LSU INFORMATION TECHNOLOGY SERVICES

Introduction to Xeon Phi programming: Part I

Shaohao Chen

High performance computing @ Louisiana State University

Outline of Xeon Phi Programming

Part I

- Intel Xeon Phi and its computing features
- Usage of Xeon Phi in HPC
- Xeon Phi programming: native mode, offloading

Part II

- Xeon Phi programming: symmetric processing
- Optimization, debugging and profiling
- Xeon-Phi enabled applications

Part I

- Intel Xeon Phi and its computing features
- Usage of Xeon Phi in HPC
- Xeon Phi programming:

  native mode

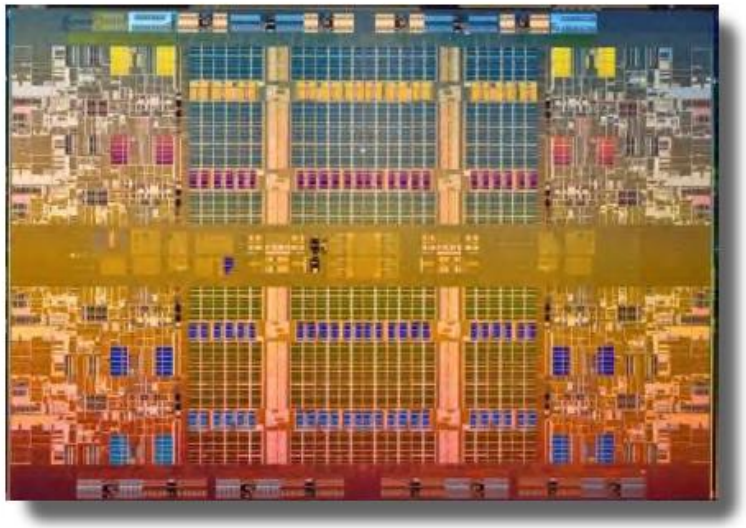
  offloading

Multi-core vs. Many-core

LSU INFORMATION TECHNOLOGY SERVICES

Xeon

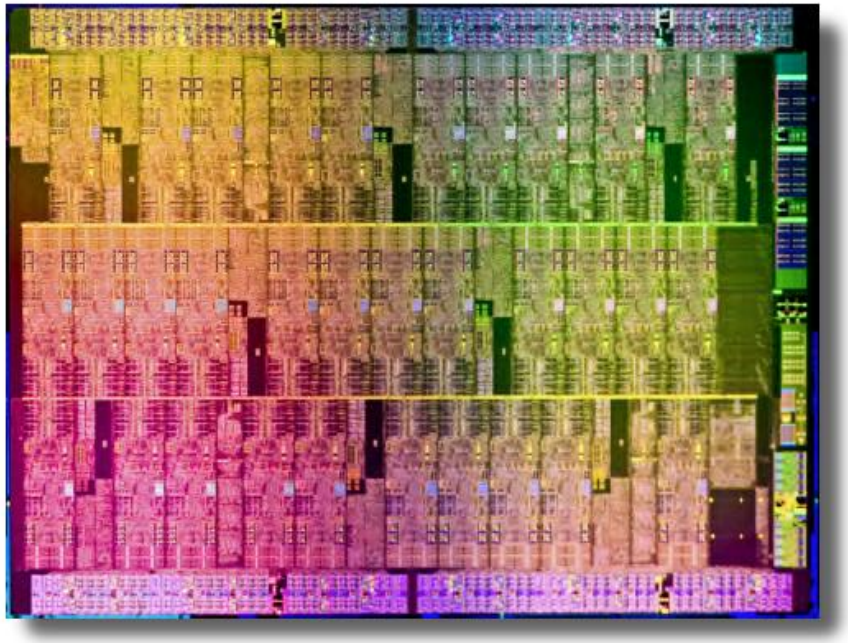
Multi-core Architecture



- 8 ~ 12 cores
- Single-core 2.5 ~ 3 GHz
- 256-bit vectors

Xeon Phi

Many Integrated Core (MIC) Architecture



- 61 cores (244 logical)
- Single-core ~ 1.2 GHz
- 512-bit vectors

Intel Xeon Phi coprocessor (accelerator)

(parameters for Xeon Phi 7120P)

- Add-on to CPU-based system
- PCI express (6.66 ~ 6.93 GB/s)
- IP-addressable
- 16 GB memory
- 61 x86 64-bit cores (244 threads)
- single-core 1.2 GHz
- 512-bit vector registers
- 1.208 TeraFLOPS = 61 cores * 1.238 GHz * 16 DP FLOPs/cycle/core

Current: Knight Corner (KNC)

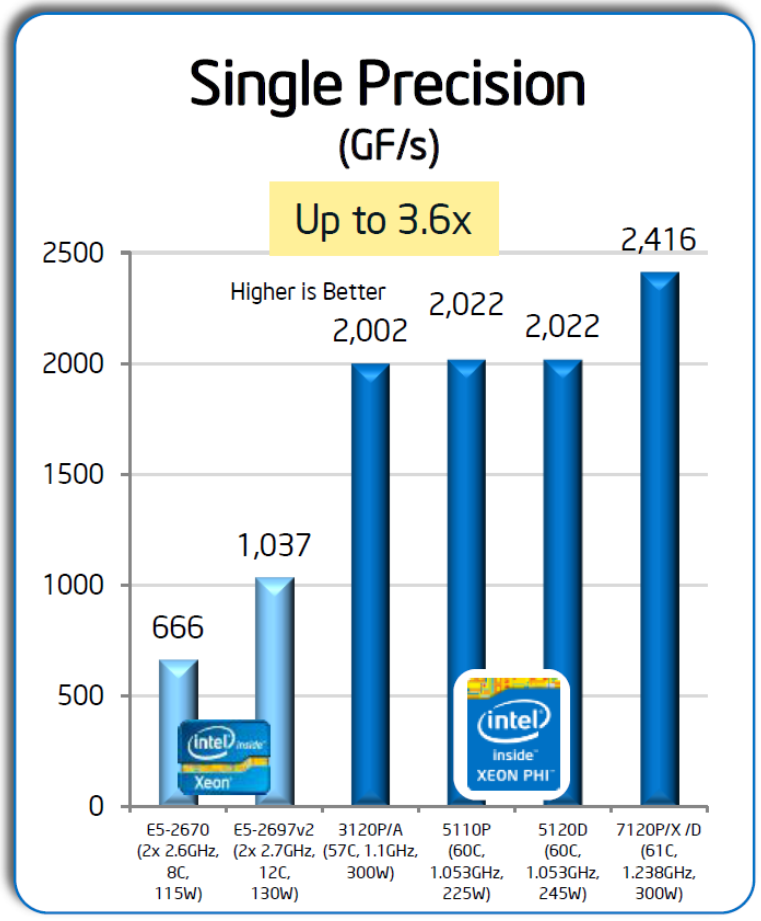
Next (2016): Knight Landing (KNL)

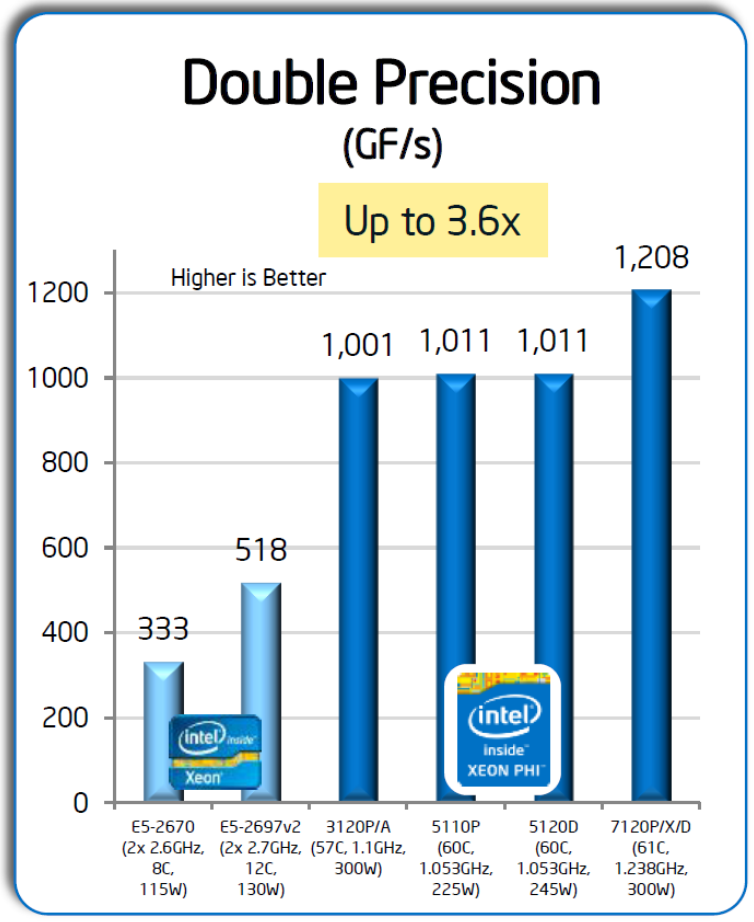
Xeon Phi Computing Performance

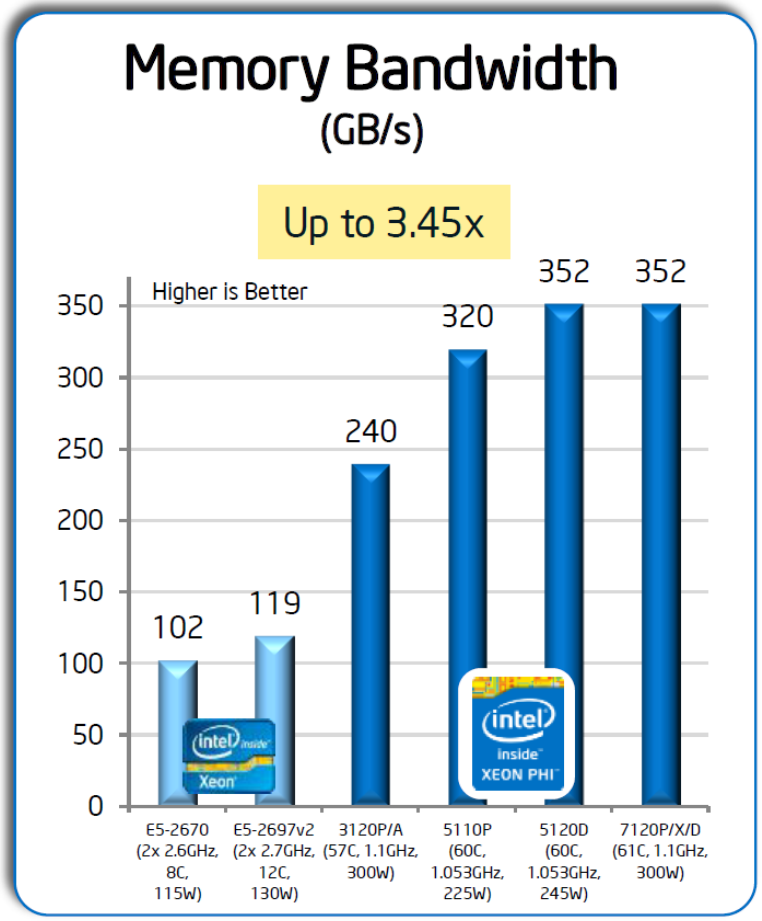
Theoretical Maximums

Updated

(2S Intel® Xeon® processor E5-2670 & E5-2697v2 vs. Intel® Xeon Phi™ coprocessor)



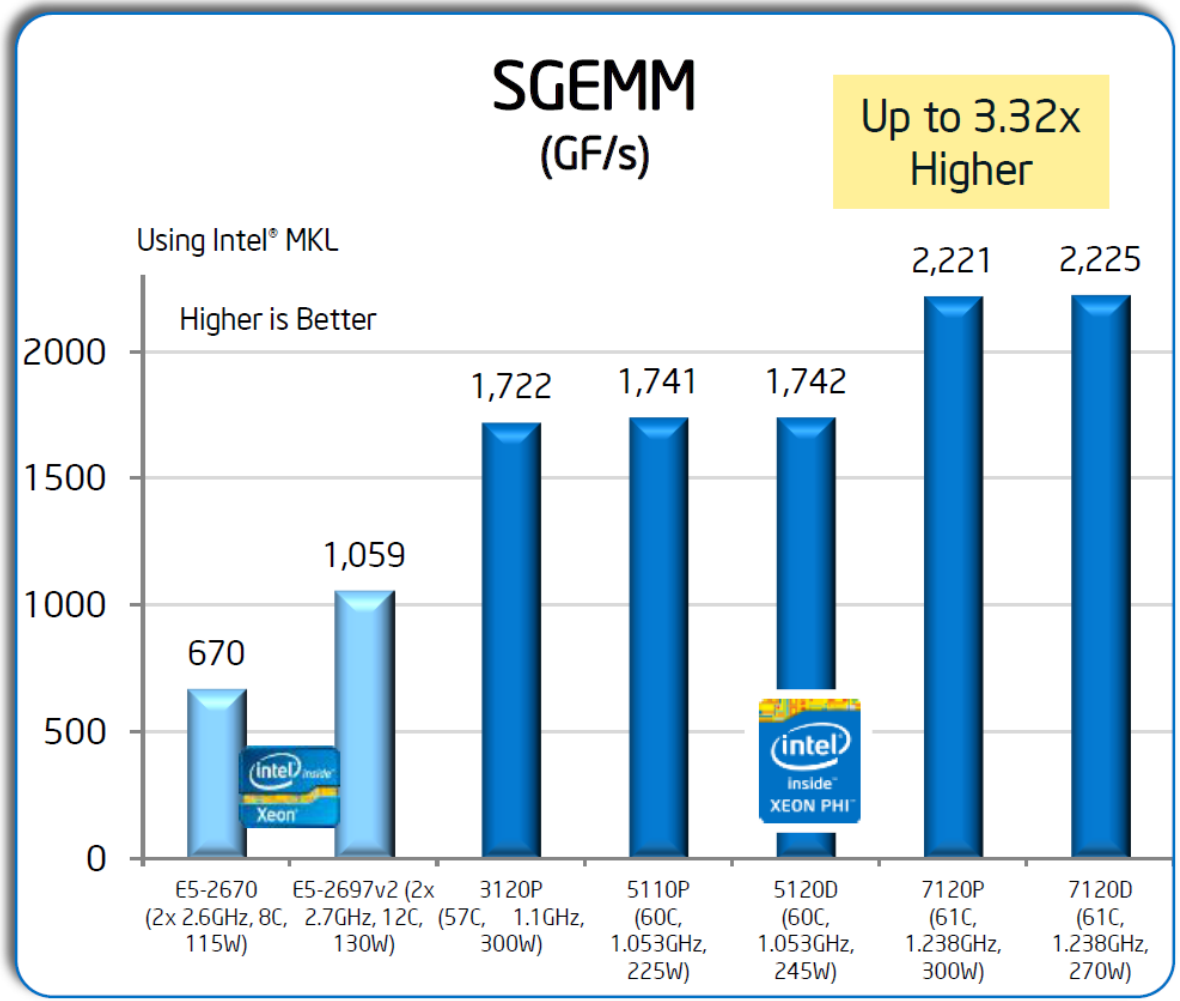


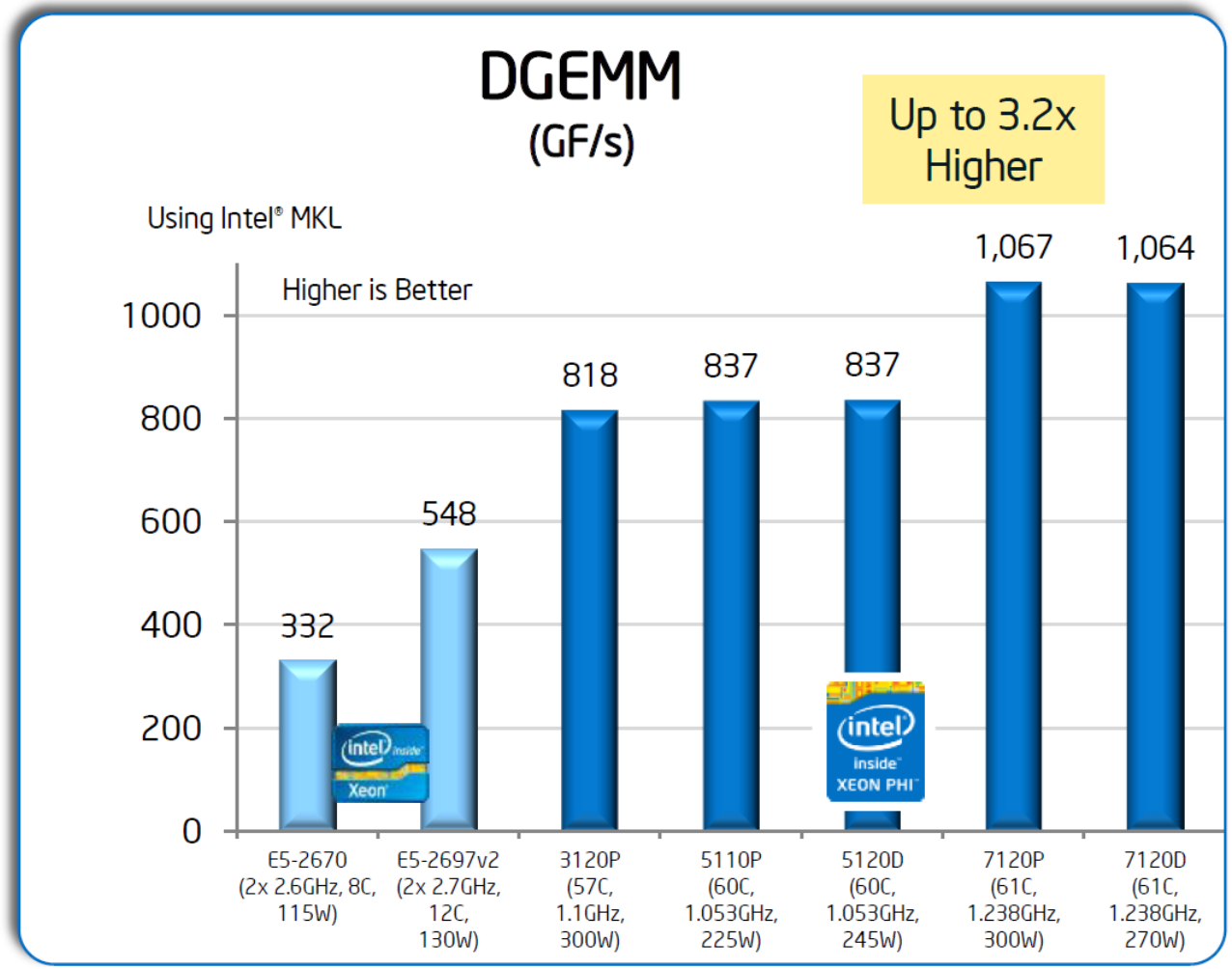


Source from Intel website

Synthetic Benchmark Summary (1 of 2)

Updated





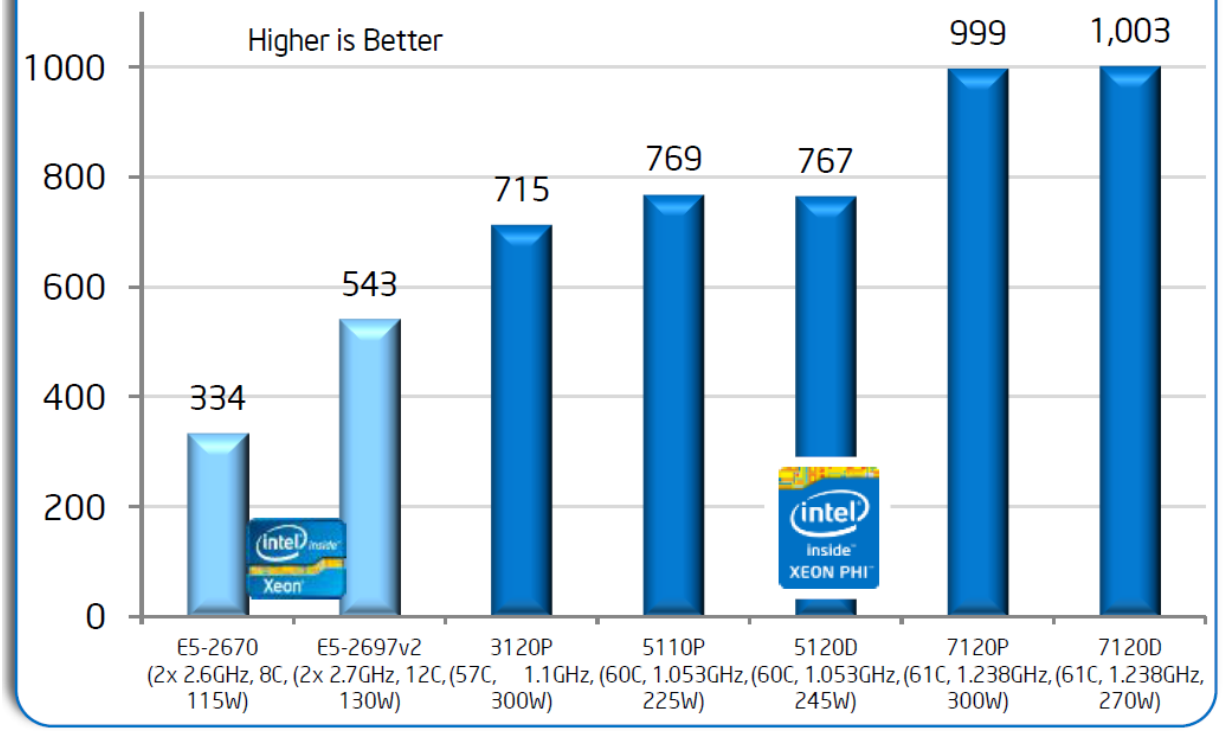
Synthetic Benchmark Summary (2 of 2)

Updated

Linpack[1] (GF/s)

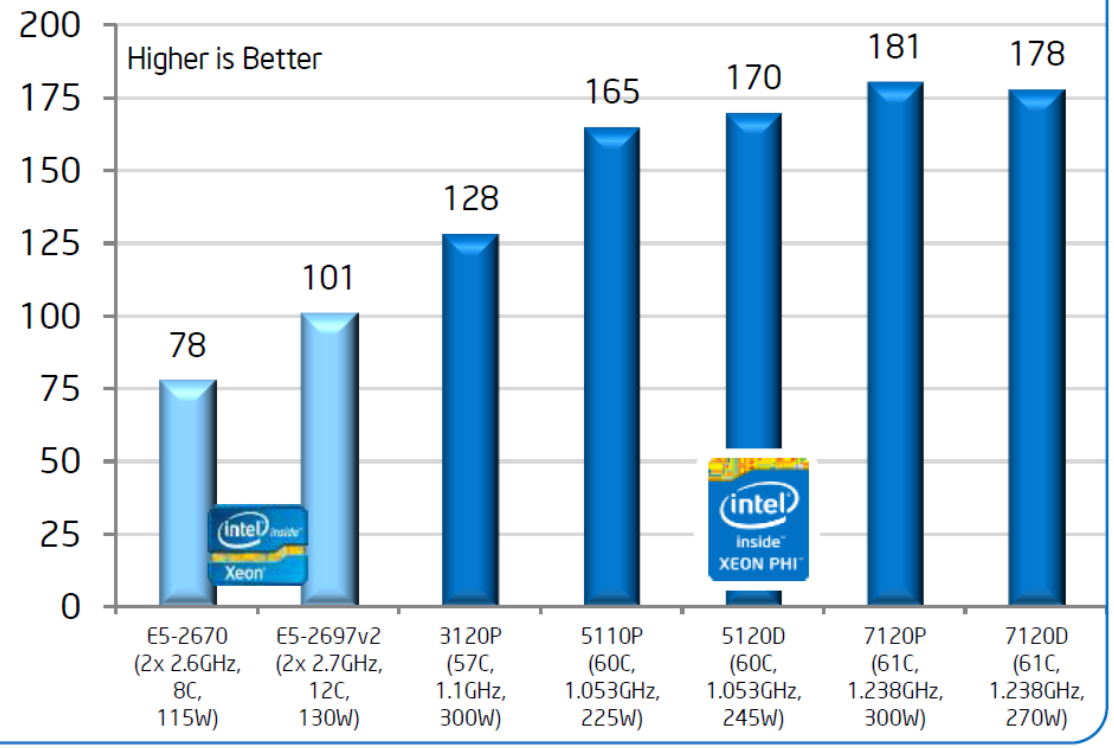
Up to 3.0x Higher

Using Intel® MKL



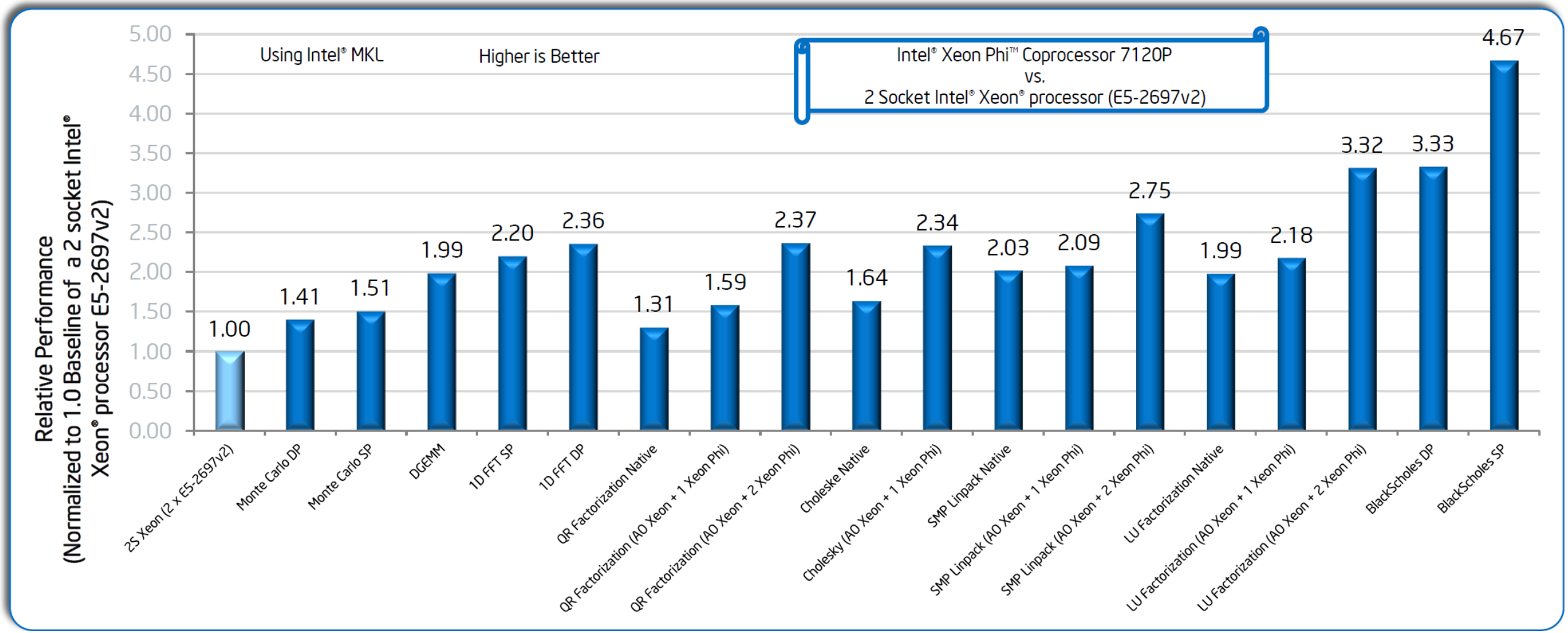
STREAM Triad (GB/s)

Up to 2.3x Higher



Intel® Xeon Phi™ Coprocessor vs. 2S Intel® Xeon® processor (Intel® MKL)

Updated



Native = Benchmark run 100% on coprocessor. AO = Automatic Offload Function = Xeon + Xeon Phi together

Segment	Customer	Application	Performance Increase[1] vs. 2S Xeon*
Energy	Acceleware	8th order isotropic variable velocity	Up to 2.23x
	Sinopec	Seismic Imaging	Up to 2.53x[2]
	CNPC (China Oil & Gas)	GeoEast Pre-Stack Time Migration (Seismic)	Up to 3.54x[2]
Financial Services	Financial Services	BlackScholes SP Monte Carlo SP	Up to 7.5x Up to 10.75x
Physics	Jefferson Labs	Lattice QCD	Up to 2.79x
Finite Element	Sandia Labs	miniFE (Finite Element Solver)	Up to 2x[3] Up to 1.3x[5]
Solid State Physics	ZIB (Zuse-Institut Berlin)	Ising 3D (Solid State Physics)	Up to 3.46x
Digital Content Creation/Video	Intel Labs	Ray Tracing (incoherent rays)	Up to 1.88x[4]
	NEC	Video Transcoding	Up to 3.0x[2]
Astronomy	CSIRO/ASKAP (Australia Astronomy)	tHogbom Clean (Astronomy image smear removal)	Up to 2.27x
	TUM (Technische Universität München)	SG++ (Astronomy Adaptive Sparse Grids/Data Mining)	Up to 1.7x
Fluid Dynamics	AWE (Atomic Weapons Establishment - UK)	Cloverleaf (2D Structured Hydrodynamics)	1.77x

Intel Xeon Phi vs. Nvidia GPU

Disadvantages

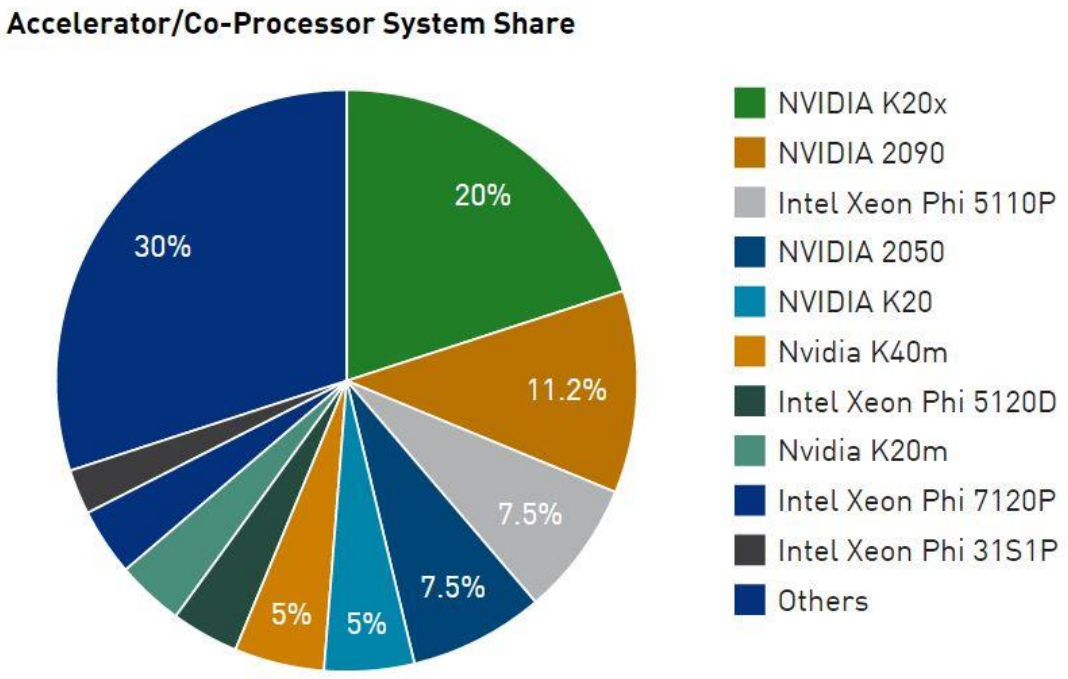
- Less acceleration
- In terms of computing power, one GPU beats one Xeon Phi for most cases currently.

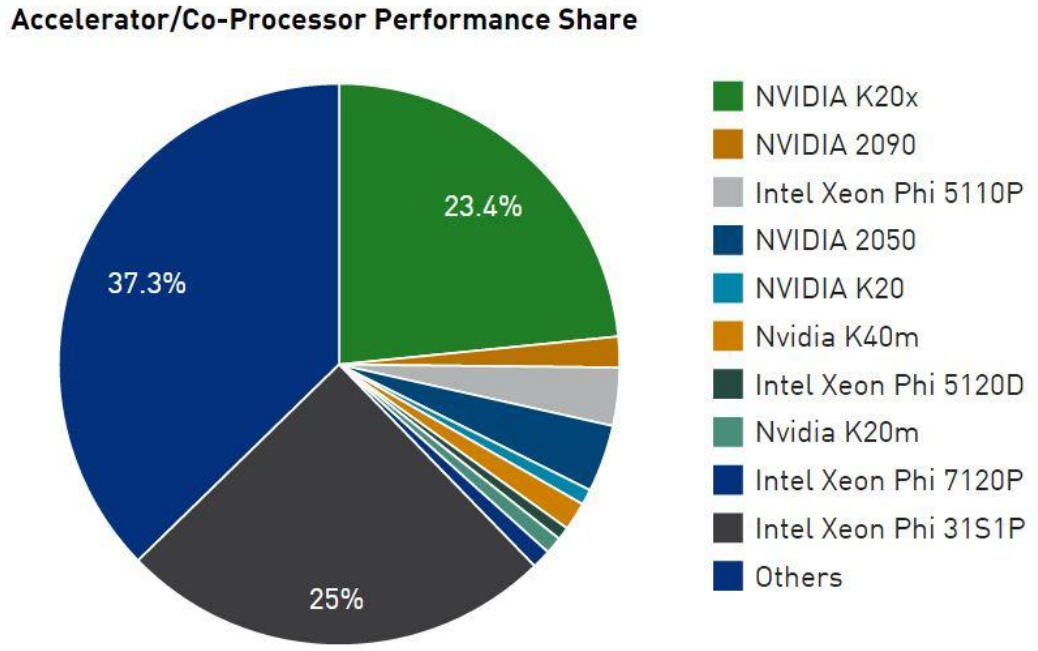
Advantages

- X86 architecture
- IP-addressable
- Traditional parallelization (OpenMP, MPI)
- Easy programming, minor changes from CPU codes
- Offload: minor change of source code.
- New. Still a lot of room for improvement.

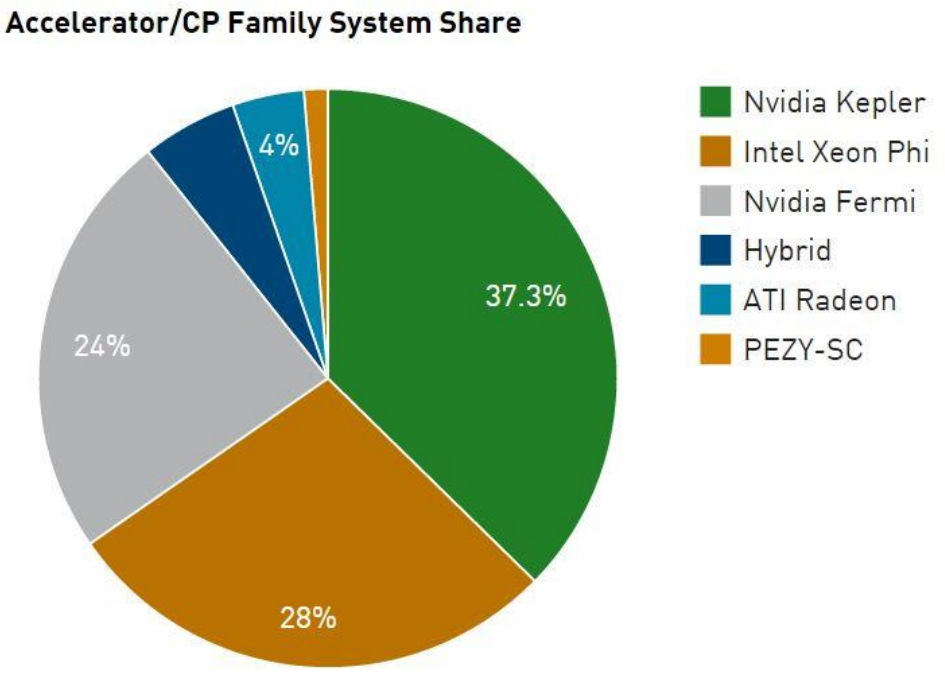
Usage of Xeon Phi in HPC

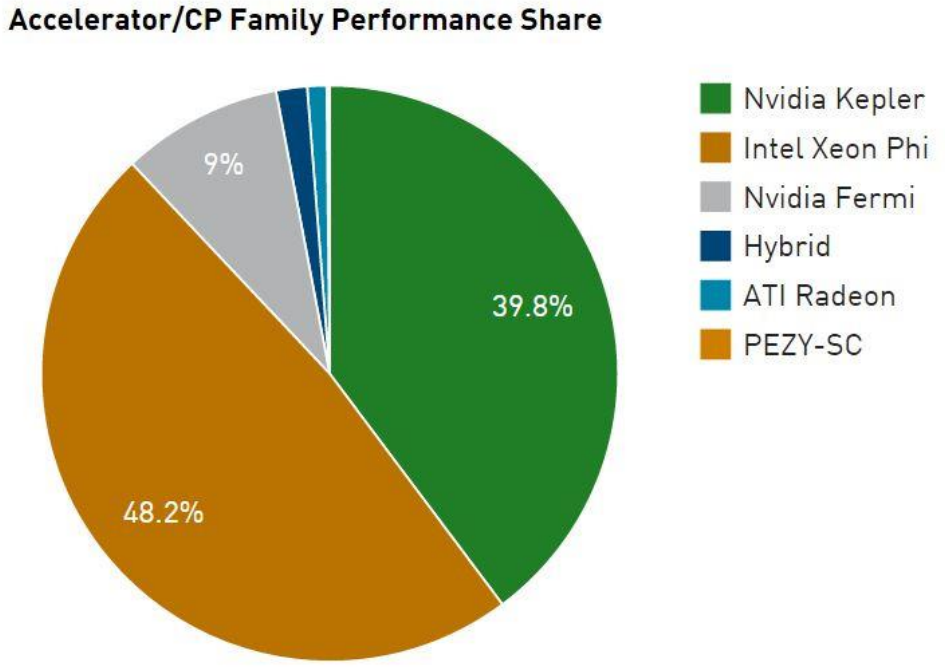
- Statistics of accelerators in top 500 supercomputers (Nov 2014 list)





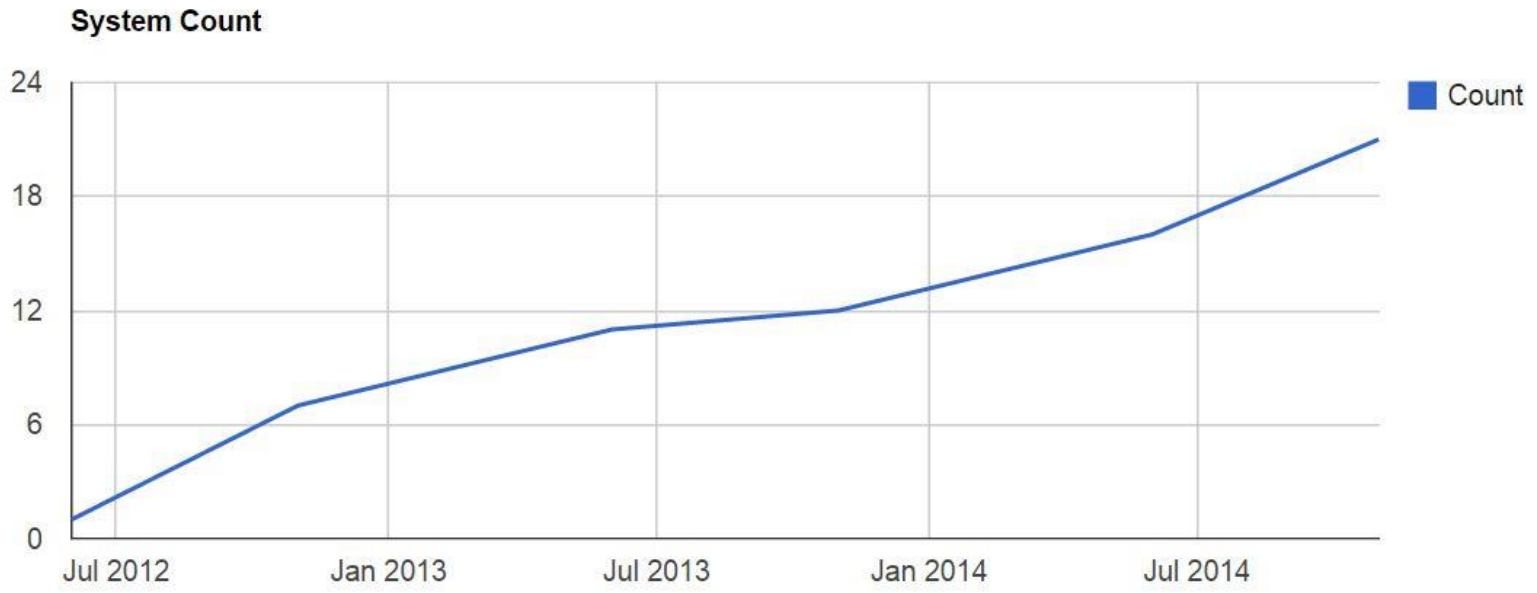
• Accelerator share





ACCELERATOR/CP FAMILY	COUNT	SYSTEM SHARE (%)	RMAX (GFLOPS)	RPEAK (GFLOPS)	CORES
Nvidia Kepler	28	5.6	42,284,740	63,953,116	1,111,247
Intel Xeon Phi	21	4.2	51,245,010	83,159,331	4,660,830
Nvidia Fermi	18	3.6	9,587,021	20,497,761	768,032
Hybrid	4	0.8	1,850,234	2,839,491	277,480
ATI Radeon	3	0.6	1,148,600	2,280,349	94,304
PEZY-SC	1	0.2	178,107	395,264	262,784

- Number of supercomputers with Xeon Phi coprocessors

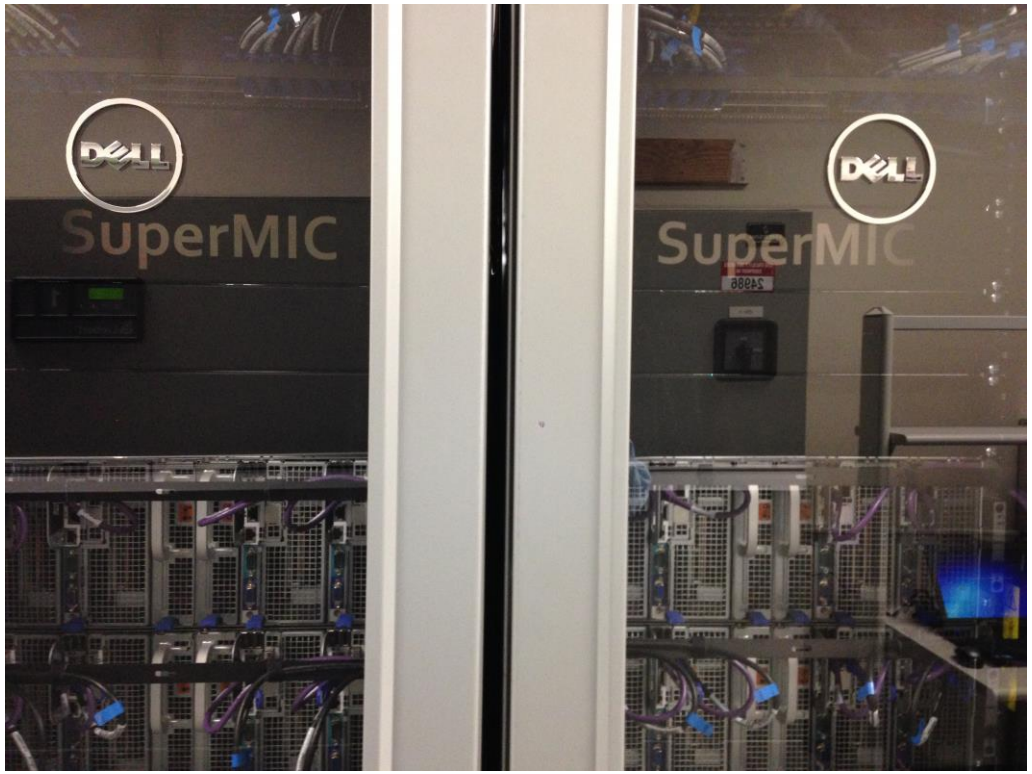


LIST	COUNT	SYSTEM SHARE (%)	RMAX (GFLOPS)	RPEAK (GFLOPS)	CORES
Nov 2014	21	4.2	51,245,010	83,159,331	4,660,830
Jun 2014	16	3.2	47,390,611	75,032,394	4,234,766
Nov 2013	12	2.4	45,244,135	72,197,351	4,079,172
Jun 2013	11	2.2	42,131,863	67,790,175	3,830,503
Nov 2012	7	1.4	4,302,764	6,309,356	337,301
Jun 2012	1	0.2	118,600	180,992	9,800

- Typical supercomputers with Xeon Phi coprocessors

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
88	Louisiana State University United States	**SuperMIC** - Dell C8220X Cluster, Intel Xeon E5-2680v2 10C 2.8GHz, Infiniband FDR, Intel Xeon Phi 7120P Dell	45,866	557.0	925.1	370

SuperMIC @LSU



- 360 Compute Nodes
  - Two 2.8GHz 10-Core Ivy Bridge-EP E5-2680 Xeon 64-bit Processors
  - Two Intel Xeon Phi 7120P Coprocessors
  - 64GB DDR3 1866MHz Ram
  - 500GB HD
  - 56 Gigabit/sec Infiniband network interface
  - 1 Gigabit Ethernet network interface

Ref: http://www.hpc.lsu.edu/resources/hpc/system.php?system=SuperMIC

SuperMIC	
Hostname	smic.hpc.lsu.edu
Peak Performance/TFlops	1000
Compute nodes	360
Processor/node	2 Deca-core
Processor Speed	2.8GHz
Processor Type	Intel Xeon 64bit
Nodes with Accelerators	360
Accelerator Type	Xeon Phi 7120P
OS	RHEL v6
Vendor	
Memory per node	64 GB
Detailed Cluster Description	
User Guide	
Available Software	

A typical compute node on SuperMIC

Two Intel® Xeon® CPUs	One Intel® Xeon Phi™ coprocessor	Two Intel® Xeon Phi™ coprocessors
		
• host 20 cores 64 GB memory	◈ mic0 61 cores (244 logical) 16 GB memory	◈ mic1 61 cores (244 logical) 16 GB memory

- Theoretical maximum acceleration:

One Xeon Phi / Two Xeons = 1208 GFLOPS / 448 GFLOPS = 2.7

(Tow Xeons + Two Xeon Phis) / Two Xeons = (2*1208 + 448) GFLOPS / 448 GFLOPS = 6.4

Xeon Phi programming

- Native mode
  - vectorization performance
- Offloading
  - Explicit offload
  - MKL automatic offload
  - MPI + offload
- Symmetric processing
  - for one node
  - for on multi nodes

Getting started …

❑ Window 1 (run jobs)

- ssh username@smic.hpc.lsu.edu # login SuperMIC
- qsub -I -A allocation_name -l nodes=2:ppn=20,walltime=hh:mm:ss # interactive session

❑ Window 2 or 3 (monitor performance)

- ssh -X username@smic.hpc.lsu.edu # login SuperMIC with graphics
- ssh -X smic{number} # login the compute node with graphics
- micsmc & (or micsmc-gui &) # open Xeon phi monitor from the host
- ssh mic0 # login mic0
- top # monitor processes on Xeon Phi

Show Xeon Phi information

```
[shaohao@smic021 ~]$ lspci | grep Co-processor
03:00.0 Co-processor: Intel Corporation Device 225c (rev 20)
83:00.0 Co-processor: Intel Corporation Device 225c (rev 20)

[shaohao@smic021 ~]$ micinfo
……
 Cores
            Total No of Active Cores : 61
            Voltage                  : 1052000 uV
            Frequency                : 1238095 kHz
……
 GDDR
            GDDR Vendor              : Samsung
            GDDR Version             : 0x6
            GDDR Density             : 4096 Mb
            GDDR Size                : 15872 MB
```

Native mode

Computer codes

execute natively

❑ An example (vector_omp.c): vector addition, parallelized with OpenMP.

- No change to normal CPU source codes.

❑ Compilation

- Always compile codes on the host. Compiler is not available on Xeon Phi.
- icc -O3 -openmp vector_omp.c -o vec.omp.cpu # CPU binary
- icc -O3 -openmp -mmic vector_omp.c -o vec.omp.mic # MIC binary

❑execute CPU binary on the host

- export LD_LIBRARY_PATH=/usr/local/compilers/Intel/composer_xe_2013.5.192/compiler/lib/intel64
  # specify libs for CPU
- export OMP_NUM_THREADS=20 # set OepnMP threads on host. Maximum is 20.
- ./vec.omp.cpu # run on the host

❑execute MIC binary on Xeon Phi natively

- ssh mic0 # login mic0
- export LD_LIBRARY_PATH=/usr/local/compilers/Intel/composer_xe_2013.5.192/compiler/lib/mic
  # specify libs for MIC
- export OMP_NUM_THREADS=244 # Set OepnMP threads on mic0. Maximum is 244.
- ./vec.omp.mic # Run natively on mic0

❑Exercise 1: Native run and affinity setting

i) Compile vector_omp.c with and without the flag -mmic, then execute the binaries on the host and on Xeon Phi respectively.

ii) Set up the affinity environment (e.g. export KMP_AFFINITY= compact,granularity=fine,verbose), then execute the MIC binary natively on Xeon Phi. Change “compact” to “scatter” or “balanced” then run it again. Observe the outputs.

Vectorization performance

❑Compare performance with and without vectorization

- An example (vector.c): a serial code for vector addition.
- icc -O3 -openmp -mmic vector.c -o vec.mic # vectorized by default
- icc -O3 -openmp -mmic -no-vec vector.c -o novec.mic # no vectorization
- Vectorized code is around 10 times faster!

❑Exercise 2: vectorization

i) Compile vector.c with and without the flag -no-vec, then run the binaries and compare the computational time.

ii) Compile vector.c with the flag -vec-report3, then vary the vector report number from 1 to 7. Observe the outputs.

Summary for Native mode

- Add flag -mmic to create MIC binary files.
- ssh to MIC and execute MIC binary natively.
- Vectorization is critical.
- Monitor MIC performance with micsmc.

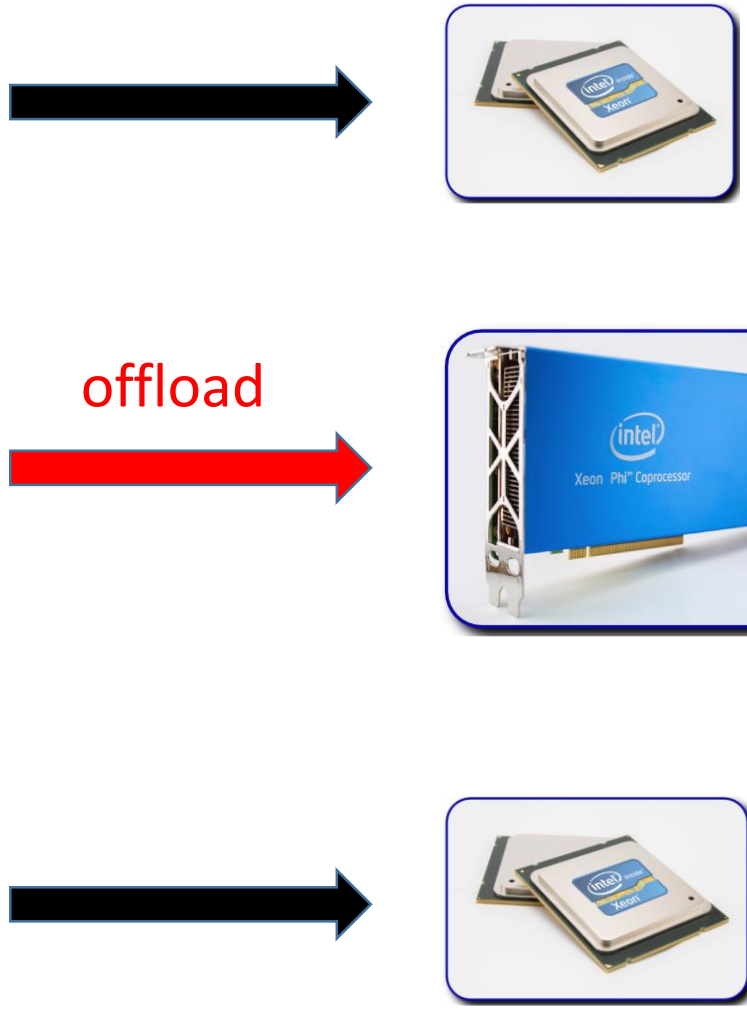
Offloading

❑ A C code with explicit offload (off02block.c)

```
int totalProcs;
int maxThreads;

#pragma offload target(mic:0)
 { // begin offload block
   totalProcs = omp_get_num_procs();
   maxThreads = omp_get_max_threads();
 } // end offload block

printf( "   total procs: %d\n", totalProcs );
printf( "   max threads: %d\n", maxThreads );
```



Explicit Offload: compilation and run

❑Compile

- The same as compiling normal CPU codes. Without -mmic.
- icc -openmp name.c -o name.off # C
- ifort -openmp name.f90 -o name.off # Fortran

❑Execute offloading jobs from the host

- export MIC_ENV_PREFIX=MIC # set the prefix if launch from the host.
- export MIC_OMP_NUM_THREADS=240 # set number of threads for MIC (The default is the maximum, that is 240, not 244. Leave one core with 4 threads to execute offloading.)
- ./name.off # launch from the host

❑Exercise 3 (a): report offloading information

i) Compile off02block.c, then launch the binary from the host.

ii) Export the value of OFFLOAD_REPORT in the range of 1, 2 and 3. Then run it again and analyze the outputs.

iii) Compile off02block.c with the flag -opt-report-phase=offload. Observe the outputs.

❑Exercise 3 (b):

Do exercise 3 (a) with the Fortran code off02block.f90 .

Offload an OpenMP region

- Spread OpenMP threads to 240 workers (logical threads) of MIC.
- Assign values to a vector (C code: off03omp.c)

```
double a[500000];    // placed on host
int i;      // placed on host
 #pragma offload target(mic:0)    // auto pass i and a in and out of MIC
  #pragma omp parallel for
    for ( i=0; i<500000; i++ )  {
      a[i] = (double)i;
    }
 printf( "\n\t last val  = %f \n", a[500000-1]);   // output
```

❑ Assign values to a vector (Fortran code: off03omp.f90)

```
integer, parameter :: N = 500000
real :: a(N)        ! placed on host
!dir$ offload target(mic:0)   ! auto pass i and a in and out of MIC
!$omp parallel do
do i=1,N
    a(i) = real(i)
end do
!$omp end parallel do
print*, "last val is ", a(N)   ! output
```

- Exercise 4 (a): Compile and run off03omp.c. Report offloading information and analyze data transfer between host and MIC.

- Exercise 4 (b): Compile and run off03omp.f90. Report offloading information and analyze data transfer between host and MIC.

- Note: Data transfer between CPU and MIC influences the performance.

  Less transferred data is better.

Control data transfer between host and MIC

- in, out, inout (C code: off06stack.c)

```
……
  double a[100000], b[100000], c[100000], d[100000];
  int i;
  for ( i=0; i<100000; i++ )  {
    a[i] = 1.0;      b[i] = (double)(i);
  }
  #pragma offload target(mic:0)  in( a ) out( c, d ) inout( b )
  #pragma omp parallel for
  for ( i=0; i<100000; i++ )  {
      c[i] =  a[i] + b[i];    d[i] =  a[i] - b[i];    b[i] = -b[i];
  }
……
```

❑ in, out, inout (Fortran code: off06stack.f90)

```
.....
integer, parameter :: N = 100000
real(8)          :: a(N), b(N), c(N), d(N)
do i=1,N
   a(i) = 1.0d0;     b(i) = real(i)
 end do
!dir$ offload target(mic)  in( a ), out( c, d ), inout( b )
!$omp parallel do
     do i=1,N
        c(i) =  a(i) + b(i);       d(i) =  a(i) - b(i);       b(i) = -b(i)
     end do
!$omp end parallel do
.....
```

Transfer dynamic arrays

- length(), alloc_if, free_if (C code: off07heap.c)

```
……
int i;     int N = 5000000;      double *a, *b;
a = ( double* ) memalign( 64, N*sizeof(double) );
b = ( double* ) memalign( 64, N*sizeof(double) );
for ( i=0; i<N; i++ )  { a[i] = (double)(i);  }
#pragma offload target(mic:0)  in(  a : length(N) alloc_if(1) free_if(1)  ), \
  out(  b : length(N) alloc_if(1) free_if(1)  )      // length(N) is required for dynamic arrays
#pragma omp parallel for
     for ( i=0; i<N; i++ )  {
        b[i] = 2.0 * a[i];
     }
……
```

❑ alloc_if, free_if (Fortran code: off07heap.f90)

```
……
real(8), allocatable  :: a(:), b(:)
integer, parameter :: N = 5000000
allocate(  a(N), b(N)  )
do i=1,N
   a(i) = real(i)
end do
!dir$ offload target(mic)   in(  a : alloc_if(.true.) free_if(.true.) ), out( b : alloc_if(.true.) free_if(.true.) )
!$omp parallel do
do i=1,N
      b(i) = 2.0 * a(i)
 end do
!$omp end parallel do
```

❑Exercise 5 (a): control data transfer

i) Compile and run off06stack.c and off07heap.c.

ii) Report offloading information and analyze data transfer between host and MIC in these cases.

❑Exercise 5 (b):

Do exercise 5 (a) with the Fortran codes off06stack.f90 and off07heap.f90.

Place valuables on MIC

- attribute decorations (C code: off05global.c)

```
……
__attribute__( ( target(mic:0) ) ) int myGlobalInt;  // global valuable, available on MIC
int main( void ) {
    ……
    int myLocalInt = 123;       // local valuable, not available on MIC
    __attribute__( ( target(mic:0) ) ) int myStaticInt; // local valuable, available on MIC
    #pragma offload target(mic:0){   // auto IN: myLocalInt; auto OUT: myGlobalInt, myStaticInt
      myGlobalInt = 2 * myLocalInt;
      myStaticInt = 2 * myGlobalInt;
    }
     ……
}
```

❑attribute decorations (Fortran code: off05global.f90)

```
module mymodvars
   !dir$ attributes offload:mic :: mymoduleint
   integer    :: mymoduleint       ! global valuable, available on MIC
end module mymodvars
program main
   use mymodvars
   implicit  none
   integer  :: mylocalint = 123    ! local valuable, not available on MIC
   integer, save :: mysaveint      ! local valuable, available on MIC
   !dir$ offload begin target(mic)     ! auto IN: mylocalint; auto OUT: mymoduleint, mysaveint
      mymoduleint = 2 * mylocalint
      mysaveint   = 2 * mymoduleint
   !dir$ end offload
……
end program
```

Exercise 6 (a): declspec/attribute decorations

i) Compile and run off05global.c.

ii) Report offloading information (export OFFLOAD_REPORT=3) and analyze data transfer between host and MIC.

iii) Replace all __attribute__((target(mic:0))) with __declspec(target(mic:0)), then compile and run it again.

iv) Remove all attribute/declspec decorations, then compile the code with the flag "-offload-attribute-target=mic". Run it again.

Exercise 6 (b):

Do sections i, ii and iv of exercise 5 (a) with the Fortran code off05global.f90 .

Asynchronous offload

- When offloading works to the MIC, the host is empty.
- When the MIC is busy, can the host do some other works? Yes.
- How? Asynchronous offload!
- Be careful about synchronizing the works between the host and the MIC.

❑ wait, signal, offload_wait (C code: off08asynch.c)

```
……
int n   = 123;
 #pragma offload target(mic:0)  signal( &tag ){    //record this offload event &tag
     printf( "\n\t logical cores on mic: %d\n\n", omp_get_num_procs() );   // total MIC threads
     printf( "\n\t maximum threads on mic: %d\n\n", omp_get_max_threads() );  // used MIC threads
     incrementSlowly( &n );    // n increases 1 then sleep 2 seconds on MIC
}
……    // the host can do something else here, while MIC is busy.
#pragma offload_wait target(mic:0)  wait( &tag ) {   //Host does not execute the following codes until the event &tag is finished.
     printf( "\n\t logical cores: %d\n\n", omp_get_num_procs() );      // total host threads
     printf( "\n\t maximum threads: %d\n\n", omp_get_max_threads() );   // used host threads
}
 if ( n == 123 )  { printf( "\n\tThe offload increment has NOT finished...\n" ); } // n does not change without waiting
else { printf( "\n\tThe offload increment DID finish successfully...\n" ); } // n increases 1 after waiting
……
```

❑ wait, signal, offload_wait (Fortran code: off08asynch.f90)

```
……
 integer  :: n = 123
!dir$ offload begin target(mic:0)  signal( tag )     ! record this offload event &tag
   call incrementslowly( n )      ! n increases 1 then sleep 2 seconds on MIC
!dir$ end offload
……      ! the host can do something else here, while MIC is busy.

!dir$ offload_wait target(mic:0)  wait( tag )     //Host does not execute the following codes until event &tag is finished.
print *, "  procs: ", omp_get_num_procs()

if ( n .eq. 123 )  then
   print *, "  The offload increment has NOT finished... ", "  n: ", n          // n does not change without waiting
 else
   print *, "  The offload increment DID finish successfully... ", "  n: ", n     // n increases 1 after waiting
 endif
……
```

❑Exercise 7 (a): asynchronous offload

i) Compile and run 0ff08asynch.c . Does the value of n increase?

ii) Comment out the line with wait, then compile and run again. Does the value of n increase? Why?

iii) Change the key word offload_wait to offload, then compile and run again. Observe the number of threads in the output. Is it changed? Why?

❑Exercise 7 (b):

Do sections i and ii of exercise 7 (a) with the Fortran code off08asynch.f90. For section iii, output the number of MIC threads instead of CPU threads.

Data-only offload

❑ Offload_transfer, nocopy (C code: off09transfer.c)

```
……
 a = ( double* ) memalign( 64, N*sizeof(double)      // allocate alligned memory on host
 b = ( double* ) memalign( 64, N*sizeof(double) );

#pragma offload_transfer target(mic:0)  nocopy( a : length(N) alloc_if(1) free_if(0) ) \
   nocopy( b : length(N) alloc_if(1) free_if(0) ) signal( &tag1 )   // allocate memory on MIC

 for ( i=0; i<N; i++ )  { a[i] = (double)(i);    // assign value on host

// after tag1 is finished, copy a from host to MIC,
#pragma offload target(mic:0) wait( &tag1 )  in(    a : length(N) alloc_if(0) free_if(0) ) \
   out(   b : length(N) alloc_if(0) free_if(0) )  signal( &tag2 )
#pragma omp parallel for
      for ( i=0; i<N; i++ )  {  b[i] = 2.0 * a[i];       // calculate b on MIC
```

```
// (…… continued from the previous slide)
#pragma offload_transfer target(mic:0) wait( &tag2 ) \        // after tag2 is finished
  nocopy( a : length(N) alloc_if(0) free_if(1) ) \    // deallocate a on mic
  out( b : length(N) alloc_if(0) free_if(1) ) \     // copy b from mic to host, deallocate b on mic
  signal( &tag3 )

#pragma offload_wait target(mic:0) wait( &tag3 )      // wait until tag3 is finished
 {
    printf( "\n\t last a val  = %f",     a[N-1]);          // print values on the host
    printf( "\n\t last b val  = %f\n\n", b[N-1]);
 }

……
```

Automatic offload with Intel MKL

- Intel Math Kernel Library (MKL):

highly vectorized and threaded Linear Algebra, Fast Fourier Transforms (FFT), Vector Math and Statistics functions.

Intel MKL Automatic Offloading environment variables

Environment Variable	Equivalent Support Function	Purpose
MKL_MIC_ENABLE	mkl_mic_enable	Enabling and disabling automatic offload
MKL_MIC_WORKDIVISION MKL_MIC[0,1]_WORKDIVISION MKL_HOST_WORKDIVISION	mkl_mic_set_workdivision	Controlling work division
MKL_MIC_MAX_MEMORY	mkl_mic_set_max_memory	Controlling maximum memory used by Automatic Offload

an example for auto-offload

- Matrix product and addition: C = alpha*A*B + beta*C

- ao_intel.c

```
……
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m, n, p, alpha, A, p, B, n, beta, C, n);    // Double-precision General Matrix Multiplication
……
```

- ao_intel.f

```
……
CALL DGEMM('N','N',M,N,P,ALPHA,A,M,B,P,BETA,C,M)
……
```

Auto-offload: compile

- No modification of the normal CPU source code!

- Compilation (the same for normal CPU code)
  - icc -openmp -mkl ao_intel.c -o ao_intel
  - ifort -openmp -mkl ao_intel.f -o ao_intel

Auto-offloading: run

- Run auto-offloading jobs
  - export MKL_MIC_ENABLE=1 # enable auto offload, also set the prefix MIC_
  - export OMP_NUM_THREADS=20 # set CPU threads
  - export MIC_OMP_NUM_THREADS=240 # set MIC threads from the host
  - export OFFLOAD_REPORT=2 # offload report level
  - ./ao_intel

- Depending on the problem size and the current status of the devices, the MKL runtime will determine how to divide the work between the host CPU's and the Xeon Phi's

Exercise 8 (a): automatic offload with MKL

i) Compile and run ao_intel.c. Observe the usage of MICs on the “micsmc” monitor.

ii) Compare the computational time with and without automatic offload.

iii) Change the number of threads on the host and on the MICs. Observe the variation of computational time.

iv) Increase the problem size from small to large and observe the results. At what threshold(s) does MKL begin to use the MIC?

Exercise 8 (b):

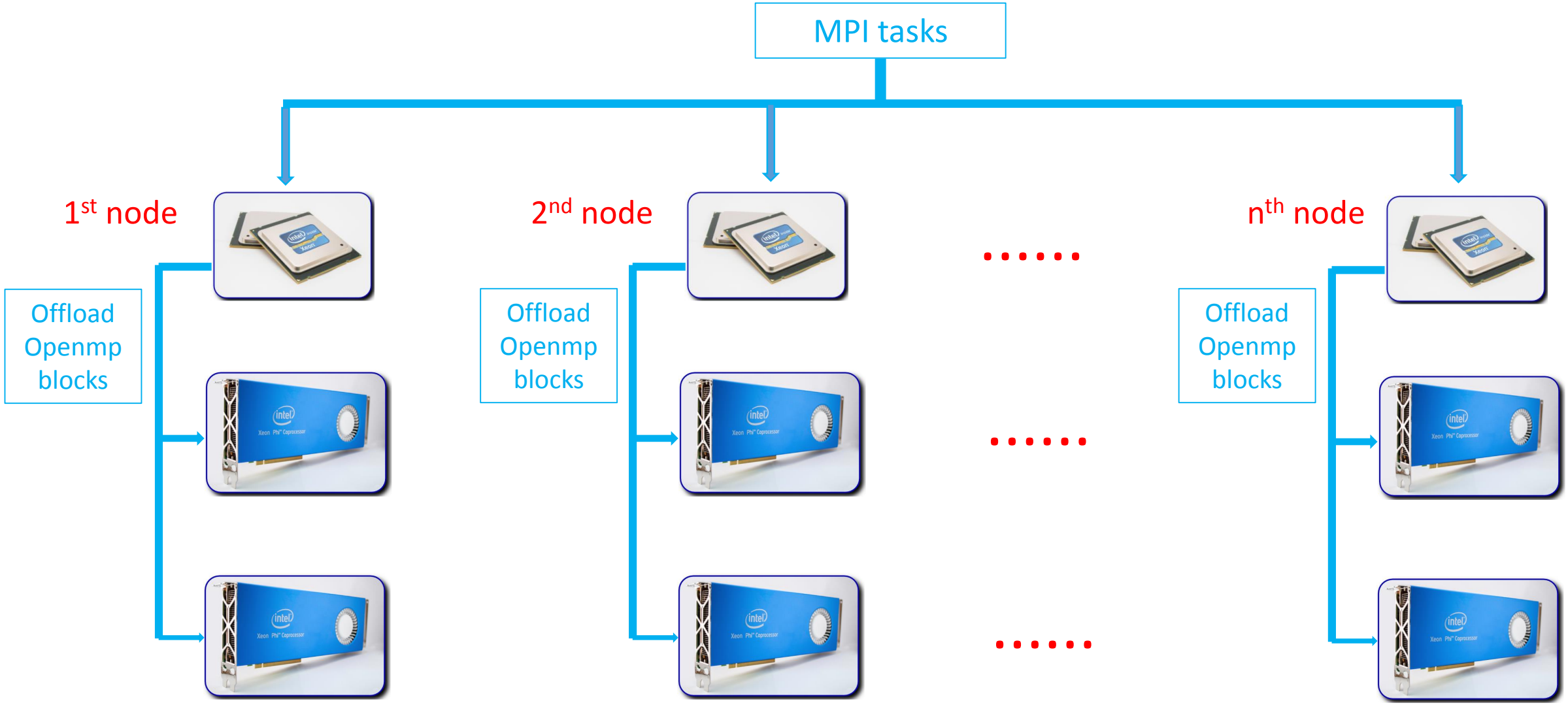
Do exercise 8 (a) with the Fortran code ao_intel.f .

Using MPI and offload together

- MPI is required to run jobs on multi nodes.
- Offloading of MPI functions? No.

  Calling MPI functions within an offload region is not supported.
- Offload OpenMP blocks in MPI-OpenMP hybrid codes? Yes!

MPI + Offload


MPI tasks
1st node
2nd node
nth node
Offload Openmp blocks
Offload Openmp blocks
Offload Openmp blocks

An example for MPI + Offload: Calculate the value of pi

- Numerical integration:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

- Parallelization scheme:

  Distribute the integration grids into various MPI tasks,

  then spread every MPI task into various OpenMP threads.

MPI-OpenMP hybrid codes with offload

- Calculate pi (C code: pi_hybrid_off.c)

```
……
MPI_Init( &argc, &argv );    //  MPI functions
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
……
#pragma  offload target (mic:myrank) in(start_int,end_int) //Offload OepnMP block of each MPI task to one MIC
#pragma omp parallel private(iam,i,np){    //  OpenMP block
   iam = omp_get_thread_num() ;
   np=omp_get_num_threads() ;
   printf("Thread %5d of %5d in MPI task %5d of %5d\n",iam,np,myrank,nprocs);
   ……
}
……
```

❑ Calculate pi (Fortran code: pi_hybrid_off.f90)

```
.....
call mpi_init(ierr) //  MPI functions
      mpi_comm_size(mpi_comm_world,nprocs,ierr)
call mpi_comm_rank(mpi_comm_world,myrank,ierr)
.....
!dir$ offload begin target (mic:myrank) in(start_int,end_int) //Offload OepnMP block of each MPI task to one MIC
   !$omp parallel private(iam,i,np) //  OpenMP block
       iam = omp_get_thread_num()
       np=omp_get_num_threads()
       write(*,*) iam, myrank, np,nprocs
       .....
   !$omp end parallel
!dir$  end offload
.....
```

MPI + Offload: compile

- Use Intel MPI implementation (a better option than MVAPICH2)
  - module switch mvapich2/2.0/INTEL-14.0.2 impi/4.1.3.048/intel64
  - module load impi/4.1.3.048/intel64

- Compile
  - mpiicc -O3 -openmp pi_hybrid_off.c -o pi_hybrid.off
  - mpiifort -O3 -openmp pi_hybrid_off.f90 -o pi_hybrid.off

MPI + Offload: run

- Run with mpiexec.hydra
  - export OFFLOAD_REPORT=2 # level-2 offload report
  - export MIC_ENV_PREFIX=MIC # make prefix simple
  - export MIC_OMP_NUM_THREADS=240 # number of threads on MIC
  - export MIC_KMP_AFFINITY=scatter # affinity type on MIC
  - mpiexec.hydra -n 2 -machinefile nodefile ./pi_hybrid.off # specify node names in node file

Exercise 9 (a): Offload OpenMP blocks in MPI-OpenMP hybrid codes

i) Compile pi_hybrid_off.c, then run it on one node and two nodes respectively, with two MPI tasks per node. Observe the usage of MICs on the micsmc monitor.

ii) Change the number of threads on the MICs. Observe the variation of computational time.

iii) Change the number of MPI tasks to 1 per node, run it again. How many MICs on each node are utilized now?

iv) Compare the computational time of the following cases: 1) without offloading; 2) offload to one MIC; 3) offload to two MICs.

Exercise 9 (b):

Do exercise 9 (a) with the Fortran code pi_hybrid_off.f90 .

Summary for offloading

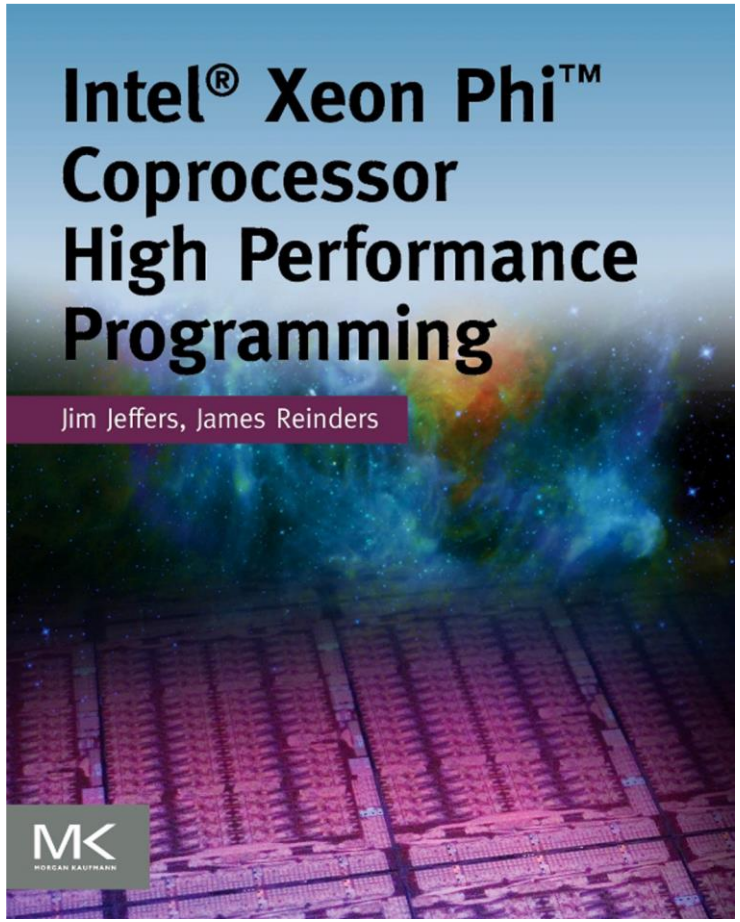
- Explicitly offload blocks by adding lines started with #pragma offload or !dir$ offload in C or Forthran source codes respectively.
- Control data transfer with in, out and inout.
- Place valuables on MIC with attribute or declspec decorations.
- Use wait and offload_wait for asynchronous offload.
- Use offload_transfer for data-only offload.
- Auto offload MKL functions by setting MKL_MIC_ENABLE=1.
- Offload OpenMP blocks in MPI-OpenMP hybrid codes.

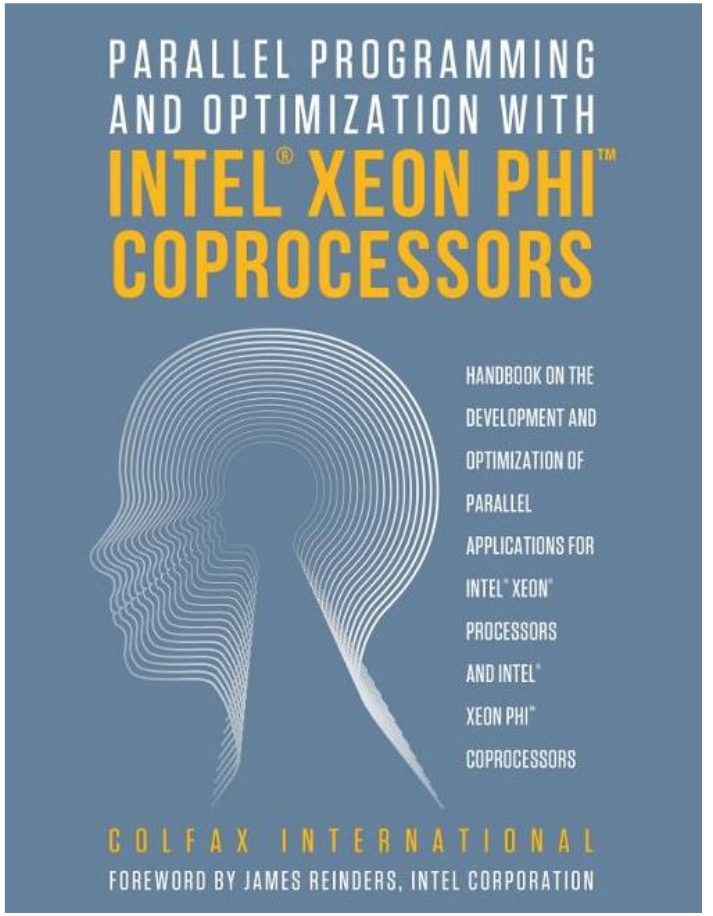
Next training: Xeon Phi programming II

Date: April 1, 2015
Time: 9:30 AM - 11:30 AM

- Xeon Phi programming: symmetric processing
- Optimization, Debug and profiling
- Xeon-Phi enabled packages

References





- User guide of SuperMIC: http://www.hpc.lsu.edu/docs/guides.php?system=SuperMIC