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

Vadim Karpusenko and Andrey Vladimirov
Colfax International

March 11, 2014

Abstract

Intel Xeon Phi coprocessors allow symmetric heterogeneous clustering models, in which MPI processes are run fully on coprocessors, as opposed to offload-based clustering. These symmetric models are attractive, because they allow effortless porting of CPU-based applications to clusters with manycore computing accelerators.

However, with the default software configuration and without specialized networking hardware, peer-to-peer communication between coprocessors in a cluster is quenched by orders of magnitude compared to the capabilities of Gigabit Ethernet networking hardware. This situation is remedied by InfiniBand interconnects and the software supporting them.

In this paper we demonstrate the procedures for configuring a cluster with Intel Xeon Phi coprocessors connected with Gigabit Ethernet as well as InfiniBand interconnects. We measure and discuss the latencies and bandwidths of MPI messages with and without the advanced configuration with InfiniBand support. The paper contains a discussion of MPI application tuning in an InfiniBand-enabled cluster with Intel Xeon Phi Coprocessors, a case study on the impact of the InfiniBand protocol, and a set of recommendations for accommodating the non-uniform RDMA performance across the PCIe bus in high performance computing applications.

Table of Contents

List of Abbreviations **2**

1 Message Passing with the Intel MIC Architecture **3**

2 Peer-to-Peer Messaging between Coprocessors . **4**
- 2.1 Ethernet 4
- 2.2 InfiniBand 6

3 Configuration of a Heterogeneous Cluster **7**
- 3.1 Network Configuration 7
- 3.2 Network File Sharing 8
- 3.3 Account Management 9
- 3.4 MPSS Configuration 9
- 3.5 InfiniBand Configuration 10

4 MPI Performance Measurements **11**
- 4.1 Performance with Ethernet 12
- 4.2 Performance with InfiniBand 14
  - 4.2.1 Messages from Hosts 14
  - 4.2.2 Messages between Coprocessors . . 14
  - 4.2.3 Messages within a Coprocessor . . 15
- 4.3 Tuning the Fabrics 15
- 4.4 Tuning Collective Communication 15
- 4.5 MPI Performance in Heterogeneous Cluster: Summary 15

5 Case Study: Asian Options **16**

6 Conclusions **17**

Colfax International (http://www.colfax-intl.com/) is a leading provider of innovative and expertly engineered workstations, servers, clusters, storage, and personal supercomputing solutions. Colfax International is uniquely positioned to offer the broadest spectrum of high performance computing solutions, all of them completely customizable to meet your needs - far beyond anything you can get from any other name brand. Ready-to-go Colfax HPC solutions deliver significant price/performance advantages, and increased IT agility, that accelerates your business and research outcomes. Colfax International's extensive customer base includes Fortune 1000 companies, educational institutions, and government agencies. Founded in 1987, Colfax International is based in Sunnyvale, California and is privately held.

LIST OF ABBREVIATIONS

API Application Programming Interface

CCL Coprocessor Communication Link

GPGPUs General-Purpose Graphics Processing Units

HCA Host Channel Adapter

IMB Intel MPI Benchmark

MIC Many Integrated Core

MKL Intel Math Kernel Library

MPI Message Passing Interface

MPSS Manycore Platform Software Stack

NFS Network File Sharing

NIC Network Interconnect

OFA OpenFabrics Alliance

OFED Open Fabrics Enterprise Edition

PCIe Peripheral Component Interconnect Express

QPI Quick Path Interconnect

RDMA Remote Direct Memory Access

1. MESSAGE PASSING WITH THE INTEL MIC ARCHITECTURE

Message Passing Interface (MPI) is a standard that defines the syntax and semantics of library routines enabling parallel programming in distributed memory systems. MPI aims to provide applications with scalability, portability, and high performance. Applications using MPI do not need to explicitly manage network functionality, and therefore developers can focus on designing the parallel algorithm. At the same time, under the hood, MPI works on customized, proprietary networks, such as Mellanox InfiniBand and Intel True Scale, with extremely high throughput and low latency, absolutely transparently to the developers.

Among the multiple existing implementations of MPI, the Intel MPI Library (see also [2]) is presently the only one with support for Intel Xeon Phi coprocessors in a cluster. Intel Xeon Phi coprocessors feature the Many Integrated Core (MIC) architecture, which may yield greater performance for highly parallel applications than general-purpose Intel Xeon CPUs of comparable cost and thermal design power.

Applications using Xeon Phi coprocessors may use offload programming, which is similar to the CUDA framework for General-Purpose Graphics Processing Units (GPGPUs). With offload programming, the CPU application is equipped with directives, which send computing-intensive parts of the code and related data from the host system memory to the coprocessor. Unlike GPGPUs, Xeon Phi coprocessors can run performance-critical functions compiled from literally the same C, C++ or Fortran codes as the same functions for the CPU architecture. The offload clustering model is schematically illustrated in Figure 1.

Additionally, Xeon Phi coprocessors can operate as independent IP-addressable manycore nodes in a computing cluster, with MPI processes running on coprocessors without the involvement of the host CPUs (see Figure 2 for an illustration and [3] for a case study). The symmetric clustering model is attractive to users of existing MPI applications for CPU-based clusters, because it allows to use coprocessors and attain speedups in HPC applications without re-structuring the code to implement data offload.

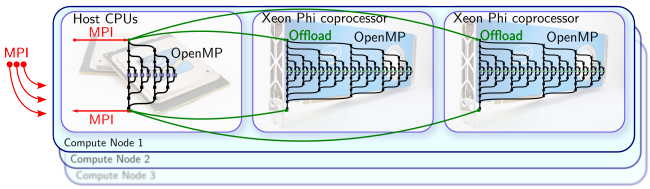

Figure 1: Traditional heterogeneous clustering with offload programming. MPI processes run only on the processors of compute nodes, utilizing Intel Xeon Phi coprocessors via an offload mechanism. Peer-to-peer communication between coprocessors is impossible, and application must utilize offload Application Programming Interface (API) to move code and data to and from the coprocessors.

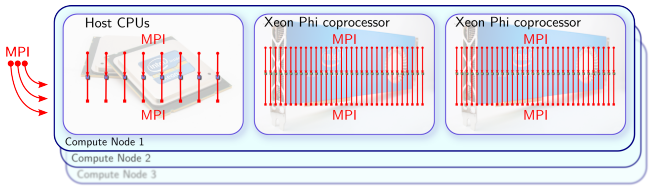

Figure 2: Symmetric heterogeneous clustering. MPI processes are launched directly on coprocessors. Peer-to-peer communication between coprocessors is possible, with network fabric virtualized in the operating system. In this mode, MPI applications for CPUs may be executed on coprocessors without code modification (however, tuning may be required).

As mentioned above, MPI can use different network configurations and fabrics for communication. In particular, this applies to peer-to-peer communication between coprocessors in a symmetric cluster. In this paper we compare MPI communication performance between coprocessors with the TCP protocol over the Ethernet fabric to the DAPL protocol over the InfiniBand fabric.

In Section 2 we discuss the software stack enabling Xeon Phi coprocessors to use networking fabrics for peer-to-peer communication. In Section 3 we demonstrate the system administration procedures for enabling networking with Ethernet and InfiniBand in a symmetric heterogeneous cluster with Xeon Phi coprocessors. Section 4 provides MPI benchmark results for our configuration, and Section 5 reports a case study on the impact of improved communication on application performance. In Conclusions (Section 6), we point out the wins and drawbacks of InfiniBand-based peer-to-peer communication in a MIC-enabled cluster, and discuss the implications for HPC workload optimization.

2. PEER-TO-PEER MESSAGING BETWEEN COPROCESSORS

2.1. ETHERNET

Intel Xeon Phi coprocessors are Peripheral Component Interconnect Express (PCIe) end-point devices. They do not have Ethernet or InfiniBand ports to connect them directly to the network. However, the Linux operating system μOS on coprocessors and the Many-core Platform Software Stack (MPSS) on hosts collaborate to virtualize networking on the coprocessor.

When MPSS is installed and configured with default parameters, it provides a "static pair" network topology. In this configuration,

i) the host's Ethernet Network Interconnect (NIC) is connected to the private network of the cluster (in our example, the cluster subnet is 10.33.0.0/15);

ii) on the host, virtual NICs `mic0`, `mic1`, etc., are created with static IP addresses and subnets of 172.31.1.254/24, 172.31.2.254/24, etc.

iii) on the host, the file `/etc/hosts` contains hostnames and IP addresses of the Xeon Phi coprocessors inside the local system: host `mic0` at 172.31.1.1, host `mic1` at 172.31.2.1, etc.

iv) in the μOS of each coprocessor, a virtual NIC `mic0` (or `mic1`, `mic2`, etc.) appears, configured to obtain a static IP address 172.31.1.1 (or 172.31.2.1, 172.31.3.1, etc), and

v) in the μOS of each coprocessor, the host file `/etc/hosts` contains the hostname of the host and and of other coprocessors: `host` at `172.31.1.254`, `mic0` at 172.31.1.1, etc.

In other words, each Xeon Phi coprocessor is placed onto a private subnet inside of the machine, and communication between the host and all coprocessors, as well as peer-to-peer messages between coprocessors, are possible. Physically, communication takes place over the PCIe bus, however, this communication is virtualized as a TCP/IP-capable NIC. Listing 1 illustrates the static pair configuration.

Another possible network configuration is flat network topology, also referred to in documentation as external bridge configuration (see Figure 3). For the flat network model, the native Linux bridging mechanism is used. In this configuration,

i) a bridge `br0` is created on the host and connected via the host's NICs to the cluster network,

ii) the host's primary network interface `eth0` is connected to `br0`

iii) all coprocessors' virtual interfaces `mic0`, `mic1`, etc., are connected to `br0`, and obtain IP addresses on the cluster private network.

This allows to send direct TCP/IP packages addressed to any coprocessor in a remote machine configured in the same way. The end result of the flat network configuration is shown in Listing 2.

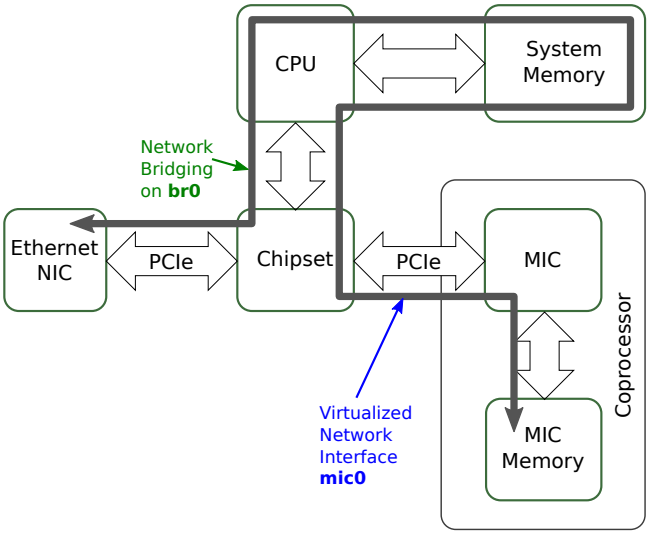

Figure 3: TCP/IP packet path with bridged networking from an Intel Xeon Phi coprocessor inside a symmetric heterogeneous cluster.

In Section 3.1 we will describe the procedure for static pair and network configuration and for flat network configuration with an external bridge. The latter configuration is necessary not only to run symmetric MPI applications with the MIC architecture over Ethernet. Flat network on the Ethernet connection of compute nodes also allows to configure InfiniBand connection for symmetric MPI.

```
[root@c001-n002 ~]# # We are on host c001-n002
[root@c001-n002 ~]# cat /etc/hosts
...
10.33.1.2 c001-n002 # Private cluster network
172.31.1.1  c001-n002-mic0 mic0 # Network...
172.31.2.1  c001-n002-mic1 mic1 # ...within...
172.31.3.1  c001-n002-mic2 mic2 # ...this...
172.31.4.1  c001-n002-mic3 mic3 # ...machine.
[root@c001-n002 ~]#
[root@c001-n002 ~]# ifconfig
...
eth0  Link encap:Ethernet  HWaddr 00:25:90:C3...
      inet addr:10.33.1.2  Bcast:10.35.255.255..
...



mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.1.254  Bcast:172.31.1.255
...
mic1  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.2.254  Bcast:172.31.2.255
...
mic2  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.3.254  Bcast:172.31.3.255
..
mic3  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.4.254  Bcast:172.31.4.255
...
[root@c001-n002 ~]# ssh mic0
[root@c001-n002-mic0 ~]# # Now we are on mic0
[root@c001-n002-mic0 ~]# cat /etc/hosts
...
172.31.1.254        host c001-n002
172.31.1.1        c001-n002-mic0 mic0
172.31.2.1        c001-n002-mic1 mic1
172.31.3.1        c001-n002-mic2 mic2
172.31.4.1        c001-n002-mic3 mic3
[root@c001-n002-mic0 ~]#
[root@c001-n002-mic0 ~]# ifconfig
...
mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:172.31.1.1  Bcast:172.31.1.255..
...
```

Listing 1: Default static pair network configuration of host `c001-n002` with four Intel Xeon Phi coprocessors. The host and all coprocessors within a machine can communicate with each other using TCP/IP, however, communication from coprocessors to remote coprocessors (on other machines) is not possible. This configuration is suitable for the offload clustering model (Figure 1) or for symmetric MPI applications running on only one compute node (Figure 2).

```
[root@c001-n002 ~]# # We are on host c001-n002
[root@c001-n002 ~]# cat /etc/hosts
...
10.33.1.2 c001-n002 # Private cluster network
10.33.1.22  c001-n002-mic0 mic0 # Now also on..
10.33.1.42  c001-n002-mic1 mic1 # ...private...
10.33.1.62  c001-n002-mic2 mic2 # ...cluster...
10.33.1.82  c001-n002-mic3 mic3 # ...network.
[root@c001-n002 ~]#
[root@c001-n002 ~]# ifconfig
...
br0   Link encap:Ethernet  HWaddr 00:25:90:C3...
      inet addr:10.33.1.2  Bcast:10.35.255.255..
...
eth0  Link encap:Ethernet  HWaddr 00:25:90:C3...
...

mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

mic1  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

mic2  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

mic3  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
...

[root@c001-n002 ~]# ssh mic0
[root@c001-n002 ~]# # Now we are on mic0
[root@c001-n002-mic0 ~]# cat /etc/hosts
...
10.33.1.2        host c001-n002
10.33.1.22        c001-n002-mic0 mic0
10.33.1.42        c001-n002-mic1 mic1
10.33.1.62        c001-n002-mic2 mic2
10.33.1.82        c001-n002-mic3 mic3
[root@c001-n002-mic0 ~]#
[root@c001-n002-mic0 ~]# ifconfig
...
mic0  Link encap:Ethernet  HWaddr 4C:79:BA:1A...
      inet addr:10.33.1.22  Bcast:10.33.1.255...
...
```

Listing 2: External bridge network configuration of host `c001-n002` with four Intel Xeon Phi coprocessors. TCP/IP packets can be sent directly between coprocessors in remote machines configured in this network topology. This configuration is appropriate for symmetric MPI applications utilizing the coprocessors in native mode (i.e., without offload) as shown in Figure 2. At the same time, cluster configured with external bridging can still be used for offload applications.

2.2. INFINIBAND

Intel MPI for Intel Xeon Phi coprocessors has built-in support for the Open Fabrics Enterprise Edition (OFED) communications stack, based on OpenFabrics Alliance (OFA) development effort. OFED allows to take advantage of Remote Direct Memory Access (RDMA) capable transport over the InfiniBand fabric, providing microsecond latencies and high bandwidth communication channels.

There are two routes for RDMA transport in clusters with Xeon Phi coprocessors:

1) the *scif0* virtual InfiniBand adapter for *intra-node* (i.e., within a compute node) communication within a single platform. This mechanism abstracts communication details over the PCIe bus between local (belonging to one node) coprocessors, and between the host and one of the local coprocessors.

2) the Coprocessor Communication Link (CCL) – a proxy driver that allows *inter-node* (i.e., between nodes) RDMA by virtualizing direct access to a hardware InfiniBand Host Channel Adapter (HCA) from Intel Xeon Phi coprocessors (see Figure 4).

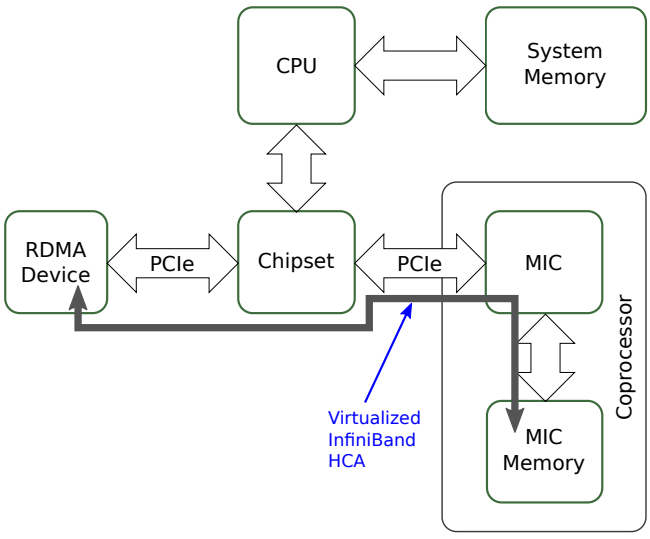

Figure 4: RDMA transport with CCL between Intel Xeon Phi coprocessors inside a symmetric heterogeneous cluster. Diagram based on [4].

The end result of OFED configuration on a system with Xeon Phi coprocessors and an InfiniBand HCA is shown in Listing 3, where a real HCA `mlx4_0` on the host, a virtualized HCA `mlx4_0` on the coprocessor, and virtual adapters `scif0` on host and on coprocessor, are shown.

```
[root@c001-n002 ~]# # We are on host c001-n002
[root@c001-n002 ~]# service openibd status

  HCA driver loaded
...
[root@c001-n002 ~]# service ofed-mic status
Status of OFED Stack:
host                                       [  OK  ]
mic0 mic1 mic2 mic3                        [  OK  ]
[root@c001-n002 ~]# ibv_devices
    device                 node GUID
    ------              ----------------
    scif0               4c79bafffe1a0ec9
    mlx4_0              f45214030012ecf0
[root@c001-n002 ~]#
[root@c001-n002 ~]# ssh mic0
[root@c001-n002 ~]# # Now we are on mic0
[root@c001-n002-mic0 ~]# ibv_devices
    device                 node GUID
    ------              ----------------
    mlx4_0              f45214030012ecf0
    scif0               4c79bafffe1a0ec8
[root@c001-n002-mic0 ~]#
```

Listing 3: InfiniBand on a system with four Intel Xeon Phi coprocessors and a Mellanox InfiniBand HCA. The service `openibd` (part of OFED) loads modules enabling the operation of the InfiniBand HCA for communication with other systems connected to the same InfiniBand network. The service `ofed-mic` (part of MPSS) creates a virtual InfiniBand interface `scif0` for communication between local coprocessors, and virtualizes an InfiniBand HCA on the coprocessor as device `mlx4_0`.

With the Ethernet and InfiniBand networks configured across all machines in the cluster, the presence of these networks is transparent to MPI applications. Intel MPI provides an abstraction for the developers to separate the code from the hardware and software implementation of communication fabrics and protocols.

In Section 3 we show how to set up and configure networking in a symmetric heterogeneous cluster with the MIC architecture, and in Section 4 we will demonstrate how to execute MPI applications on the configured cluster.

3. CONFIGURATION OF A HETEROGENEOUS CLUSTER

This sections explains how to configure a cluster with Intel Xeon Phi coprocessors and InfiniBand interconnects for symmetric heterogeneous MPI workloads. We are assuming here that all nodes come with a CentOS 6.5 Linux OS installed, and basic networking over Ethernet is configured. We explain the steps for manual configuration of MPSS, bridged networking, and OFED on compute nodes, and the reader can adapt this procedure to the provisioning system of their cluster, be it imaging, kickstart, or cluster management software.

The procedures described in this section are largely based on the methodology proposed in [5], with updates reflecting the features and specifics of the current MPSS version 3.1.2.

3.1. NETWORK CONFIGURATION

In our cluster, the hostnames of compute nodes are `c001-n001`, `c001-n002`, etc. In our naming scheme, the first part of the hostname corresponds to the number of cabinet in the cluster, and the second part — to the index number of the chassis. Our cluster has only one cabinet, but our naming system can be easily extended to multiple cabinets.

IP addresses of the Ethernet adapters of compute nodes follow the pattern `10.33.C.N`, where `C` is the cabinet number, and `N` is the node number, with a netmask `255.254.0.0`. IP addresses of the InfiniBand adapters of the corresponding nodes are `10.34.C.N` with a netmask `255.254.0.0`. Note that the Ethernet and InfiniBand IP address ranges do not overlap.

Finally, Intel Xeon Phi coprocessors have the same hostnames as their host system with the additional suffix `-micX`. For example, in the first cabinet, compute node number two has hostname `c001-n002`, and corresponding MIC cards are `c001-n002-mic0`, `c001-n002-mic1`, etc. Their IP addresses follow the pattern `10.33.C.M`, where `M=N+(X+1)*20`, and `X` is the zero-based number of the coprocessor. This scheme allows to have up to 11 Intel Xeon Phi coprocessors per chassis (today's systems support up to 8), and up to 19 compute nodes in a cabinet.

As the first step, create `/etc/hosts` file with the IP addresses of all machines and Intel Xeon Phi coprocessors in the cluster, and place it on the head node. MPSS may modify this file during the configuration process, so if you install and configure MPSS on the head node, keep a backup copy of the hosts file. *NOTE:* Intel Xeon Phi coprocessors are required to have the same name as the host system with `-micX` suffix. Otherwise, Intel MPI may not function properly.

```
[root@head-node ~]# cat /etc/hosts
127.0.0.1   localhost
::1         localhost
10.32.0.1     head-node
10.33.1.1     c001-n001
10.33.1.21    c001-n001-mic0
10.33.1.41    c001-n001-mic1
10.33.1.61    c001-n001-mic2
10.33.1.81    c001-n001-mic3
10.33.1.2     c001-n002
10.33.1.22    c001-n002-mic0
10.33.1.42    c001-n002-mic1
10.33.1.62    c001-n002-mic2
10.33.1.82    c001-n002-mic3
```

Listing 4: Master `/etc/hosts` file must be stored on the head node, from which the MPI jobs will be launched.

To implement the flat network topology we need to configure network bridging on the host system. With the bridged configuration, the host NIC will serve IP addresses assigned to all the Intel Xeon Phi coprocessors in the system. In order for that to happen, the administrator must create a bridge device `br0` and connect to it the Ethernet NIC `eth0`. This is similar to the procedure of configuring external network access for a virtual machine via a network bridge. The corresponding network configuration files for the bridge `br0` and the Ethernet interface `eth0` are located in the folder `/etc/sysconfig/network-scripts`. Listing 5 shows an example of these files for node `c001-n001`.

Configuration files presented here are not using DHCP for dynamic address distribution. Static assignment of the IP address and network mask are used instead (`BOOTPROTO=static`). IP address and network mask of the host system should be assigned to the bridge interface `br0`. To disable Network Manager interference with the configuration,

```
[root@c001-n001 ~]# \
> cat /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge
BOOTPROTO=static
ONBOOT=yes
NM_CONTROLLED=no
IPADDR=10.33.1.1
NETMASK=255.254.0.0
[root@c001-n001 ~]#
[root@c001-n001 ~]# \
> cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=Ethernet
BRIDGE=br0
BOOTPROTO=none
ONBOOT=yes
NM_CONTROLLED=no
HWADDR="00:E0:81:E4:84:EC"
```

Listing 5: Configure bridging on compute nodes.

`NM_CONTROLLED=no` should be specified for all interfaces. `ONBOOT=yes` parameter means that the interface is enabled automatically at every boot. After modifying these files, the service `network` must be restarted. Service `NetworkManager`, if present, must be disabled.

At this point, all nodes must be connected to the network with their respective IP addresses, and ping between them must succeed.

As a preparation for configuring Xeon Phi coprocessors, packet forwarding should be enabled (see Listing 6) on the host. This will allow coprocessors within one system to communicate with each other over the TCP protocol.

```
[root@c001-n001 ~]# echo 1 > \
> /proc/sys/net/ipv4/ip_forward
[root@c001-n001 ~]# cat /etc/sysctl.conf \
> | grep forward
# Controls IP packet forwarding
net.ipv4.ip_forward = 1
```

Listing 6: Configure packet forwarding on compute nodes.

We have not yet configured networking for coprocessors. This is postponed until MPSS configuration in (see Section 3.4).

3.2. Network File Sharing

Network File Sharing (NFS) allows file sharing not only with compute nodes, but also with Intel Xeon Phi coprocessors. NFS-sharing with coprocessor is useful for three purposes:

1) Sharing `/home` shares the SSH keys in `~/.ssh/`, which allows to easily configure passwordless access from the head node to compute nodes and to coprocessors in compute nodes,

2) Sharing `/home` allows users to simplify symmetric heterogeneous MPI calculation setup (when coprocessors are used in the native mode), and

3) Sharing `/opt/intel` with coprocessor simplifies the transfer of Intel libraries (specifically, Intel Math Kernel Library (MKL), MPI and OpenMP libraries) to Intel Xeon Phi coprocessors.

In order to enable file sharing with coprocessors, `/etc/exports` file must be modified on the head node (of NFS server) as shown in Listing 7. We assume that the head node is in the subnet `10.33.0.0/15` (in our cluster, we use the IP address `10.32.0.1` for the head node).

```
[root@head-node ~]# cat /etc/exports
/opt/intel 10.33.0.0/15(rw,no_root_squash)
/home 10.33.0.0/15(rw,no_root_squash)
[root@head-node ~]# exportfs -ra
[root@head-node ~]# mkdir /opt/mic
[root@head-node ~]# ln -s /opt/mic /opt/intel/mic
```

Listing 7: Configure NFS server on the head node to share `/home` and `/opt/intel`.

Note that, in order to work around a missing feature in the MPSS configuration while mounting `/opt/intel`, we must create a local directory `/opt/mic` on the head node (and on all compute nodes) and point a symbolic link `/opt/intel/mic` to `/opt/mic`. This is necessary because some configuration files specific to the compute node are installed into `/opt/intel/mic`. Without the symbolic link, different nodes will overwrite each other's configuration, while pointing the link outside of the shared directory `/opt/intel` allows each node to have a unique configuration.

Then, in order to mount `/opt/intel` and `/home` from the server, the following lines must be appended to `/etc/fstab` on compute nodes:

```
[root@c001-n002 ~]# tail -f /etc/fstab
head-node:/home      /home      nfs defaults 1 2
head-node:/opt/intel /opt/intel nfs defaults 1 2
```

Listing 8: Configure NFS client to mount `/home` and `/opt/intel`.

An alternative configuration is to install Intel Cluster Studio on compute nodes, rather than NFS-mount it from the head node. With locally installed Intel Cluster Studio, the role of the NFS server that provides `/opt/intel` to coprocessors is played by the local compute node. That is, each compute node must be configured as an NFS-server, and `/etc/exports` on compute nodes must allow sharing `/opt/intel` with `10.33.C.N/24` (see Section 3.1 for an explanation of this IP address pattern). In this case, it is not necessary to create a symbolic link pointing from `/opt/intel/mic` to `/opt/mic`.

Further details of standard NFS server and client configuration and security are beyond the scope of this paper.

3.3. Account Management

Intel Xeon Phi coprocessors inherit user privilege policies from the host system. Therefore, to run the application under a specific user account, we need to create one and set up the corresponding SSH keys (see Listing 9).

Assuming that `/home` is NFS-shared from the `head-node` to all compute nodes, we create the SSH key for each user only once, on the head node. We copy the public SSH key to the `authorized_keys` file in order to enable passwordless authentication across all nodes and coprocessors NFS-sharing `/home`.

Note that non-empty passwords must be set for all users that will access Intel Xeon Phi coprocessors, even though access to coprocessors uses SSH key authentication and skips password prompt. This is required for authentication on coprocessors.

Also, care must be taken to ensure that numerical user IDs and group IDs for each user are identical on the head node and on compute nodes (use `--uid` and `--gid` arguments of `adduser` if necessary). This is required for correct NFS functioning.

```
[root@c001-n001 ~]# adduser cfxuser
[root@c001-n001 ~]# passwd cfxuser
...
[root@c001-n002 ~]# adduser cfxuser
[root@c001-n002 ~]# passwd cfxuser
...
[root@head-node ~]# adduser cfxuser
[root@head-node ~]# passwd cfxuser
[root@head-node ~]# su cfxuser
[cfxuser@head-node ~]# ssh-keygen
...
[cfxuser@head-node ~]# cat \
> .ssh/id_rsa.pub >> .ssh/authorized_keys
[cfxuser@head-node ~]# chmod \
> 600 ~/.ssh/authorized_keys
```

Listing 9: Configure users on compute nodes and their keys on the head node.

3.4. MPSS Configuration

Next step is to install and configure MPSS for Intel Xeon Phi coprocessors. For MPSS version 3.1.2, download the archive from the Intel Registration Center (or from the archive for older versions), unpack it and install the RPMs using `yum`, as shown in Listing 10.

```
[root@c001-n001 ~]# tar -xf \
>                  mpss-3.1.2-rhel-6.5.tar
[root@c001-n001 ~]# cd mpss-3.1.2/
[root@c001-n001 ~]# yum install *.rpm
```

Listing 10: Installing MPSS

In our system configuration, we disable Linux distribution updates on compute nodes after installation. This ensures that the installed kernel version is the same as the kernel for which MPSS was compiled. If installed kernel does not match the version for which MPSS was compiled, MPSS kernel modules and OFED modules will not work, and must be recompiled. The procedure for recompilation is described in Section 8 of the MPSS User Guide [6].

In order to configure MPSS, we will use the `micctrl` tool. `micctrl` allows to manage the con-

figuration of the μOS of Intel Xeon Phi coprocessors through command line arguments. Listing 11 shows the initialization of the configuration files for Intel Xeon Phi coprocessors, configuring bridged connection and setting up the network configuration, and adding NFS folders previously configured on the host system.

```
[root@c001-n001]# micctrl --initdefaults
...
[root@c001-n001]# micctrl --addbridge=br0 \
> --type=external --ip=10.33.1.1 --netbits=15
[root@c001-n001]# micctrl --network=static \
> --bridge=br0 \
> --ip=10.33.1.21:10.33.1.41:10.33.1.61:\
>10.33.1.81

[root@c001-n001]# micctrl --addnfs=/opt/intel \
>                 --dir=/opt/intel
[root@c001-n001]# micctrl --addnfs=/home \
>                 --dir=/home
```

Listing 11: Configuring MPSS with bridging and NFS

As of MPSS 3.1.2, the NFS server for coprocessors set by `micctrl --addnfs` is always the hosting compute node, and it is not possible to specify a different NFS server in the command line. Therefore, if `/home` and/or `/opt/intel` must be mounted from the head node, rather than from the hosting compute node, the administrator must manually modify the files `/var/mpss/mic*/etc/fstab` and change the location of the NFS server to the correct IP address. For example, Listing 8 shows how `/home` can be mounted from the head node (10.32.0.1), and `/opt/intel` – from the local compute node.

```
[root@c001-n001]# cat /var/mpss/mic0/etc/fstab \
        | tail -2
10.32.0.1:/home      /home      nfs nolock 1 1
10.33.1.1:/opt/intel /opt/intel nfs nolock 1 1
```

Listing 12: Modified `/etc/fstab` for coprocessors mounts `/home` from the head node instead of the local compute node. As of MPSS 3.1.2, this modification must be performed manually.

Now everything is ready for starting MPSS as shown in Listing 13. With MPSS started on all devices, it must be possible to ping all hosts and all coprocessors from any host or coprocessor in the cluster.

```
[root@head-node]# ssh c001-n001
[root@c001-n001]# service mpss start
Starting Intel(R) MPSS:                [  OK  ]
mic0: online (mode: linux image: /usr/share/...
mic1: online (mode: linux image: /usr/share/...
mic2: online (mode: linux image: /usr/share/...
mic3: online (mode: linux image: /usr/share/...
[root@c001-n001]# exit
[root@head-node]# ping -q -c 1 c001-n001
PING c001-n001 (10.33.1.1) 56(84) bytes of data.

--- c001-n001 ping statistics ---
1 packets transmitted, 1 received ...
rtt min/avg/max/mdev = 0.330/0.330/0.330/0.000ms
[root@head-node]#
[root@head-node]# ping -c 1 c001-n001-mic0
PING c001-n001-mic0 (10.33.1.21) 56(84) bytes...

--- c001-n001-mic0 ping statistics ---
1 packets transmitted, 1 received ...
rtt min/avg/max/mdev = 0.714/0.714/0.714/0.000ms
[root@head-node]#
[root@head-node]# ssh c001-n002
[root@c001-n002]# service mpss start
...
```

Listing 13: Starting MPSS on compute nodes and verifying bridged networking for coprocessors.

3.5. INFINIBAND CONFIGURATION

To install and configure InfiniBand support for Intel MPI for the Intel MIC architecture, OFED must be installed on all compute nodes. With MPSS 3.1.2, CentOS 6.5 Linux and a Mellanox HCA (or without an HCA, for improved intra-node communication via the `scif0` virtual network), download and install OFED version 1.5.4.1 from the OFA Web site. To install the package, run the Perl installation script `install.pl`. During the installation, answer "No" to installing libraries `dapl*`, `compat-dapl*` and 32-bit library support. After installing OFED, install additional MPSS packages from subdirectory `mpss-3.1.2/ofed`, as shown in Listing 14. Note that if the Linux kernel is different from the kernel for which the MPSS OFED packages were compiled, the OFED modules will need to be recompiled prior to installation. Refer to Section 2.3 of the MPSS User Guide [6] for more information.

Although not specifically called for by the documentation, we have found that a system reboot may be helpful at this point in the configuration process.

```
[root@c001-n001 ~]# cd OFED-1.5.4.1
[root@c001-n001 OFED-1.5.4.1]# ./install.pl
...
[root@c001-n001 OFED-1.5.4.1]# cd ../mpss-3.1.2
[root@c001-n001 mpss-3.1.2]# yum install\
> --disablerepo=* --skip-broken ofed/*.rpm
...
```

Listing 14: Installing OFED 1.5.4.1 from source and MPSS OFED packages as RPMs.

After installing OFED, follow Listing 15 to:

1) configure a static IP address for the InfiniBand connection,

2) start service `openibd` on all compute nodes,

3) start service `opensmd` on exactly one of the compute nodes (unless the network switch already provides a subnet manager, in which case do not start this service on any nodes),

4) start service `ofed-mic` in order to activate CCL, which creates the virtual intra-node interface `scif0` and inter-node interfaces `mlx4_0` on the host and on coprocessors, and

5) verify the InfiniBand device status by running `ibstatus`. The intra-node and inter-node interfaces must be present, and their state must be listed as "ACTIVE".

In order to ensure that InfiniBand support for Xeon Phi coprocessors has been started, log in to one of the coprocessors and execute `ibv_devinfo` (Listing 16. Make sure that the intra-node interface `scif0` and the inter-node interface `mlx4_0` are listed in the `PORT_ACTIVE` state.

4. MPI Performance Measurements

The cluster on which the performance measurements were taken consists of two Colfax ProEdge™ SXP8600p workstations, each with four Intel Xeon Phi 31S1P coprocessors. Two interconnects were used for networking within the cluster:

```
[root@c001-n001 ~]# cat \
> /etc/sysconfig/network-scripts/ifcfg-ib0
DEVICE=ib0
HWADDR=80:00:00:48:FE:80:00:00:00:00:00:00...
TYPE=InfiniBand
UUID=43a82878-51ee-427d-bec2-312d6b868508
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=10.34.1.1
NETMASK=255.254.0.0
[root@c001-n001 ~]# service network restart
[root@c001-n001 ~]# service openibd start
[root@c001-n001 ~]# service ofed-mic start
[root@c001-n001 ~]# ibstatus
Infiniband device 'mlx4_0' port 1 status:
        default gid:  fe80:0000:0000:...
        base lid:     0x1
        sm lid:               0x1
        state:               4: ACTIVE
        phys state:   5: LinkUp
        rate:              40 Gb/sec (4X QDR)
        link_layer:    InfiniBand

Infiniband device 'scif0' port 1 status:
        default gid:  fe80:0000:0000:...
        base lid:     0x3e8
        sm lid:               0x1
        state:        4: ACTIVE
        phys state:   5: LinkUp
        rate:         40 Gb/sec (4X QDR)
        link_layer:   Ethernet

[root@c001-n001 ~]#
```

Listing 15: Start/check the InfiniBand services.

1. Intel Gigabit Ethernet adapters (model I350) installed in the systems and connected to a D-Link Gigabit Ethernet switch (model DGS-1024D), and

2. Mellanox InfiniBand ConnectX-3 Single-Port VPI 4X QDR adapters (model version MCX353A-FCAT) connected to a 36-port Mellanox Infiniscale IV switch (model IS5025).

The systems were running the CentOS 6.5 Linux operating system with kernel 2.6.32-431.el6.x86_64, MPSS 3.1.2, Intel MPI 4.1.1.036, and OFED 1.5.4.1. We used the Intel MPI Benchmark (IMB) shipped with Intel MPI for performance measurements.

```
[root@c001-n001 ~]# ssh mic0
[root@c001-n001-mic0 ~]# ibv_devinfo
hca_id:        mlx4_0
  transport:       InfiniBand (0)
  fw_ver:          2.30.8000
  node_guid:       f452:1403:0012:ed70
  sys_image_guid: f452:1403:0012:ed73
  vendor_id:       0x02c9
  vendor_part_id: 4099
  hw_ver:          0x0
  phys_port_cnt:  1
    port:  1
      state:       PORT_ACTIVE (4)
      max_mtu:     2048 (4)
      active_mtu: 2048 (4)
      sm_lid:      1
      port_lid:    1
      port_lmc:    0x00
      link_layer: IB

hca_id:  scif0
  transport:       SCIF (2)
  fw_ver:          0.0.1
  node_guid:       460f:d8ff:fe1d:8294
  sys_image_guid: 460f:d8ff:fe1d:8294
  vendor_id:       0x8086
  vendor_part_id: 0
  hw_ver:          0x1
  phys_port_cnt:  1
    port:  1
      state:       PORT_ACTIVE (4)
      max_mtu:     4096 (5)
      active_mtu: 4096 (5)
      sm_lid:      1
      port_lid:    1001
      port_lmc:    0x00
      link_layer: SCIF
```

Listing 16: CCL at work: intra-node interface `scif0` and the inter-node interface `mlx4_0` are virtualized on an Intel Xeon Phi coprocessor.

4.1. Performance with Ethernet

First, we would like to study the scenario in which the InfiniBand interconnects are not installed, and MPI communication takes place over Ethernet. We can emulate this scenario by setting the environment variable `I_MPI_FABRICS=tcp`, which forces Intel MPI to use the TCP/IP protocol over Ethernet for communication.

Listing 17 demonstrates the procedure for launching the benchmark. The output of the benchmark application was used to report the latencies and bandwidth shown in the plots.

Figure 5 demonstrates the performance of the PingPong benchmark with the TCP protocol. For intra-node

```
[cfxuser@head-node ~]# export I_MPI_MIC=1
[cfxuser@head-node ~]# export I_MPI_FABRICS=tcp
[cfxuser@head-node ~]# export IMBHOST=\
>${I_MPI_ROOT}/bin64/IMB-MPI1
[cfxuser@head-node ~]# export IMBMIC=\
>${I_MPI_ROOT}/mic/bin/IMB-MPI1
[cfxuser@head-node ~]# mpirun \
> -np 1 -host c001-n001 ${IMBHOST} PingPong :\
> -np 1 -host c001-n002-mic0 ${IMBMIC}
...
```

Listing 17: Execution of the Intel MPI benchmark with the Ethernet fabric. In this example, one of the PingPong endpoints is the CPU on node 1, and the other is the coprocessor 0 on node 2.

communication, MPI uses virtualized Ethernet NICs `mic0`, `mic1`, etc., for data transport. For inter-node communication (from host to remote hosts and remote coprocessors), the bridged connection of `eth0` with `mic0`, `mic1`, etc., is used.

The latency of short messages for all communication going through the bridge to coprocessors is between 300 and 500 μs. For direct connections (between the CPU and local coprocessors, and between two CPUs), the latency is 50 to 100 μs. The bandwidth of large messages approaches about 110 MB/s for host-to-host connection, which is consistent with the hardware limitation of the Gigabit Ethernet network. However, for all communication involving coprocessors, the bandwidth plateaus at 20-25 MB/s. This is orders of magnitude below the hardware limitation of both the network interconnects, and the PCIe bus. As we have reasoned in the previous work [3], the convenience of symmetric heterogeneous clustering over Ethernet comes at the cost of reduced communication efficiency.

However, in the next section we will see that this is not the case for the InfiniBand fabric.

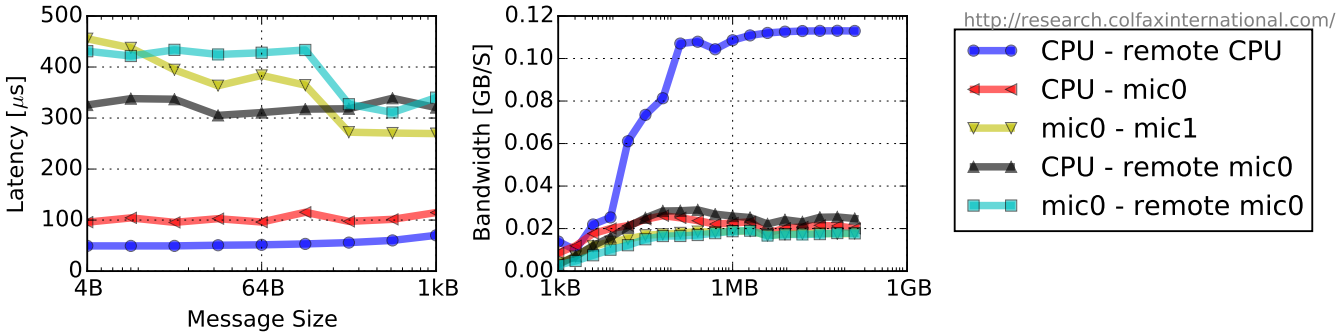

Figure 5: Intel MPI Benchmark, PingPong test over the Ethernet fabric (`I_MPI_FABRICS=tcp`). CPU indicates an MPI process run on the host, and `mic*` are MPI processes executed on the respective coprocessor. "Remote" means located on a different chassis.

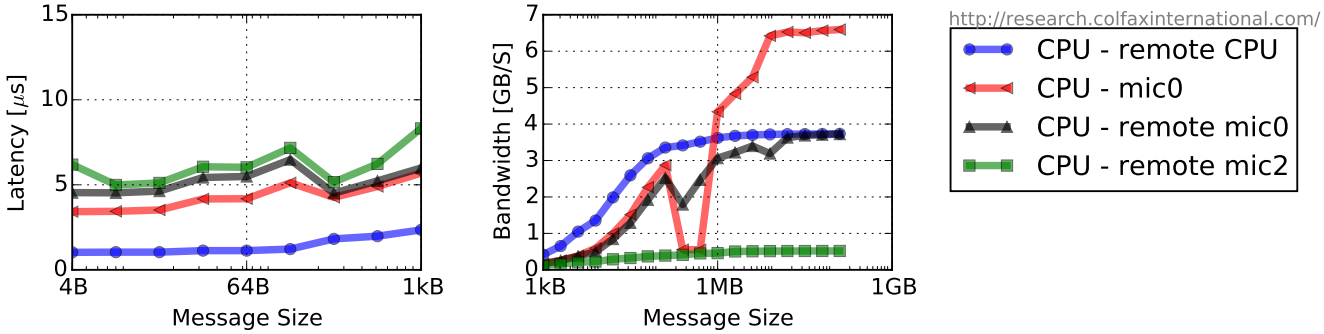

Figure 6: Intel MPI Benchmark, PingPong test over the InfiniBand fabric (`I_MPI_FABRICS=dapl`) for communication originating in a CPU host. See also caption for Figure 5.

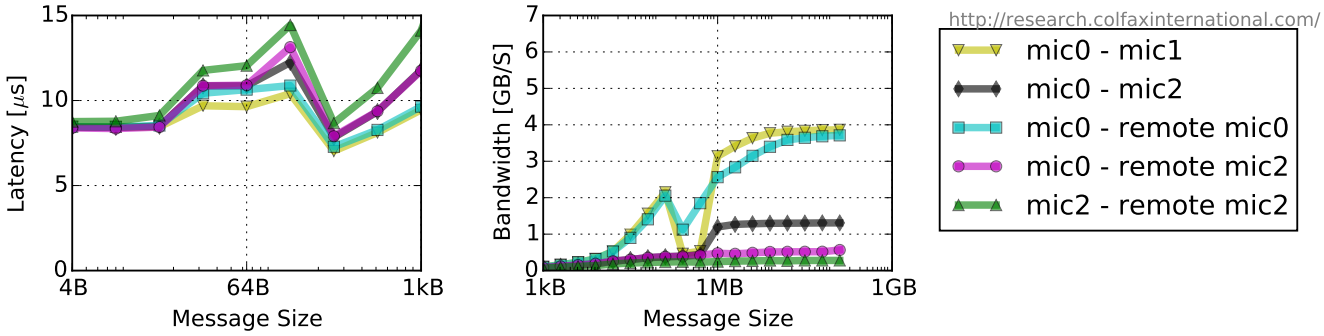

Figure 7: Intel MPI Benchmark, PingPong test over the InfiniBand fabric (`I_MPI_FABRICS=dapl`) for communication originating in a MIC host. See also caption for Figure 5.

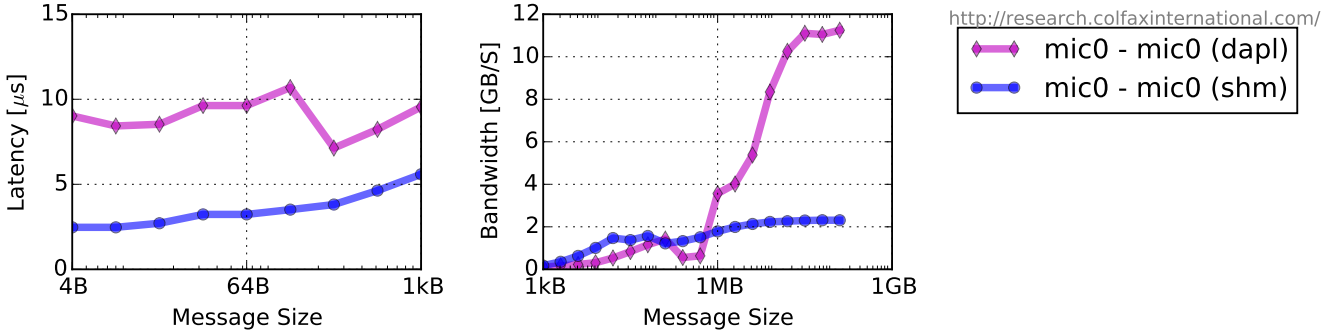

Figure 8: Intel MPI Benchmark, PingPong test over the shared-memory and InfiniBand fabrics (`I_MPI_FABRICS=shm` and `I_MPI_FABRICS=shm:dapl`) for communication within a MIC host. See also caption for Figure 5.

4.2. Performance with InfiniBand

The next step in benchmarking MPI communication over the InfiniBand fabric. Intel MPI can detect and use this fabric automatically, and no additional run configuration needs to be done. However, if automated configuration fails, or if the user wants to prevent other fabrics from being used, the environment variable `I_MPI_FABRICS=dapl` must be set in order to request communication using the DAPL protocol.

We discuss the performance results for three groups of communication participants: messages originating from hosts, messages between Xeon Phi coprocessors, and messages within a single Xeon Phi coprocessor.

4.2.1. Messages from Hosts

In Figure 6, latency and bandwidth of the PingPong test from the host CPU to a remote CPU, a local coprocessor, and a remote coprocessor are shown. Immediately apparent is the fact that in this case, both metrics are orders of magnitude better than with Ethernet and TCP protocol.

The communication latency for short messages is down to 1.5–8 μs. This is two orders of magnitude better than with the TCP protocol over Ethernet, and one order of magnitude better than the time of transferring control to the coprocessor in the offload programming model (see [7] for offload performance measurement).

The bandwidth for large messages from CPU to the local Xeon Phi reaches 6.5 GB/s, which is near the limits of the PCIe bus bandwidth for a PCIe v2.0 x16 device. Bandwidth from host to remote host and remote coprocessor reaches 3.8 GB/s, which is consistent with the limit of the 4X QDR InfiniBand interconnect.

Communication bandwidth and latency from host to local coprocessors `mic1`, `mic2` and `mic3`, as well as to remote `mic0` and `mic1` all are very close to the metrics from host to `mic0`, and therefore not shown.

However, messages from the host to remote `mic2` and `mic3` achieve only 0.5 GB/s. As we will see in the next section, this is caused by the fact that the InfiniBand HCA is installed on CPU1 on all hosts, and coprocessors `mic2` and `mic3` are on CPU2.

4.2.2. Messages between Coprocessors

In Figure 7, the latency and bandwidth of MPI messages from `mic0` to other local and remote coprocessors are plotted. While the latency is comparable in all these cases, and is of order 10 μs, the bandwidth depends on the pair of communicating devices.

In the case of intra-node communication:

i) Going from `mic0` to local `mic1`, the bandwidth reaches almost 4 GB/s. The same bandwidth is achieved between `mic2` and local `mic3` (not shown in the figure).

ii) However, going from `mic0` to local `mic2` (or `mic3`, not shown), the bandwidth drops down to 1.3 GB/s. This is due to the location of the coprocessors on the PCIe bus. While `mic0` and `mic1` are connected to PCIe slots controlled by CPU1, `mic2` and `mic3` are connected to PCIe slots controlled by CPU2. The PCIe transport between endpoint devices is slower across the Quick Path Interconnect (QPI) connecting the CPUs than within the domain controlled by the same CPU.

Likewise, in the case of inter-node communication, the fact that the InfiniBand HCA is connected to the PCIe slot on CPU1 determines the communication efficiency pattern:

i) Communication between `mic0` and remote (i.e., on another compute node) `mic1` or `mic0` (not shown), the bandwidth approaches the InfiniBand connection limit of 4 GB/s.

ii) However, from `mic0` to remote `mic2`, the bandwidth drops to 0.4 GB/s, because the data must be sent from the remote HCA (on CPU1) to `mic2` on the same machine (which is on CPU2).

iii) Even worse is the traffic from `mic2` to remote `mic2` (or `mic3`, not shown), which is around 0.25 GB/s. In this case, the data are sent first across the local PCIe bus to CPU1, then across the InfiniBand network, and then from remote HCA to CPU2 on that machine.

4.2.3. Messages within a Coprocessor

A special case is communication between multiple MPI processes within a single coprocessor. By default, MPI uses a mixed communication protocol, which is equivalent to setting `I_MPI_FABRICS=shm:dapl`. With this setting, communication between MPI ranks within the same compute device (CPU or coprocessor) takes place using shared memory copy, and communication outside of the device uses DAPL.

However, in some cases, it may be beneficial to use just shared-memory copy or only DAPL, as evident from Figure 8. Indeed, the latency of short messages is better in the case of the `shm` protocol (2 μs versus 10μs for `dapl`), and the bandwidth of large messages is greater with the `dapl` protocol (up to 11 GB/s versus 2 GB/s for `shm`). Measurements for Figure 8 were obtained by running IMB with `I_MPI_FABRICS=shm` and with `I_MPI_FABRICS=dapl`.

4.3. Tuning the Fabrics

The DAPL protocol is supported by multiple libraries in the OFED stack, and these libraries are tuned to different message sizes and fabrics. Intel MPI automatically decides which providers of DAPL to use depending on the detected system configuration at runtime. However, that choice can be overridden.

In our system, MPI chose three DAPL providers for communication: `ofa-v2-mlx4_0-1` for short messages, `ofa-v2-scif0` for long messages within a compute node, and `ofa-v2-mcm-1` for long messages outside a compute node. This choice can be diagnosed by setting the environment variable `I_MPI_DEBUG=2` (or a higher value) for IMB. In order to change this selection, the environment variable `I_MPI_DAPL_PROVIDER_LIST` can be used. This variable takes a comma-separated list of up to 3 providers: one for short messages, another for intra-node long messages, and the third one for inter-node long messages. See a blog post by James Tullos for more details. The complete list of supported DAPL providers can be found in the file `/etc/dat.conf`. The threshold between "short" and "long" messages is controlled by the environment variable `I_MPI_DAPL_DIRECT_COPY_THRESHOLD`.

We have verified empirically that the choice made by the Intel MPI library is indeed optimal.

4.4. Tuning Collective Communication

Parallel transfers (e.g., `MPI_SendRecv`) and collective MPI communication functions (e.g., `MPI_Allgather`) use configurable algorithms for moving data across the cluster. For the best performance with a specific machine file, utility `mpitune` can find the optimal algorithms. This utility is a part of Intel MPI. The list of parameters tuned by `mpitune` can be found in the file `$I_MPI_ROOT/etc64/options.xml`, and the instructions for using `mpitune` are available in [2].

4.5. MPI Performance in Heterogeneous Cluster: Summary

The summary of our measurements is depicted in Figure 9. In this image, the devices are shown with rounded-corner rectangles, and MPI communication efficiency is drawn with connecting lines. The thickness of the lines is proportional to the bandwidth of the largest messages. Also, the bandwidth, along with the latency for the shortest messages, is indicated in the label of each line.

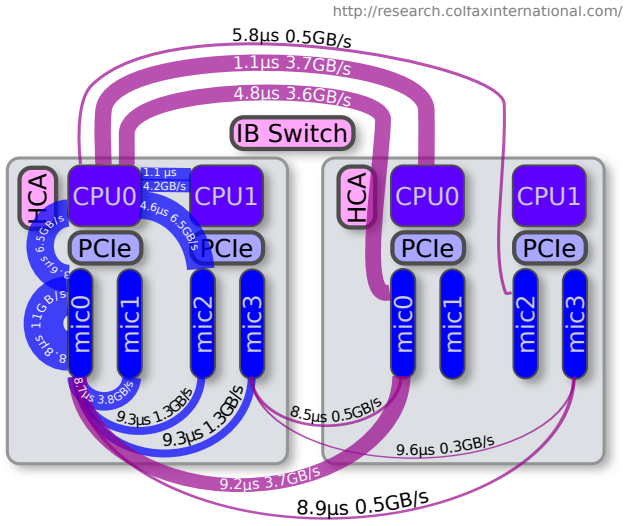

Figure 9: Latency of short MPI messages and bandwidth of long MPI messages with DAPL in our testbench cluster. Thickness of lines is proportional to the bandwidth

5. Case Study: Asian Options

Reduced latency and increased bandwidth of MPI communication in the presence of InfiniBand interconnects is undoubtedly beneficial to any parallel application with data traffic. In order to illustrate the benefits, we benchmark the Asian option pricing application previously reported in [3], this time on our cluster with InfiniBand.

The application is a Monte Carlo code, which distributes an array of calculations for different sets of option parameters across compute devices (CPUs and Intel Xeon Phi coprocessors), using the boss-worker model. The input parameters sent from the boss process to each worker process are the drift rate, variability of the underlying stock asset, and the strike price, and the output returned by the worker are the discounted pay-offs computed using the Monte Carlo method. The amount of exchanged data in each message is very small (32 bytes), and therefore the performance of this application is sensitive to network latency. In order to simplify the interpretation of results, this time we place the worker processes only on Xeon Phi coprocessors.

In the previous publication [3], we focused on the case where work items distributed by the boss process are large, and the communication time is therefore insignificant. In this paper, we consider the case of small work items, which are processed very quickly. The sizes of the work-items, measured in the quantity of random values, in this study range from $W = 2^{16}$ to $W = 2^{24}$, for which the computation time of each work-item ranges from a few microseconds to several milliseconds. Here, $W = M \times N$, where $M = 2^{15}$ is the number of Monte Carlo paths required to achieve the desired accuracy, and $N \in \{2^1, 2^4, 2^9\}$ is the number of time intervals for price averaging per Asian option definition. In order to improve performance, we used a hybrid MPI+OpenMP approach, with seven 32-threaded workers on each Intel Xeon Phi coprocessor. Compared to the setup with 228 threads per process, this approach incurs smaller thread synchronization overhead. 228 threads is the number of logical cores in the Intel Xeon Phi 31S1P coprocessor with 57 cores and 4-way hyper-threading.

Figure 10 shows the performance of this Monte Carlo application, measured in the quantity of random values generated and used per second, as a function of the number of compute devices. In our cluster, four Xeon Phi coprocessors per node are installed, so cases with 6 and 8 coprocessor utilize two compute nodes.

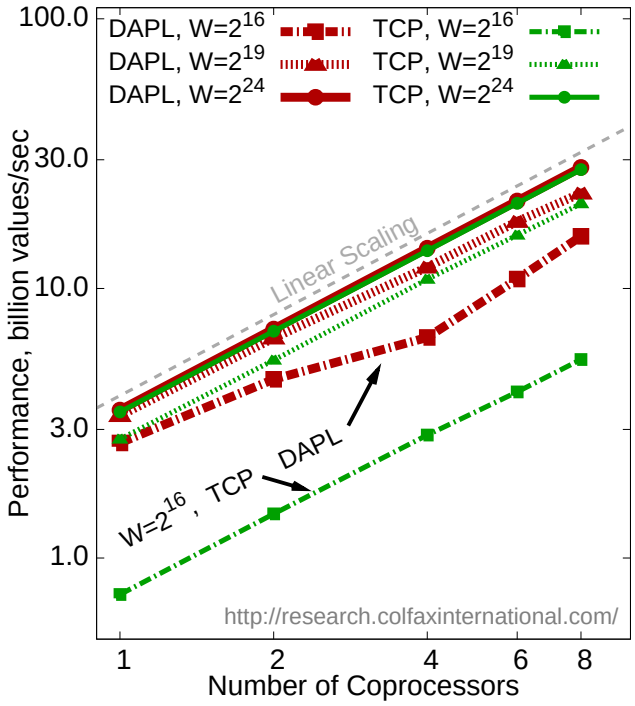

Figure 10: Asian option pricing calculation from [3] with Ethernet and InfiniBand fabrics. Small work item sizes W make the application sensitive to the network latency.

As Figure 10 shows, for large work-item sizes, the performance of the application with Ethernet (labelled "TCP") and with InfiniBand (labelled "DAPL") is identical. However, for smaller work-items, both fabrics lose performance, because each worker has to remain idle for relatively longer periods of time while awaiting a scheduling command from the boss process. The Ethernet case loses performance with decreasing work-item size faster than the InfiniBand case, which is consistent with the lower latency of InfiniBand.

The source code of the application can be obtained at [3], however, the value of M and N and the machine file need to be changed in order to reproduce the results shown in Figure 10.

6. CONCLUSIONS

MPI communication with the TCP protocol saturates the Gigabit Ethernet network bandwidth only for communication between CPU hosts in a cluster. This is satisfactory for applications utilizing the offload model for Xeon Phi coprocessors (where MPI ranks are placed only on CPUs). However, for heterogeneous MPI applications with ranks placed natively on coprocessors, peer-to-peer communication over Ethernet is orders of magnitude slower than the hardware limits.

We have demonstrated that installing InfiniBand controllers and related software on top of the MPSS leads to a tremendous improvement of MPI communication between hosts and coprocessors in a cluster. That includes intra-node communication between CPUs and coprocessors, intra-node communication between coprocessors, and inter-node communication between both types of devices. We have also shown the steps required to reproduce our network and software configuration in CentOS Linux 6.5 with MPSS 3.1.2 and OFED 1.5.4.1 with Mellanox interconnects.

With the `scif0` virtual InfiniBand adapter, MPI communication from hosts to local coprocessors achieves the nominal PCIe bandwidth of 6.5 GB/s with a latency of a few μs. MPI communication between all devices (CPUs and coprocessors) connected to the same CPU as the HCA achieves a nominal unidirectional bandwidth of the QDR 4X interconnect, 4 GB/s.

However, for communication that involves passing messages across the PCIe bus from one CPU to another, a considerable drop in bandwidth is observed. Going from `mic0` to `mic2` on the same dual-socket machine, the bandwidth drops to 1.3 GB/s. Going from `mic0` to `mic2` on a remote machine, the attained bandwidth is only 0.5 GB/s.

The situation with non-uniform communication bandwidth across the heterogeneous cluster with Xeon Phi coprocessors may sometimes be accommodated by considering and tuning the application's communication pattern. For instance,

a) In communication patterns where messages are passed to the nearest neighbor in a ring, the user may change the order of MPI ranks. Pattern `mic0` → `mic2` → `mic3` → `mic1` → remote `mic0`, may be more efficient than `mic0` → `mic1` → `mic2` → `mic3` → remote `mic0`, because in the latter (default) pattern, the link from `mic3` to remote `mic0` is the slowest (around 0.5 GB/s), while in the former (optimized) pattern, links are no worse than from `mic0`→`mic2` or `mic3`→`mic1` (around 1.3 GB/s). This optimization requires only a permutation of the corresponding lines in the MPI machine file.

b) For MPI bandwidth-bound applications, it may be possible to reduce the work share of coprocessors `mic2` and `mic3` installed on CPU2 (if the HCA is installed on CPU1), in order to improve the load balance across the cluster.

c) Additionally, the programmer may consider using offload-like communication pattern, where `mic0` ... `mic3` transfer data to the local CPU, the local CPU sends it to the remote CPU, from which the data are moved to coprocessors in the remote machine.

d) Finally, the programmer may decide that for a particular application, the offload approach is a better choice than symmetric heterogeneous clustering because of the better bandwidth and more uniform RDMA performance between MPI processes.

Despite degraded bandwidth in some cases, all communication paths involving Intel Xeon Phi coprocessors with InfiniBand are faster than with Gigabit Ethernet by one to two orders of magnitude. This is a considerable motivation for equipping compute nodes with InfiniBand interconnects. Even with a standalone workstation without a physical HCA, incorporating the OFED software into the MPSS installation makes sense. In this case, MPSS will employ OFED to create a virtual adapter `scif0`, which will be used by MPI for faster communication between coprocessors.

Software means of improving the non-uniform communication across the bus are investigated, and solutions have been proposed [8]. One of the future publications on Colfax Research will study the possibility of improving inter-node traffic by installing two HCAs, one on CPU1 and another on CPU2. We will also watch for updates in the standard Intel software (MPSS and MPI) for Xeon Phi coprocessors, and for upgrades of the PCIe bus in forthcoming computing platforms.

ACKNOWLEDGEMENTS

We thank Intel's Andrey Semin and Michael Hebenstreit for help with troubleshooting and with the interpretation of our results.

REFERENCES

[1] Landing page for this paper "Configuration and Benchmarks...". http://research.colfaxinternational.com/post/2014/03/11/InfiniBand-for-MIC.aspx.

[2] Intel MPI Library Reference Manual for Linux* OS. http://software.intel.com/sites/products/documentation/hpc/ics/impi/41/lin/Reference_Manual/index.htm.

[3] Hetetogeneous Clustering with Homogeneous Code. http://research.colfaxinternational.com/post/2013/10/17/Heterogeneous-Clustering.aspx.

[4] Intel Xeon Phi Coprocessor System Software Developers Guide. http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide.

[5] Michael Hebenstreit. Configuring Intel Xeon Phi coprocessors inside a cluster. http://software.intel.com/en-us/articles/configuring-intel-xeon-phi-coprocessors-inside-a-cluster.

[6] Intel Manycore Platform Software Stack (MPSS) User's Guide. http://registrationcenter.intel.com/irc_nas/3778/MPSS_Users_Guide.pdf.

[7] Colfax International. *Parallel Programming and Optimization with Intel Xeon Phi Coprocessors*. ISBN: 978-0-9885234-1-8. Colfax International, 2013. http://www.colfax-intl.com/xeonphi/book.html.

[8] S. Potluri et al., in Proceedings of SC'13. MVAPICH-PRISM: a proxy-based communication framework using InfiniBand and SCIF for intel MIC clusters. http://dx.doi.org/10.1145/2503210.2503288.