Programming and Optimization for Intel® Architecture

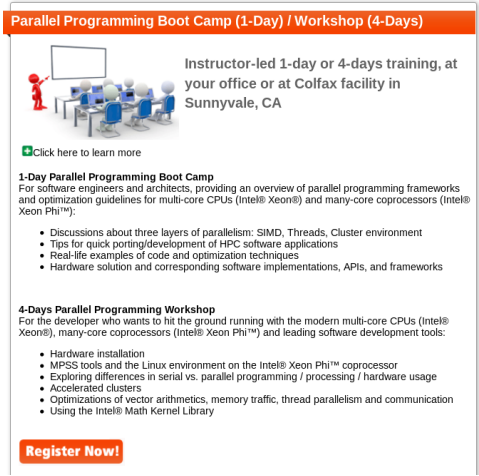
The Hands-On Workshop (HOW) Series

Colfax International — @colfaxintl

June 2016 , Rev. 02d

About This Document

This document represents the materials of a Web-based training “Programming and Optimization with Intel Architecture“ developed and run by Colfax International.



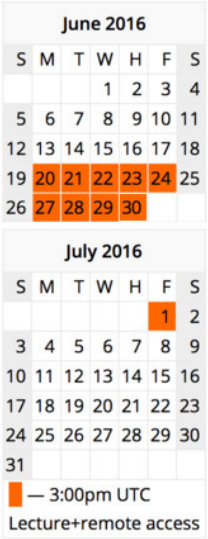
colfaxresearch.com/how-series

Disclaimer

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.

Course Roadmap

1. Why Intel Parallel Architectures?
   - Parallelism and specialization – June 20
   - Programming model continuity – June 20
2. Programming models for Xeon Phi coprocessors
   - Native programming – June 20
   - Offload programming – June 21
3. Expressing Parallelism
   - Introduction to vectorization – June 22
   - Crash-course on OpenMP – June 23
4. Optimization – intro on June 24
   - Vectorization tuning – June 27
   - Multi-threading – June 28, 29
   - Memory traffic – June 30
5. Distributed Computing: MPI – July 1



HOW Online

Course page: colfaxresearch.com/how-16-06

- Slides (including this one), code downloads
- Video of recorded sessions
- Chat (during webinars or offline)

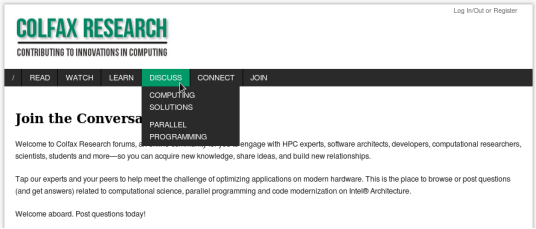
Additional resources:

- More workshops like this one: colfaxresearch.com/training
- Video courses: colfaxresearch.com/video-courses

Get Your Questions Answered

Chat (current):
colfaxresearch.com/how-16-06

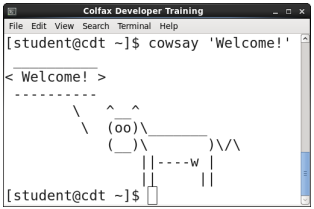
Forums (technical):
colfaxresearch.com/discussion



Email (organizational):
`training@colfax-intl.com`

Hands-On Exercises and Remote Access

- 96 people receive a remote access token
- Can access the system the entire 3 weeks of the workshop



- Not among the 96? Stay tuned: follow along with instructor, use own system, or wait for a seat
- Use it or lose it: if you do not log in for a while, remote access token goes to next student on the list

§2. Distributed Computing

Computing Hardware

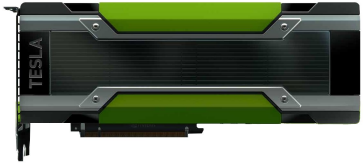
Computing Platforms

Workstations

Servers

Clusters

Computing Accelerators



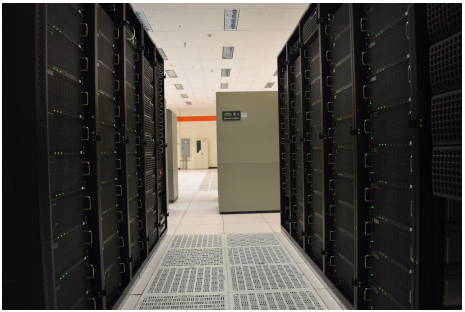
GPGPUs

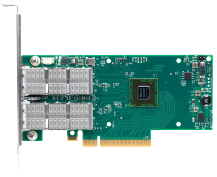
Coprocessors

FPGAs

Clusters

Clusters often use Gigabit Ethernet for administration and InfiniBand for communication.





Parallel Programming Frameworks

Data-Parallel :

intrinsics, vector classes, automatic vectorization, array notation, kernels

Shared Memory :

Pthreads, Threading Building Blocks, OpenMP

Message Passing :

TCP/IP, MPI

MPI and Heterogeneous Computing

Structure of MPI Applications: Hello World

```
#include "mpi.h"
#include <stdio.h>
int main (int argc, char *argv[]) {
  MPI_Init (&argc, &argv);  // Initialize MPI environment
  if (ret != MPI_SUCCESS) {
    MyErrorLogger("...");
    MPI_Abort(MPI_COMM_WORLD, ret);
  }
  int i, rank, size, namelen;
  char name[MPI_MAX_PROCESSOR_NAME];
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name (name, &namelen);
  printf ("Hello World from rank %d running on %s!\n", rank, name);
  if (rank == 0) printf("MPI World size = %d processes\n", size);
  MPI_Finalize (); // Terminate MPI environment
}
```

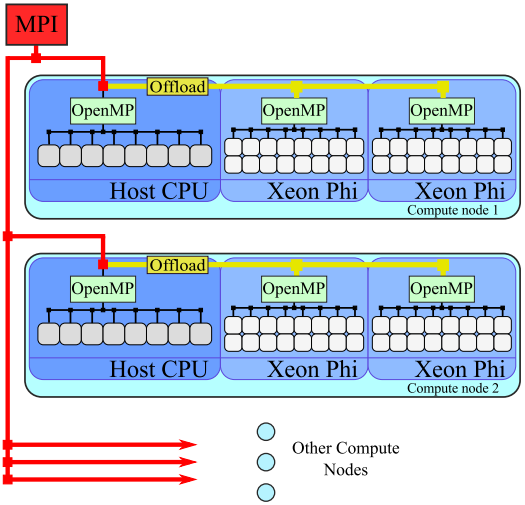

Compiling and Running MPI Applications on Host

```
vega@lyra% mpiicpc -o HelloMPI HelloMPI.cc
vega@lyra% mpirun -host localhost -np 2 ./HelloMPI
Hello World from rank 1 running on lyra!
Hello World from rank 0 running on lyra!
MPI World size = 2 processes
```

- Set up MPI environment variables
- Use wrapper script `mpiicpc` to compile
- Use automated tool `mpirun` to launch

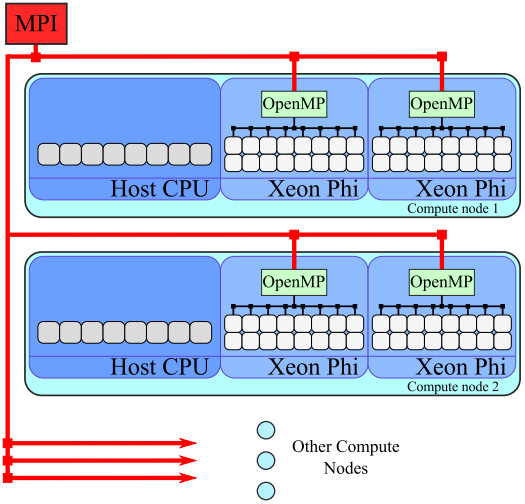
Scaling Across a Cluster with Coprocessors

- We put MPI processes only on CPUs
- Subdivide data between coprocessors
- Concurrent offload from multiple host threads
- Synchronize data between nodes with MPI



Scaling Across a Cluster with Coprocessors with MPI

- We put MPI processes only on CPUs
- Subdivide data between coprocessors
- Concurrent offload from multiple host threads
- Synchronize data between nodes with MPI



Compiling and Running Native MPI Applications on Coprocessors

```
vega@lyra% export I_MPI_MIC=1
vega@lyra% mpiicpc -mmic -o HelloMPI.MIC HelloMPI.c
vega@lyra% scp HelloMPI.MIC mic0:~/
vega@lyra% mpirun -host mic0 -np 2 ~/HelloMPI.MIC
Hello World from rank 1 running on lyra-mic0!
Hello World from rank 0 running on lyra-mic0!
MPI World size = 2 processes
```

- Enable the MIC architecture in Intel MPI: `I_MPI_MIC=1`
- Copy or NFS-share MPI library & executables with coprocessor
- Use `mpiicpc` with `-mmic` to compile
- Launch as if `mic0` is a remote host

Heterogeneous MPI Applications: Host + Coprocessors

```
vega@lyra% mpirun \
> -host mic0 -n 2 ~/Hello.MIC : \
> -host mic1 -n 2 ~/Hello.MIC : \
> -host localhost -n 2 ~/Hello
Hello World from rank 5 running on localhost!
Hello World from rank 4 running on localhost!
Hello World from rank 2 running on mic1!
Hello World from rank 3 running on mic1!
Hello World from rank 1 running on mic0!
Hello World from rank 0 running on mic0!
MPI World size = 6 ranks
```

- Specify Xeon executable for host processes
- Specify Xeon Phi executable for coprocessor processes

Heterogeneous MPI Applications: Machine File

```
vega@lyra% cat hosts.txt
localhost:2
mic0:2
mic1:2
vega@lyra% export I_MPI_MIC_POSTFIX=.MIC
vega@lyra% mpirun -machinefile hosts.txt ~/Hello
Hello World from rank 0 running on localhost!
Hello World from rank 1 running on localhost!
Hello World from rank 2 running on mic1!
Hello World from rank 3 running on mic1!
Hello World from rank 4 running on mic0!
Hello World from rank 5 running on mic0!
MPI World size = 6 ranks
```

- Specify Xeon executable for host processes
- MIC executable obtained by appending `I_MPI_MIC_POSTFIX`

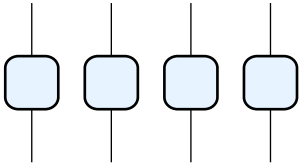
Compiling and Running MPI applications

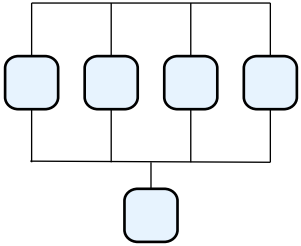
1. Compile and link with the MPI wrapper of the compiler:
   - `mpiicc` for C,
   - `mpiicpc` for C++,
   - `mpiifort` for Fortran 77 and Fortran 95.
2. Set up MPI environment variables *and* `I_MPI_MIC=1`
3. NFS-share or copy the MPI library and the application executable to the coprocessors
4. Launch with the tool `mpirun`
   - Colon-separated list of executables and hosts (argument `-host` *hostname*),
   - Alternatively, use the machine file to list hosts
   - Coprocessors have hostnames defined in `/etc/hosts`

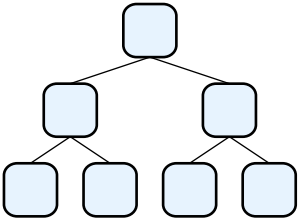
Communication Patterns

Parallel Patterns

Embarrassingly parallel, reduction, fork-join.





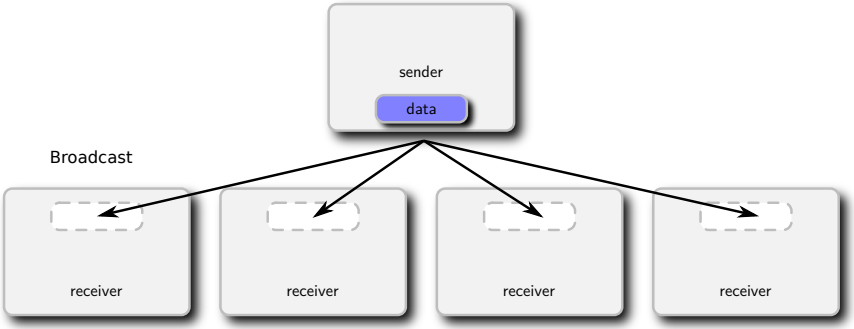


Point to Point Communication

```
1  if (rank == sender) {
2
3   char outgoingMsg[messageLength];
4   strcpy(outgoingMsg, "/Jenny");
5   MPI_Send(&outgoingMsg, messageLength, MPI_CHAR, receiver, tag, MPI_COMM_WORLD);
6
7
8  } else if (rank == receiver) {
9
10  char incomingMsg[messageLength];
11  MPI_Recv (&incomingMsg, messageLength, MPI_CHAR, sender,
12  tag, MPI_COMM_WORLD, &stat);
13  printf ("Received message with tag %d: '%s'\n", tag, incomingMsg);
14
15
16 }
```

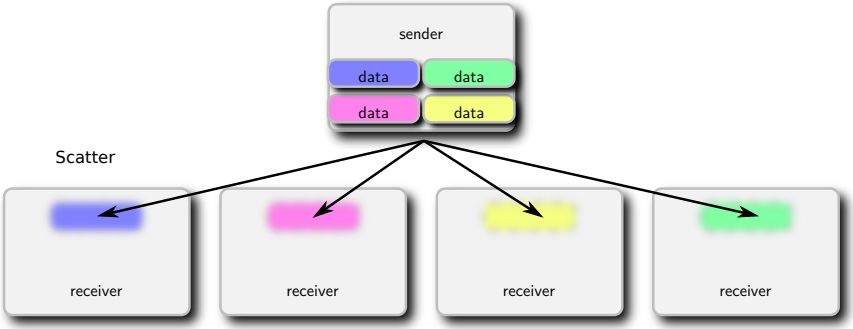

Collective Communication: Broadcast

```
1 int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype,
2     int root, MPI_Comm comm );
```



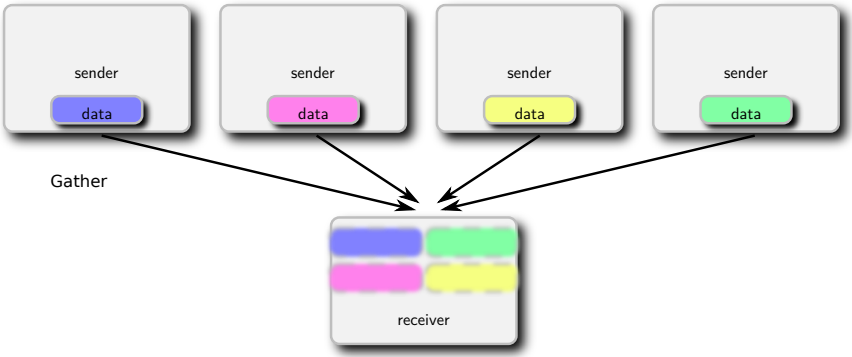
Collective Communication: Scatter

```
1 int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf,
2     int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm);
```



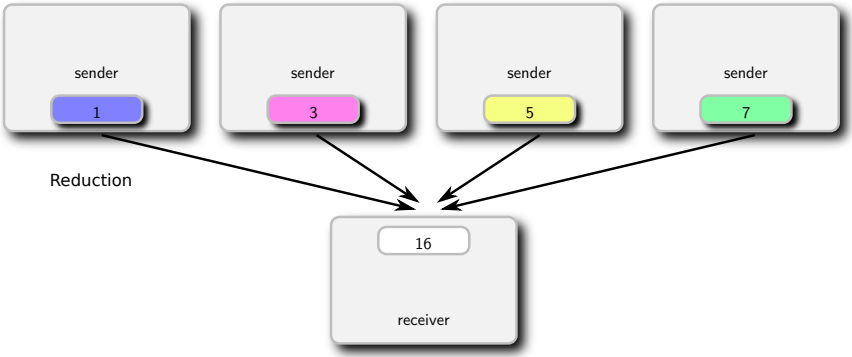
Collective Communication: Gather

```
1 int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
2     void *recvbuf, int recvcnt, MPI_Datatype recvtype,
3     int root, MPI_Comm comm);
```



Collective Communication: Reduction

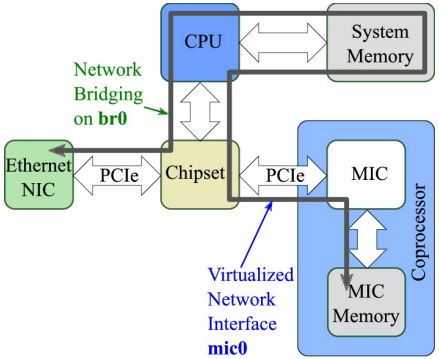
```
1 int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,
2     MPI_Op op, int root, MPI_Comm comm);
```

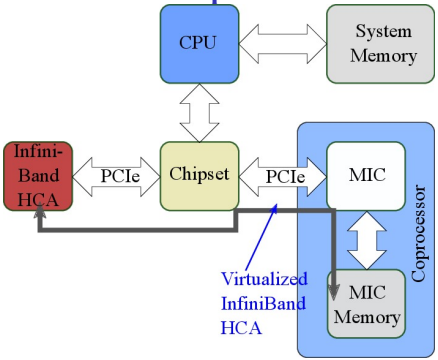


Available reducers: max/min, minloc/maxloc, sum, product, AND, OR, XOR (logical or bitwise).

Coprocessors and InfiniBand

Peer-to-Peer Communication between Coprocessors

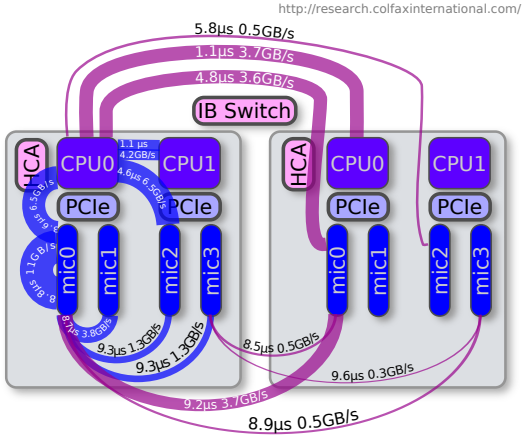




- Left: Gigabit Ethernet bridging on host allows to place coprocessors on the same subnet as hosts
- Right: Coprocessor Communication Link (CCL) – virtualization of an InfiniBand device on each coprocessor

MPI Fabric Selection

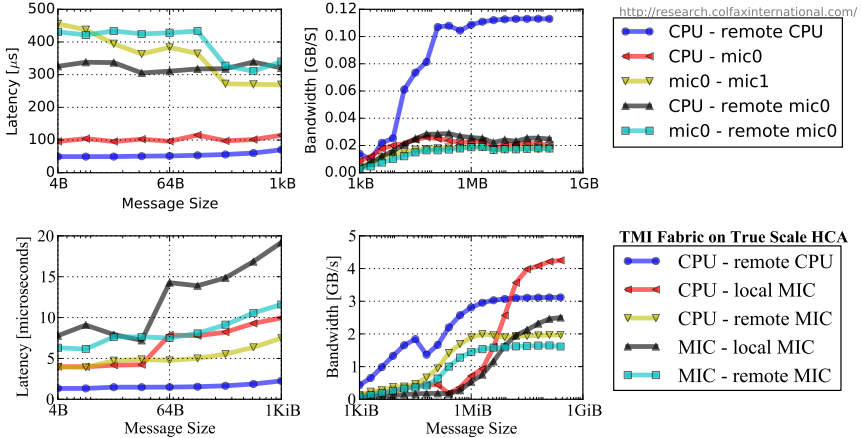
- MPI communication between CPU and coprocessors as efficient as offload
- Peer-to-peer communication not uniform, but better than with Gigabit Ethernet
- Control: environment variable `I_MPI_FABRICS`



Our publication with details:
http://xeonphi.com/papers/p2p

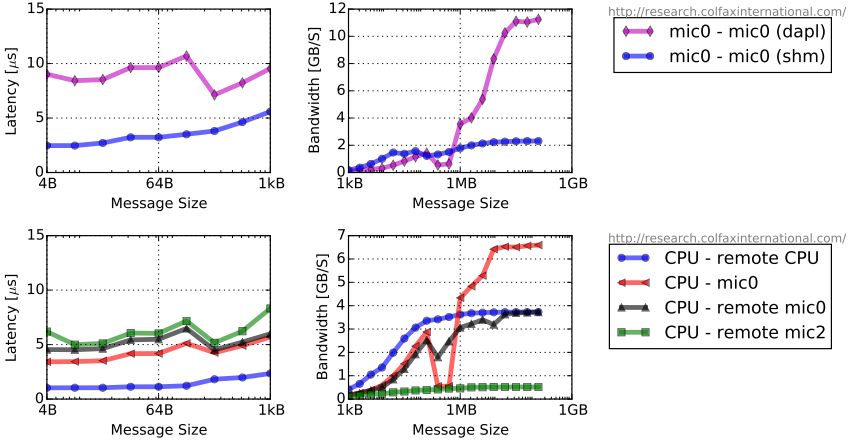
Gigabit Ethernet versus Intel True Scale Interconnects

- Ethernet+TCP between coprocessors slower than the hardware limit
- InfiniBand approaches the hardware limit from CPU to coprocessors



Intra-Device and Intra-Node Communication

- Fabrics for messages within a single coprocessor:
- `shm` provides better latency, `dapl` – greater bandwidth

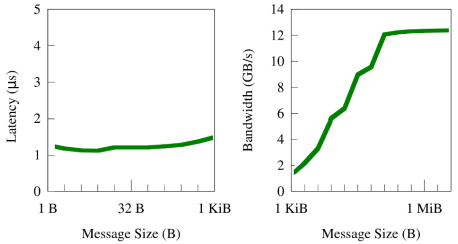


Intra-Device and Intra-Node Communication

Intel® Omni-Path Fabric

```
File Edit View Search Terminal Help
[ryo@c010-n003 ~]$ cat /sys/module/hfi1/parameters/desct_intr
64
[ryo@c010-n003 ~]$ cat /sys/module/hfi1/parameters/sdma_descq_cnt
2048
[ryo@c010-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                12306.10
1048576               12338.90
2097152               12354.31
4194304               12379.14
[ryo@c010-n003 ~]$
```

Intel Omni-Path Fabric Performance (OSU benchmark)*



* Pre-production A0 hardware and Alpha-level software

www.colfax-intl.com

COLFAX
Customized Solutions

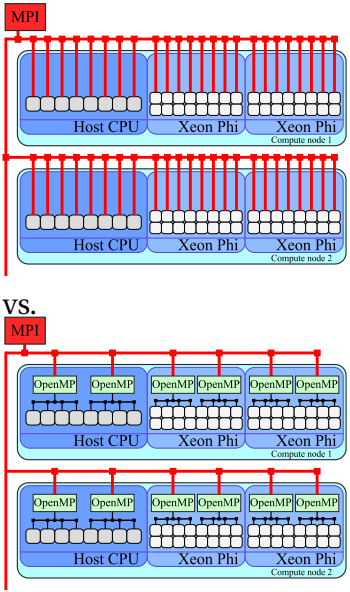
(slide from Colfax's presentation at SC'15)

Inter-Operation with OpenMP

Hybrid MPI+OpenMP

Using OpenMP inside of MPI processes:

- Reduces the memory footprint
- Decreases the number of MPI ranks, which reduces communication
- May incur thread synchronization overhead
- Optimal number of threads in MPI processes must be established empirically

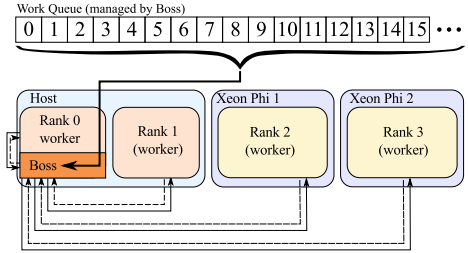


Hybrid MPI+OpenMP

- For MPI calls from multiple MPI threads, use `-mt_mpi`
- MPI pins processes to cores and sets OpenMP affinity for them.
- To tune pinning: `I_MPI_PIN`, `I_MPI_PIN_DOMAIN`
- To diagnose process pinning: `I_MPI_DEBUG=4`
- More information in the MPI Reference Manual

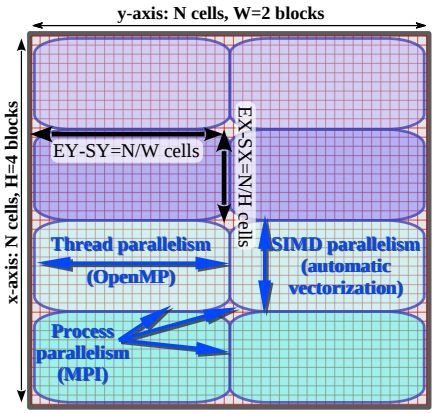
Multi-Threading within MPI Processes

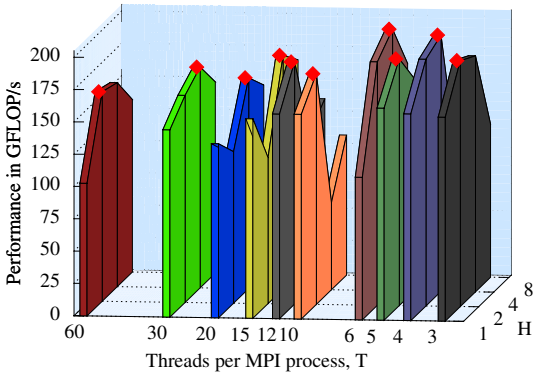
```
1 if(myRank == 0) { // Rank 0 has both a boss and a worker inside:
2     const int nThreads = omp_get_max_threads();  omp_set_nested(1);
3 #pragma omp parallel sections num_threads(2)
4     {
5 #pragma omp section
6       { DistributeWork(nOptions, option, mpiWorldSize); } // Boss
7 #pragma omp section
8       { omp_set_num_threads(nThreads-1); // Worker in rank 0:
9         ReceiveWork(option, payoff, myRank, optioncount);  } // ...
```



Example of OpenMP and MPI Inter-Operation

Number of threads per process may be a tuning parameter:





Case study: this paper

Example: Asian Options, Heterogeneous Distributed Computing

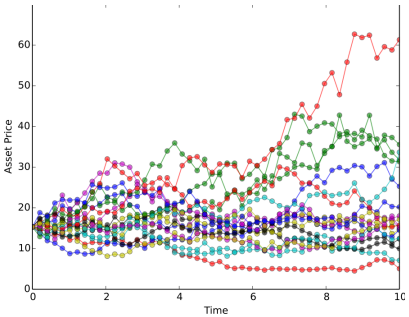
Example: the Monte Carlo Method of Asian Option Pricing

1) Simulate Random-Walk of Asset Price

$$dS(t) = \mu S(t)dt + \sigma S(t)dB(t)$$

Using the Solution

$$S(t) = S(0)e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma\sqrt{t}N(0,1)}$$



2) Perform Asian Option Price Averaging

$$\langle S \rangle_{\text{arithm}} = \frac{1}{N}\sum_{i=0}^{N-1} S(t_i),$$

$$\langle S \rangle_{\text{geom}} = \exp\left(\frac{1}{N}\sum_{i=0}^{N-1} \log S(t_i)\right)$$

3) Compute Discounted Pay-off

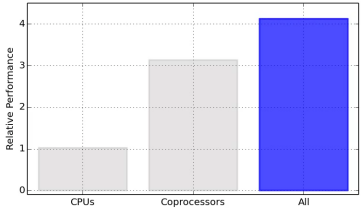
$$P_{\text{put}} = e^{-rT}\mathbb{E}\left(\max\left\{0; K - \langle S \rangle\right\}\right),$$

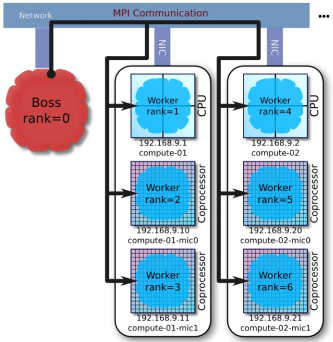
$$P_{\text{call}} = e^{-rT}\mathbb{E}\left(\max\left\{0; \langle S \rangle - K\right\}\right)$$

More information: this paper

Asian Option Pricing: Heterogeneous Clustering

Heterogeneous Clustering with Homogeneous Code:
Asian Option Pricing



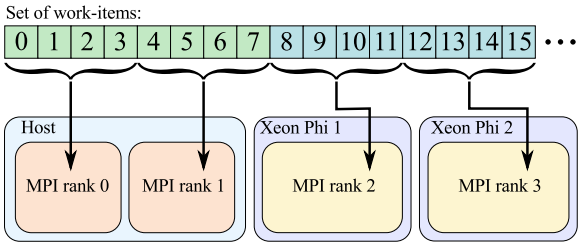


COLFAX
Customized Solutions

http://xeonphi.com/papers/heterogeneous

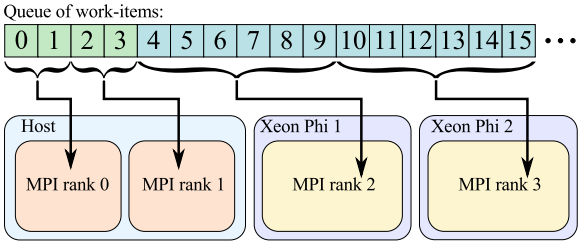
Heterogeneous Calculation without Load Balancing

```
1  const double optionsPerProcess = double(nOptions)/double(mpiWorldSize);
2  const int myFirstOption = int(optionsPerProcess*(myRank));
3  const int myLastOption  = int(optionsPerProcess*(myRank+1));
4
5  // Static, even load distribution: assign options to ranks
6  for (int i = myFirstOption; i < myLastOption; i++)
7    ComputeOptionPayoffs(option[i], payoff[i]);
```

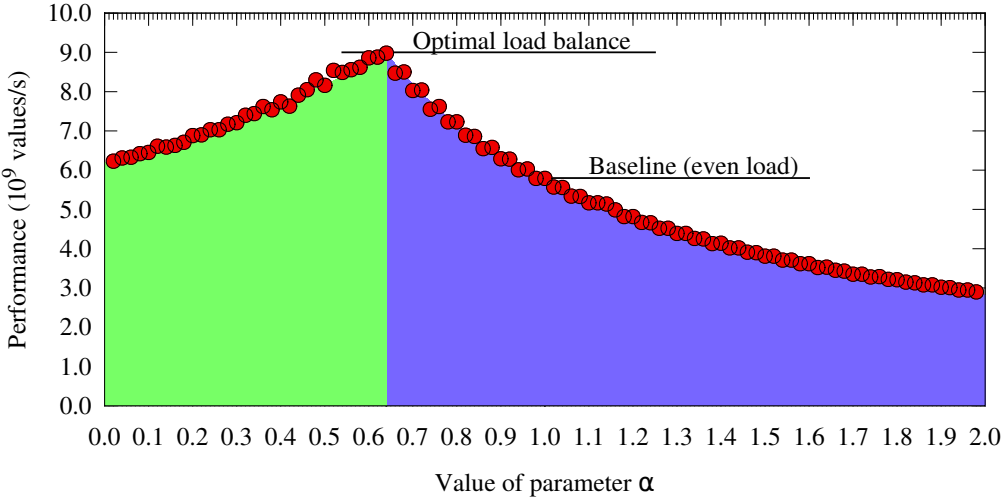


Static Load Balancing

```
1 if (rankTypes[myRank] == 0) { // I am a MIC-based rank
2   double optionsPerProc = double(lastOptForCPUs)/double(cpuRanks.size());
3   myFirstOpt = int(optionsPerProc*(myGroupRank));
4   myLastOpt  = int(optionsPerProc*(myGroupRank+1.0));
5 } else { // I am a CPU-based rank
6   double optionsPerProc = double(nOpts-lastOptForCPUs)/double(micRanks.size());
7   myFirstOpt=lastOptForCPUs+int(optionsPerProc*(myGroupRank));
8   myLastOpt=lastOptForCPUs+int(optionsPerProc*(myGroupRank+1.0)); }
```

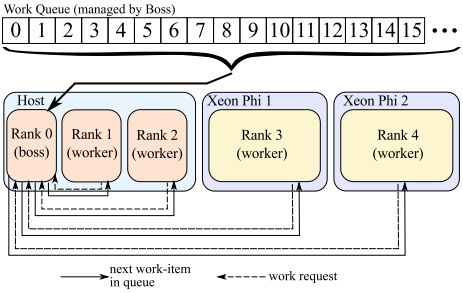


Static Load Balancing: Parameter Tuning

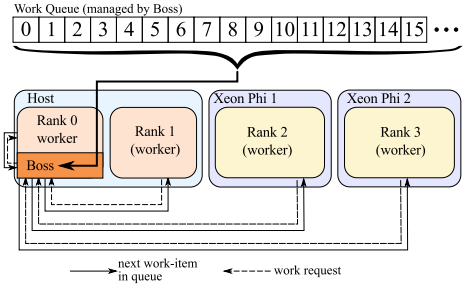


Dynamic Load Balancing

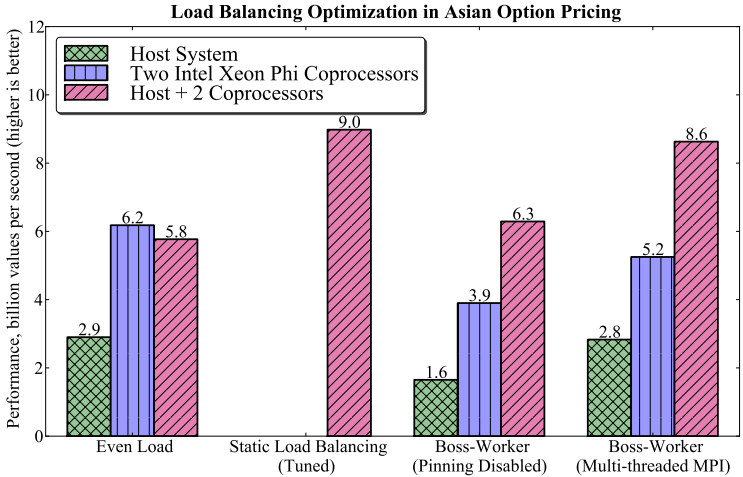
```
1  if (myRank == 0) // Boss's branch
2    DistributeWork(nOptions, option, mpiWorldSize);
3  else // Workers' branch
4    ReceiveWork(option, payoff, myRank);
```



or



Performance with Different Scheduling Modes



Refer to the book for explanation on the last two results.

§3. Summary

What We Learned

Session 1 Native programming for Xeon Phi

Session 2 Offload programming for Xeon Phi

Session 3 Expressing vectorization

Session 4 Expressing thread parallelism

Session 5 Optimization overview (N-body)

Session 6 Optimizing vectorization

Session 7 Optimizing multi-threading (common errors)

Session 8 Optimizing multi-threading (advanced tweaks)

Session 9 Optimizing memory access

Session 10 Distributed computing

Play It Again

HOW Series Archive

Your Turn

Let us know what you think via email!

Learn More

HOW “Tools” Series

Hands-On Workshop (HOW “Tools” Series): webinars on efficient programming for the Intel architecture with the help of dedicated software development tools

colfaxresearch.com/how-tools-16-06

Developer's Guide to Knights Landing

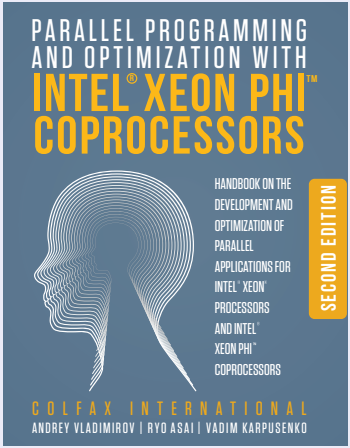
colfaxresearch.com/knl-webinar/

Textbook

ISBN: 978-0-9885234-0-1 (508 pages, Electronic or Print)

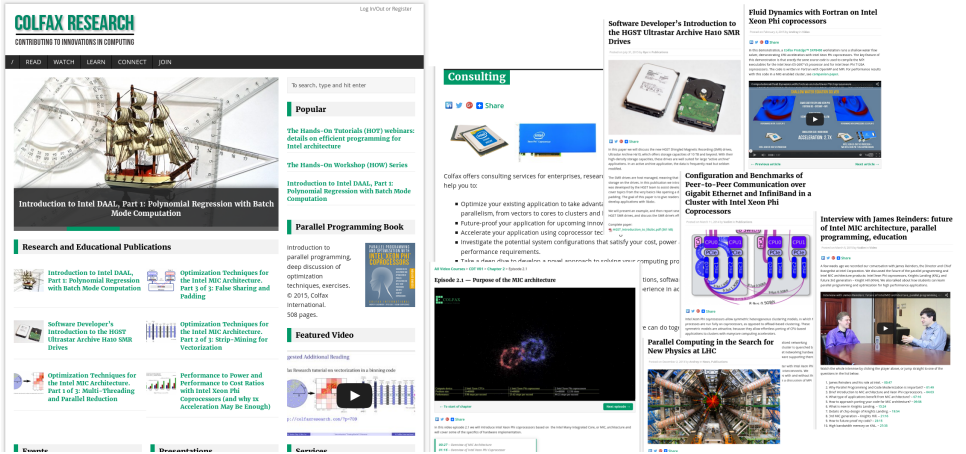
Parallel Programming
and Optimization with
Intel® Xeon Phi™
Coprocessors

Handbook on the Development and
Optimization of Parallel Applications
for Intel® Xeon® Processors
and Intel® Xeon Phi™ Coprocessors



http://xeonphi.com/book

Colfax Research



http://colfaxresearch.com/