



PROGRAMMING AND OPTIMIZATION FOR INTEL[®] ARCHITECTURE

Hands-On Workshop (HOW) Series "Deep Dive"
Session 2

Colfax International — colfaxresearch.com

May 2017

While best efforts have been used in preparing this training, Colfax International makes no representations or warranties of any kind and assumes no liabilities of any kind with respect to the accuracy or completeness of the contents and specifically disclaims any implied warranties of merchantability or fitness of use for a particular purpose. The publisher shall not be held liable or responsible to any person or entity with respect to any loss or incidental or consequential damages caused, or alleged to have been caused, directly or indirectly, by the information or programs contained herein. No warranty may be created or extended by sales representatives or written sales materials.

- ▶ **Module I. Programming**
 - 01. Intel Architecture and Modern Code – May 15
 - 02. Xeon Phi, Coprocessors, Omni-Path – May 16
- ▶ **Module II. Expressing Parallelism**
 - 03. Automatic vectorization – May 17
 - 04. Multi-threading with OpenMP – May 18
 - 05. Distributed Computing, MPI – May 19
- ▶ **Module III. Optimization**
 - 06. Optimization Overview: N-body – May 22
 - 07. Scalar tuning, Vectorization – May 23
 - 08. Common Multi-threading Problems – May 24
 - 09. Multi-threading, Memory Aspect – May 25
 - 10. Access to Caches and Memory – May 26

May 2017						
S	M	T	W	H	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Course page:

colfaxresearch.com/how-17-05

- ▶ Slides
- ▶ Code
- ▶ Video
- ▶ Chat

More workshops:

colfaxresearch.com/training



GET YOUR QUESTIONS ANSWERED: CHAT



colfaxresearch.com/how-17-05

GET YOUR QUESTIONS ANSWERED: FORUMS



Forum

Colfax Cluster

Discussion of Colfax Cluster usage policies, troubleshooting.

Developer Training, HOW Series

Questions about any of the Colfax trainings? Usage of training servers, experience with specific exercises, inquiries on what's inside, suggestions for future trainings - post them here.

Performance Optimization and Parallelism

Discuss with Colfax Research and colleagues any topics related to computational science, parallel programming, performance optimization and code modernization.

colfaxresearch.com/discussion

- ▶ All registrants receive an invitation from `cluster@colfaxresearch.com`
- ▶ Queue-based access to Intel Xeon E5, Intel Xeon Phi (KNC and KNL)
- ▶ Can access the cluster the entire 2 weeks of the workshop





§2. ROADMAP OF INTEL ARCHITECTURE

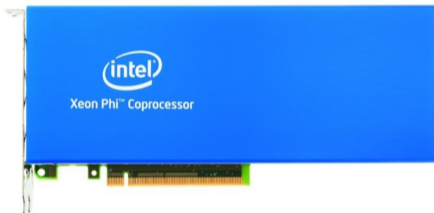
Intel Xeon Processor



Current: Broadwell
Upcoming: Skylake

Multi-Core Architecture

Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

Intel Xeon Phi Processor, 2nd generation*







* socket and coprocessor versions

Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture

INTEL XEON PHI PROCESSORS

	Knights Corner	Knights Landing	Knights Mill	Knights Hill
Lith	22 nm	14 nm	14 nm	10 nm
Models	71xx P/A	72xx, 72xx F	?	?
Form-factors	coprocessor 	processor  coprocessor  processor with fabric 	?	?



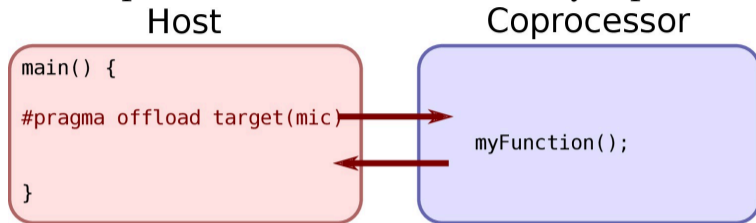
§3. PROGRAMMING COPROCESSORS



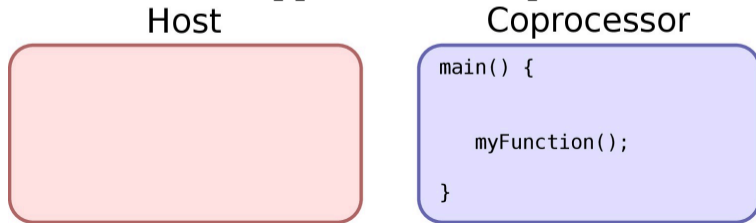
OFFLOAD AND NATIVE MODELS

OFFLOAD AND NATIVE MODELS

- ▶ Offload model (explicit/virtual-shared memory/OpenMP 4.0):



- ▶ Native model (standalone application/MPI process):



OFFLOAD OR NATIVE: HOW TO DECIDE

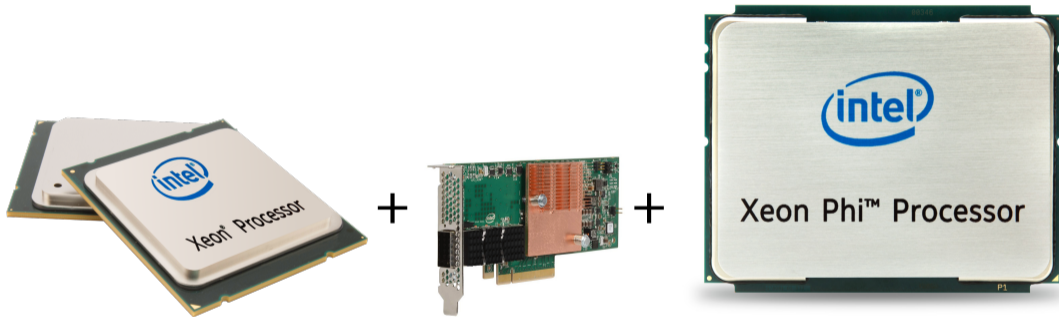
Native	Offload
≤ 16 GiB	> 16 GiB
All parallel	Parallel + serial phases
Complex data structures	Bitwise-copyable data
Any arithmetic intensity	(FLOPs/transfer) \gg 1000

Native = same code on CPU and MIC

Offload = must insert directives in code

OFFLOAD OVER FABRIC

Heterogeneous computing is possible even with bootable KNL



Tutorial: **Offload over Fabric to Intel Xeon Phi Processor**



NATIVE PROGRAMMING (KNC)

INTEL COMPILERS + INTEL XEON PROCESSOR

“Hello World” application:

```
1 #include <stdio>
2 #include <unistd.h>
3 int main(){
4     printf("Hello world! I have %ld logical processors.\n",
5         sysconf(_SC_NPROCESSORS_ONLN ));
6 }
```

Compile and run on host CPU:

```
vega@lyra% icpc hello.cc -xhost
vega@lyra% ./a.out
Hello world! I have 48 logical processors.
vega@lyra%
```

NATIVE EXECUTION ON AN INTEL XEON PHI COPROCESSOR (KNC)

Compile and run the same code on the coprocessor in the native mode:

```
vega@lyra% icpc hello.cc -mmic # Cross-compile
vega@lyra% scp a.out mic0:~/ # Put executable on coprocessor
a.out 100% 10KB 10.4KB/s 00:00
vega@lyra% ssh mic0 # Log in to coprocessor
vega@mic0% pwd
/home/lyra
vega@mic0% ls
a.out
vega@mic0% ./a.out # Launch application
Hello world! I have 244 logical processors.
vega@mic0%
```

- ▶ Use `-mmic` to produce executable for MIC architecture
- ▶ Must transfer executable to coprocessor (or NFS-share) and run from shell
- ▶ Native MPI applications work the same way (need Intel MPI library)

NATIVE APPLICATIONS WITH AUTOTOOLS

- ▶ Use the Intel compiler with flag `-mmic`
- ▶ Knights Landing: `-xMIC-AVX512`
- ▶ Eliminate assembly and unnecessary dependencies
- ▶ Use `--host=x86_64` to avoid “program does not run” errors

Example, the GNU Multiple Precision Arithmetic Library (GMP):

```
vega@lyra% wget https://ftp.gnu.org/gnu/gmp/gmp-5.1.3.tar.bz2
vega@lyra% tar -xf gmp-5.1.3.tar.bz2
vega@lyra% cd gmp-5.1.3
vega@lyra% ./configure CC=icc CFLAGS="-mmic" --host=x86_64 --disable-assembly
...
vega@lyra% make
...
```



EXPLICIT OFFLOAD (LEO)

EXPLICIT OFFLOAD: PRAGMA-BASED APPROACH

“Hello World” in the explicit offload model:

```
1 #include <stdio>
2 int main() {
3     printf("Hello World from host!\n");
4     #pragma offload target(mic)
5     {
6         printf("Hello World from coprocessor!\n"); fflush(stdout);
7     }
8     printf("Bye\n");
9 }
```

Application runs on the host, but some parts of code and data are moved (“offloaded”) to the coprocessor.

Detailed syntax in the [Intel C++ Compiler Reference](#).

COMPILING AND RUNNING AN OFFLOAD APPLICATION

```
vega@lyra% icpc hello_offload.cc -o hello_offload
vega@lyra% ./hello_offload
Hello World from host!
Bye
Hello World from coprocessor!
```

- ▶ No additional arguments (for Intel compiler)
- ▶ Launch on host as a regular application
- ▶ Code inside of `#pragma offload` is offloaded automatically
- ▶ Console output on coprocessor buffered, mirrored to the host
- ▶ If no coprocessor available, default behavior is error; may be overridden to fall back to host



OFFLOADING FUNCTIONS AND DATA

OFFLOADING MULTIPLE FUNCTIONS

```
1 #pragma offload_attribute(push, target(mic))
2 void MyFunctionOne() {
3   // ... implement function as usual
4 }
5
6 void MyFunctionTwo() {
7   // ... implement function as usual
8 }
9 #pragma offload_attribute(pop)
```

- ▶ To mark a long block of code with the offload attribute, use `#pragma offload_attribute(push/pop)`

OFFLOADING DATA: LOCAL SCALARS AND ARRAYS

```
1 void MyFunction() {  
2     const int N = 1000;  
3     int data[N];  
4     #pragma offload target(mic)  
5     {  
6         for (int i = 0; i < N; i++)  
7             data[i] = 0;  
8     }
```

- ▶ Scope-local scalars and known-size arrays offloaded automatically
- ▶ Data is copied from host to coprocessor at the start of offload
- ▶ Data is copied back from coprocessor to host at the end of offload
- ▶ Bitwise-copyable data only (arrays of basic types and scalars)
C++ classes, etc. should use virtual-shared memory model

DATA MARSHALLING FOR DYNAMICALLY ALLOCATED DATA

```
1 double *p1=(double*)malloc(sizeof(double)*N);  
2 double *p2=(double*)malloc(sizeof(double)*N);  
3  
4 #pragma offload target(mic) in(p1 : length(N)) out(p2 : length(N))  
5 {  
6     // ... perform operations on p1[] and p2[]  
7 }
```

- ▶ #pragma offload recognizes clauses in, out, inout and nocopy
- ▶ Data size (length), alignment, redirection, and other properties may be specified
- ▶ Marshalling is required for pointer-based data

OPTIONAL OFFLOAD, FALL-BACK TO HOST

```
1 #pragma offload target(mic) optional
2 {
3     printf("Hello World! I have %d logical processors.\n",
4         sysconf(_SC_NPROCESSORS_ONLN )); fflush(stdout);
5 }
```

```
vega@lyra% icpc Offload-Fallback.cc -o Offload-Fallback
vega@lyra% ./Offload-Fallback
Hello World! I have 244 logical processors.
vega@lyra% sudo systemctl stop mpss # Disabling coprocessors
vega@lyra% ./Offload-Fallback
Hello World! I have 48 logical processors.
```



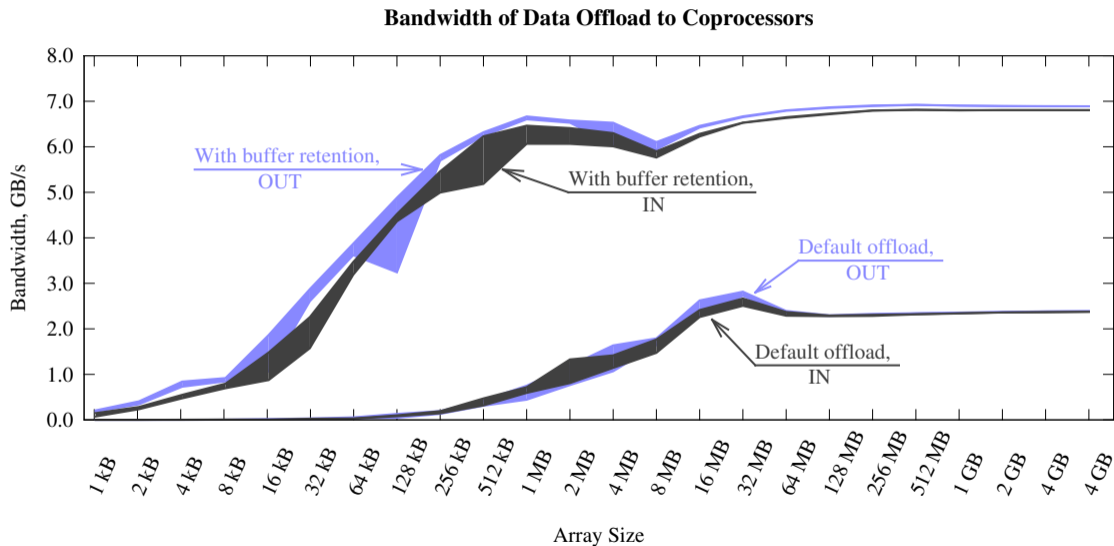
MEMORY ALLOCATION CONTROL

MEMORY RETENTION AND DATA PERSISTENCE ON COPROCESSOR

- ▶ By default, memory on coprocessor is allocated before, deallocated after offload
- ▶ Specifiers `alloc_if` and `free_if` allow to avoid allocation/deallocation
- ▶ Data transfer across the PCIe bus rate is ≈ 7 GB/s
- ▶ To allocate memory on the coprocessor – 0.5-2.0 GB/s

```
1 #pragma offload target(mic:0) in(p : length(N) alloc_if(1) free_if(0) )
2 { /* allocate memory for array p on coprocessor, do not deallocate */ }
3
4 #pragma offload target(mic:0) in(p : length(N) alloc_if(0) free_if(0) )
5 { /* re-use previously allocated memory buffer on coprocessor */ }
6
7 #pragma offload target(mic:0) in(p : length(0) alloc_if(0) free_if(0) )
8 { /* re-use previously transferred data on coprocessor */ }
9
10 #pragma offload target(mic:0) out(p : length(N) alloc_if(0) free_if(1) )
11 { /* re-use memory and deallocate at the end of offload */ }
```

OFFLOAD LATENCY WITH AND WITHOUT MEMORY/DATA RETENTION





ADDITIONAL OFFLOAD CONTROLS

TARGET-SPECIFIC CODE

- ▶ During MIC architecture compilation, preprocessor macro `__MIC__` is defined.
- ▶ Allows to fine-tune application performance or output where necessary

```
1 __attribute__((target(mic))) void MyFunction() {  
2 #ifdef __MIC__  
3     printf("I am running on a coprocessor.\n");  
4     const int tuningParameter = 16;  
5 #else  
6     printf("I am running on the host.\n");  
7     const int tuningParameter = 8;  
8 #endif  
9     // ... Proceed, using the variable tuningParameter  
10 }
```

OFFLOAD DIAGNOSTICS

```
vega@lyra% export OFFLOAD_REPORT=2
vega@lyra% ./offload-application
Transferring some data to and from coprocessor...
Done. Bye!
[Offload] [MIC 0] [File]                offload-application.cc
[Offload] [MIC 0] [Line]                6
[Offload] [MIC 0] [CPU Time]            0.505982 (seconds)
[Offload] [MIC 0] [CPU->MIC Data]      1024 (bytes)
[Offload] [MIC 0] [MIC Time]           0.000409 (seconds)
[Offload] [MIC 0] [MIC->CPU Data]      1024 (bytes)
vega@lyra%
```

- ▶ Set environment variable `OFFLOAD_REPORT` to 1 or 2 for automatic collection and output of offload information.
- ▶ Unset or set `OFFLOAD_REPORT=0` to disable offload diagnostics

ENVIRONMENT VARIABLE FORWARDING WITH OFFLOAD

- ▶ By default, all host environment variables on the host will be copied to the coprocessor when offload starts.
- ▶ In order to have different values for an environment variable on host and coprocessor, set `MIC_ENV_PREFIX`
- ▶ The prefix is dropped when variables are copied to coprocessor

```
vega@lyra% # This sets the value of OMP_NUM_THREADS on the host:
vega@lyra% export OMP_NUM_THREADS=48
vega@lyra%
vega@lyra% # This enables special rules for variable copying:
vega@lyra% export MIC_ENV_PREFIX=XEONPHI
vega@lyra%
vega@lyra% # This sets the value of OMP_NUM_THREADS on the coprocessor:
vega@lyra% export XEONPHI_OMP_NUM_THREADS=240
```



OFFLOAD IN OPENMP 4.0

OPENMP 4.0 TARGET OFFLOAD

- ▶ Another API for offload: `#pragma omp target`
- ▶ Part of the OpenMP 4.0 standard
- ▶ Designed as portable solution (coprocessors, GPGPUs)
- ▶ On Xeon Phi, uses the same back-end as `#pragma offload`

```
1 #pragma omp target
2 {
3 #pragma omp parallel for
4   for(int i=0; i<size; i++)
5     data[i] = 0;
6 }
```

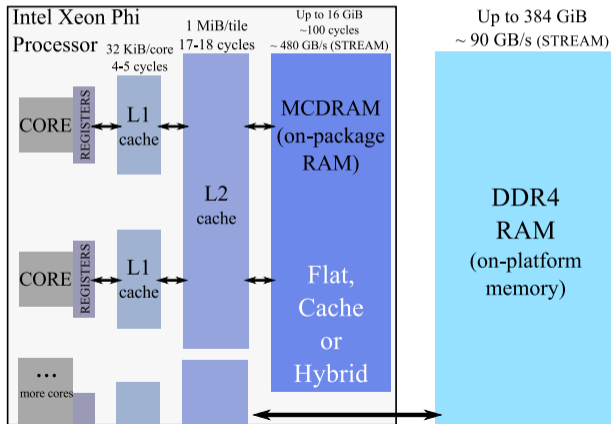
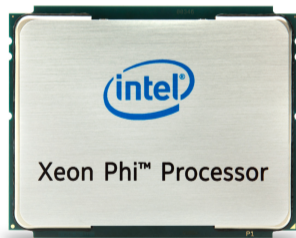
Application runs on the host, but some parts of code and data are moved (“offloaded”) the coprocessor. Scope-local scalars and stack arrays offloaded automatically.



§4. HIGH-BANDWIDTH MEMORY

KNL MEMORY ORGANIZATION (BOOTABLE)

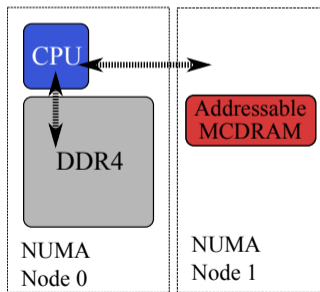
- ▶ On-package high-bandwidth memory (HBM) – MCDRAM
- ▶ Optimized for arithmetic performance and bandwidth (not latency)



HIGH-BANDWIDTH MEMORY MODES

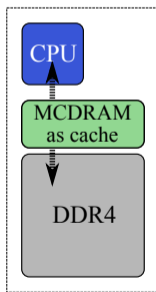
Flat Mode

- ▶ MCDRAM treated as a NUMA node
- ▶ Users control what goes to MCDRAM



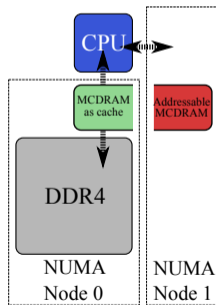
Cache Mode

- ▶ MCDRAM treated as a Last Level Cache (LLC)
- ▶ MCDRAM is used automatically



Hybrid Mode

- ▶ Combination of Flat and Cache
- ▶ Ratio can be chosen in the BIOS



RUNNING APPLICATIONS IN HBM WITH NUMACTL

- ▶ Finding information about the NUMA nodes in the system.

```
user@knl% # In Flat mode of MCDRAM
user@knl% numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... 254 255
node 0 size: 98207 MB
node 1 cpus:
node 1 size: 16384 MB
```

- ▶ Binding the application to HBM (Flat/Hybrid)

```
user@knl% icc myapp.c -o runme -xMIC_AVX512
user@knl% numactl --membind 1 ./runme
// ... Application running in HBM ... //
```

ALLOCATION IN HBM WITH MEMKIND LIBRARY

```
1 #include <hbwmalloc.h>
2 const int n = 1<<10;
3 // Allocation to MCDRAM
4 double* A = (double*) hbw_malloc(sizeof(double)*n);
5 // No replacement for _mm_malloc. Use posix_memalign
6 double* B;
7 int ret = hbw_posix_memalign((void**) &B, 64, sizeof(double)*n);
8 .....
9 // Free with hbw_free
10 hbw_free(A); hbw_free(B);
```

COMPILATION WITH MEMKIND LIBRARY AND HBWMALLOC

To compile C/C++ applications:

```
user@knl% icpc -lmemkind foo.cc -o runme
user@knl% g++ -lmemkind foo.cc -o runme
```

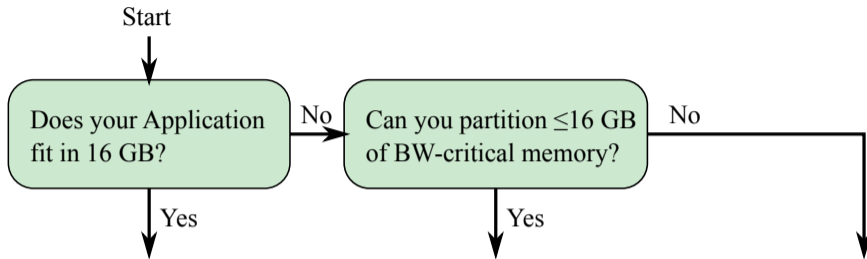
Open source distribution of Memkind library can be found at:

memkind.github.io/memkind

Learn more:

colfaxresearch.com/knl-mcdram

FLOW CHART FOR BANDWIDTH-BOUND APPLICATIONS



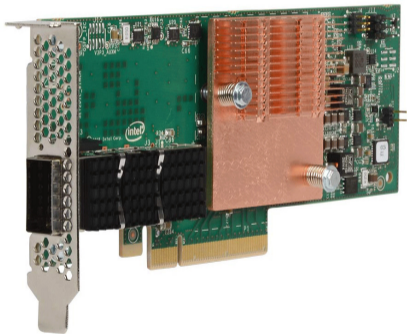
numactl	Memkind	Cache mode
<ul style="list-style-type: none"> ▶ Simply run the whole program in MCDRAM ▶ No code modification required 	<ul style="list-style-type: none"> ▶ Manually allocate BW-critical memory to MCDRAM ▶ Memkind calls need to be added. 	<ul style="list-style-type: none"> ▶ Allow the chip to figure out how to use MCDRAM ▶ No code modification required



§5. INTEL OMNI-PATH ARCHITECTURE

INTEL'S HPC COMMUNICATION FABRIC

Intel Omni-Path Architecture - low-latency, high-bandwidth, scalable communication fabric for HPC applications.



Discrete



Integrated

INTEL OMNI-PATH FABRIC 100 WITH INTEL XEON PROCESSORS

First generation: 100 Gbps bandwidth, \approx 1 microsecond latency

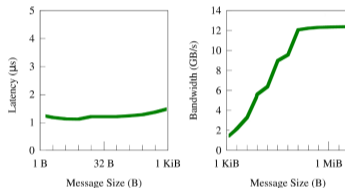
Intel® Omni-Path Fabric

```

File Edit View Search Terminal Help
[ryoc010-n003 ~]# cat /sys/module/hfi1/parameters/desc_t_intr
64
[ryoc010-n003 ~]# cat /sys/module/hfi1/parameters/sdma_desc_cnt
2048
[ryoc010-n003 ~]# mpirun -np 2 -ppn 1 -host c010-n003,c010-n004 \
-PSM -genv I_MPI_PIN_PROCESSOR_LIST=0 ~/osu_bw

# OSU MPI Bandwidth Test v5.0
# Size      Bandwidth (MB/s)
1           1.77
2           3.54
4           7.22
8           14.74
16          26.96
32          57.22
64          114.45
128         232.93
256         405.45
512         753.79
1024        1386.73
2048        2264.14
4096        3389.89
8192        5612.19
16384       6375.83
32768       8975.61
65536       9558.37
131072     12061.21
262144     12222.53
524288     12386.10
1048576    12338.90
2097152    12354.31
4194304    12379.14
[ryoc010-n003 ~]#
  
```

Intel Omni-Path Fabric Performance (OSU benchmark)*



* Pre-production A0 hardware and Alpha-level software

www.colfax-intl.com

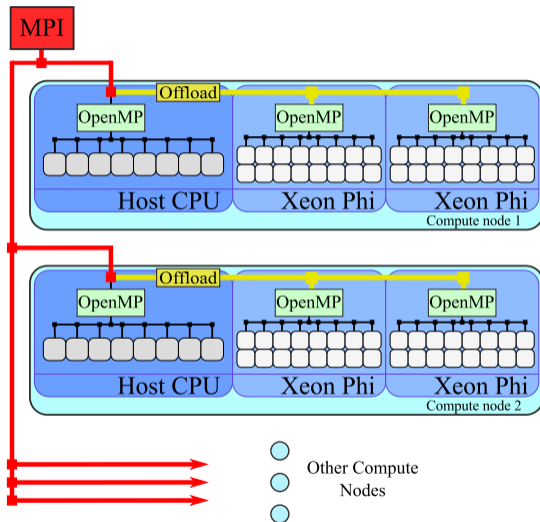


- ▶ Rely on MPI for platform-independent communication
- ▶ Intel MPI: set `I_MPI_FABRICS=tmi`.

HETEROGENEOUS DISTRIBUTED COMPUTING WITH XEON PHI

Option 1: MPI+OpenMP with Offload.

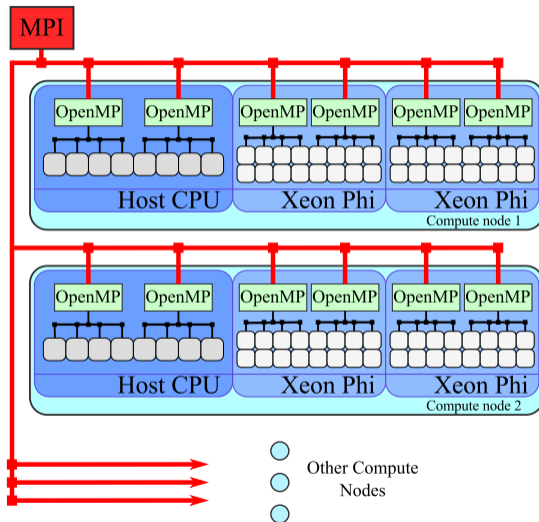
- ▶ MPI processes are multi-threaded with OpenMP.
- ▶ MPI runs only on CPUs.
- ▶ MPI processes offload to coprocessor(s).
- ▶ OpenMP in offload regions.



HETEROGENEOUS DISTRIBUTED COMPUTING WITH XEON PHI

Option 2: Symmetric hybrid MPI+OpenMP.

- ▶ MPI processes on hosts
- ▶ Native MPI processes on the coprocessor.
- ▶ Multi-threading with OpenMP.



REVIEW AND WHAT'S NEXT

- ▶ Coprocessor programming: native and offload models
- ▶ High-bandwidth memory: cache mode or flat mode
- ▶ Intel OPA: use MPI for transparent, portable programming

Next session: expressing data parallelism, vectorization.

COLFAX RESEARCH

CONTRIBUTING TO INNOVATIONS IN COMPUTING

[Log In/Out](#) or [Register](#)

READ WATCH LEARN CONNECT JOIN

To search, type and hit enter

Popular

The Hands-On Tutorials (HOT) webinars: details on efficient programming for Intel architecture

The Hands-On Workshop (HOW) Series

Introduction to Intel DAAL, Part 1: Polynomial Regression with Batch Mode Computation

Parallel Programming Book

Introduction to parallel programming, deep discussion of optimization techniques, exercises.

© 2015, Colfax International, 508 pages.

Research and Educational Publications

Introduction to Intel DAAL, Part 1: Polynomial Regression with Batch Mode Computation

Optimization Techniques for the Intel MIC Architecture, Part 3 of 3: False Sharing and Padding

Software Developer's Introduction to the HGST Ultrastar Archive H700 SMR Drives

Optimization Techniques for the Intel MIC Architecture, Part 2 of 3: Strip-Mining for Vectorization

Optimization Techniques for the Intel MIC Architecture, Part 1 of 3: Multi-Threading and Parallel Reduction

Performance to Power and Performance to Cost Ratios with Intel Xeon Phi Coprocessors (and why ix Acceleration May Be Enough)

Featured Video

See Research material on vectorization in a streaming code

https://www.youtube.com/watch?v=7p-708

Events

Registration

Consulting

[f](#) [t](#) [+](#) [s](#) [Share](#)

Colfax offers consulting services for enterprises, research help you:

- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and
- Future-proof your application for upcoming innovations
- Accelerate your application using coprocessor tech
- Investigate the potential system configurations that satisfy your cost, power, performance requirements.
- Take a clean slate to develop a novel approach to reduce your computing pro

48 Videos (Categories: CPU VSI - Chapter 1 - Episode 1.1)

Episode 2.1 — Purpose of the MIC architecture

1 - Purpose of the design

[f](#) [t](#) [+](#) [s](#) [Share](#)

[f](#) [t](#) [+](#) [s](#) [Share](#)

Software Developer's Introduction to the HGST Ultrastar Archive H700 SMR Drives

Introduction by Dr. Jay R. Robinson

[f](#) [t](#) [+](#) [s](#) [Share](#)

[f](#) [t](#) [+](#) [s](#) [Share](#)

Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors

Introduction by Dr. Jay R. Robinson

[f](#) [t](#) [+](#) [s](#) [Share](#)

[f](#) [t](#) [+](#) [s](#) [Share](#)

Configuration and Benchmarks of Peer-to-Peer Communication over Gigabit Ethernet and InfiniBand in a Cluster with Intel Xeon Phi Coprocessors

Introduction by Dr. Jay R. Robinson

[f](#) [t](#) [+](#) [s](#) [Share](#)

[f](#) [t](#) [+](#) [s](#) [Share](#)

Interview with James Reinders: future of Intel MIC architecture, parallel programming, education

Presented by Dr. Jay R. Robinson

[f](#) [t](#) [+](#) [s](#) [Share](#)

[f](#) [t](#) [+](#) [s](#) [Share](#)

Parallel Computing in the Search for New Physics at LHC

Introduction by Dr. Jay R. Robinson

[f](#) [t](#) [+](#) [s](#) [Share](#)

[f](#) [t](#) [+](#) [s](#) [Share](#)

http://colfaxresearch.com/