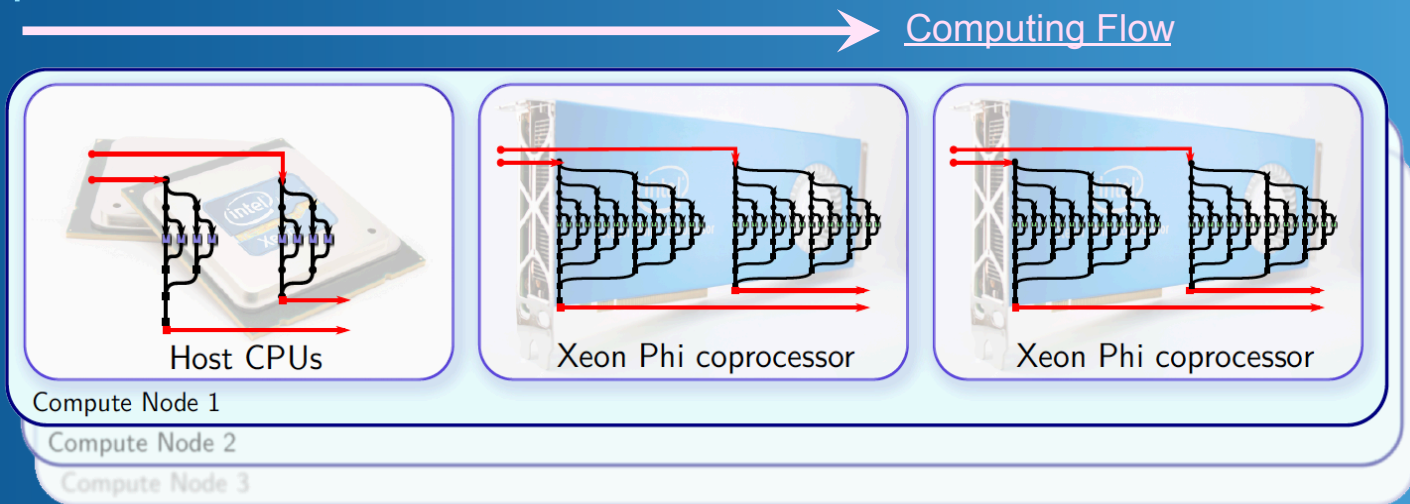


■ Very Rough Concept: Assume

- 2 nodes (definitely needs MPI)
- 2 sockets per node, 2 cores per socket (either MPI or openMP)

In Xeon/Xeon Phi?

Conceptual Flow of Parallelization in Xeon Phi™



- Inter-node or Inter-multicore communication: MPI (solid red line)
- Additional SIMD units: Processors and Coprocessors (Xeon Phi™)
- Excellent for dense matrix operation or EP-type tasks: Vectorization
 - Still nice for sparse-data-involved PDE problems – Will be back

Execution Modes

Native, Offload and Symmetric Modes

- Native Mode

Host (Main CPUs)



Coprocessors

```
main() {  
  
    myFuntion();  
  
}
```

- Offload Mode

Host (Main CPUs)

```
main() {  
  
    #pragma offload target(mic)  
  
}
```

Coprocessors

```
myFuntion();
```

Native Mode

Running Applications on Coprocessors

- A Simple “Hello World” Application (hello.c)

```
01 #include <stdio.h>
02 #include <unistd.h>
03 int main(){
04     printf("Hello world! I have %ld logical cores.\n", \
05         sysconf(_SC_NPROCESSORS_ONLN));
06 }
```

- Returns # of processors currently online

- Compile and Run on Host

```
user@host% icc hello.c
user@host% ./a.out
Hello world! I have 32 logical cores.
user@host%
```


Native Mode

Running Applications on Coprocessors (cont'd)

- Compile and Run on Coprocessors in Native Mode

```
user@host% icc hello.c -mmic
user@host% scp a.out mic0:~/
a.out 100% 10KB 10.4KB/s 00:00
user@host% ssh mic0
user@mic0% pwd
/home/user
user@mic0% ls
a.out
user@mic0% ./a.out
Hello world! I have 244 logical cores.
user@mic0%
```

- **Xeon Phi 7120a**
 - 61 physical cores
 - 4 hardware threads

- “-mmic” Option: Runs on coprocessors; Compile and link on host
- Executable MUST be transferred to coprocessor cards
- Native MPI applications work the same way (needs Intel MPI)

Native Mode

Super-easy Porting of User Applications for Native Execution

- Simple CPU applications can be compiled for native execution
 - By presenting the flag “-mmic” to the Intel compiler
 - So far just for native executions: Performance improvement normally requires additional works for optimization.

```
user@host% icpc -c myobject1.cc -mmic
user@host% icpc -c myobject2.cc -mmic
user@host% icpc -o myapplication myobject1.o myobject2.o -mmic
```

- Same for coprocessor-only MPI applications

```
user@host% mpiicpc -c myobject1.cc -mmic
user@host% mpiicpc -c myobject2.cc -mmic
user@host% mpiicpc -o myapplication myobject1.o myobject2.o -mmic
```

Native Mode

Sample Code: HelloMPI.c

```
01  #include <stdlib.h>
02  #include <stdio.h>
03  #include "mpi.h"
04  #define MASTER 0
05  #define TAG_HELLO 4
06
07  void main(int argc, char *argv[])
08  {
09      int i, id, remote_id, num_procs, namelen;
10      MPI_Status stat;
11      char name[MPI_MAX_PROCESSOR_NAME];
12
13      if (MPI_Init (&argc, &argv) != MPI_SUCCESS)
14      {
15          printf ("Failed to initialize MPI\n");
16          return;
17      }
18
19      MPI_Comm_size (MPI_COMM_WORLD, &num_procs);
20      MPI_Comm_rank (MPI_COMM_WORLD, &id);
21      MPI_Get_processor_name (name, &namelen);
```

Native Mode

Sample Code: HelloMPI.c (cont'd)

```
22     if (id == MASTER)
23     {
24         printf ("Hello world: rank %d of %d running on %s\n", id,num_procs,name);
25         for (i = 1; i<num_procs; i++)
26         {
27             MPI_Recv(&remote_id,1,MPI_INT,i,TAG_HELLO,MPI_COMM_WORLD,&stat);
28             MPI_Recv(&num_procs,1,MPI_INT,i,TAG_HELLO,MPI_COMM_WORLD,&stat);
29             MPI_Recv(&namelen,1,MPI_INT,i,TAG_HELLO,MPI_COMM_WORLD,&stat);
30             MPI_Recv(name,namelen+1,MPI_CHAR,i,TAG_HELLO,MPI_COMM_WORLD, &stat);
31             printf("Hello world: rank %d of %d running on %s\n", remote_id, \
32                 num_procs,name);
33         }
34     }
35     else
36     {
37         MPI_Send (&id, 1, MPI_INT, MASTER, TAG_HELLO, MPI_COMM_WORLD);
38         MPI_Send (&num_procs, 1, MPI_INT, MASTER, TAG_HELLO, MPI_COMM_WORLD);
39         MPI_Send (&namelen, 1, MPI_INT, MASTER, TAG_HELLO, MPI_COMM_WORLD);
40         MPI_Send (name, namelen+1, MPI_CHAR, MASTER, TAG_HELLO, MPI_COMM_WORLD);
41     }
42 }
```

Native Mode

Running MPI Applications on Coprocessors

- Copy or NFS-share Intel® MPI library to (with) coprocessors
- Enable I_MPI_MIC
 - Set to either 1 or “enable” to use Intel® MPI in coprocessors
- Compile and link the code with `–mmic` option and Intel® MPI (host)
- Copy executable to coprocessors
- Run the executable as if mic is a remote host.

```
user@host% export I_MPI_MIC=1
user@host% mpiicpc -mmic -o HelloMPI.MIC HelloMPI.c
user@host% scp HelloMPI.MIC mic0:~/
user@host% mpirun -host mic0 -np 2 ~/HelloMPI.MIC
Hello world: rank 0 of 2 running on host-mic0
Hello world: rank 1 of 2 running on host-mic0
```

Symmetric Mode

Running MPI Applications on both Host and Coprocessors

- Copy or NFS-share Intel® MPI library to (with) coprocessors
- Home and it's sub-directories in host and mic should be synchronized
- Enable I_MPI_MIC and set I_MPI_MIC_POSTFIX, e.g. to '.mic'
 - I_MPI_MIC_POSTFIX: the postfix to be added to the name of executable
- Compile and link the code Intel® MPI in two ways (host)
 - For executable in host: With a normal way (name of executable: HelloMPI)
 - For executable in mic: With -mmic option (name of executable: HelloMPI.mic)
- Copy mic-executable to coprocessors
 - Path should be identical to what has cpu-executable
- Create a host file and run on host with host-executable

Symmetric Mode

Running MPI Applications on both Host and Coprocessors (cont'd)

```
user@host% export I_MPI_MIC=1
user@host% export I_MPI_MIC_POSTFIX=.mic
user@host% mpiicc HelloMPI.c -o HelloMPI
user@host% mpiicc -mmic HelloMPI.c -o HelloMPI.mic
user@host% scp HelloMPI.mic host-mic0:~/
user@host% cat host_file
host
host-mic0
user@host% mpirun -hostfile host_file -np 2 ~/HelloMPI
Hello world: rank 0 of 2 running on host
Hello world: rank 1 of 2 running on host-mic0
```

ex: host_file
host:2
host-mic0:2

- # of ranks can be controlled: one way with host_file
- More diverse ways and details for MPI runs in symmetric mode
 - <https://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems>

Offload Mode

Explicit Offload Model

- Revisit to “Hello World”: In an explicit offload mode (hello_offload.cpp)

```
01 #include <stdio.h>
02
03 int main(int argc, char * argv[] ) {
04
05     printf("Hello World from host!\n");
06
07     #pragma offload target(mic)
08     {
09         printf("Hello World from coprocessor!\n"); fflush(0);
10     }
11
12     printf("Bye\n");
13 }
```

- Application runs on host
 - Some parts of code and data are moved (“offloaded”) to coprocessors

Offload Mode

Compiling and Running an Offload Application

- Compile, link and execute on host

```
user@host% icpc hello_offload.cpp -o hello_offload
user@host% ./hello_offload
Hello World from host!
Bye
Hello World from coprocessor!
```

- Code inside **#pragma offload** runs in Xeon Phi™ coprocessors
- Console output on coprocessors is buffered and mirrored to host
- With no coprocessor installed,
 - Code inside **#pragma offload** may fall back to host
 - Compiler complains “unknown preprocessor **#pragma offload**” with an warning
 - Easily to make the code be versatile in both CPU-only and offload mode

Offload Mode

Sample Code: offloadExample.c

```
01  #include <stdio.h>
02  #include <unistd.h>
03
04  #define ALLOC alloc_if(1) free_if(0)
05  #define REUSE alloc_if(0) free_if(0)
06  #define FREE  alloc_if(0) free_if(1)
07  #define LOCAL alloc_if(1) free_if(1)
08
09  int main(){
10
11      int *ncore = (int*)malloc(sizeof(int)*2);
12      int phi_tid = 0;
13
14      ncore[0] = sysconf(_SC_NPROCESSORS_ONLN); // # of cores in host
15      ncore[1] = 0; // initialize
16
17      printf("[HOST] # of CPUs in (host, mic): (%d, %d)\n", ncore[0], ncore[1]);
```

Offload Mode

Sample Code: offloadExample.c (cont'd)

```

18  #pragma offload target(mic:phi_tid) \
19      in(ncore[0:2]                : ALLOC) \
20      out(ncore[1:1]                : REUSE)
21      {
22          ncore[1] = sysconf(_SC_NPROCESSORS_ONLN); // # of cores in mic
23      }
24      printf("[HOST] # of CPUs in (host, mic): (%d, %d)\n", ncore[0], ncore[1]);
25
26      ncore[0] = -1; ncore[1] = -1;
27
28  #pragma offload target(mic:phi_tid) \
29      in(ncore[0:2]                : REUSE) \
30      out(ncore[0:2]                : FREE)
31      {
32          ncore[0] = sysconf(_SC_NPROCESSORS_ONLN);
33          ncore[1] = sysconf(_SC_NPROCESSORS_ONLN);
34      }
35      printf("[HOST] # of CPUs in (mic, mic): (%d, %d)\n", ncore[0], ncore[1]);
36      free(ncore);
37      return 0;
38  }

```

Offload Mode

Running Applications in Offload Mode

```
user@host% icc offloadExample.c -o offloadExample
user@host% ./offloadExample
[HOST] # of CPUs in (host, mic): (20, 0)
[HOST] # of CPUs in (host, mic): (20, 244)
[HOST] # of CPUs in (mic, mic): (244, 244)
```

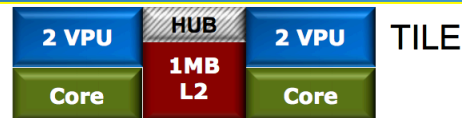
- General format: `#pragma offload target(mic) opt(data: clause)`

Option	Description	Clause	Description
in	copy (host→mic)	alloc(1) free(1)	data is local in offload
out	copy (mic→host)	alloc(1) free(0)	allocated in offload, but will be used in next offload
inout	copy (host↔mic)	alloc(0) free(1)	Has been used enough, so don't need it any more
nocopy	reuse with new offload	alloc(0) free(0)	Ready to reuse

Knights Landing (KNL)

Self-bootable: “a cluster full of coprocessors”

Knights Landing Overview



Stand-alone, Self-boot CPU

Up to 72 new Silvermont-based cores

4 Threads per core. 2 AVX 512 vector units

Binary Compatible¹ with Intel® Xeon® processor

2-dimensional Mesh on-die interconnect

MCDRAM: On-Package memory: 400+ GB/s of BW²

DDR memory

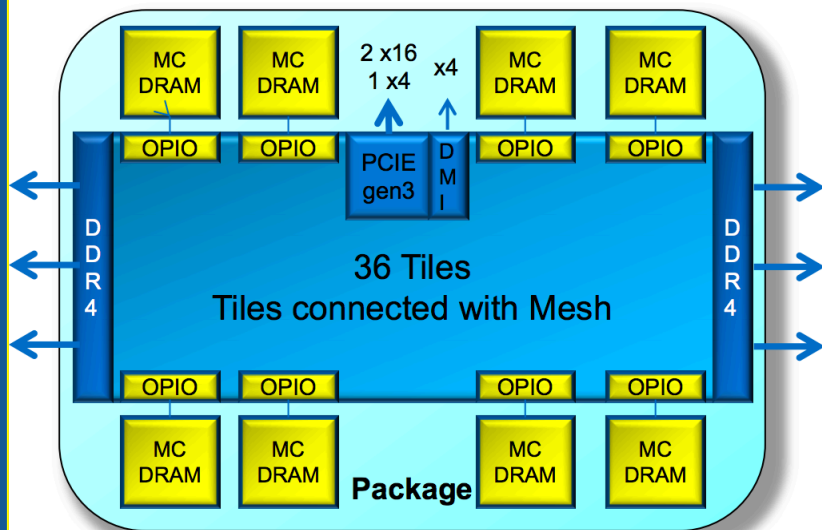
Intel® Omni-path Fabric

3+ TFLOps (DP) peak per package

~3x ST performance over KNC

< 16GB

< 384GB

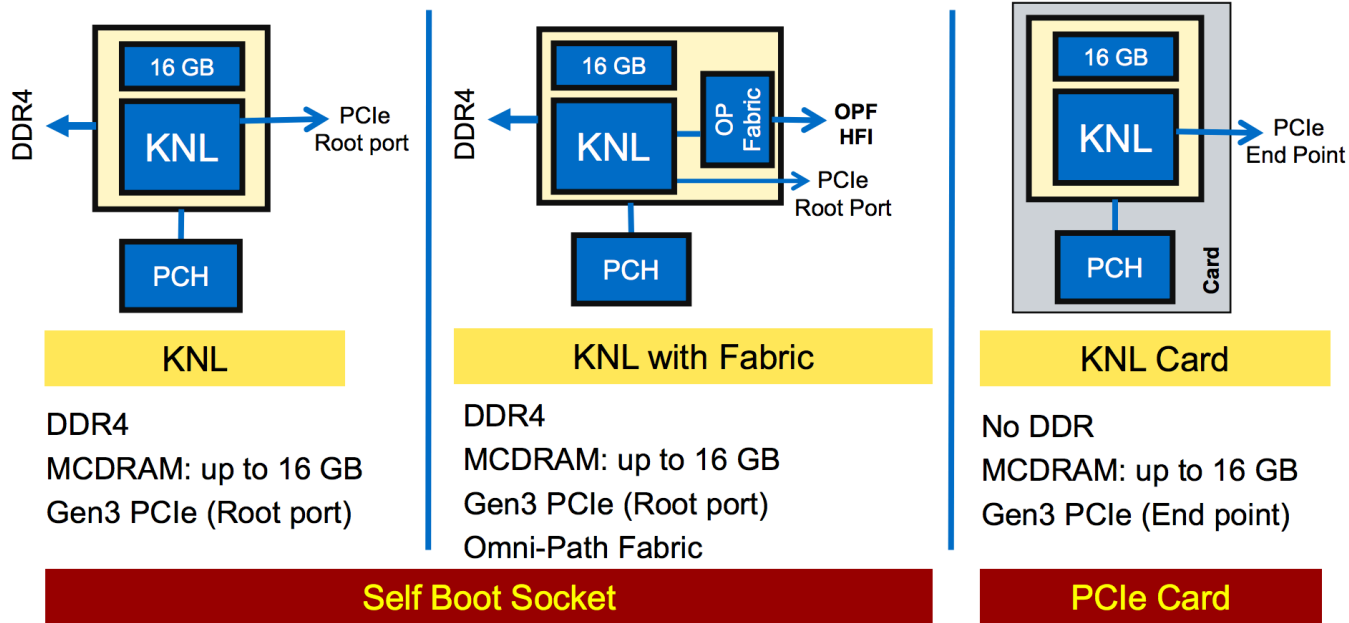


A. Sodani, Intel® Xeon Phi™ Processor “Knights Landing” Architectural Overview, ISC (2015)

Knights Landing (KNL)

Self-bootable: “a cluster full of coprocessors”

Knights Landing Products



A. Sodani, Intel® Xeon Phi™ Processor “Knights Landing” Architectural Overview, ISC (2015)

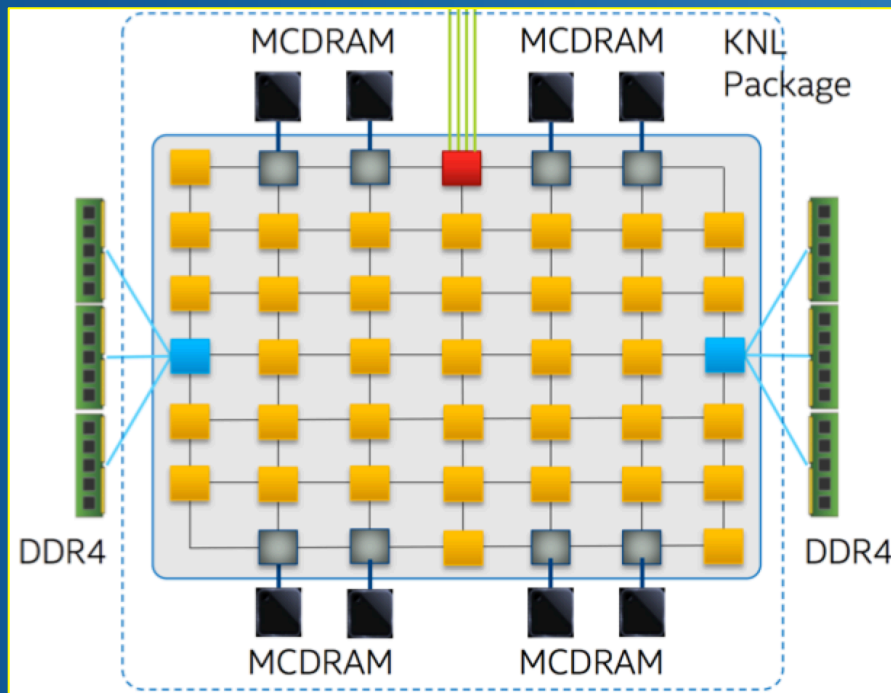
Are we happy?
No worries?

15Q4/16Q1

17Q4/18Q1

Knights Landing (KNL)

Potential Slots of Bottleneck in Performance, and When?



A. Sodani, Intel® Xeon Phi™ Processor “Knights Landing” Architectural Overview, ISC (2015)

- **Are we happy? No worries?**
- **Memory Access Pattern**
 - Do we have the most efficient path for each core?
 - When we have “multiple cores” access a single block of memory?
- **So, HBW (MCD)RAM**
 - 16G enough? Not sure before “do”ing
- **Why do we worry?**
 - will be good anyway for dense-matrix operations which fully use the power of vectorization
 - Large-scale PDE problems

Summary or More?

Intel Xeon Phi™ Coprocessors

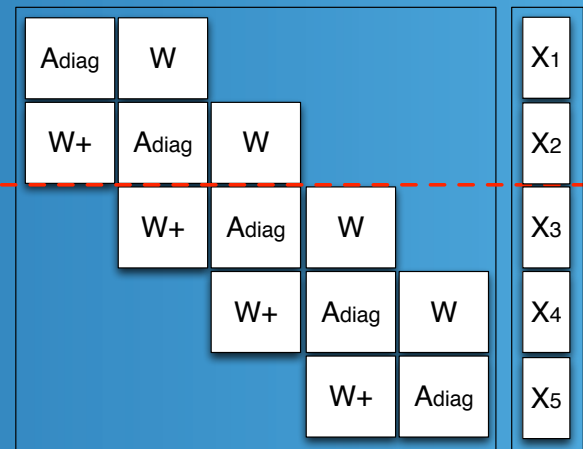
- Environment of MIC Programming:
 - Characteristics of Xeon Phi™ Coprocessors
 - Performance: Dense Operations
 - Vectorization and Parallelization
- Execution Modes (Programming Models)
 - Native Mode
 - Symmetric Mode
 - Offload Mode
- Very short discussions: Knights Landing
- Go more if we have time: Introduction to Ongoing Research

Large-scale, Parallel Computing?

A viewpoint from computing category

- EP (Embarrassingly Parallelism): ex) Monte Carlo (Event-Driven) etc.
- Communication w/ Dense Array-involved OP.: ex) MD / FFT etc.
- Communication w/ Sparse Array-involved OP.: ex) Most PDEs etc.
 - A system matrix describing simulation domain
 - CFD, Materials, Electronics, Structural Dynamics
 - Vectorization: can't avoid huge cache-miss
- Sparse-matrix MV-multiplication
 - Conditional statement: ignore “zero” elements

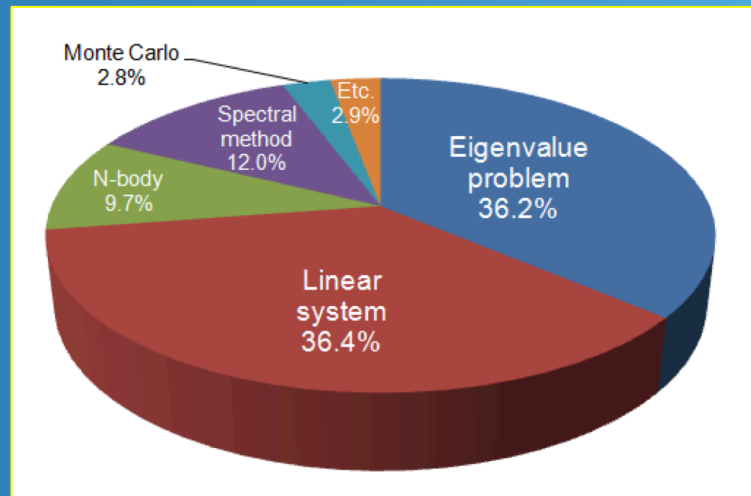
How large is the related numerical society?



Group: Sparse Matrix (PDE) Operation

A simple statistic from KISTI 2013-2014 HPC utilizations

- Type of jobs submitted to TACHYON-II
- PDE-involved jobs > 70+alpha (%)
- Eigenvalue PDE problem
 - 36.2 (%): Electronic structure, Heavy-ion acceleration, Resonance frequency etc.
- Linear system PDE problem
 - 36.4 (%): Most of linear PDE equations, including (linear) Poisson, Drift-diffusion, Heat, Black-Scholes etc.



Nanoelectronics

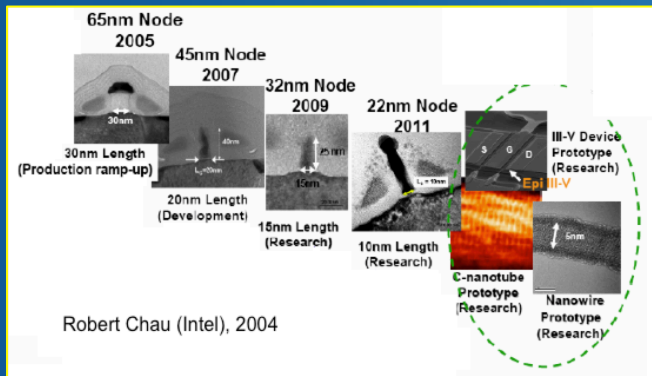
Power of Xeon Phi™ Coprocessors in solving PDEs?

- Extremely Large (Sparse) System Matrices

Nanoelectronics Computing: Overview

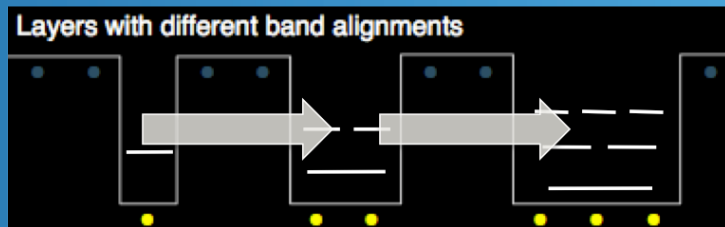
“TINY” Device Size, “LARGE” Computing Power

Size Downscaling



Materials and Electrical Properties

- Quantum Effects
- Energy-level quantization, Tunneling



Computing Load: Hamiltonian

- System matrix of 3D device structure
- DOF directly proportional to # of atoms



30nm³ Si box ~ 1M atoms

10 basis for 1 atom?

→ Involves H of 10M DOFs

Governing Equations

- Schrödinger PDE
- Non-equilibrium Green's Function

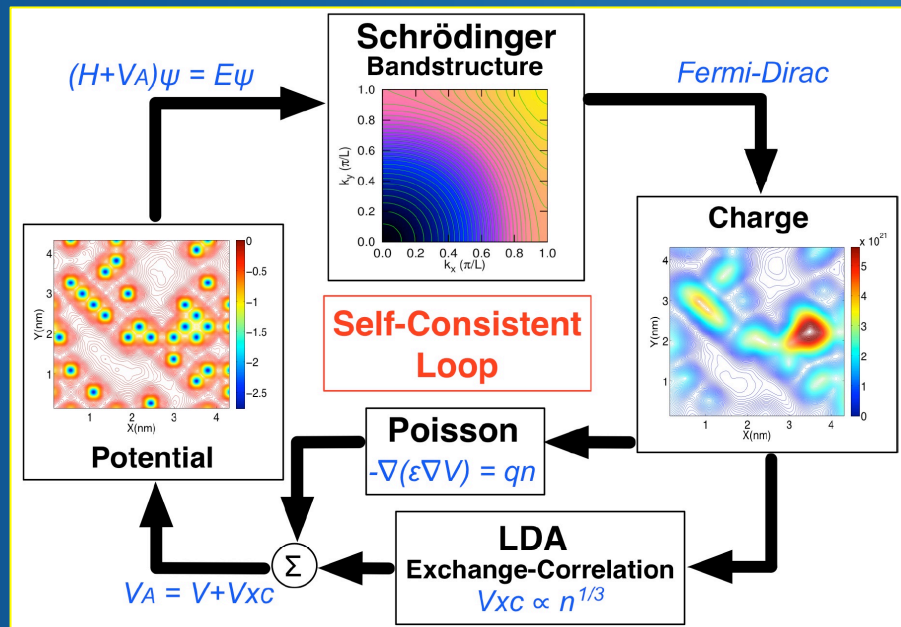
$$H\Psi = E\Psi \quad \text{Normal eigenvalue}$$

$$G(E) = (E - H - \Sigma(E))^{-1}$$

Inverse Matrix

Nanoelectronics Computing: Focus

Electronic Structure of Realistically Sized Devices: Under External Bias



H. Ryu *et al.*, *Nanoscale* **5**, 18, 8666 (2013)

- Schrödinger-Poisson Loop
 - Self-consistent Field
 - Materials Properties under “External” Biases
 - Targeted for multi-million atom systems: Physically Realizable
- Intel® Parallel Computing Center
 - KISTI designated since 2014
 - Development and Production of Qualified Research Outcomes

Development Strategy: Parallelization

Large-scale Schrödinger, Poisson Eqns.

Schrödinger Equation

- Normal Eigenvalue Problem (Electronic Structure Calc.)

$$H\Psi = E\Psi$$

- Hamiltonian is always symmetric

Poisson Equation

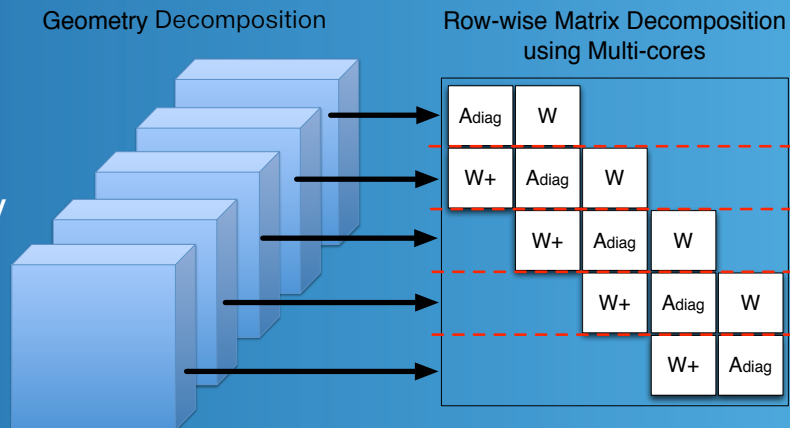
- Linear System Problem

$$-\nabla(\epsilon\nabla V) = \rho$$

- Poisson matrix is always symmetric

Steps for Parallelization

- Domain Decomposition on CPU-level
 - System matrices directly mapped to real-space
- Suitable algorithm for numerical solvers for scalability
 - Should be purely iterative
 - Direct algorithm is bad, (e.g.) No memory reduction with increasing # of CPUs



Numerical Algorithm: Eigenvalue Solver

LANCZOS Iterations

- **Schrödinger Eqns. with LANCZOS Algorithm**
→ C. Lanczos, *J. Res. Natl. Bur. Stand.* 45, 255
- Original Matrix (Hamiltonian) → T Matrix by Basis-Reduction
- Overall Steps for Iteration: Purely Scalable Algebraic Operations

Lanczos vector

\mathbf{v}_i : ($N \times 1$) vectors ($i = 0, \dots, K$); \mathbf{a}_i and \mathbf{b}_i : scalars ($i = 1, \dots, K$)

$\mathbf{v}_0 \leftarrow \mathbf{0}$, \mathbf{v}_1 = random vector with norm 1 ;

$\mathbf{b}_1 \leftarrow 0$;

loop for ($j=1$; $j \leq K$; $j++$)

$\mathbf{w}_j \leftarrow \mathbf{A}\mathbf{v}_j$;

$\mathbf{a}_j \leftarrow \mathbf{w}_j \bullet \mathbf{v}_j$;

$\mathbf{w}_j \leftarrow \mathbf{w}_j - \mathbf{a}_j \mathbf{v}_j - \mathbf{b}_j \mathbf{v}_{j-1}$;

$\mathbf{b}_{j+1} \leftarrow \|\mathbf{w}_j\|$;

$\mathbf{v}_{j+1} \leftarrow \mathbf{w}_j / \mathbf{b}_{j+1}$;

construct T matrix;

end loop



$$T = \begin{pmatrix} a_1 & b_2 & 0 & \cdots & \cdots & 0 \\ b_2 & a_2 & b_3 & & & \vdots \\ 0 & b_3 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & b_{k-1} & 0 \\ \vdots & & & b_{k-1} & a_{k-1} & b_k \\ 0 & \cdots & \cdots & 0 & b_k & a_k \end{pmatrix}$$

Numerical Algorithm: Poisson Solver

Conjugate Gradient (CG) Iterations

- **Poisson Eqns. with Conjugate Gradient Algorithm**
→ A Problem of Solving Linear Systems
- Convergence guaranteed: symmetric and positive definite system matrices
- Overall Steps for Iteration: Purely Scalable Algebraic Operations

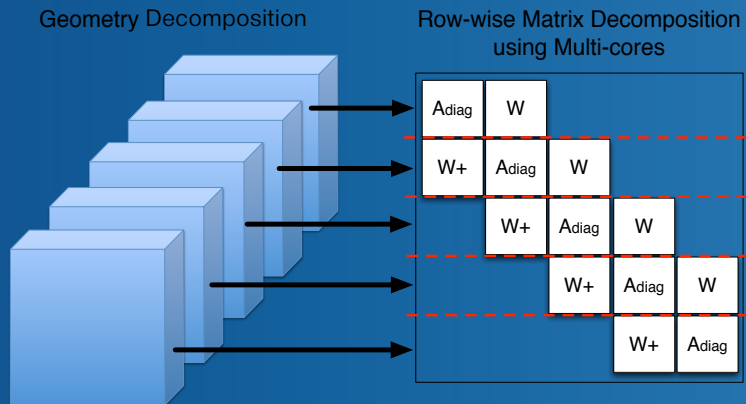
```

We want to solve  $\mathbf{Ax} = \mathbf{b}$ . First compute  $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ ,  $\mathbf{p}_0 = \mathbf{r}_0$ 
loop for (j=1; j<=K; j++)
     $\mathbf{a}_j \leftarrow \langle \mathbf{r}_j, \mathbf{r}_j \rangle / \langle \mathbf{Ap}_j, \mathbf{p}_j \rangle$ ;
     $\mathbf{x}_{j+1} \leftarrow \mathbf{x}_j + \mathbf{a}_j \mathbf{p}_j$ ;
     $\mathbf{r}_{j+1} \leftarrow \mathbf{r}_j - \mathbf{a}_j \mathbf{Ap}_j$ ;
    if ( $\|\mathbf{r}_{j+1}\| / \|\mathbf{r}_0\| < \epsilon$ )
        declare  $\mathbf{r}_{j+1}$  is the solution of  $\mathbf{Ax} = \mathbf{b}$  and break the loop
     $\mathbf{c}_j \leftarrow \langle \mathbf{r}_{j+1}, \mathbf{r}_{j+1} \rangle / \langle \mathbf{r}_j, \mathbf{r}_j \rangle$ ;
     $\mathbf{p}_{j+1} \leftarrow \mathbf{r}_{j+1} + \mathbf{c}_j \mathbf{p}_j$ ;
end loop
    
```

Performance on a CPU-level

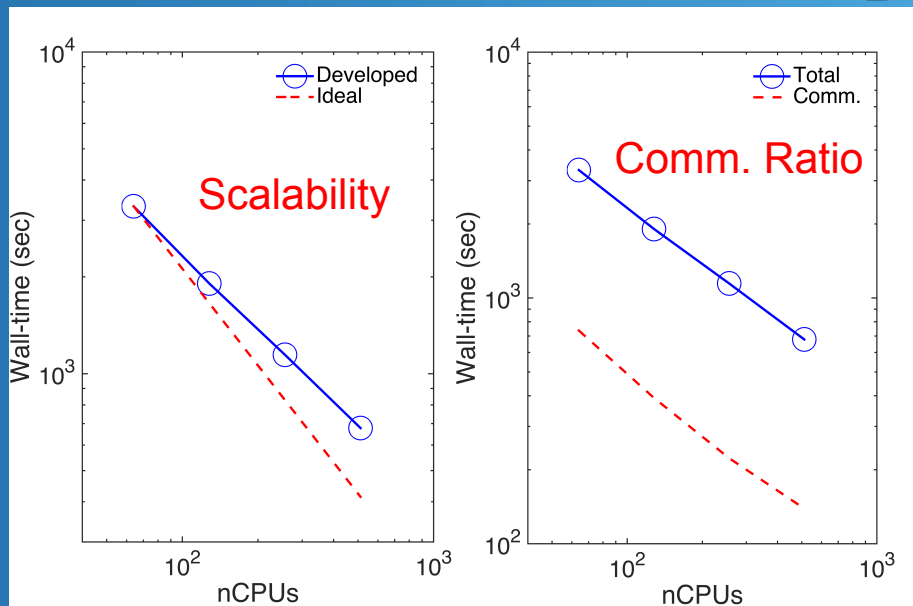
1D Domain Decomposition: Lanczos Iteration

Domain Decomposition



Performance BMT @ KISTI Tachyon-II

- 278x16x16nm [100] Si nanowires
- 4194304 atoms
- Find 5 lowest energy-levels



- Scalability upto ~80%: Communication Bottleneck
- How to reduce wall-time more with less # of nodes?
 - BUT with NO LOSS in computing time

Strategy: Performance Improvement

Asynchronous Offloading

The real bottleneck of computing

- Vector dot-product is not expensive: All-reduce, but small communication loads
- Vector communication is not a big deal: only communicates between adjacent layers
- Sparse-matrix-vector multiplication itself is indeed a big deal

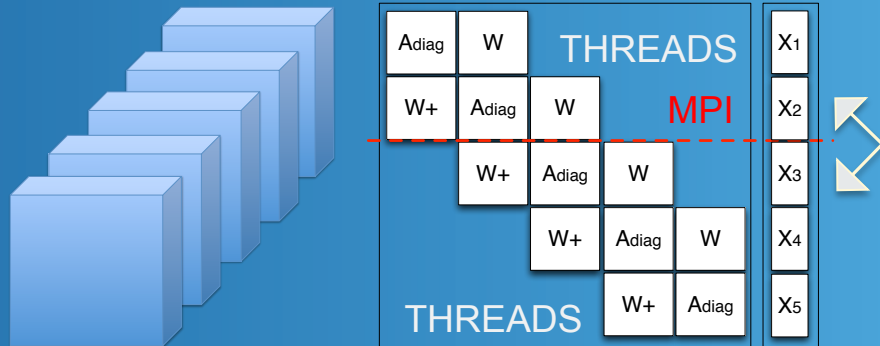
Schrödinger (LANCZOS)

```

loop for (j=1; j<=K ; j++)
   $w_j \leftarrow A v_j$ ;
   $a_j \leftarrow w_j \bullet v_j$ ;
   $w_j \leftarrow w_j - a_j v_j - b_j v_{j-1}$ ;
   $b_{j+1} \leftarrow \|w_j\|$ ;
   $v_{j+1} \leftarrow w_j / b_{j+1}$ ;
  
```

- Matrix-vector multiplier: Comm??
- Vector dot product: Reduction
- Others: No communication

Communication Pattern for MV multiplier



Food for Thoughts

- Sparse MV multiplier: performance vectorization?
- Synchronous offload: CPU should be idle: Asynchronous?

Strategy: Performance Improvement

Asynchronous Offloading

The real bottleneck of computing

- Vector dot-product is not expensive: All-reduce, but small communication loads
- Vector communication is not a big deal: only communicates between adjacent layers
- Sparse-matrix-vector multiplication itself is indeed a big deal

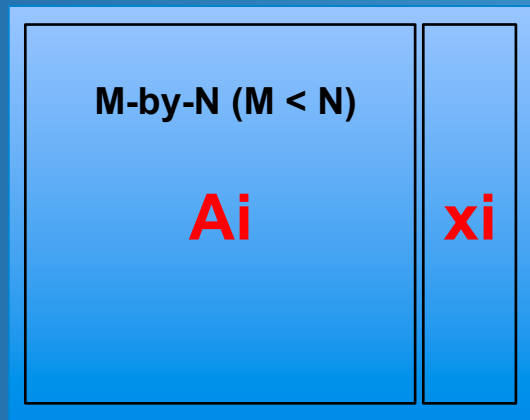
Schrödinger (LANCZOS)

```

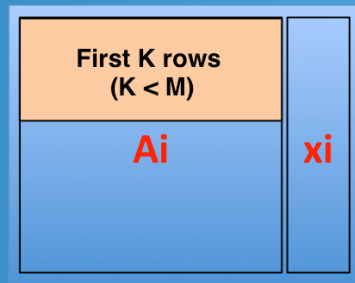
loop for (j=1; j<=K ; j++)
   $\underline{w}_j \leftarrow \underline{A} \underline{v}_j ;$ 
   $\underline{a}_j \leftarrow \underline{w}_j \bullet \underline{v}_j ;$ 
   $\underline{w}_j \leftarrow \underline{w}_j - \underline{a}_j \underline{v}_j - \underline{b}_j \underline{v}_{j-1} ;$ 
   $\underline{b}_{j+1} \leftarrow \|\underline{w}_j\| ;$ 
   $\underline{v}_{j+1} \leftarrow \underline{w}_j / \underline{b}_{j+1} ;$ 
  
```

- Matrix-vector multiplier: Comm??
- Vector dot product: Reduction
- Others: No communication

“Truncated M”V Multiplier



Coprocessor



CPU



Some Results: Performance Improvement

Asynchronous Offloading: Server spec is moderate

Benchmark in a testbed 1 @ KISTI

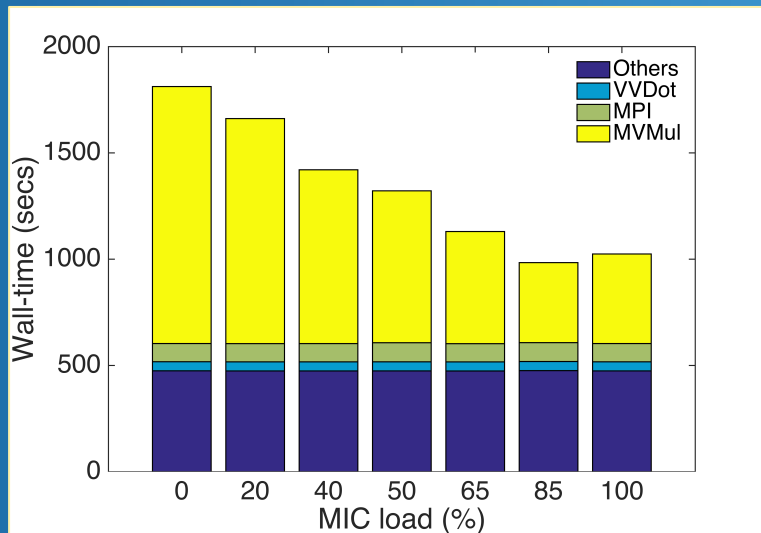
- 1 node: 4 Xeon E5-2603 v2 CPUs (1.8GHz), 16GB M w/ a 3120 card (6GB M)
- Problem size: A P-atom integrated in 22nm \times 22nm \times 22nm Si box (DOF: 5.12M, M \sim 3.8GB)
- Target: Run 5000 Lanczos iterations to find as many eigenvalues as possible
- Cond: 1 MPI process with 4 threads per CPU, 224 threads in MIC (56 \times 4)

Schrödinger (LANCZOS)

```

loop for (j=1; j<=K ; j++)
   $w_j \leftarrow A v_j$ ;
   $a_j \leftarrow w_j \bullet v_j$ ;
   $w_j \leftarrow w_j - a_j v_j - b_j v_{j-1}$ ;
   $b_{j+1} \leftarrow ||w_j||$ ;
   $v_{j+1} \leftarrow w_j / b_{j+1}$ ;
  
```

- Matrix-vector multiplier: Comm??
- Vector dot product: Reduction
- Others: No communication



BMT Summary

- 85% MIC load: Best
- $\sim 3x$ Improvement in MVMul, $\sim 2x$ in Total

Some Results: Performance Improvement

Asynchronous Offloading: Server spec is super-excellent

Benchmark in a testbed 2 @ KISTI – super-excellent spec

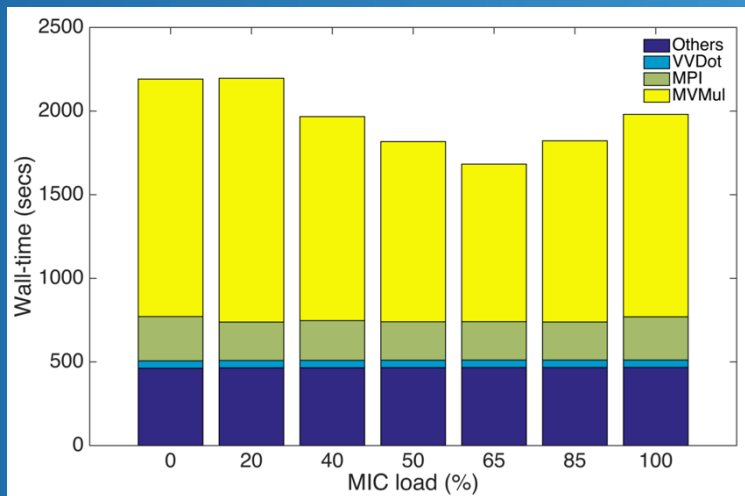
- 2 nodes: 20 Xeon E5-2680 v2 CPUs (2.8GHz), 256GB M w/ two 7120 card (32GB M total)
- Problem size: A P-atom integrated in 22nm \times 71nm \times 71nm Si box (DOF: 54M, M \sim 41GB)
- Target: Run 5000 Lanczos iterations to find as many eigenvalues as possible
- Cond: 4 MPI process with 10 threads per CPU, 240 threads in MIC (60 \times 4)

Schrödinger (LANCZOS)

```

loop for (j=1; j<=K ; j++)
   $w_j \leftarrow A v_j$ ;
   $a_j \leftarrow w_j \bullet v_j$ ;
   $w_j \leftarrow w_j - a_j v_j - b_j v_{j-1}$ ;
   $b_{j+1} \leftarrow ||w_j||$ ;
   $v_{j+1} \leftarrow w_j / b_{j+1}$ ;
  
```

- Matrix-vector multiplier: Comm??
- Vector dot product: Reduction
- Others: No communication



BMT Summary

- 65% MIC load: Best
- $\sim 1.5x$ Improvement in MVMul, $\sim 1.3x$ in Total

H. Ryu *et al.*, SC 15
Intel Theatre Presentation

Possibility: Meaning of 1.5x~3x Speedup?

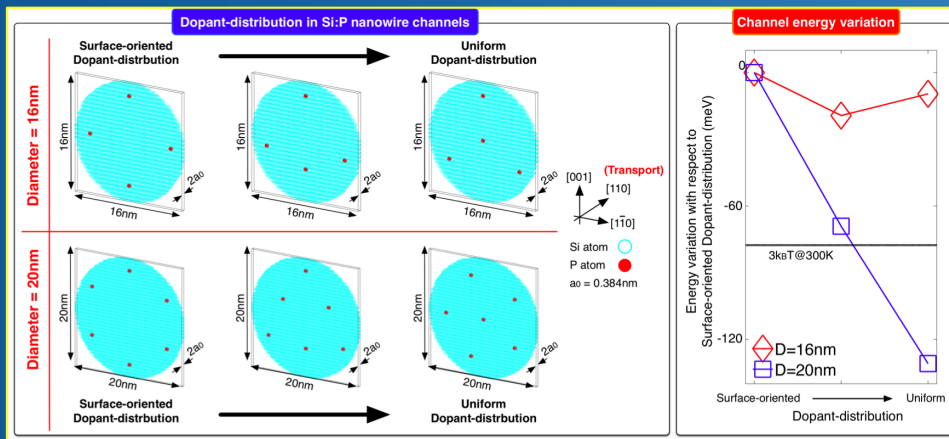
When the computing is not CPU-bound


- Recall: Sparse Matrix is not a BEST example
 - It rather is “WORST” example while having a big community
- 20 cores w/ 2 coprocessor cards vs. 30 cores?
 - # of nodes: already 66%~33% with rough estimation
- Tachyon-II: Yes, it is old but
 - 50M USD one time vs. 6M every year?
 - Electricity and Land: Coprocessor is already a green computing
- Upcoming Knight-Landing: Self-bootable.
 - Computing Capability would reduce # of nodes (req. for some performance)
 - Sparse Matrix Operation can even improved with High BW memory
 - Money Money Money...

Utility: Developed SP Solver

Nanoelectronics/Materials Research

- A Study of Dopant-distribution in Free-standing Si nanowires
- P-doped Si nanowires: Attractive materials for designs of ultrathin interconnector.
- Dopant-distribution should be uniform (in general) to achieve conducting nanowires.
- **Problem:** What's the size limit of highly P-doped nanowires that allow uniform dopant-distribution? – Needs to be understood to provide a design guideline for field engineers.





Letter
pubs.acs.org/NanoLetter

Atomistic Study on Dopant-Distributions in Realistically Sized, Highly P-Doped Si Nanowires

Hyun Ryu^{*,†}, Jongseob Kim[‡], and Ki-Ha Hong^{*,†}

전자신문

초박형 실리콘 나노전선 공정 원리 밝혀

휴온 KISTI 선임연구원, 슈퍼컴퓨터 활용해 계산

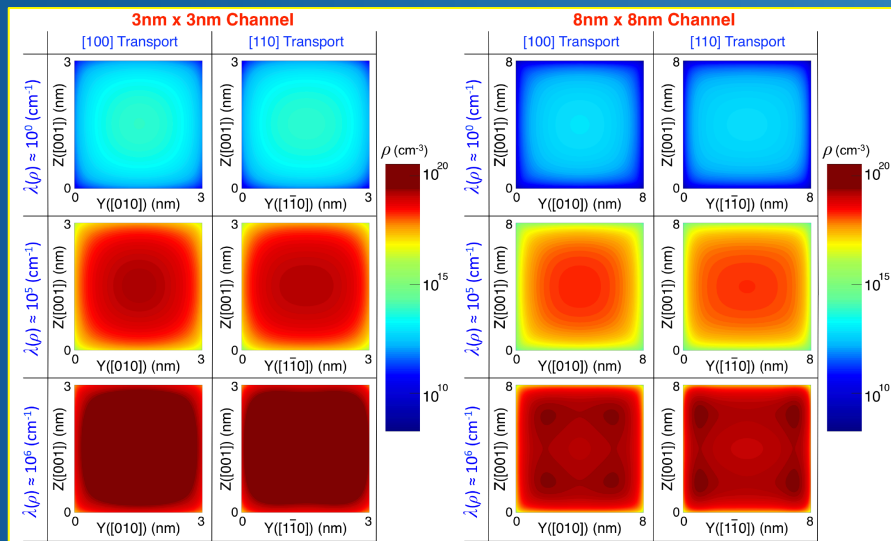
반도체 집적도를 획기적으로 개선할 초박형 실리콘 나노전선 공정 원리가 밝혀졌다. 한국과학기술정보연구원(KISTI-원장 한선화)은 휴온 슈퍼컴퓨팅본부 슈퍼컴퓨팅응용실선임연구원이 슈퍼컴퓨터를 활용한 계산을 통해 불순물 반도체 기반의 초박형 전선 공정을 위한 이론적 배경을 세계 최초로 제시했다고 5일 밝혔다. 휴 박사는 불순물인 '인(Phosphorus)' 원자의 분포 경향과 실리콘 나노선 크기 사이의 상관관계를 이론적으로 밝혔다. 휴 박사는 실리콘 1㎞당 약 1019개의 인 원자가 섞인 10⁻⁶㎞ 이상의 고농도 실리콘 나노선을 원자 수준으로 묘사한 뒤 양자원리(슈뢰딩거 방정식)를 적용해 슈퍼컴퓨터로 이 전자구조를 밝혀냈다. 휴 박사는 이 연구로 안정적인 전기전도도를 지닌 균일한 나노전선을 개발할 수 있을 것으로 기대했다. 제한된 면적에 반도체의 집적도를 높이기 위해서는 소자와 소자를 연결하는 전선의 굵기를 얇게 할수록 유리하다. 하지만 이 전선의 전도도에 영향을 미치는 불순물 원자분포 경향에 대한 이론적 배경이 없어 균일한 품질을 만들기 어려웠다. 이 연구는 인텔 초병렬 컴퓨팅 지원사업(UPC)의 일환으로 수행 중인 '파이 코프로세서(Phi coprocessor)' 기반의 고성능 슈퍼딥러닝 병렬계산 SW 개발 및 이의 반도체 소자 설계 활용연구'로 진행됐다. 이 연구결과는 지난 3일 나노과학 국제학술지인 '나노 레터스(Nano Letters)' 온라인판에 게재됐다. 대전=박희범기자 hbpark@etnews.com

H. Ryu et al., Nano Letters 15, 1, 450-456 (2015)

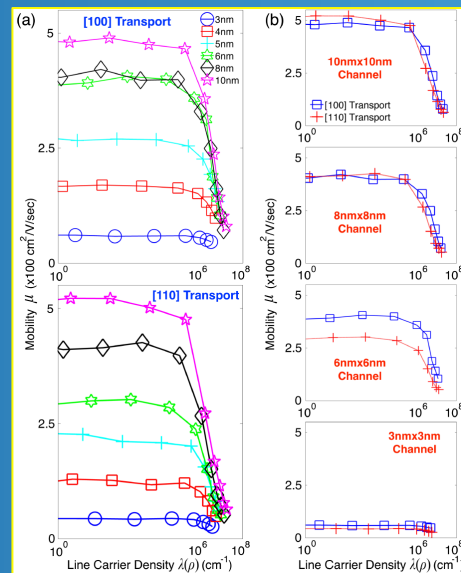
Utility: Developed SP Solver

Nanoelectronics/Materials Research

- Mobility behavior in Free-standing P-doped Si nanowires
- **Problem:** Investigation Major Scattering Mechanisms suppressing Carrier Transport
- Below 10nm CMOS Technology, presenting Design Guideline to Experimentalists.



H. Ryu, *Nanoscale Research Letters* 11, 1, 36 (2016)



Grand Summary

Intel Xeon Phi™ Coprocessors

- Environment of MIC Programming:
 - Characteristics of Xeon Phi™ Coprocessors
 - Performance: Dense Operations
 - Vectorization and Parallelization
- Execution Modes (Programming Models)
 - Native Mode
 - Symmetric Mode
 - Offload Mode
- Very short discussions: Knights Landing
- Validity on Large-scale Sparse-matrix-involved PDE computing
 - KISTI Intel® Parallel Computing Center: Schrödinger-Poisson
 - Research Activities