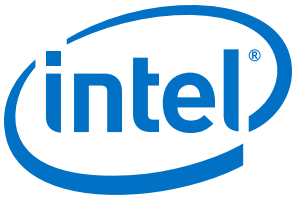
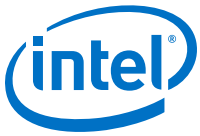


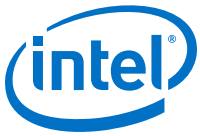
Intel® Omni-Path Fabric Host Software

User Guide

November 2015

Order No.: H76470-1.0

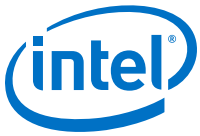




Revision History

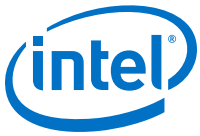
Date	Revision	Description
November 2015	1.0	Document has been updated for Revision 1.0
September 2015	0.7	Document has been updated for Revision 0.7.
April 2015	0.5	Alpha release of document.

Contents



Figures

Tables



Preface

This manual is part of the documentation set for the Intel® Omni-Path Fabric (Intel® OP Fabric), which is an end-to-end solution consisting of adapters, edge switches, director switches and fabric management and development tools.

The Intel® OP Fabric delivers a platform for the next generation of High-Performance Computing (HPC) systems that is designed to cost-effectively meet the scale, density, and reliability requirements of large-scale HPC clusters.

Both the Intel® OP Fabric and standard InfiniBand* are able to send Internet Protocol (IP) traffic over the fabric, or *IPoFabric*. In this document, however, it is referred to as *IP over IB* or *IPoIB*. From a software point of view, IPoFabric and IPoIB behave the same way and, in fact, use the same `ib_ipoib` driver to send IP traffic over the ib0 and/or ib1 ports.

Intended Audience

The intended audience for the Intel® Omni-Path (Intel® OP) document set is network administrators and other qualified personnel.

Documentation Set

The following are the list of the complete end-user publications set for the Intel® Omni-Path product. These documents can be downloaded from https://downloadcenter.intel.com/.

- Hardware Documents:
  - *Intel® Omni-Path Fabric Switches Hardware Installation Guide*
  - *Intel® Omni-Path Fabric Switches GUI User Guide*
  - *Intel® Omni-Path Fabric Switches Command Line Interface Reference Guide*
  - *Intel® Omni-Path Edge Switch Platform Configuration Reference Guide*
  - *Intel® Omni-Path Fabric Managed Switches Release Notes*
  - *Intel® Omni-Path Fabric Externally-Managed Switches Release Notes*
  - *Intel® Omni-Path Host Fabric Interface Installation Guide*
  - *Intel® Omni-Path Host Fabric Interface Release Notes*
- Software Documents:
  - *Intel® Omni-Path Fabric Software Installation Guide*
  - *Intel® Omni-Path Fabric Suite Fabric Manager User Guide*
  - *Intel® Omni-Path Fabric Suite FastFabric User Guide*
  - *Intel® Omni-Path Fabric Host Software User Guide*
  - *Intel® Omni-Path Fabric Suite Fabric Manager GUI Online Help*

- *Intel® Omni-Path Fabric Suite Fabric Manager GUI User Guide*
- *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*
- *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide*
- *Intel® Omni-Path Fabric Performance Tuning User Guide*
- *Intel® Omni-Path Host Fabric Interface Platform Configuration Reference Guide*
- *Intel® Omni-Path Fabric Software Release Notes*
- *Intel® Omni-Path Fabric Manager GUI Release Notes*

Documentation Conventions

This guide uses the following documentation conventions:

- *Note:* provides additional information.
- ***Caution:*** indicates the presence of a hazard that has the potential of causing damage to data or equipment.
- ***Warning:*** indicates the presence of a hazard that has the potential of causing personal injury.
- Text in blue font indicates a hyperlink (jump) to a figure, table, or section in this guide. Links to Web sites are also shown in blue. For example:

  See License Agreements on page 10 for more information.

  For more information, visit www.intel.com.
- Text in **bold** font indicates user interface elements such as a menu items, buttons, check boxes, or column headings. For example:

  Click the **Start** button, point to **Programs**, point to **Accessories**, and then click **Command Prompt**.
- Text in `Courier` font indicates a file name, directory path, or command line text. For example:

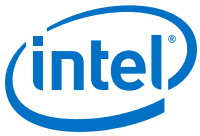
  Enter the following command: `sh ./install.bin`
- Key names and key strokes are shown in underlined bold uppercase letters. For example:

  Press **CTRL+P** and then press the **UP ARROW** key.
- Text in *italics* indicates terms, emphasis, variables, or document titles. For example:

  For a complete listing of license agreements, refer to the *Intel® Software End User License Agreement*.

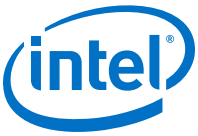
License Agreements

This software is provided under one or more license agreements. Please refer to the license agreement(s) provided with the software for specific detail. Do not install or use the software until you have carefully read and agree to the terms and conditions of the license agreement(s). By loading or using the software, you agree to the terms of the license agreement(s). If you do not wish to so agree, do not install or use the software.



Technical Support

Technical support for Intel® Omni-Path products is available 24 hours a day, 365 days a year. Please contact Intel Customer Support or visit www.intel.com for additional detail.



1.0 Introduction

The *Intel® Omni-Path Fabric Host Software User Guide* shows end users how to use the installed software to set up and administer the fabric. End users include both the cluster administrator and the Message-Passing Interface (MPI) application programmers, who have different but overlapping interests in the details of the technology.

For specific instructions about installing the Intel® Omni-Path Host Fabric Interface adapters see the *Intel® Omni-Path Host Fabric Interface Installation Guide*, and for the initial installation of the Fabric Software see the *Intel® Omni-Path Fabric Software Installation Guide*.

1.1 Interoperability

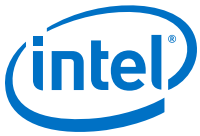
Intel® Omni-Path Fabric participates in the standard subnet management protocols for configuration and monitoring.

Note: In addition to supporting verbs, Intel provides a high-performance vendor-specific protocol for Intel® Omni-Path known as Performance Scaled Messaging 2 (PSM2). MPIs run over PSM2 will not inter-operate with other adapters.

See the OpenFabrics Alliance website at www.openfabrics.org for more information.

1.2 Related Materials

- *Intel® Omni-Path Host Fabric Interface Installation Guide*
- *Intel® Omni-Path Fabric Software Installation Guide*
- Release Notes



2.0 Step-by-Step Cluster Setup and MPI Usage Checklists

This section describes how to set up your cluster to run high-performance Message Passing Interface (MPI) jobs.

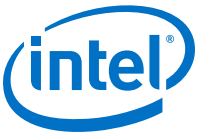
2.1 Cluster Setup

Perform the following tasks when setting up the cluster. These include BIOS, adapter, and system settings. See the "Intel® Omni-Path Performance Tuning User Guide" for information regarding fabric performance tuning.

1. Make sure that hardware installation has been completed according to the instructions in the *Intel® Omni-Path Host Fabric Interface Installation Guide*, and/or the *Intel® Omni-Path Fabric Switches Hardware Installation Guide*, and that software installation and driver configuration has been completed according to the instructions in the *Intel® Omni-Path Fabric Software Installation Guide*. To minimize management problems, the compute nodes of the cluster must have very similar hardware configurations and identical software installations.
2. Check that the BIOS is set properly according to the instructions in the *Intel® Omni-Path Host Fabric Interface Installation Guide*.
3. Set up the Distributed Subnet Administration Provider (DSAP) to correctly synchronize your virtual fabrics (optional). See Intel Distributed Subnet Administration on page 21.
4. Check other performance tuning settings. See the *Intel® Omni-Path Fabric Performance Tuning User Guide*.
5. Set up the host environment to use `ssh`. See the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide* for information on using the opasetupssh CLI command, and the *Intel® Omni-Path Fabric Suite FastFabric User Guide*for information on using the FastFabric TUI to set up `ssh`.
6. Verify the cluster setup. See the `opainfo` CLI command in the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*.

2.2 Using MPI

1. Verify that the Intel hardware and software has been installed on all the nodes you will be using, and that `ssh` is set up on your cluster (see all the steps in the preceeding section, Cluster Setup).
2. Set up Open MPI. See Setup on page 41.
3. Compile Open MPI applications. See Compiling Open MPI Applications on page 42.
4. Create an `mpihosts` file that lists the nodes where your programs will run. See Create the mpi_hosts File on page 43.
5. Run Open MPI applications. See Running Open MPI Applications on page 43.



6. Configure MPI programs for Open MPI. See Configuring MPI Programs for Open MPI on page 44.
7. To test using other MPIs that run over PSM2, such as MVAPICH2, and Intel MPI, see Using Other MPIs on page 56.
8. Use the MPI Selector Utility to switch between multiple versions of MPI. See Managing MPI Versions with the MPI Selector Utility on page 58.
9. Refer to Intel® Omni-Path Cluster Setup and Administration on page 15, and the document *Intel® Omni-Path Fabric Performance Tuning User Guide* for information regarding fabric performance tuning.
10. Refer to Using Other MPIs on page 56 to learn about using other MPI implementations.

3.0 Intel® Omni-Path Cluster Setup and Administration

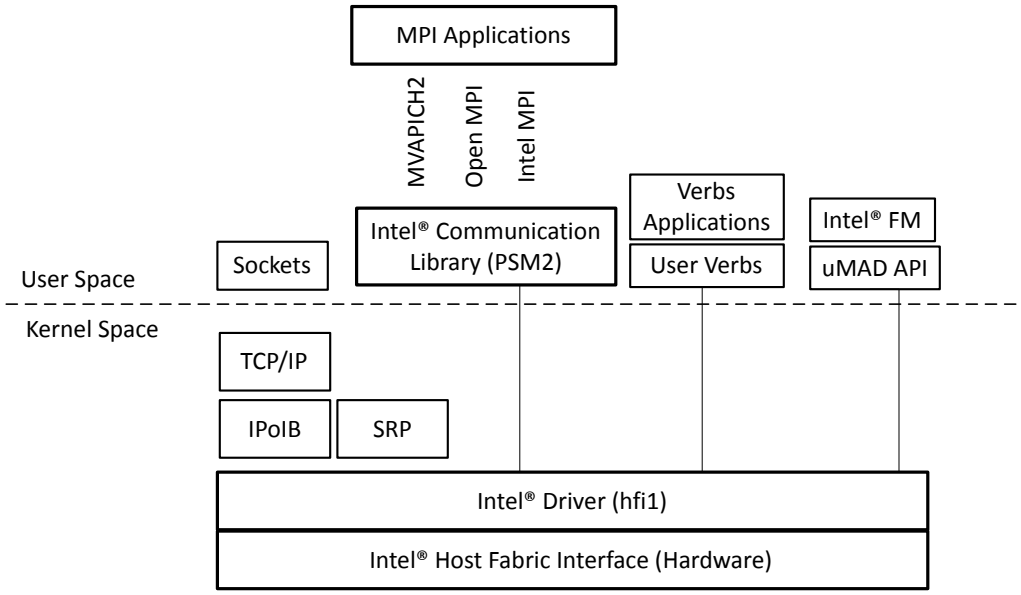
This section describes what the cluster administrator needs to know about the Intel® Omni-Path software and system administration.

3.1 Introduction

The Intel® Omni-Path Host Fabric Interface driver, Intel® Performance Scaled Messaging for Intel® Omni-Path (PSM2), accelerated Message-Passing Interface (MPI) stack, the protocol and MPI support libraries, and other modules are components of the Intel® Omni-Path Software. This software provides the foundation that supports the MPI implementation. The following figure, illustrates these relationships.

Note: Intel recommends using only MPI over PSM2 for best performance.

Figure 1. Intel OFA Delta Software Structure



3.2 Installed Layout

This section describes the default installed layout for the Intel® Omni-Path Software and Intel-supplied Message Passing Interfaces (MPIs).

Intel-supplied Open MPI and MVAPICH2 RPMs, with Performance Scaled Messaging for Intel® Omni-Path (PSM2) support and compiled with the Intel, GCC, and the PGI compilers, are installed in directories using the following format:

/usr/mpi/<*compiler*>/<*mpi*>-<*mpi_version*>-hfi

For example:

```
/usr/mpi/gcc/openmpi-X.X.X-hfi
```

Intel® Omni-Path Fabric utility programs, are installed in:

- /usr/sbin
- /opt/opa/*
- /opt/opafm

Documentation is found in:

- /usr/share/man
- */usr/share/doc/opa-fm-X.X.X.X*

Intel® Omni-Path user documentation can be found on the Intel web site on the software download page for your distribution.

Configuration files are found in:

- /etc/sysconfig
- /etc/sysconfig/opa

Init scripts are found in:

- /etc/init.d

The Intel® Omni-Path Fabric driver modules are installed in:

- /lib/modules/<kernel name>/updates/drivers/infiniband/hw/hfi1

Most of the other Intel® Omni-Path Fabric modules are installed under the /opt/opa subdirectory. Other modules are installed under:

- /lib/modules/<kernel name>/updates/kernel/drivers/net

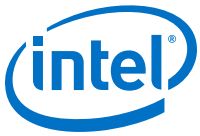
The RDS modules are installed under:

- /lib/modules/<kernel name>/updates/kernel/net/rds

3.3 Intel® Omni-Path Fabric and OpenFabrics Driver Overview

The Intel® Omni-Path Host Fabric Interface hfi1 kernel module provides low-level Intel hardware support, and is the base driver for both Message Passing Interface (MPI)/Performance Scaled Messaging (PSM2) programs and general OpenFabrics protocols such as IPoIB. The driver also supplies the Subnet Management Agent (SMA) component.

The following is a list of the optional configurable OpenFabrics components and their default settings:



- IPoIB network interface

  This component is required for TCP/IP networking for running IP traffic over the Intel® Omni-Path Fabric link. It is not running until it is configured.

- OpenSM

  This component is not supported with Intel® Omni-Path Fabric. Intel recommends using the Intel® Omni-Path Fabric Suite Fabric Manager (FM), which is included with the Intel® Omni-Path Fabric Suite or optionally available within the Intel switches. The Fabric Manager can be installed on one or more nodes with only one node being the master Subnet Manager (SM).

- SCSI RDMA Protocol (SRP)

  This component is not running until the module is loaded and the SRP devices on the fabric have been discovered.

Other optional drivers can now be configured and enabled, as described in IPoIB Network Interface Configuration on page 17.

Complete information about starting, stopping, and restarting the Intel® Omni-Path services are in Managing the Intel® Omni-Path Fabric Driver.

3.4 IPoIB Network Interface Configuration

The following instructions show you how to manually configure your OpenFabrics IPoIB network interface. Intel recommends using the Intel® Omni-Path Fabric Host Software Installation package or the opaconfig tool. For larger clusters, Intel® Omni-Path Fabric Suite FastFabric (FF) can be used to automate installation and configuration of many nodes. These tools automate the configuration of the IPoIB network interface. This example assumes that you are using `sh` or `bash` as your shell, all required Intel® Omni-Path and OpenFabric's RPMs are installed, and your startup scripts have been run (either manually or at system boot). For this example, the IPoIB network is 10.1.17.0 (one of the networks reserved for private use, and thus not routeable on the Internet), with a /8 host portion. In this case, the netmask must be specified. This example assumes that no hosts files exist, the host being configured has the IP address 10.1.17.3, and DHCP is not used.

Note: Instructions are only for this static IP address case.

1. Type the following command (as a root user):

```
ifconfig ib0 10.1.17.3 netmask 0xffffff00
```

   Note: You have the option to use the opa software Textual User Interface (TUI) to do this. `/.INSTALL` or `opaconfig` will take you to the OPA Software TUI where there is a process for doing this via TUI.

2. To verify the configuration, type:

```
ifconfig ib0
ifconfig ib1
```

The output from this command will be similar to:

```
ib0   Link encap:InfiniBand HWaddr
00:00:00:02:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
inet addr:10.1.17.3  Bcast:10.1.17.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:4096  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:128
RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

3. Type:

```
ping -c 2 -b 10.1.17.255
```

The output of the `ping` command will be similar to the following, with a line for each host already configured and connected:

```
WARNING: pinging broadcast address
PING 10.1.17.255 (10.1.17.255) 517(84) bytes of data.
174 bytes from 10.1.17.3: icmp_seq=0 ttl=174 time=0.022 ms
64 bytes from 10.1.17.1: icmp_seq=0 ttl=64 time=0.070 ms (DUP!)
64 bytes from 10.1.17.7: icmp_seq=0 ttl=64 time=0.073 ms (DUP!)
```

The IPoIB network interface is now configured.

4. Restart (as a root user) by typing:

```
systemctl restart openibd
```

Note: IPoIB-CM (Connected Mode) is enabled by default. The setting in `/etc/infiniband/openib.conf` is `SET_IPOIB_CM=yes`. To use datagram mode, change the setting to `SET_IPOIB_CM=no`. Setting can also be changed when asked during initial installation (`./INSTALL`).

3.5 IPoIB Administration

The following sections provide information on the administration of the IPoIB Driver.

3.5.1 Stop, Start and Restart the IPoIB Driver

Intel recommends using the Intel® Omni-Path Fabric Installer TUI or `opaconfig` command to enable autostart for the IPoIB driver. Refer to the *Intel® Omni-Path Fabric Software Installation Guide* for more information.

Use the following command line commands to stop, start, and restart the IPoIB driver.

- To stop the IPoIB driver: `systemctl stop openibd`
- To start the IPoIB driver: `systemctl start openibd`
- To restart the IPoIB driver: `systemctl restart openibd`

3.5.2 Configure IPoIB

Intel recommends using the Intel® OPA Software TUI, or `opaconfig` command to configure the auto-start of the IPoIB driver. Refer to the *Intel® Omni-Path Fabric Software Installation Guide* for more information on using the Intel® OPA Software TUI.

To configure the IPoIB driver using the command line, perform the following steps.

1. For each IP Link Layer interface, create an interface configuration file, `/etc/sysconfig/network-scripts/ifcfg-NAME`, where *NAME* is the network interface name. The following is an example of the `ifcfg-NAME` file:

```
DEVICE=ib0
BOOTPROTO=static
IPADDR=10.228.216.153
BROADCAST=10.228.219.255
NETWORK=10.228.216.0
NETMASK=255.255.252.0
ONBOOT=yes
NM_CONTROLLED=no
```

   Note: For IPoIB, the `INSTALL` script for the adapter now helps you create the `ifcfg` files.

2. Restart the IPoIB driver with the following:

```
systemctl restart openibd
```

3.6 IB Bonding

IB bonding is a high-availability solution for IPoIB interfaces. It is based on the Linux Ethernet Bonding Driver and was adopted to work with IPoIB. The support for IPoIB interfaces is only for the active-backup mode. Other modes are not supported.

3.6.1 Interface Configuration Scripts

Create interface configuration scripts for the ibX and bondX interfaces. Once the configurations are in place, perform a server reboot, or a service network restart. For SLES operating systems (OS), a server reboot is required. Refer to the following standard syntax for bonding configuration by the OS.

3.6.1.1 Red Hat Enterprise Linux* (RHEL)

See the *Intel® Omni-Path Fabric Software Release Notes* for versions of RHEL that are supported by Intel® Omni-Path Fabric Host Software.

Add the following lines to the RHEL file `/etc/modprobe.d/hfi.conf`:

```
alias bond0 bonding
        options bonding miimon=100 mode=1 max_bonds=1
```

3.6.1.2 SuSE Linux Enterprise Server * (SLES)

See the *Intel® Omni-Path Fabric Software Release Notes* for versions of SLES that are supported by Intel® Omni-Path Fabric Host Software

Verify that the following line is set to the value of yes in `/etc/sysconfig/boot`:

```
RUN_PARALLEL="yes"
```

3.6.2 Verify IB Bonding is Configured

After the configuration scripts are updated, and the service network is restarted or a server reboot is accomplished, use the following CLI commands to verify that IB bonding is configured.

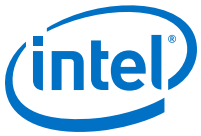
```
cat /proc/net/bonding/bond0
ifconfig
```

Example of `cat /proc/net/bonding/bond0` output:

```
cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: vX.X.X (mm dd, yyyy)

Bonding Mode: fault-tolerance (active-backup) (fail_over_mac)
Primary Slave: ib0
Currently Active Slave: ib0
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: ib0
MII Status: up
Link Failure Count: 0
Permanent HW addr: 80:00:04:04:fe:80

Slave Interface: ib1
MII Status: up
Link Failure Count: 0
Permanent HW addr: 80:00:04:05:fe:80
```

Example of `ifconfig` output:

```
st2169:/etc/sysconfig ifconfig
bond0     Link encap:InfiniBand  HWaddr
80:00:00:02:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
       inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
       inet6 addr: fe80::211:7500:ff:909b/64 Scope:Link
       UP BROADCAST RUNNING MASTER MULTICAST  MTU:65520  Metric:1
       RX packets:120619276 errors:0 dropped:0 overruns:0 frame:0
       TX packets:120619277 errors:0 dropped:137 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:10132014352 (9662.6 Mb)  TX bytes:10614493096 (10122.7 Mb)

ib0     Link encap:InfiniBand  HWaddr
80:00:00:02:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
       UP BROADCAST RUNNING SLAVE MULTICAST  MTU:65520  Metric:1
       RX packets:118938033 errors:0 dropped:0 overruns:0 frame:0
       TX packets:118938027 errors:0 dropped:41 overruns:0 carrier:0
       collisions:0 txqueuelen:256
       RX bytes:9990790704 (9527.9 Mb)  TX bytes:10466543096 (9981.6 Mb)
```



```
ib1     Link encap:InfiniBand  HWaddr
80:00:00:02:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00
       UP BROADCAST RUNNING SLAVE MULTICAST  MTU:65520  Metric:1
       RX packets:1681243 errors:0 dropped:0 overruns:0 frame:0
       TX packets:1681250 errors:0 dropped:96 overruns:0 carrier:0
       collisions:0 txqueuelen:256
       RX bytes:141223648 (134.6 Mb)  TX bytes:147950000 (141.0 Mb)
```

3.7 Intel Distributed Subnet Administration

As Intel® Omni-Path Fabric clusters are scaled into the Petaflop range and beyond, a more efficient method for handling queries to the Fabric Manager (FM) is required. One of the issues is that while the Fabric Manager can configure and operate that many nodes, under certain conditions it can become overloaded with queries from those same nodes.

For example, consider a fabric consisting of 1,000 nodes, each with 4 processors. When a large Message Passing Interface (MPI) job is started across the entire fabric, each process needs to collect path records for every other node in the fabric. Each process is going to be querying the subnet manager for these path records at roughly the same time. This amounts to a total of 3.9 million path queries just to start the job.

In the past, MPI implementations have side-stepped this problem by hand crafting path records themselves, but this solution cannot be used if advanced fabric management techniques such as virtual fabrics and advanced topologies configurations are being used. In such cases, only the subnet manager itself has enough information to correctly build a path record between two nodes.

The open source daemon `ibacm` accomplishes the task of collecting path records by caching paths as they are used. `ibacm` also provides a plugin architecture, which Intel has extended with the Distributed Subnet Administration Provider (DSAP) to enhance the `ibacm` solution. DSAP works with `ibacm` by allowing each node to locally replicate the path records needed to reach the other nodes on the fabric. At boot time, each `ibacm`/DSAP pairing queries the subnet manager for information about the relevant parts of the fabric, backing off whenever the subnet manager indicates that it is busy. Once this information is in the DSAP's database, it is ready to answer local path queries from MPI or other applications. If the fabric changes (due to a switch failure or a node being added or removed from the fabric) the DSAP updates the affected portions of the database. The DSAP can be installed and run on any node in the fabric, however it is only needed on nodes running shared memory (SHMEM) and MPI applications.

3.7.1 Applications that use the DSAP Plugin

The Performance Scaled Messaging (PSM2) Library for Intel® Omni-Path has been extended to take advantage of the Distributed Subnet Administration Provider (DSAP) plugin to the `ibacm` daemon. Therefore, all MPIs that use the Intel PSM2 library can take advantage of the DSAP. Other applications must be modified specifically to take advantage of it.

If you plan to use the DSAP, refer to the header file `/usr/include/Infiniband/opasadb_path.h` for information on using the DSAP APIs. This file can be found on any node where the `libopasadb` library is installed. The libopasadb library is found in the Intel® Omni-Path Fabric Suite FastFabric-AR package, which is named `opa-address-resolution`. For further assistance, please contact Intel Support.

3.7.2 DSAP Configuration File

The Distributed Subnet Administration Provider (DSAP) configuration file is `/etc/rdma/dsap.conf`. It has several settings, but normally administrators will only need to deal with two or three of them.

3.7.2.1 SID

The SID is the primary configuration setting for the DSAP, and it can be specified multiple times. The SIDs identify applications which will use the DSAP to determine their path records. The default configuration for the DSAP includes all the SIDs defined in the default Fabric Manager (FM) configuration for use by MPI.

Each SID= entry defines one Service ID that will be used to identify an application. In addition, multiple SID= entries can be specified. For example, if a virtual fabric has three sets of SIDs associated with it (0x0a1 through 0x0a3, 0x1a1 through 0x1a3, and 0x2a1 through 0x2a3), you would define this as:

```
SID=0x0a1
SID=0x0a2
SID=0x0a3
SID=0x1a1
SID=0x1a2
SID=0x1a3
SID=0x2a1
SID=0x2a2
SID=0x2a3
```

Note: A SID of zero is not supported at this time. Instead, the OPP libraries treat zero values as “unspecified”.

3.7.2.2 ScanFrequency

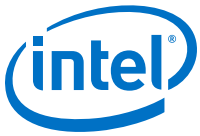
Periodically, the DSAP will completely re-synchronize its database. This also occurs if the Fabric Manager (FM) is restarted. `ScanFrequency` defines the minimum number of seconds between complete re-synchronizations. It defaults to 600 seconds, or 10 minutes. On very large fabrics, increasing this value may help reduce the total amount of SM traffic. For example, to set the interval to 15 minutes, add this line to the bottom of the `dsap.conf` file:

```
ScanFrequency=900
```

3.7.2.3 LogFile

Normally, the DSAP logs special events through syslog to `/var/log/messages`. This parameter allows you to specify a different destination for the log messages. For example, to direct DSAP messages to their own log, add this line to the bottom of the `dsap.conf` file:

```
LogFile=/var/log/dsap.log
```



3.7.2.4 Dbg

This parameter controls how much logging the DSAP will do. It can be set to a number between one and seven, where one indicates no logging and seven includes informational and debugging messages. To change the `Dbg` setting for DSAP, find the line in `dsap.conf` that reads `Dbg=5` and change it to a different value, between 1 and 7. The value of `Dbg` changes the amount of logging that the DSAP generates as follows:

- Dbg=1 or Dbg=2: Alerts and Critical Errors

  Only errors that will cause the DSAP to terminate will be reported.

- Dbg=3: Errors

  Errors will be reported, but nothing else. (Includes Dbg=1 and Dbg=2)

- Dbg=4: Warnings

  Errors and warnings will be reported. (Includes Dbg=3)

- Dbg=5: Normal

  Some normal events will be reported along with errors and warnings. (Includes Dbg=4)

- Dbg=6: Informational Messages

  In addition to the normal logging, DSAP will report detailed information about its status and operation. Generally, this will produce too much information for normal use. (Includes Dbg=5)

- Dbg=7: Debugging

  This should only be turned on at the request of Intel Support. This will generate so much information that system operation will be impacted. (Includes Dbg=6)

3.7.2.5 Other Settings

The remaining configuration settings for the DSAP are generally only useful in special circumstances and are not needed in normal operation. The sample `dsap.conf` configuration file contains a brief description of each.

3.7.3 Virtual Fabrics and the Distributed SA Provider

OFA applications can be identified by a Service ID (SID). The Intel® Omni-Path Fabric Suite Fabric Manager (FM) uses SIDs to identify applications. One or more applications can be associated with a Virtual Fabric using the SID. The Distributed Subnet Administration Provider (DSAP) is designed to be aware of Virtual Fabrics, and to only store records for those Virtual Fabrics that match the SIDs in the DSAP's configuration file. The DSAP recognizes when multiple SIDs match the same Virtual Fabric and will only store one copy of each path record within a Virtual Fabric. SIDs that match more than one Virtual Fabric will be associated with a single Virtual Fabric. The method for handling cases of multiple VFs matching the same SID is discussed later. The Virtual Fabrics that do not match SIDs in the DSAP's database will be ignored.

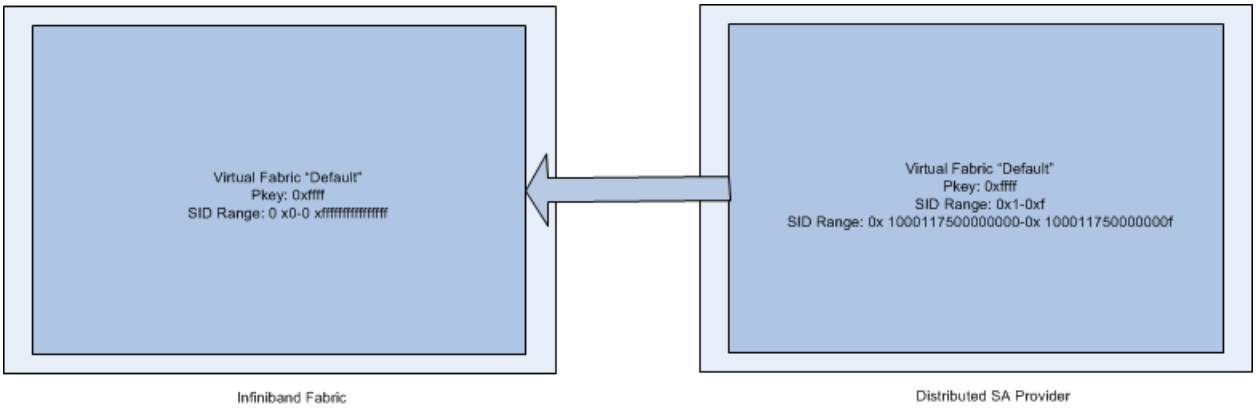
3.7.3.1 Configuring the DSAP

In order to absolutely minimize the number of queries made by the DSAP, it is important to configure it correctly, both to match the configuration of the Fabric Manager (FM) and to exclude those portions of the fabric that will not be used by applications using the DSAP. The configuration file for the DSAP is named `/etc/rdma/dsap.conf`.

3.7.3.2 Default Configuration

As shipped, the Fabric Manager (FM) creates a single application virtual fabric, called "Default" and maps all nodes and Service IDs (SIDs) to it, and the DSAP ships with a configuration that lists a set of 31 SIDs, 0x1000117500000000 through 0x100011750000000f, and 0x1 through 0xf. This results in an arrangement like the one shown in the following figure.

Figure 2. Distributed SA Provider Default Configuration



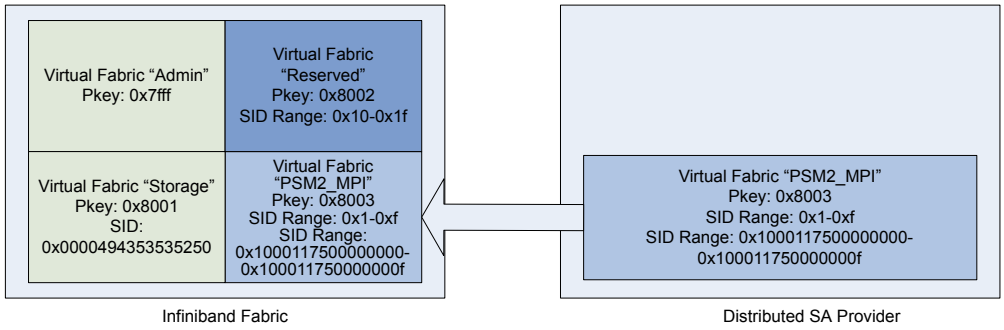
If you are using the Fabric Manager in its default configuration, and you are using the standard Intel PSM2 SIDs, this arrangement will work fine and you will not need to modify the DSAP's configuration file - but notice that the DSAP has restricted the range of SIDs it cares about to those that were defined in its configuration file. Attempts to get path records using other SIDs will not work, even if those other SIDs are valid for the fabric. When using this default configuration it is necessary that MPI applications only be run using one of these 31 SIDs.

3.7.3.3 Multiple Virtual Fabrics Example

If you are configuring the physical fabric, you may want to limit how much bandwidth MPI applications are permitted to consume. To do so, you may re-configure the Fabric Manager (FM), turning off the "Default" Virtual Fabric and replacing it with several other Virtual Fabrics.

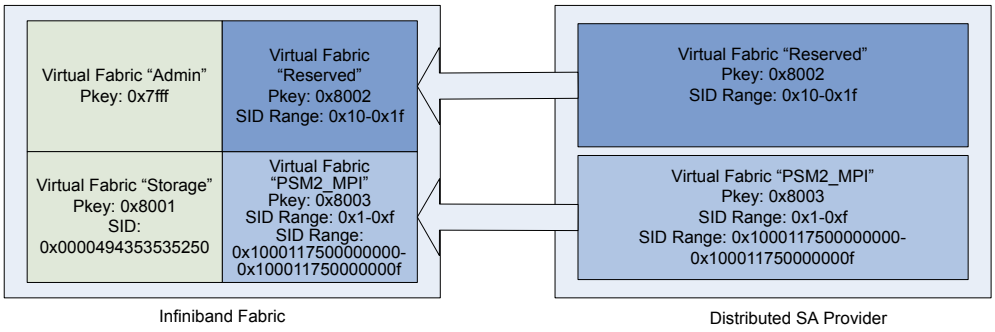
In Figure 3 on page 25, the administrator has divided the physical fabric into four virtual fabrics: "Admin" (used to communicate with the Fabric Manager), "Storage" (used by SRP), "PSM2_MPI" (used by regular MPI jobs) and a special "Reserved" fabric for special high-priority jobs.

Figure 3. Distributed SA Provider Multiple Virtual Fabrics Example



Due to the fact that the DSAP was not configured to include the SID Range 0x10 through 0x1f, it has simply ignored the "Reserved" virtual fabric. Adding those SIDs to the dsap.conf file solves the problem as shown in Figure 4 on page 25.

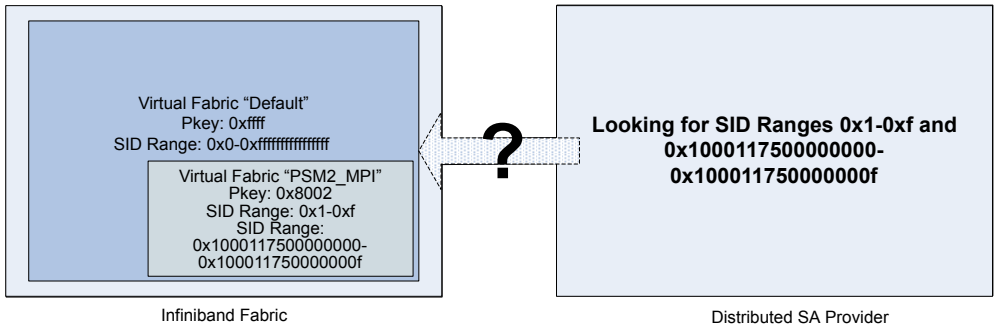
Figure 4. Distributed SA Provider Multiple Virtual Fabrics Configured Example



3.7.3.4 Virtual Fabrics with Overlapping Definitions

As defined, SIDs should never be shared between Virtual Fabrics. Unfortunately, it is very easy to accidentally create such overlaps. The following figure shows an example with overlapping definitions.

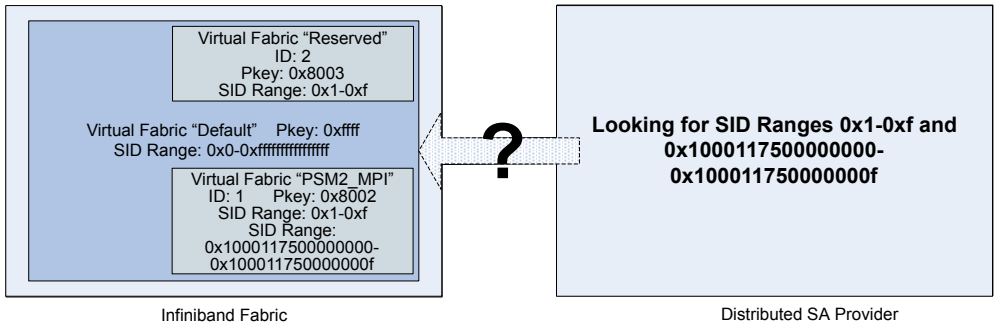
Figure 5. Virtual Fabrics with Overlapping Definitions



In the previous figure, the fabric administrator enabled the "PSM2_MPI" Virtual Fabric without modifying the "Default" Virtual Fabric. As a result, the DSAP sees two different Virtual Fabrics that match its configuration file.

In the following, the person administering the fabric has created two different Virtual Fabrics without turning off the Default - and two of the new fabrics have overlapping SID ranges.

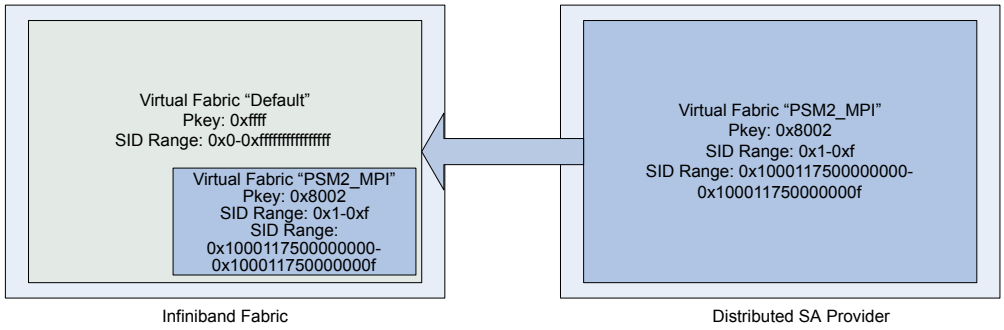
Figure 6. Virtual Fabrics with PSM2_MPI Virtual Fabric Enabled



In Figure 6 on page 26, the administrator enabled the "PSM2_MPI" fabric, and then added a new "Reserved" fabric that uses one of the SID ranges that "PSM2_MPI" uses. When a path query has been received, the DSAP deals with these conflicts as follows:

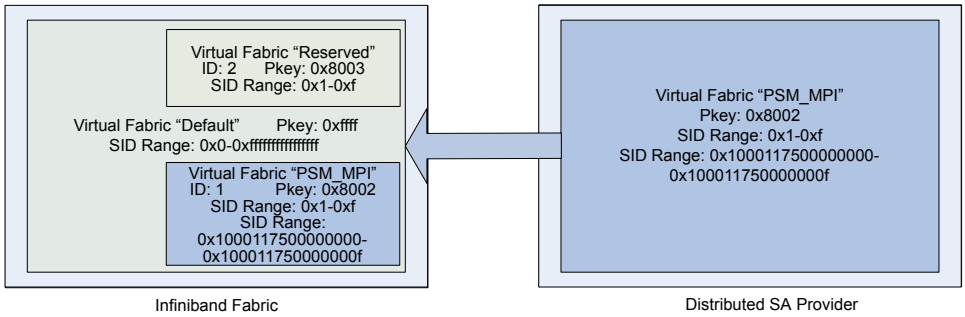
First, any Virtual Fabric with a pkey of 0xffff or 0x7fff is considered to be an Admin or Default Virtual Fabric. This Admin or Default Virtual Fabric is treated as a special case by the DSAP and is used only as a last resort. Stored SIDs are only mapped to the default Virtual Fabric if they do not match any other Virtual Fabrics. Thus, in the first example, in the previous figure, the DSAP will assign all the SIDs in its configuration file to the "PSM2_MPI" Virtual Fabric as shown in the following figure.

Figure 7. Virtual Fabrics with all SIDs assigned to PSM2_MPI Virtual Fabric



Second, the DSAP handles overlaps by taking advantage of the fact that Virtual Fabrics have unique numeric indexes. These indexes are assigned by the Fabric Manager (FM) in the order which the Virtual Fabrics appear in the configuration file. These indexes can be seen by using the command `opasaquery -o vfinfo` command. The DSAP will always assign a SID to the Virtual Fabric with the lowest index, as shown in the following figure. This ensures that all copies of the DSAP in the fabric will make the same decisions about assigning SIDs. However, it also means that the behavior of your fabric can be affected by the order in which you configured the virtual fabrics.

Figure 8. Virtual Fabrics with Unique Numeric Indexes



In the previous figure, Virtual Fabrics with Unique Numeric Indexes, the DSAP assigns all overlapping SIDs to the "PSM2_MPI" fabric because it has the lowest Index.

Note: The DSAP makes these assignments in order to allow the fabric to work despite configuration ambiguities. The proper solution in these cases is to redefine the fabric so that no node will ever be a member of two Virtual Fabrics that service the same SID.

3.8 MTU Size

You may specify the size of the Maximum Transfer Unit (MTU) for each of the virtual fabrics (vFabric) defined on a Host Fabric Interface (HFI). The values may be set to one of the following: 2048 bytes, 4096 bytes, 8192 bytes, and 10240 bytes, but the size is limited to the capability of the port/link. Intel recommends that you use the default value, which is 8192 bytes.

Intel has found that the default MTU size of 8K provides the best fabric performance, and recommends that you do not change it. However, there is a provision for making changes in MTU size in the Intel® Omni-Path Fabric Suite Fabric Manager. Refer to the *Intel® Omni-Path Fabric Suite Fabric Manager User Guide*.

Note: Performance Scaled Messaging for Intel® Omni-Path (PSM2) has an environment variable, *PSM2_MTU* that can be set to help tune the system. *PSM2_MTU* can be set to a value equal to or smaller than the MTU size specified for the vFabric being used, but not larger. The PSM2 will use this value as its maximum payload. A *PSM2_MTU* value that is set lower than the vFabric MTU will have the effect of PSM2 behaving as if the overall fabric is configured with a smaller MTU size. *PSM2_MTU* can be assigned one of the number designations, or one of the discrete numerical values from the Value in Bytes column in the preceding table.

3.9 Managing the Intel® Omni-Path Fabric Driver

The startup script for the Intel® Omni-Path Host Fabric Interface (HFI) is installed automatically as part of the software installation, and normally does not need to be changed. It runs as a system service.

The primary configuration file for the Intel® Omni-Path Fabric driver and other modules and associated daemons is `/etc/rdma/rdma.conf`.

Normally, this configuration file is set up correctly at installation and the drivers are loaded automatically during system boot once the software has been installed. However, the HFI driver has several configuration variables that set reserved buffers for the software, define events to create trace records, and set the debug level.

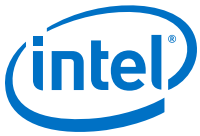
If you are upgrading, your existing configuration files will not be overwritten.

3.9.1 Intel® Omni-Path Driver File System

The Intel® Omni-Path driver supplies a file system for exporting certain binary statistics to user applications. By default, this file system is mounted in the `/sys/kernel/debug` directory when the Intel® Omni-Path script is invoked with the `start` option (for example, at system startup). The file system is unmounted when the Intel® Omni-Path script is invoked with the `stop` option (for example, at system shutdown).

The following is a sample layout of a system with two cards:

```
/sys/kernel/debug/0/flash
/sys/kernel/debug/0/port2counters
/sys/kernel/debug/0/port1counters
/sys/kernel/debug/0/portcounter_names
/sys/kernel/debug/0/counter_names
/sys/kernel/debug/0/counters
```



```
/sys/kernel/debug/driver_stats_names
/sys/kernel/debug/driver_stats
/sys/kernel/debug/1/flash
/sys/kernel/debug/1/port2counters
/sys/kernel/debug/1/port1counters
/sys/kernel/debug/1/portcounter_names
/sys/kernel/debug/1/counter_names
/sys/kernel/debug/1/counters
```

The `driver_stats` file contains general driver statistics. There is one numbered subdirectory per Intel® Omni-Path device on the system. Each numbered subdirectory contains the following per-device files:

- `port1counters`
- `port2counters`
- `flash`

The `driver1counters` and `driver2counters` files contain counters for the device, for example, interrupts received, bytes and packets in and out, and so forth. The `flash` file is an interface for internal diagnostic commands.

The file `counter_names` provides the names associated with each of the counters in the binary `port#counters` files, and the file `driver_stats_names` provides the names for the stats in the binary `driver_stats` files.

3.10 More Information on Configuring and Loading Drivers

See the `modprobe(8)`, `modprobe.conf(5)`, and `lsmod(8)` man pages for more information. Also see the file `/usr/share/doc/initscripts-*/sysconfig.txt` for more general information on configuration files.

4.0 Intel® True Scale/Intel® Omni-Path Coexistence

It is possible to have Intel® True Scale and Intel® Omni-Path coexist within the same server. Doing so, however, requires some special procedures. It is important to keep the following points in mind.

- Intel® True Scale and Intel® Omni-Path do not interoperate. You cannot connect a Intel® True Scale adapter card to an Intel® Omni-Path switch. Likewise, you cannot connect an Intel® Omni-Path adapter card to a Intel® True Scale switch.
- Each fabric must have its own Intel® Omni-Path Fabric Suite Fabric Manager node.
- Any node you intend to use in a coexistence scenario must be running an Intel® Omni-Path-compatible Linux* distribution. See the *Intel® Omni-Path Fabric Software Release Notes* for versions of Linux* distributions that are supported by Intel® Omni-Path Fabric Host Software
- Some MPIs *can* work with both Intel® True Scale and Intel® Omni-Path network adapters. However, to do so, those MPIs must be recompiled such that they are able to correctly choose the library support for the underlying hardware.

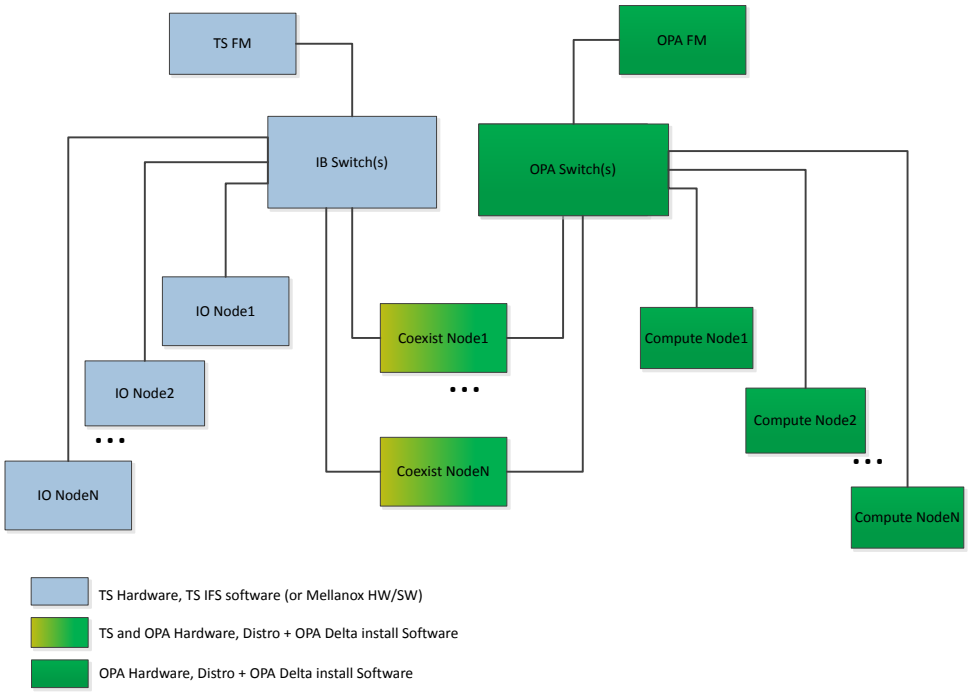
4.1 Coexist Nodes

In a mixed fabric (Intel® True Scale and Intel® Omni-Path coexisting), select nodes will have both Intel® True Scale HCAs and Intel® Omni-Path HFIs installed. Note that verbs continues to work on both cards without special configuration.

4.2 Configurations

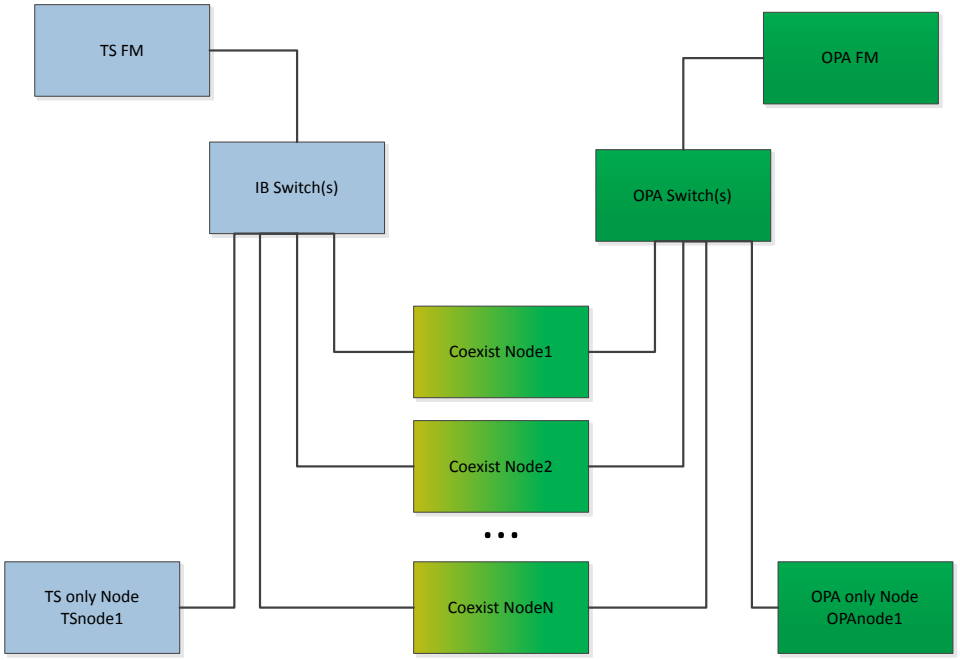
There are two supported scenarios for creating Intel® True Scale/Intel® Omni-Path coexistence configurations. The first is where the Intel® True Scale hardware serves as an InfiniBand* storage network, and the Intel® Omni-Path hardware is used for computing. This configuration allows for the continued use of existing Intel® True Scale infrastructure for data storage, and for taking advantage of the performance improvements that are inherent in Intel® Omni-Path for compute operations and fabric management. The following figure illustrates this configuration.

Figure 9. Intel® True Scale/Intel® Omni-Path



The second scenario is referred to as a rolling upgrade from Intel® True Scale to Intel® Omni-Path. In this scenario, Intel® Omni-Path hardware is added to an existing Intel® True Scale fabric over time. Once the Intel® Omni-Path hardware is installed and configured, the Intel® True Scale hardware is decommissioned.

Figure 10. Intel® True Scale/Omni-Path Rolling Upgrade

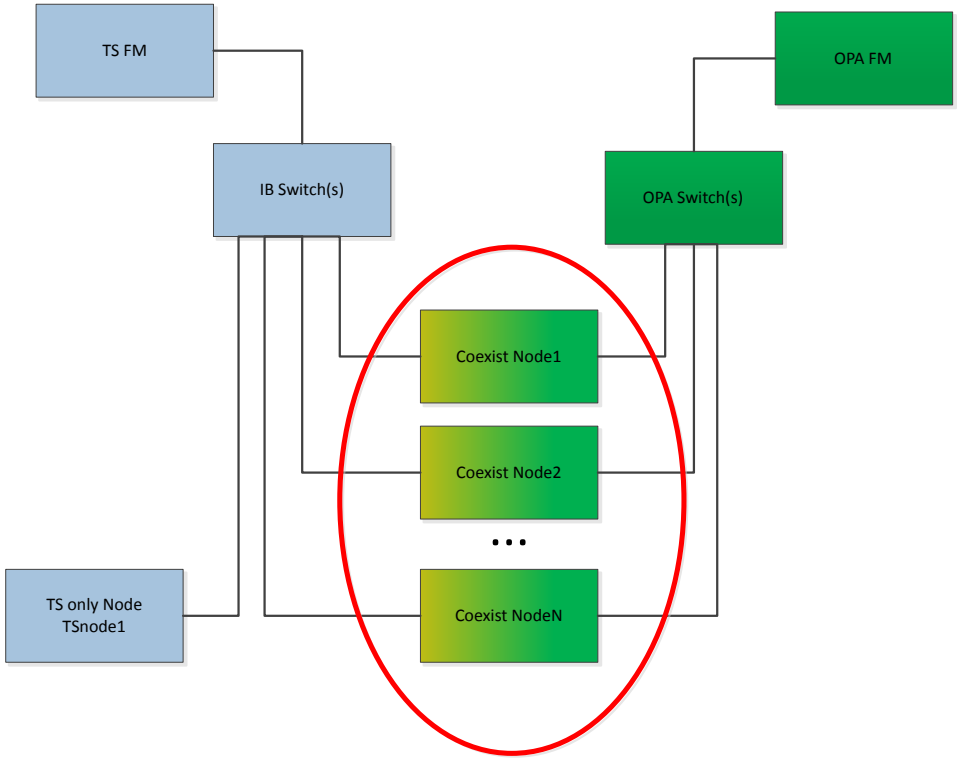


4.3 Coexist Node Details

Coexist Nodes support full distro verbs. All standard verbs applications, including Lustre, IPoIB, SRP, and Verbs MPIs, are supported by the distro software for both Intel® True Scale and Intel® Omni-Path hardware.

Monitoring and Management of the InfiniBand* fabric is accomplished via *Intel® True Scale Fabric Suite FastFabric* and/or Open Fabrics commands. Refer to *Intel® True Scale Fabric Suite FastFabric User Guide* and OFA distro documentation.

Figure 11. Coexist Node Details

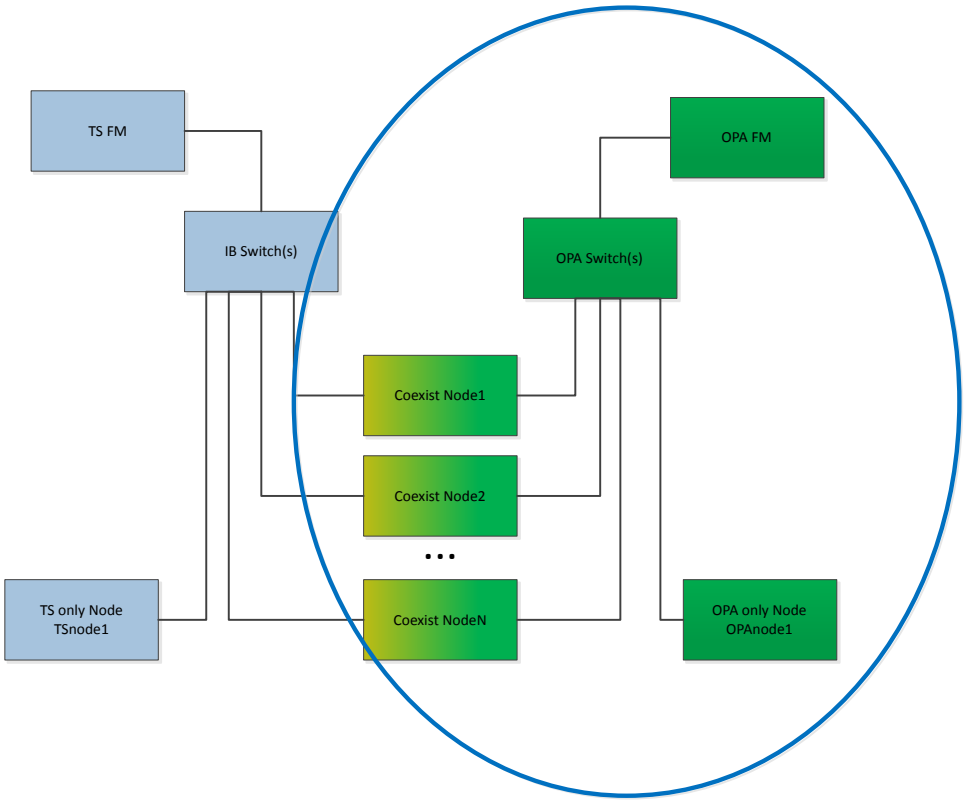


4.4 Omni-Path Node Details

Intel® Omni-Path Fabric Suite Fabric Manager runs on an Intel® Omni-Path-only node, and manages the Intel® Omni-Path hardware, whether in Intel® Omni-Path nodes or coexist nodes. Full Intel® Omni-Path support is provided on all these nodes.

The Intel® Omni-Path Delta install is installed on all nodes with Intel® Omni-Path hardware. The Delta Install will only upgrade some distro packages when deemed necessary by the installation scripts, based on the distro detected during the installation process.

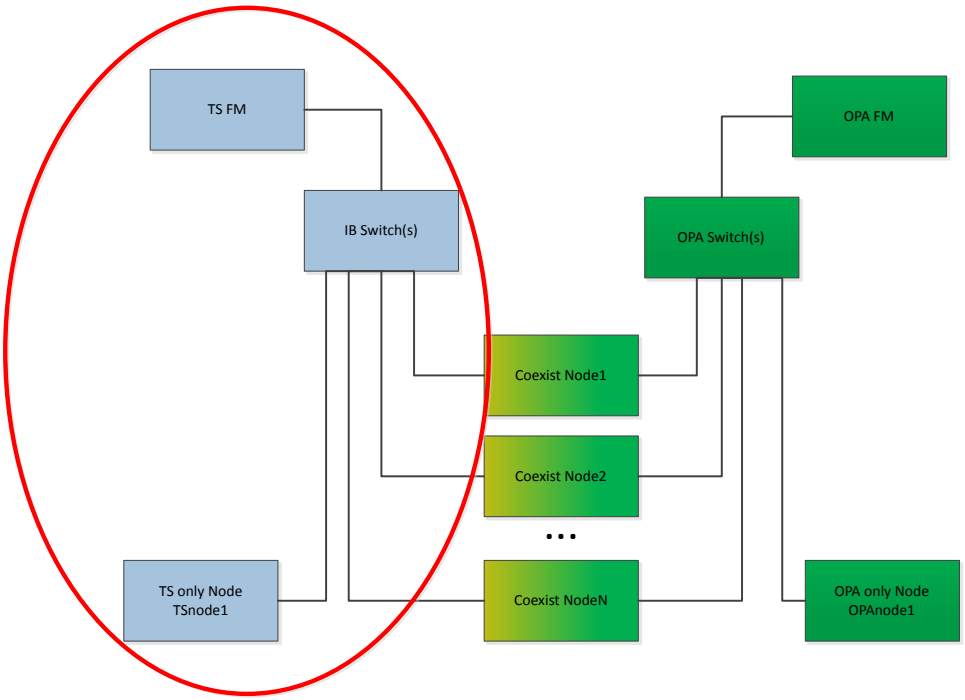
Figure 12. Intel® Omni-Path Node Details



4.5 Intel® True Scale Node Details

Full support for *Intel® True Scale Fabric Suite* software is available on the *Intel® True Scale Fabric Suite Fabric Manager* node. This includes support for OFED+ and all iba_* tools, such as iba_report, iba_top, and iba_port. Intel® True Scale-only compute nodes retain full *Intel® True Scale Fabric Suite* software support. Note, however, that dist_sa is not supported on Intel® True Scale nodes in a coexistence configuration.

Figure 13. Intel® True Scale Node Details

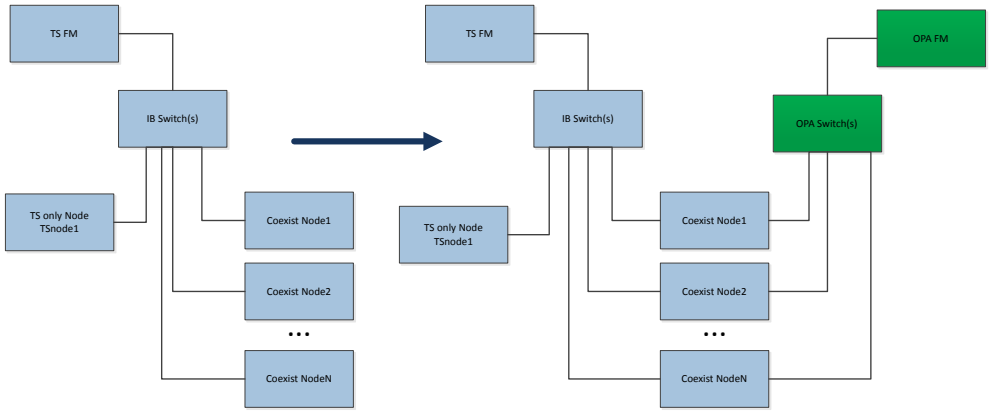


4.6 Installation on An Existing Intel® True Scale Cluster

The following scenario describes the case of adding Intel® Omni-Path to an existing Intel® True Scale cluster.

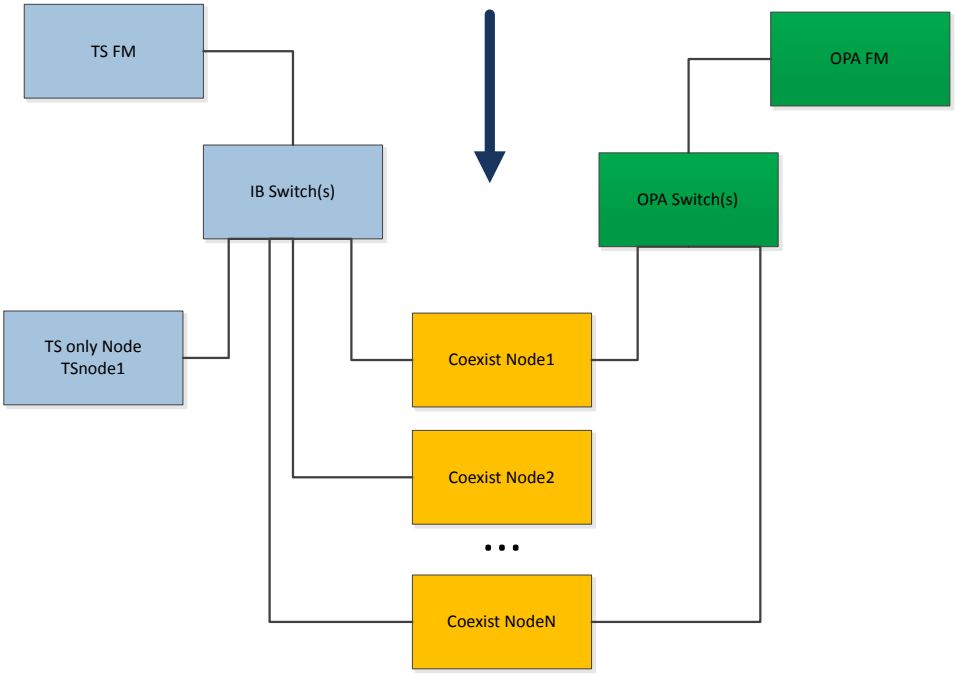
Add an Intel® Omni-Path Fabric Suite Fabric Manager node, Intel® Omni-Path switches, and an Intel® Omni-Path HFI to the Intel® True Scale nodes that are to operate as dual mode nodes. With the hardware in place and connected, install the standard OPA software on the OPA Fabric Manager node. Follow the standard Intel® Omni-Path installation procedures, as described in the *Intel® Omni-Path Fabric Software Installation Guide*.

Figure 14. Adding Intel® Omni-Path Hardware and Software

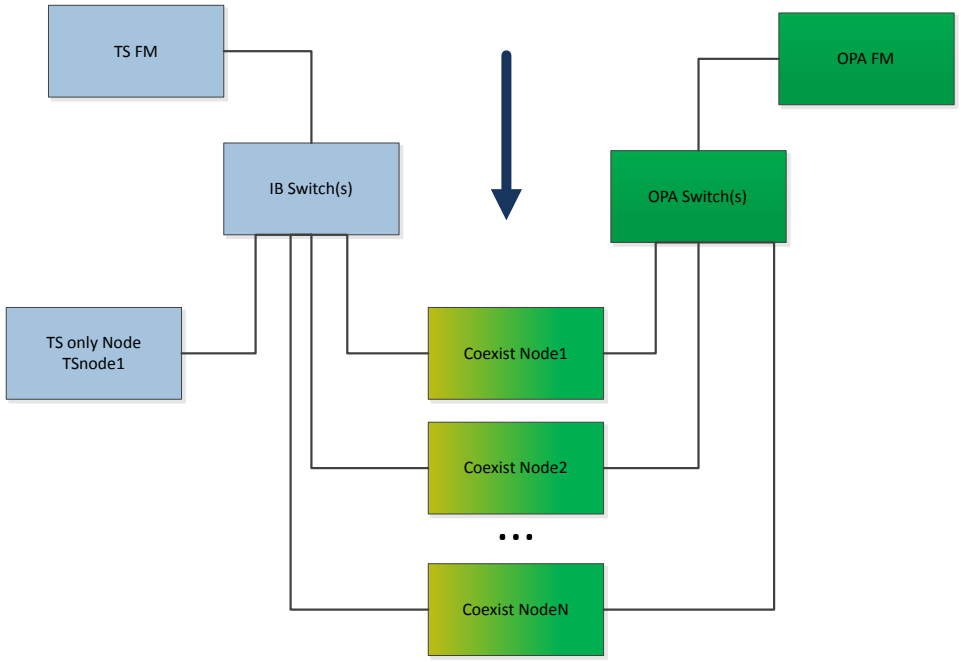


Next, install the Base Distro (RHEL) onto the coexist nodes. Note that Intel® True Scale IFS should *not* be installed on the coexist nodes.

Figure 15. Adding the Base Distro

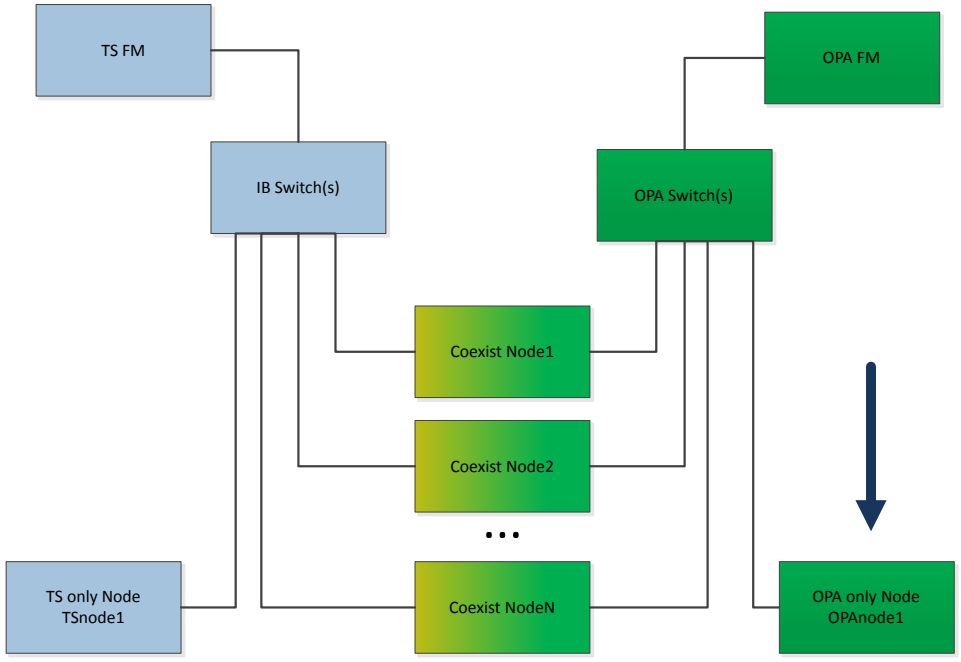


Install Intel® Omni-Path Delta software on the coexist nodes. Follow the standard Omni-Path installation procedures, as described in the *Intel® Omni-Path Fabric Software Installation Guide*.



Add Intel® Omni-Path-only nodes to the Intel® Omni-Path Fabric.

Figure 16. Adding Nodes to Omni-Path Fabric



4.7 PSM2 Compatibility

For more compatibility when running with Open MPI, Open MPI can be recompiled in order to run on the same node at the same time on both TS and OPA hardware. Note that updating Intel MPI to version 5.1 Gold allows Intel® True Scale and Intel® Omni-Path hardware to operate simultaneously without use of the compat package, as does Open MPI.

The following table indicates the compatibility of the MPI libraries with PSM1 and PSM2 versions and Intel® True Scale/Intel® Omni-Path hardware.

Table 1. PSM1 and PSM2 Compatibility Matrix

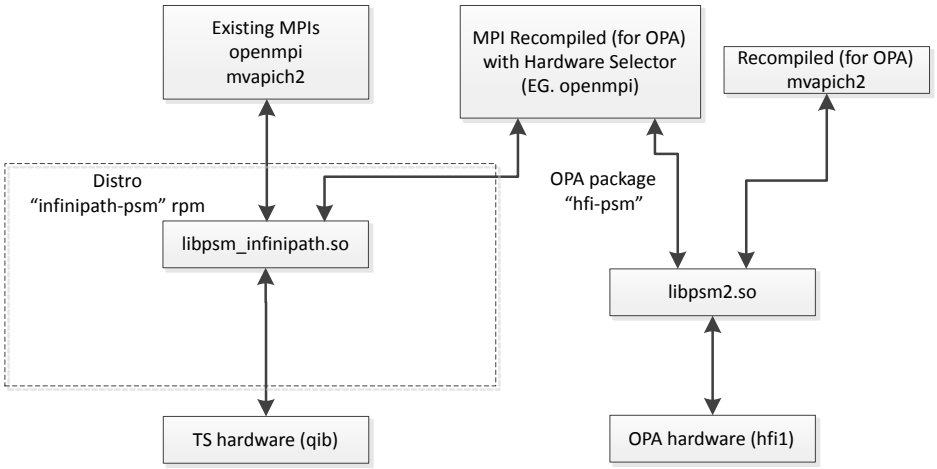
Library	PSM1 (TS Hardware)	PSM1-Compat (OPA Hardware)	PSM2 (OPA Hardware)
Open MPI (recompiled for PSM2)	X		X
MVAPICH2 (recompiled for PSM2)			X
Intel MPI 5.1 Gold (recompiled for PSM2)	X		X
Open MPI (existing)	X	X	
MVAPICH2 (existing)	X	X	
Intel MPI (existing)	X	X	

4.7.1 PSM Standard Configuration

Existing MPIs that were compiled for TS can be linked to an optionally-installed compat RPM via an LD_LIBRARY_PATH directive. The compat RPM contains a library that has a binary-compatible API for linking old MPIs into PSM2. OPA IFS defaults to including new MPIs that are compiled against the new PSM2 library. MPIs such as OpenMPI, which have run time selectors of the underlying hardware, can be used to select the hardware via their mechanisms, as is show in the preceding figure. The following figure shows the linkage when an MPI is run with the LD_LIBRARY_PATH specifying the compat RPM library.

Note: In this environment, even OpenMPI would run over PSM2 if the True Scale hardware is selected.

Figure 17. PSM Standard Configuration



4.7.2 Using the PSM1 Interface on OPA Hardware

The hfi1-psm-compat package provides a recompiled infinipath-psm library placed in `/usr/lib64/psm2-compat`. This package allows existing MPIs to be run on Intel® Omni-Path hardware without being recompiled.

LD_LIBRARY_PATH must be prepended with /usr/lib64/psm2-compat on the mpirun command line, with `/etc/profile.d`, `/etc/profile`, or added to the users shell environment .

Use PSM2 with existing OpenMPI

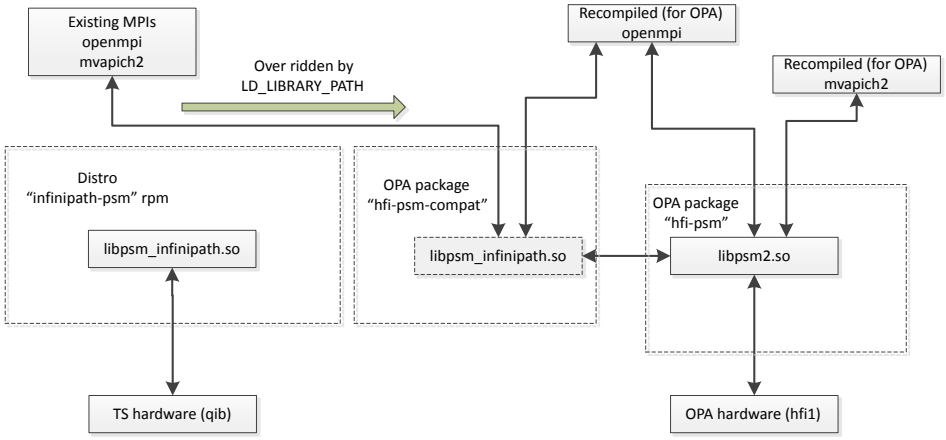
```
mpirun -np 2 -x
LD_LIBRARY_PATH=/usr/lib64/psm2-compat:$LD_LIBRARY_PATH
```

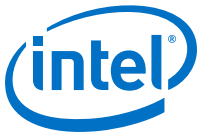
Use PSM2 with existing Intel MPI or MVAPICH2

```
mpirun -np 2 -x LD_LIBRARY_PATH=/usr/lib64/psm2-compat:$LD_LIBRARY_PATH
```

When `LD_LIBRARY_PATH` is not modified, existing MPIs will continue to use the library from the infinipath-psm package and will run on Intel® True Scale hardware.

Figure 18. Overriding LD_LIBRARY_PATH to Run Existing MPIs on OPA Hardware





5.0 Running MPI on Intel HFIs

This section provides information on using the Message-Passing Interface (MPI) on Intel® Omni-Path Host Fabric Interfaces (HFIs). Examples are provided for setting up the user environment, and for compiling and running MPI programs.

5.1 Introduction

The MPI standard is a message-passing library or collection of routines used in distributed-memory parallel programming. It is used in data exchange and task synchronization between processes. The goal of MPI is to provide portability and efficient implementation across different platforms and architectures.

5.1.1 MPIs Packaged with Intel® Omni-Path Fabric Host Software

The high-performance open-source MPIs packaged with Intel OFA Delta in the Intel® Omni-Path Fabric Host Software include Open MPI and MVAPICH2. These MPIs are offered in versions built with the Intel® Omni-Path high-performance Performance Scaled Messaging for Intel® Omni-Path (PSM2) interface and versions built to run over IB Verbs. There are also the commercial MPIs that are not packaged with Intel OFA Delta that make use of the PSM2 application programming interface (API) and run over IB Verbs, including Intel MPI . For more information on other MPIs, see Using Other MPIs.

5.2 Open MPI

Open MPI is an open source MPI implementation from the Open MPI Project. Pre-compiled versions of Open MPI that run over PSM2 and are built with the GCC, PGI, and Intel compilers are available with the Intel download. Open MPI that runs over verbs is also available.

Open MPI can be managed with the mpi-selector utility, as described in Managing MPI Versions with the MPI Selector Utility on page 58.

5.2.1 Installation

Follow the instructions in the *Intel® Omni-Path Fabric Software Installation Guide* for installing Open MPI.

5.2.2 Setup

- When using the `mpi-selector` tool, the necessary `$PATH` and `$LD_LIBRARY_PATH` setup is done.

- When not using the `mpi-selector` tool, put the Open MPI installation directory in the PATH by adding the following to `PATH`:

```
$mpi_home/bin
```

  Where *$mpi_home* is the directory path where Open MPI is installed.

If not using the `mpi-selector`, it is also necessary to set `$LD_LIBRARY_PATH`.

5.2.3 Compiling Open MPI Applications

Intel recommends that you use the included wrapper scripts that invoke the underlying compiler (see Table 2 on page 42).

Table 2. Open MPI Wrapper Scripts

Wrapper Script Name	Language
mpicc	C
mpiCC, mpicxx, or mpic++	C++
mpif77	Fortran 77
mpif90	Fortran 90

To compile your program in C, type the following:

```
$ mpicc mpi_app_name.c -o mpi_app_name
```

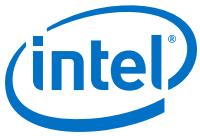
These scripts all provide the command line options listed in Table 3 on page 42.

Table 3. Command Line Options for Scripts

Command	Meaning
man mpicc (mpif90, mpicxx, etc.)	Provides help
-showme	Lists each of the compiling and linking commands that would be called without actually invoking the underlying compiler
-showme:compile	Shows the compile-time flags that would be supplied to the compiler
-showme:link	Shows the linker flags that would be supplied to the compiler for the link phase.

These wrapper scripts pass most options on to the underlying compiler. Use the documentation for the underlying compiler (gcc, icc, pgcc, etc.) to determine what options to use for your application.

Intel strongly encourages using the wrapper compilers instead of attempting to link to the Open MPI libraries manually. This allows the specific implementation of Open MPI to change without forcing changes to linker directives in users' Makefiles.



5.2.4 Create the mpi_hosts File

Create an MPI hosts file in the same working directory where Open MPI is installed. The MPI hosts file contains the host names of the nodes in your cluster that run the examples, with one host name per line. Name this file `mpi_hosts`.

Details on the `mpi_hosts` file, including information regarding format options, can be found in mpi_hosts File Details on page 48.

5.2.5 Running Open MPI Applications

The `mpi-selector --list` command evokes the MPI Selector, and provides the following list of MPI options, including a number of Open MPI choices.

- mvapich2_gcc-X.X
- mvapich2_gcc_hfi-X.X
- mvapich2_intel_hfi-X.X
- mvapich2_pgi_hfi-X.X
- openmpi_gcc-X.X.X
- openmpi_gcc_hfi-X.X.X
- openmpi_intel_hfi-X.X.X
- openmpi_pgi_hfi-X.X.X

The first choice will use verbs by default, and any with the `_hfi` string will use PSM for Intel® Omni-Path (PSM2) by default. If you chose `openmpi_gcc_hfi-X.X.X`, for example, the following simple mpirun command would run using PSM2:

```
$ mpirun -np 4 -machinefile mpi_hosts mpi_app_name
```

To run over IB Verbs instead of the default PSM2 transport in `openmpi_gcc_hfi-X.X.X`, use this `mpirun` command line:

```
$ mpirun -np 4 -machinefile mpi_hosts -mca btl sm,openib,self -mca mtl ^psm,psm2
mpi_app_name
```

The following command enables shared memory:

```
-mca btl sm
```

The following command enables openib transport and communication to self:

```
-mca btl openib,self
```

The following command disables both PSM and PSM2 transport:

```
-mca mtl ^psm,^psm2
```

In these commands, `btl` stands for *byte transport layer* and `mtl` for *matching transport layer*.

PSM2 transport works in terms of MPI messages. OpenIB transport works in terms of byte streams.

Alternatively, you can use Open MPI with a sockets transport running over IPoIB, for example:

```
$ mpirun -np 4 -machinefile mpi_hosts -mca btl sm -mca btl tcp,self --mca
btl_tcp_if_exclude eth0 -mca btl_tcp_if_include ib0 -mca mtl ^psm2 <mpi_app_name>
```

Note that `eth0` and `psm` are excluded, while `ib0` is included. These instructions may need to be adjusted for your interface names.

Note that in Open MPI, `machinefile` is also known as the `hostfile`.

5.2.6 Further Information on Open MPI

For more information about Open MPI, see:

http://www.open-mpi.org/

http://www.open-mpi.org/faq

5.2.7 Configuring MPI Programs for Open MPI

When configuring an MPI program (generating header files and/or Makefiles) for Open MPI, you usually need to specify `mpicc`, `mpicxx`, and so on as the compiler, rather than `gcc`, `g__`, etc.

Specifying the compiler is typically done with commands similar to the following, assuming that you are using `sh` or `bash` as the shell:

```
$ export CC=mpicc
$ export CXX=mpicxx
$ export F77=mpif77
$ export F90=mpif90
```

The shell variables will vary with the program being configured. The following examples show frequently used variable names. If you use `csh`, use commands similar to the following:

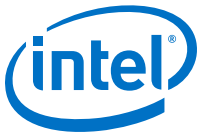
```
$ setenv CC mpicc
```

You may need to pass arguments to `configure` directly, for example:

```
$ ./configure -cc=mpicc -fc=mpif77 -c__=mpicxx -c__linker=mpicxx
```

You may also need to edit a Makefile to achieve this result, adding lines similar to:

```
CC=mpicc
F77=mpif77
F90=mpif90
CXX=mpicxx
```



In some cases, the configuration process may specify the linker. Intel recommends that the linker be specified as `mpicc`, `mpif90`, etc. in these cases. This specification automatically includes the correct flags and libraries, rather than trying to configure to pass the flags and libraries explicitly. For example:

```
LD=mpif90
```

These scripts pass appropriate options to the various compiler passes to include header files, required libraries, etc. While the same effect can be achieved by passing the arguments explicitly as flags, the required arguments may vary from release to release, so it is good practice to use the provided scripts.

5.2.8 To Use Another Compiler

Open MPI and all other Message Passing Interfaces (MPIs) that run on Intel® Omni-Path support a number of compilers, in addition to the default GNU Compiler Collection (GCC, including gcc, g__ and gfortran) versions 3.3 and later. These include the PGI 8.0, through 11.9; and Intel 9.*x*, 10.1, 11.*x*, and 12.*x*.

The easiest way to use other compilers with any MPI that comes with Intel OFED_ is to use `mpi-selector` to change the selected MPI/compiler combination, see Managing MPI Versions with the MPI Selector Utility on page 58.

These compilers can be invoked on the command line by passing options to the wrapper scripts. Command line options override environment variables, if set.

Table 4 on page 45 and Table 5 on page 45 show the options for each of the compilers.

In each case, `.....` stands for the remaining options to the `mpicxx` script, the options to the compiler in question, and the names of the files that it operates.

Table 4. Intel Compilers

Compiler	Command
C	`$ mpicc -cc=icc .....`
C++	`$ mpicc -CC=icpc`
Fortran 77	`$ mpif77 -fc=ifort .....`
Fortran 90/95	`$ mpif90 -f90=ifort .....` `$ mpif95 -f95=ifort .....`

Table 5. Portland Group (PGI) Compilers

Compiler	Command
C	`mpicc -cc=pgcc .....`
C++	`mpicc -CC=pgCC`
Fortran 77	`mpif77 -fc=pgf77 .....`
Fortran 90/95	`mpif90 -f90=pgf90 .....` `mpif95 -f95=pgf95 .....`

Also, use `mpif77`, `mpif90`, or `mpif95` for linking; otherwise, `.true.` may have the wrong value.

If you are not using the provided scripts for linking, link a sample program using the `–show` option as a test (without the actual build) to see what libraries to add to your link line. Some examples of the using the PGI compilers follow.

For Fortran 90 programs:

```
$ mpif90 -f90=pgf90 -show pi3f90.f90 -o pi3f90
pgf90 -I/usr/include/mpich/pgi5/x86_64 -c -I/usr/include
pi3f90.f90 -c
pgf90 pi3f90.o -o pi3f90 -lmpichf90 -lmpich -lmpichabiglue_pgi5
```

Fortran 95 programs will be similar to the above.

For C programs:

```
$ mpicc -cc=pgcc -show cpi.c
pgcc -c cpi.c
pgcc cpi.o -lmpich -lpgftnrtl -lmpichabiglue_pgi5
```

5.2.8.1 Compiler and Linker Variables

When you use environment variables (e.g., `$MPICH_CC`) to select the compiler `mpicc` (and others) will use, the scripts will also set the matching linker variable (for example, `$MPICH_CLINKER`), if it is not already set. When both the environment variable and command line options are used (`–cc=gcc`), the command line variable is used.

When both the compiler and linker variables are set, and they do not match for the compiler you are using, the MPI program may fail to link; or, if it links, it may not execute correctly.

5.2.9 Process Allocation

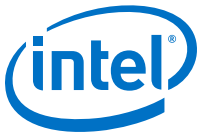
MPI ranks are processes that communicate through the PSM2 library (for best performance from the Intel® Omni-Path Fabric Suite) to get access to the HFI. We refer to these MPI ranks as PSM2 processes.

Normally MPI jobs are run with each node process (rank) being associated with a dedicated Intel HFI *hardware context* that is mapped to a CPU.

If the number of node processes is greater than the available number of hardware contexts, *software context sharing* increases the number of node programs that can be run. Each HFI supports 8 software contexts per hardware context, so up to 8 MPI processes (from the same MPI job) can share that hardware context. There is a small additional overhead for each shared context.

For the Intel® Omni-Path Host Fabric Interfaces, the maximum number of contexts available is:

- Up to 160 user hardware contexts available per HFI
- Up to 8*160 (1280) MPI ranks (processes) that can be run per HFI when the Software Context Sharing is Enabled (default mode)



The number of HW contexts available can be set by the driver, and depends on the number of processor cores in the system, including hyperthreaded cores.

The default hardware context/CPU mappings can be changed on the Intel® Omni-Path HFIs.

Optimal performance may be achieved by ensuring that the PSM2 process affinity is assigned to the CPU of the Non-Uniform Memory Access (NUMA) node local to the HFI that it is operating.

When running MPI jobs in a batch system environment where multiple jobs may be running simultaneously, it may be useful to restrict the number of Intel® Omni-Path contexts that are made available on each node running an MPI job. See Restricting Intel® Omni-PathIntel® Omni-Path Hardware Contexts in a Batch Environment.

Errors that may occur with context sharing are covered in Context Sharing Error Messages on page 48.

There are multiple ways of specifying how processes are allocated. You can use the `mpi_hosts` file, the `-np` and `-ppn` options with `mpirun`, and the `MPI_NPROCS` and `PSM2_SHAREDCONTEXTS_MAX` environment variables. How these all are set are covered later in this document.

5.2.9.1 Restricting Intel® Omni-PathIntel® Omni-Path Hardware Contexts in a Batch Environment

If required for resource sharing between multiple jobs in batch systems, you can restrict the number of Intel® Omni-Path hardware contexts that are made available on each node of an MPI job by setting that number in the `PSM2_SHAREDCONTEXTS_MAX` environment variable.

Note: Before enabling hardware context sharing, first ensure that the maximum number of hardware contexts are enabled and used. If that is done, most use cases will be satisfied.

For example, consider that the maximum 160 hardware contexts are enabled and that the driver configuration consumes 16 of these, leaving 144 for user contexts. If you are running two different jobs on nodes using Intel® OP HFIs, PSM2_SHAREDCONTEXTS_MAX can be set to `72` for each job (both of the jobs that want to share a node would have to set PSM2_SHAREDCONTEXTS_MAX=72). Each job would then have at most half of the available hardware contexts.

Note: MPIs use different methods for propagating environment variables to the nodes used for the job. See Virtual Fabric Support in PSM2 for examples. Open MPI will automatically propagate PSM2 environment variables.

Setting `PSM2_SHAREDCONTEXTS_MAX=16` as a cluster-wide default would unnecessarily penalize nodes that are dedicated to running single jobs. Intel recommends that a per-node setting, or some level of coordination with the job scheduler with setting the environment variable, should be used.

`PSM2_RANKS_PER_CONTEXT` provides an alternate way of specifying how PSM2 should use contexts. The variable is the number of ranks that will share each hardware context. The supported values are 1 through 8, where 1 is no context sharing, 2 is 2-way context sharing, 3 is 3-way context sharing, and so forth, up to 8 being 8-way context sharing. The same value of `PSM2_RANKS_PER_CONTEXT` must be

used for all ranks on a node, and typically, you would use the same value for all nodes in that job. Either PSM2_RANKS_PER_CONTEXT or PSM2_SHAREDCONTEXTS_MAX would be used in a particular job, but not both. If both are used and the settings are incompatible, then PSM will report an error and the job will fail to start up.

5.2.9.2 Context Sharing Error Messages

The error message when the context limit is exceeded is:

```
No free OPA contexts available on /dev/hfi1
```

This message appears when the application starts.

Error messages related to contexts may also be generated by mpirun. For example:

```
PSM2 found 0 available contexts on OPA device
```

The most likely cause is that the cluster has processes using all the available PSM2 contexts. Clean up these processes before restarting the job.

5.2.9.3 Running in Shared Memory Mode

Open MPI supports running exclusively in shared memory mode; no Intel® Omni-Path Host Fabric Interface is required for this mode of operation. This mode is used for running applications on a single node rather than on a cluster of nodes.

To add pre-built applications (benchmarks), add /usr/mpi/gcc/openmpi-X.X.X-hfi/tests/osu_benchmarks-X.X.X to your PATH (or if you installed the MPI in another location: add *$MPI_HOME*/tests/osu_benchmarks-X.X.X to your PATH).

To enable shared memory mode, use a single node in the mpi_hosts file. For example, if the file were named onehost and it is in the working directory, the following would be entered:

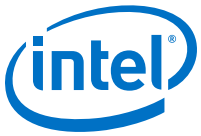
```
$ cat /tmp/onehost
idev-64 slots=8
```

Enabling the shared memory mode as previously described uses a feature of Open-MPI host files to list the number of slots, which is the number of possible MPI processes (aka ranks) that you want to run on the node. Typically this is set equal to the number of processor cores on the node. A hostfile with 8 lines containing 'idev-64' would function identically.

You can use this hostfile and run $ mpirun -np=2 -hostfile onehost osu_latency to measure MPI latency between two cores on the same host using shared-memory, or $ mpirun -np=2 -hostfile onehost osu_bw to measure MPI unidirectional bandwidth using shared memory.

5.2.10 mpi_hosts File Details

As noted in Create the mpi_hosts File on page 43, a hostfile (also called *machines file, nodefile*, or *hostsfile*) has been created in your current working directory. This file names the nodes that the node programs may run.



The two supported formats for the `hostfile` are:

```
hostname1
hostname2
...
```

or

```
hostname1 slots=process_count
hostname2 slots=process_count
...
```

In the first format, if the `-np` count (number of processes to spawn in the `mpirun` command) is greater than the number of lines in the machine file, the hostnames will be repeated (in order) as many times as necessary for the requested number of node programs.

Also in the first format, if the `-np` count is less than the number of lines in the machine file, `mpirun` still processes the entire file and tries to pack processes to use as few hosts as possible in the `hostfile`. This is a different behavior than MVAPICH2.

In the second format, `process_count` can be different for each host, and is normally the number of available cores on the node. When not specified, the default value is one. The value of `process_count` determines how many node processes will be started on that host before using the next entry in the `hostfile` file. When the full `hostfile` is processed, and there are additional processes requested, processing starts again at the start of the file.

It is generally recommended to use the second format and various command line options to schedule the placement of processes to nodes and cores. For example, the `mpirun` option `-npernode` can be used to specify (similar to the Intel MPI option `-ppn`) how many processes should be scheduled on each node on each pass through the `hostfile`. In the case of nodes with 8 cores each, if the `hostfile` line is specified as `hostname1 slots=8 max-slots=8`, then Open MPI will assign a maximum of 8 processes to the node and there can be no over-subscription of the 8 cores.

There are several alternative ways of specifying the `hostfile`:

- The command line option `-hostfile` can be used as shown in the following command line:

  ```
  $mpirun -np n -hostfile mpi_hosts [other options] program-name
  ```

  or `-machinefile` is a synonym for `-hostfile`. In this case, if the named file cannot be opened, the MPI job fails.

  An alternate mechanism to `-hostfile` for specifying hosts is the `-H`, `-hosts`, or `--host` followed by a host list. The host list can follow one of the following examples:

  ```
  host-01, or
  host-01,host-02,host-04,host-06,host-07,host-08
  ```

- In the absence of the -hostfile option, the -H option, mpirun uses the file ./mpi_hosts, if it exists.

  If you are working in the context of a batch queuing system, it may provide a job submission script that generates an appropriate mpi_hosts file. More details about how to schedule processes to nodes with Open MPI refer to the Open MPI website:

  http://www.open-mpi.org/faq/?category=running#mpirun-scheduling

5.2.11 Using Open MPI's mpirun

The script mpirun is a front end program that starts a parallel MPI job on a set of nodes in a cluster. mpirun may be run on any x86_64 machine inside or outside the cluster, as long as it is on a supported Linux distribution, and has TCP connectivity to all Intel® Omni-Path cluster machines to be used in a job.

The script starts, monitors, and terminates the node processes. mpirun uses ssh (secure shell) to log in to individual cluster machines and prints any messages that the node process prints on stdout or stderr, on the terminal where mpirun is invoked.

The general syntax is:

```
$ mpirun [mpirun_options...] program-name [program options]
```

program-name is usually the pathname to the executable MPI program. When the MPI program resides in the current directory and the current directory is not in your search path, then *program-name* must begin with '*./*', for example:

```
./program-name
```

Unless you want to run only one instance of the program, use the -np option, for example:

```
$ mpirun -np n [other options] program-name
```

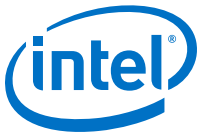
This option spawns *n* instances of *program-name*. These instances are called *node processes*.

Generally, mpirun tries to distribute the specified number of processes evenly among the nodes listed in the hostfile. However, if the number of processes exceeds the number of nodes listed in the hostfile, then some nodes will be assigned more than one instance of the process.

Another command line option, -npernode, instructs mpirun to assign a fixed number *p* of node processes to each node, as it distributes *n* instances among the nodes:

```
$ mpirun -np n -npernode p -hostfile mpi_hosts program-name
```

This option overrides the slots=*process_count* specifications, if any, in the lines of the mpi_hosts file. As a general rule, mpirun distributes the *n* node processes among the nodes without exceeding, on any node, the maximum number of instances



specified by the `slots=process_count` option. The value of the `slots=process_count` option is specified by either the `-npernode` command line option or in the `mpi_hosts` file.

Typically, the number of node processes should not be larger than the number of processor cores, at least not for compute-bound programs.

This option specifies the number of processes to spawn. If this option is not set, then environment variable `MPI_NPROCS` is checked. If `MPI_NPROCS` is not set, the default is to determine the number of processes based on the number of hosts in the `hostfile` or the list of hosts `-H` or `--host`.

```
-npernode processes-per-node
```

This option creates up to the specified number of *processes per node*.

Each node process is started as a process on one node. While a node process may fork child processes, the children themselves must not call MPI functions.

There are many more `mpirun` options for scheduling where the processes get assigned to nodes. See `man mpirun` for details.

`mpirun` monitors the parallel MPI job, terminating when all the node processes in that job exit normally, or if any of them terminates abnormally.

Killing the `mpirun` program kills all the processes in the job. Use CTRL_C to kill `mpirun`.

5.2.12 Console I/O in Open MPI Programs

Open MPI directs UNIX standard input to `/dev/null` on all processes except the `MPI_COMM_WORLD` rank `0` process. The `MPI_COMM_WORLD` rank `0` process inherits standard input from `mpirun`.

Note: The node that invoked `mpirun` need not be the same as the node where the `MPI_COMM_WORLD` rank `0` process resides. Open MPI handles the redirection of `mpirun`'s standard input to the rank `0` process.

Open MPI directs UNIX standard output and error from remote nodes to the node that invoked `mpirun` and prints it on the standard output/error of `mpirun`. Local processes inherit the standard output/error of `mpirun` and transfer to it directly.

It is possible to redirect standard I/O for Open MPI applications by using the typical shell redirection procedure on `mpirun`.

```
$ mpirun -np 2 my_app < my_input > my_output
```

Note that in this example only the `MPI_COMM_WORLD` rank `0` process will receive the stream from `my_input` on stdin. The stdin on all the other nodes will be tied to `/dev/null`. However, the stdout from all nodes will be collected into the `my_output` file.

5.2.13 Environment for Node Processes

The following information can be found in the Open MPI man page and is repeated here for easy of use.

The following are the sections of the Environment for Node Processes:

5.2.13.1 Remote Execution

Open MPI requires that the `PATH` environment variable be set to find executables on remote nodes (this is typically only necessary in or ssh-based environments – batch/scheduled environments typically copy the current environment to the execution of remote jobs, so if the current environment has `PATH` and/or `LD_LIBRARY_PATH` set properly, the remote nodes will also have it set properly). If Open MPI was compiled with shared library support, and the shared libraries are not in the standard path, it may also be necessary to have the `LD_LIBRARY_PATH` environment variable set on remote nodes as well (especially to find the shared libraries required to run user MPI applications).

It is not always desirable or possible to edit shell startup files to set `PATH` and/or `LD_LIBRARY_PATH`. The `--prefix` option is provided for some simple configurations where this is not possible.

The `--prefix` option takes a single argument: the base directory on the remote node where Open MPI is installed. Open MPI will use this directory to set the remote `PATH` and `LD_LIBRARY_PATH` before executing any Open MPI or user applications. This allows running Open MPI jobs without having pre-configured the `PATH` and `LD_LIBRARY_PATH` on the remote nodes.

Open MPI adds the base-name of the current node's `bindir` (the directory where Open MPI's executables are installed) to the prefix and uses that to set the `PATH` on the remote node. Similarly, Open MPI adds the base-name of the current node's `libdir` (the directory where Open MPI's libraries are installed) to the prefix and uses that to set the `LD_LIBRARY_PATH` on the remote node. For example:

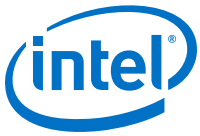
```
Local bindir:  /local/node/directory/bin
Local libdir:  /local/node/directory/lib64
```

If the following command line is used:

```
% mpirun --prefix /remote/node/directory
```

Open MPI will add `/remote/node/directory/bin` to the `PATH` and `/remote/node/directory/lib64` to the `LD_LIBRARY_PATH` on the remote node before attempting to execute anything.

Note that `--prefix` can be set on a node basis, allowing for different values for different nodes.



The `--prefix` option is not sufficient if the installation paths on the remote node are different than the local node (for example, if `/lib` is used on the local node but `/lib64` is used on the remote node), or if the installation paths are something other than a subdirectory under a common prefix.

Note that executing `mpirun` using an absolute pathname is equivalent to specifying `--prefix` without the last subdirectory in the absolute pathname to `mpirun`. For example:

```
% /usr/local/bin/mpirun ...
```

is equivalent to

```
% mpirun --prefix /usr/local
```

5.2.13.2 Exported Environment Variables

All environment variables that are named in the form `OMPI_*` will automatically be exported to new processes on the local and remote nodes. The `-x` option to `mpirun` can be used to export specific environment variables to the new processes. While the syntax of the `-x` option allows the definition of new variables. Note that the parser for this option is currently not very sophisticated, it does not understand quoted values. Users are advised to set variables in the environment and use `-x` to export them, not to define them.

5.2.13.3 Setting MCA Parameters

The `-mca` switch allows the passing of parameters to various Modular Component Architecture (MCA) modules. MCA modules have direct impact on MPI programs because they allow tunable parameters to be set at run time (such as which BTL communication device driver to use, what parameters to pass to that BTL, and so on.).

The `-mca` switch takes two arguments: *key* and *value*. The *key* argument generally specifies which MCA module will receive the value. For example, the *`key`* `btl` is used to select which BTL to be used for transporting MPI messages. The *value* argument is the value that is passed. For example:

```
mpirun -mca btl tcp,self -np 1 foo
```

Tells Open MPI to use the `tcp` and `self` BTLs, and to run a single copy of `foo` an allocated node.

```
mpirun -mca btl,self -np 1 foo
```

Tells Open MPI to use the `self` BTL, and to run a single copy of `foo` an allocated node.

The `-mca` switch can be used multiple times to specify different *key* and/or *value* arguments. If the same *key* is specified more than once, the *values* are concatenated with a comma (",") separating them.

Note that the -mca switch is simply a shortcut for setting environment variables. The same effect may be accomplished by setting corresponding environment variables before running mpirun. The form of the environment variables that Open MPI sets is:

```
OMPI_MCA_key=value
```

Thus, the -mca switch overrides any previously set environment variables. The -mca settings similarly override MCA parameters set in these two files, which are searched (in order):

1. $HOME/.openmpi/mca-params.conf: The user-supplied set of values takes the highest precedence.
2. $prefix/etc/openmpi-mca-params.conf: The system-supplied set of values has a lower precedence.

5.2.14 Environment Variables

The Chapter titled Environmental Variables, in *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide*, contains a summary of the environment variables that are relevant to any PSM including Open MPI. Open MPI provides its own environmental variables that may be more relevant for the MPI programmer or script writer, because these variables are only active after the mpirun command has been issued and while the MPI processes are active. See the Open MPI documentation for information on these Open MPI environmental variables.

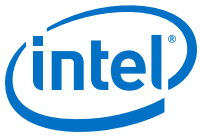
5.3 Open MPI and Hybrid MPI/OpenMP Applications

Open MPI supports hybrid MPI/OpenMP applications, provided that MPI routines are called only by the master OpenMP thread. This application is called the *funneled thread model*. Instead of MPI_Init/MPI_INIT (for C/C__ and Fortran respectively), the program can call MPI_Init_thread/MPI_INIT_THREAD to determine the level of thread support, and the value MPI_THREAD_FUNNELED will be returned.

To use this feature, the application must be compiled with both OpenMP and MPI code enabled. To do this, use the -openmp or -mp flag (depending on your compiler) on the mpicc compile line.

As mentioned previously, MPI routines can be called only by the master OpenMP thread. The hybrid executable is executed as usual using mpirun, but typically only one MPI process is run per node and the OpenMP library will create additional threads to utilize all CPUs on that node. If there are sufficient CPUs on a node, you may want to run multiple MPI processes and multiple OpenMP threads per node.

The number of OpenMP threads is typically controlled by the OMP_NUM_THREADS environment variable in the .bashrc file. (OMP_NUM_THREADS is used by other compilers' OpenMP products, but is not an Open MPI environment variable.) Use this variable to adjust the split between MPI processes and OpenMP threads. Usually, the number of MPI processes (per node) times the number of OpenMP threads will be set to match the number of CPUs per node. An example case would be a node with four CPUs, running one MPI process and four OpenMP threads. In this case, OMP_NUM_THREADS is set to four. OMP_NUM_THREADS is on a per-node basis.



See Environment for Node Processes on page 52 for information on setting environment variables.

Note: With Open MPI, and other PSM2-enabled MPIs, you will typically want to turn off PSM2's CPU affinity controls so that the OpenMP threads spawned by an MPI process are not constrained to stay on the CPU core of that process, causing over-subscription of that CPU. Accomplish this using the `HFI_NO_CPUAFFINITY=1` setting as follows:

```
OMP_NUM_THREADS=8 (typically set in the ~/.bashrc file)
mprun -np 2 -H host1,host2 -x HFI_NO_CPUAFFINITY=1 ./hybrid_app
```

Note: In this case, typically there would be 8 or more CPU cores on the host1 and host2 nodes, and this job would run on a total of 16 threads, 8 on each node. You can use 'top' and then '1' to monitor that load is distributed to 8 different CPU cores in this case.

Note: [Both the `OMP_NUM_THREADS` and `HFI_NO_CPUAFFINITY` can be set in `.bashrc` or both on the command line after `-x` options.]

Note: When there are more threads than CPUs, both MPI and OpenMP performance can be significantly degraded due to over-subscription of the CPUs

5.4 Debugging MPI Programs

Debugging parallel programs are substantially more difficult than debugging serial programs. Thoroughly debugging the serial parts of your code before parallelizing is good programming practice.

Refer to MPI Errors on page 55 and Using Debuggers on page 55 for information on debugging MPI programs.

5.4.1 MPI Errors

Almost all MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error code; either as the function return value in C functions or as the last argument in a Fortran subroutine call. Before the value is returned, the current MPI error handler is called. By default, this error handler aborts the MPI job. Therefore, you can get information about MPI exceptions in your code by providing your own handler for `MPI_ERRORS_RETURN`. See the man page for the `MPI_Errhandler_set` for details.

See the standard MPI documentation referenced in tsfofed_References for details on the MPI error codes.

5.4.2 Using Debuggers

- See http://www.open-mpi.org/faq/?category=debugging for details on debugging with Open MPI.

Note: The TotalView* debugger can be used with the Open MPI supplied in this release. Consult the TotalView* documentation for more information:

- http://www.open-mpi.org/faq/?category=running#run-with-tv

6.0 Using Other MPIs

This section provides information on using other Message Passing Interface (MPI) implementations. Detailed information on using Open MPI is provided in Running MPI on Intel HCAs, and will be covered in this section in the context of choosing among multiple MPIs or in tables which compare the multiple MPIs available.

6.1 Introduction

Support for multiple high-performance MPI implementations has been added. Most implementations run over both PSM2 and OpenFabrics Verbs (see the following table). Use the `mpi-selector-menu` command to choose which MPI to use, as described in Managing MPI Versions with the MPI Selector Utility on page 58.

Table 6. Other Supported MPI Implementations

MPI Implementation	Runs Over	Compiled With	Comments
Open MPI	PSM Verbs	GCC, Intel, PGI	Provides some MPI-2 functionality (one-sided operations and dynamic processes). Available as part of the Intel download. Can be managed by `mpi-selector`.
MVAPICH2	PSM Verbs	GCC, Intel, PGI	Provides MPI-2 Functionality. Can be managed by `MPI-Selector`.
Intel MPI	TMI/PSM	GCC (default)	Provides MPI-1 and MPI-2 functionality. Available for purchase from Intel.

Note: These MPI implementations run on multiple interconnects, and have their own mechanisms for selecting the interconnect that runs on. Basic information about using these MPIs is provided in this section. However, for more detailed information, see the documentation provided with the version of MPI that you want to use.

6.2 Installed Layout

By default, the MVAPICH2, and Open MPI are installed in the following directory tree:

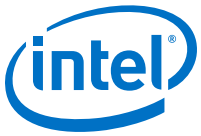
`/usr/mpi/$compiler/$mpi-mpi_version`

The Intel-supplied MPIs are precompiled with the GCC, PGI, and the Intel compilers will also have `-hif` appended after the MPI version number.

For example:

`/usr/mpi/gcc/openmpi-VERSION-hfi`

If a prefixed installation location is used, `/usr` is replaced by `$prefix`.



The following examples assume that the default path for each MPI implementation to `mpirun` is:

`/usr/mpi/$compiler/$mpi/bin/mpirun`

Again, `/usr` may be replaced by *`$prefix`*. This path is sometimes referred to as *`$mpi_home`*`/bin/mpirun` in the following sections.

See documentation for Intel MPI, and Platform MPI for their default installation directories.

6.3 Open MPI

Open MPI is an open source MPI-2 implementation from the Open MPI Project. Pre-compiled versions of Open MPI that run over PSM2 and are built with the GCC, PGI, and Intel compilers are available with the Intel download.

6.4 MVAPICH2

Supported, pre-compiled versions of MVAPICH2 built with the GNU, PGI, and Intel compilers, and that run over PSM, are included with the Intel download.

MVAPICH2 that runs over Verbs and is pre-compiled with the GNU compiler is also available.

MVAPICH2 can be managed with the MPI Selector utility, as described in Managing MPI Versions with the MPI Selector Utility on page 58.

6.4.1 Compiling MVAPICH2 Applications

As with Open MPI, Intel recommends that you use the included wrapper scripts that invoke the underlying compiler (see Table 7 on page 57).

Table 7. MVAPICH2 Wrapper Scripts

Wrapper Script Name	Language
`mpicc`	C
`mpiCC, mpicxx`	C++
`mpif77`	Fortran 77
`mpif90`	Fortran 90

To compile your program in C, type:

```
$ mpicc mpi_app_name.c -o mpi_app_name
```

To check the default configuration for the installation, check the following file:

```
/usr/mpi/$compiler/$mpi/etc/mvapich.conf
```

6.4.2 Running MVAPICH2 Applications

By default, the MVAPICH2 options in mpi-selector with 'hfi' as part of their name run over PSM once it is installed.

Here is an example of a simple `mpirun` command running with four processes:

```
$ mpirun -np 4 -hostfile mpihosts ./mpi_app_name
```

6.4.3 Further Information on MVAPICH2

For more information about MVAPICH2, see:

http://mvapich.cse.ohio-state.edu/support/mvapich2-1.7-quick-start.html

or for more detail:

http://mvapich.cse.ohio-state.edu/support/mvapich2-1.7rc2_user_guide.pdf

6.5 Managing MPI Versions with the MPI Selector Utility

When multiple MPI implementations have been installed on the cluster, you can use the MPI Selector utility to switch between them.

The MPI Selector is a utility that is installed as a part of Intel® Omni-Path Fabric Host Software. Its basic functions include:

- Listing MPI implementations that have registered with the utility
- Setting a default MPI to use (per user or site wide)
- Unsetting a default MPI to use (per user or site wide)
- Querying the current default MPI

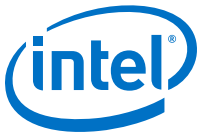
Following is an example for listing and selecting an MPI:

```
$ mpi-selector --list
mvapich2_gcc-X.X
mvapich2_gcc_hfi-X.X
mvapich2_intel_hfi-X.X
mvapich2_pgi_hfi-X.X
openmpi_gcc-X.X.X
openmpi_gcc_hfi-X.X.X
openmpi_intel_hfi-X.X.X
openmpi_pgi_hfi-X.X.X
```

The new default takes effect in the next shell that is started. See the MPI Selector man page for more information.

Each MPI registers itself with MPI Selector, and provides shell scripts mpivar.sh and mpivars.sh scripts that can be found in $prefix/mpi/<COMPILER>/<MPI>/bin directories.

For all non-GNU compilers that are installed outside standard Linux search paths, set up the paths so that compiler binaries and runtime libraries can be resolved. For example, set `LD_LIBRARY_PATH`, both in your local environment and in an rc file



(such as `.mpirunrc`, `.bashrc`, or `.cshrc`), are invoked on remote nodes. See Environment for Node Programs and Compiler and Linker Variables for information on setting up the environment for information on setting the run-time library path.

Note: The Intel-compiled versions require that the Intel compiler be installed and that paths to the Intel compiler runtime libraries be resolvable from the user's environment.

6.6 Intel MPI

The supported version of Intel MPI is included in this release of Intel® Omni-Path Software.

6.6.1 Installation and Setup

Follow the instructions for download and installation of Intel MPI from the Intel web site. The following subsections provide setup instructions for the Intel MPI:

6.6.1.1 Set Up MPI Over TMI

Intel MPI can be run over Tag Matching Interface (TMI)

1. Make sure that the TMI PSM provider is installed on every node and all nodes have the same version installed. The TMI is supplied with the Intel MPI distribution. It can be installed either with the Intel® Omni-Path Fabric Host Software installation or using the `rpm` files. For example:

```
$ rpm -qa | grep tmi
tmi-1.0-1
```

2. Verify that there is a `/etc/tmi.conf` file. It should be installed when installing the TMI. The file `tmi.conf` contains a list of TMI PSM providers. In particular it must contain an entry for the PSM2 provider in a form similar to:

```
psm2 X.X libtmip_psm2.so " " # Comments OK
```

6.6.1.2 Setup

A 64 bit version of Intel MPI is included in the Intel® Omni-Path Software distribution. 32 bit Intel MPI is not supported.

To launch MPI jobs, the Intel installation directory must be included in `PATH` and `LD_LIBRARY_PATH`.

- When using `sh` for launching MPI jobs, run the following command:

```
$ source $prefix/bin64/mpivars.sh
```

- When using `csh` for launching MPI jobs, run the following command:

```
$ source $prefix/bin64/mpivars.csh
```

6.6.1.3 Compiling Intel MPI Applications

As with Open MPI, Intel recommended that you use the included wrapper scripts that invoke the underlying compiler. The default underlying compiler is GCC, including gfortran. Note that there are more compiler drivers (wrapper scripts) with Intel MPI than are listed here (see Table 8 on page 60); check the Intel documentation for more information.

Table 8. Intel MPI Wrapper Scripts

Wrapper Script Name	Language
mpicc	C
mpiCC	C++
mpif77	Fortran 77
mpif90	Fortran 90
mpiicc	C (uses Intel C compiler)
mpiicpc	C++ (uses Intel C++ compiler)
mpiifort	Fortran 77/90 (uses Intel Fortran compiler)

To compile your program in C using the default compiler, type:

```
$ mpicc mpi_app_name.c -o mpi_app_name
```

To use the Intel compiler wrappers (mpiicc, mpiicpc, mpiifort), the Intel compilers must be installed and resolvable from the user's environment.

6.6.2 Running Intel MPI Applications

Here is an example of a simple mpirun command running with four processes:

```
$ mpirun -np 4 -f mpihosts mpi_app_name
```

For more information, follow the Intel MPI instructions for usage of mpirun, mpdboot, and mpiexec (mpirun is a wrapper script that invoked both mpdboot and mpiexec). Remember to use -r ssh with mpdboot if you use ssh.

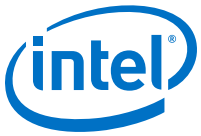
Pass the following option to mpirun to select TMI:

```
-genv I_MPI_FABRICS tmi
```

Pass the following option to mpirun to select uDAPL:

uDAPL 1.2:

```
-genv I_MPI_DEVICE rdma:OpenIB-cma
```



uDAPL 2.0:

```
-genv I_MPI_DEVICE rdma:ofa-v2-ib
```

To help with debugging, you can add this option to the Intel `mpirun` command:

TMI:

```
-genv TMI_DEBUG 1
```

uDAPL:

```
-genv I_MPI_DEBUG 2
```

6.6.3 Further Information on Intel MPI

For more information on using Intel MPI, see: http://www.intel.com/

6.7 Improving Performance of Other MPIs Over IB Verbs

Performance of MPI applications when using an MPI implementation over IB Verbs can be improved by tuning the IB MTU size.

Note: There is no manual tuning for PSMs-based MPIs. The PSMs layer determines the largest possible IB MTU for each source/destination path.

The maximum supported MTU size of HFIs is 10K.

Support for 10K IB MTU requires switch support for 10K MTU. The method to set the IB MTU size varies by MPI implementation:

- Open MPI defaults to the lower of either the IB MTU size or switch MTU size.
- MVAPICH2 defaults to an IB MTU size of 1024 bytes. This can be over-ridden by setting an environment variable, such as in the following command line:

  ```
  $ export VIADEV_DEFAULT_MTU=MTU2048
  ```

  Valid values are `MTU2048`, `MTU4096`, `MTU8192`, and `MTU10240`. This environment variable must be set for all processes in the MPI job. To do so, use `~/.bashrc` or use of `/usr/bin/env`.
- MVAPICH2 defaults to an IB MTU size of 2048 bytes, which should be sufficient for most applications.

Note: Intel recommends relying on the default MTU size of 8K for best performance.

7.0 SHMEM Description and Configuration

Overview

OpenSHMEM is a user-level communications library for one-sided operations. It implements the OpenSHMEM Application Programming Interface (API) and runs on the Intel® Omni-Path Fabric software stack. The OpenSHMEM API provides global distributed shared memory across a network of hosts.

SHMEM is quite distinct from local shared memory (often abbreviated as "shm" or even “shmem”). Local shared memory is the sharing of memory by processes on the same host running the same OS system image. SHMEM provides access to global shared memory distributed across a cluster. The OpenSHMEM API is completely different from, and unrelated to, the standard System V Shared Memory API provided by UNIX operating systems.

7.1 Interoperability

OpenSHMEM depends on the GASNet layer, which in turn depends on the Intel enhanced Performance Scaled Messaging (PSM2) protocol layer, implemented as a user-space library. OpenSHMEM is only available to run with Intel HFIs.

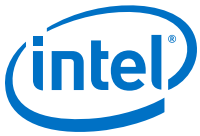
7.2 Installation

SHMEM is packaged with the Intel® Omni-Path Fabric Suite or Intel® Omni-Path Fabric Host Software. Every node in the cluster must have an Intel HFI and be running RedHat Enterprise Linux (RHEL) OS. One, or more, Message Passing Interface (MPI) implementations is required and Performance Scaled Messaging (PSM2) support must be enabled within the MPI. The following MPI Implementations are supported:

- Open MPI configured to include PSM2 support. This is provided by Intel IFS and can be found in the following directories:
  - `/usr/mpi/gcc/openmpi-X.X.X-hfi`
  - `/usr/mpi/intel/openmpi-X.X.X-hfi`
  - `/usr/mpi/pgi/openmpi-X.X.X-hfi`

  The `-hfi` suffix denotes that this is the Intel PSM2 version.
- MVAPICH2 compiled for PSM2. This is provided by Intel IFS and can be found in the following directory:
  - `/usr/mpi/gcc/mvapich2-X.X-hfi`
  - `/usr/mpi/intel/mvapich2-X.X-hfi`
  - `/usr/mpi/pgi/mvapich2-X.X-hfi`

  The `-hfi` suffix denotes that this is the Intel PSM2 version.



It is recommended that you match the compiler used to build the MPI implementation with the compiler that you are using to build your SHMEM application. For example, if you are using the Intel compilers to build your SHMEM application and wish to run with Open MPI, then use the Intel build of the Open MPI library:

```
/usr/mpi/intel/openmpi-X.X.X-hfi
```

The following C compilers are supported:

- gcc (as provided by distro) in 64-bit mode
- Intel 12.1 C compiler in 64-bit mode
- PGI 11.7 C compiler in 64-bit mode

For more information or to perform and installation with SHMEM enabled, refer the *Intel® Omni-Path Fabric Software Installation Guide*.

By default Intel SHMEM is installed with a prefix of /usr/shmem/intel into the following directory structure:

- `/usr/shmem/intel`
- `/usr/shmem/intel/bin`
- `/usr/shmem/intel/bin/mvapich2`
- `/usr/shmem/intel/bin/openmpi`
- `/usr/shmem/intel/lib64`
- `/usr/shmem/intel/lib64/mvapich2`
- `/usr/shmem/intel/lib64/openmpi`
- `/usr/shmem/intel/include`

Intel recommends that `/usr/shmem/intel/bin` is added onto your `$PATH`.

If it is not on your `$PATH`, then you will need to give full pathname scd to find the `shmemrun` and `shmemcc` wrapper scripts.

Note: There are subdirectories inside of bin for each MPI that are supported. These contain SHMEM benchmark programs that are linked directly against the MPI libraries as well as the SHMEM libraries.

7.3 Basic SHMEM Program

Examples of creating basic SHMEM programs are available in the OpenSHMEM Specification, Annex A, available at www.OpenSHMEM.org.

7.4 Compiling and Running SHMEM Programs

See the OpenSHMEM specification at www.openshmem.org , Annex B, for up-to-date information on compiling and running programs with shmrun.

Note that the MPIRUN_CMD environment variable needs to be set, but how it is set depends on the MPI used. For Open MPI:

```
export MPIRUN_CMD="mpirun -mca mtl ^psm2 -mca mtl ^psm -np %N %C"
```

This is necessary because PSM cannot be used by both MPI and SHMEM at the same time. The above environment variable prevents Open MPI from using PSM. This has no negative impact on performance; SHMEM only uses MPI for job launch and initialization.

First, the MPI, GASNet, and OpenSHMEM installations must be added to the environment paths:

```
export PATH=/usr/shmem/gcc/openshmem-$VERSION-hfi/bin:/usr/shmem/gcc/
gasnet-$VERSION-openmpi-hfi/bin:/usr/mpi/gcc/openmpi-$VERSION/bin:$PATH
```

```
export LD_LIBRARY_PATH=/usr/shmem/gcc/openshmem-$VERSION-hfi/lib:/usr/shmem/gcc/
gasnet-$VERSION-openmpi-hfi/bin:/usr/mpi/gcc/openmpi-$VERSION/
lib64:$LD_LIBRARY_PATH
```

MPI builds can also contain their own OpenSHMEM implementation. To use the Intel recommended OpenSHMEM, make sure the Intel OpenSHMEM installation is listed first, as is done in the paths above.

PSM has a restriction in which only one endpoint may be opened per process.

Intel OpenSHMEM and GASNet use MPI to launch processes. Due to this PSM restriction, that MPI must *NOT* also attempt to use PSM for communication.

If compiled with PSM support (such as with Intel's openmpi_gcc_hfi RPM package), Open MPI will try to use PSM by default. To override, set the MPIRUN_CMD variable. With Open MPI, for example:

```
export MPIRUN_CMD="mpirun -np %N -mca mtl ^psm,psm2 %P %A"
```

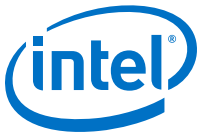
The above command line disables both the psm (v1) and psm2 MTLs. Open MPI will select another network such as verbs and/or TCP. An alternative is to use an MPI without PSM support enabled, such as MPICH.

Any site-specific mpirun options, such as specification of hosts to run on, may be set using the MPIRUN_CMD environment variable.

7.5 Slurm Integration

OpenSHMEM relies on an MPI implementation to provide a run-time environment for jobs. This includes job start-up, stdin/stdout/stderr routing, and other low performance control mechanisms. OpenSHMEM programs are typically started using `shmemrun`, which is a wrapper script around mpirun. The shmemrun script takes care of setting up the environment appropriately, and also provides a common command-line interface regardless of which underlying mpirun is used.

Integration of OpenSHMEM with slurm comes from the slurm integration provided by the MPI implementation. The slurm web pages describe 3 approaches. Please refer to points 1, 2 and 3 on the following web-page:



https://computing.llnl.gov/linux/slurm/mpi_guide.html

The following are various options listed for integration of the OpenSHMEM and slurm:

- Full Integration on page 65
- Two-step Integration on page 65
- No Integration on page 65

7.5.1 Full Integration

This approach fully integrates OpenSHMEM start-up into `slurm` and is available when running over MVAPICH2. The SHMEM program is executed using `srun` directly. For example:

```
srun -N 16 shmem-test-world
```

To run a program on 16 nodes. `slurm` starts the processes using `slurmd` and provides communication initialization. The implementation typically relies on `slurm` providing a process management interface (PMI) library and the MPI implementation using that so that each MPI process can hook into `slurm`.

The user is responsible for setting up the environment appropriately. This includes adding Intel SHMEM's library directory to LD_LIBRARY_PATH. See Compiling and Running SHMEM Programs on page 63 for more information on the environment setup.

7.5.2 Two-step Integration

This approach is integrated, but is performed in 2 steps to allocate the nodes and run the job. This is available when running over Open MPI. The run command is now:

```
salloc -N 16 shmemrun shmem-test-world
```

The `salloc` allocates 16 nodes and runs one copy of `shmemrun` on the first allocated node which then creates the SHMEM processes. `shmemrun` invokes `mpirun`, and `mpirun` determines the correct set of hosts and required number of processes based on the `slurm` allocation that it is running inside of. Since `shmemrun` is used in this approach there is no need for the user to set up the environment.

7.5.3 No Integration

This approach allows a job to be launched inside a `slurm` allocation but with no integration. This approach can be used for any supported MPI implementation. However, it requires that a wrapper script is used to generate the hosts file. `slurm` is used to allocate nodes for the job, and the job runs within that allocation, but not under the control of the `slurm` daemon. One way to use this approach is:

```
salloc -N 16 shmemrun_wrapper shmem-test-world
```

Where *shmemrun_wrapper* is a user-provided wrapper script that creates a hosts file based on the current `slurm` allocation and simply invokes `mpirun` with the hosts file and other appropriate options. Note that ssh will be used for starting processes not `slurm`.

7.6 Sizing Global Shared Memory

SHMEM provides `shmalloc`, `shrealloc` and `shfree` calls to allocate and release memory using a symmetric heap. These functions are called collectively across the processing elements (PEs) so that the memory is managed symmetrically across them. The extent of the symmetric heap determines the amount of global shared memory per PE that is available to the application.

This is an important resource and this section discusses the mechanisms available to size it. Applications can access this memory in various ways and this maps into quite different access mechanisms:

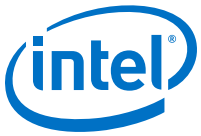
- Accessing global shared memory on my PE: This is achieved by direct loads and stores to the memory.
- Accessing global shared memory on a PE on the same host: This is achieved by mapping the global shared memory using the local shared memory mechanisms (for example, System V shared memory) operating system and then accessing the memory by direct loads and stores. This means that each PE on a host needs to map the global shared memory of each other PE on that host. These accesses do not use the adapter and interconnect.
- Accessing global shared memory on a PE on a different host: This is achieved by sending put, get, and atomic requests across the interconnect.

Note: There is a connection between the sizing of the global shared memory and local shared memory because of the mechanism used for accessing global shared memory in a PE that happens to be on the same host.

The OpenSHMEM library pre-allocates room in the virtual address space according to `$SHMEM_SHMALLOC_MAX_SIZE` (default of 4GB). It then populates this with enough pages to cover `$SHMEM_SHMALLOC_INIT_SIZE` (default 16MB). The global shared memory segment can then grow dynamically from its initial size up to its maximum size. If an allocation attempts to exceed the maximum size allocations are no longer guaranteed to succeed, and will fail if there is no room in the virtual memory space of the process following the global shared memory segment. Upon failure the call to `shmalloc` or `shrealloc` returns NULL. The only down-side of using a large maximum size is occupancy of virtual address space (48 bits for 64-bit processes is very plentiful), and set-up of page table entries by the OS. A reasonable limit is 4GB per process. One side-effect of this approach is that SHMEM programs consume a large amount of virtual memory when viewed with the "top" program. This is due to the large maximum size setting. The `RES` field of `top` indicates the actual amount of memory that is resident in memory (for example, in actual use).

If a SHMEM application program runs out of global shared memory, increase the value of `$SHMEM_SHMALLOC_MAX_SIZE`. The value of `$SHMEM_SHMALLOC_INIT_SIZE` can also be changed to pre-allocate more memory up front rather than dynamically.

By default OpenSHMEM will use the same base address for the symmetric heap across all PEs in the job. This address can be changed using the `$SHMEM_SHMALLOC_BASE_ADDR` environment variable. It will be rounded up to the



nearest multiple of the page size. The virtual address range specified by this base address and the maximum size must not clash with any other memory mapping. If any SHMEM process in a job has a memory mapping clash, the Intel SHMEM library will fail during `shmem_init()`. With 64-bit programs, a large virtual address space (for example, 48 bits in many modern processors) and a reasonably homogeneous cluster, it is expected that such failures will be rare. The default value of `$SHMEM_SHMALLOC_BASE_ADDR` has been chosen to work on the supported distributions and processors. In the rare event of a failure, the value of `$SHMEM_SHMALLOC_BASE_ADDR` can be changed using the environment variable.

Alternatively, if `$SHMEM_SHMALLOC_BASE_ADDR` is specified as `0`, then each SHMEM process will independently choose its own base virtual address for the global shared memory segment. In this case, the values for a symmetric allocation using `shmalloc()` are no longer guaranteed to be identical across the PEs. The OpenSHMEM implementation takes care of this asymmetry by using offsets relative to the base of the symmetric heap in its protocols. However, applications that interpret symmetric heap pointer values or exchange symmetric heap pointer values between PEs will not behave as expected.

It is possible for SHMEM to fail at start-up or while allocating global shared memory due to limits placed by the operating system on the amount of *local* shared memory that SHMEM can use. Since SHMEM programs can use very large amounts of memory this can exceed typical OS configurations. As long as there is sufficient physical memory for the program, the following steps can be used to solve local shared memory allocation problems:

- Check for low `ulimits` on memory:

```
ulimit -l : max locked memory (important for PSM, not SHMEM)
ulimit -v : max virtual memory
```

- Check the contents of these `sysctl` variables:

```
sysctl kernel.shmmax ; maximum size of a single shm allocation in
bytes
sysctl kernel.shmall ; maximum size of all shm allocations in “pages”
sysctl kernel.shmnmi ; maximum number of shm segments
```

- Check the size of `/dev/shm`:

```
df /dev/shm
```

- Check for stale files in `/dev/shm`:

```
ls /dev/shm
```

If any of these checks indicate a problem, ask the cluster administrator to increase the limit.

7.7 Application Programming Interface

OpenSHMEM.org provides full documentation on the OpenSHMEM Application Programming Interface, under "Get Documentation\Specification" in the site navigation frame. Select the OpenSHMEM Specification 1.1 Final document.

7.8 SHMEM Benchmark Programs

The following SHMEM micro-benchmark programs are included:

- `shmem-get-latency`: measures get latency
- `shmem-get-bw`: measures streaming get bandwidth (uni-directional)
- `shmem-get-bibw`: measures streaming get bandwidth (bi-directional)
- `shmem-put-latency`: measures put latency
- `shmem-put-bw`: measures streaming put bandwidth (uni-directional)
- `shmem-put-bibw`: measures streaming put bandwidth (bi-directional)

The programs can be used to measure round-trip get latency, one way put latency, get and put bandwidth, as well as get and put message rates.

The benchmarks must be run with an even number of processes. They are typically run on exactly two hosts with the processes equally-divided between them. The processes are split up into pairs, with one from each pair on either host and each pair is loaded with the desired traffic pattern. The benchmark automatically determines the correct mapping, regardless of the actual rank order of the processes and their mapping to the two hosts.

Alternatively, if the `-f` option is specified the benchmark is forced to use the rank order when arranging the communication pattern. In this mode and with `np` ranks, each rank `i` (in `0`, `np/2`) is paired with rank `(np / 2) _ i`. For example, this mode can be used to test SHMEM performance within a single node.

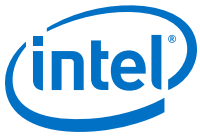
The micro-benchmarks have the command line options shown in Table 9 on page 68

Table 9. OpenSHMEM micro-benchmarks options

Option	Description
`-a INT`	`log2` of desired alignment for buffers (default = 12)
`-b INT`	Batch size, number of concurrent operations (default = 64)
`-f`	Force order for bifurcation of PEs based on rank order
`-h`	Displays the help page
`-l INT`	Set minimum message size (default = 2)
`-m INT`	Sets the maximum message size (default = 4194304)

Additional SHMEM micro-benchmark programs are included to measure get and put performance with randomized PE selection and randomized target memory locations, all-to-all communication patterns using put, barrier and reduce:

- OpenSHMEM Random Access Benchmark on page 69
- OpenSHMEM all-to-all benchmark on page 69
- OpenSHMEM barrier benchmark on page 70
- OpenSHMEM reduce benchmark on page 70



7.8.1 OpenSHMEM Random Access Benchmark

`shmem-rand`: randomized put/get benchmark

- This is actually a hybrid SHMEM/MPI code, so a binary is provided per supported MPI implementation. It has the following command line options:

  Usage: `shmem-rand [options] [list of message sizes]`.

  Message sizes are specified in bytes (default = 8)

  Options: See Table 10

Table 10. OpenSHMEM random access benchmark options

Option	Description
`-a`	Use automatic (NULL) handles for NB ops (default explicit handles)
`-b`	Use a barrier every window
`-c INTEGER`	Specify loop count (see also `-t`)
`-f`	Fixed window size (default is scaled)
`-h`	Displays the help page
`-l`	Enable communication to local ranks
`-m INTEGER[K]`	Memory size in MB (default = 8MB): or in KB with a K suffix
`-n`	Use non-pipelined mode for NB ops (default pipelined)
`-o OP`	Choose `OP` from `get`, `getnb`, `put`, `putnb`
`-p`	For blocking puts, no quiet every window (this is the default)
`-q`	For blocking puts, use quiet every window
`-r`	Use ring pattern (default is random)
`-s`	Enable communication to self
`-t FLOAT`	If the loop count is not given, run the test for this many seconds (default is 10s)
`-u`	Run in uni-directional mode
`-v`	Verbose mode (repeat for more verbose)
`-w INTEGER`	Set the window size (default = 32)
`-x INTEGER`	Window size limit (default = 16384)

7.8.2 OpenSHMEM all-to-all benchmark

`shmem-alltoall`: all-to-all put benchmark

- This is a hybrid SHMEM/MPI code, so a binary is provided per supported MPI implementation. It has the following command line options:

  Usage: `/test/shmem-alltoall [options] [list of message sizes]`

  Message sizes are specified in bytes (default 8)

  Options: See Table 11

Table 11. OpenSHMEM all-to-all benchmark options

Option	Description
`-a`	Use automatic (NULL) handles for NB ops (default explicit handles)
`-c INTEGER`	Specify loop count (see also `-t`)
`-f`	Fixed window size (default is scaled)
`-h`	Displays the help page
`-l`	Enable communication to local ranks (including self)
`-m INTEGER[K]`	Memory size in MB (default = 8MB): or in KB with a K suffix
`-n`	Use non-pipelined mode for NB ops (default pipelined)
`-o OP`	Choose `OP` from put, or `putnb`
`-p INTEGER`	Offset for all-to-all schedule (default 1, usually set to ppn)
`-r`	Randomize all-to-all schedule
`-s`	Enable communication to self
`-t FLOAT`	If the loop count is not given, run the test for this many seconds (default is 10s)
`-v`	Verbose mode (repeat for more verbose)
`-w INTEGER`	Set the window size (default = 32)
`-x INTEGER`	Window size limit (default = 16384)

7.8.3 OpenSHMEM barrier benchmark

`shmem-barrier`: barrier benchmark

- Usage: `shmem-barrier [options]`

  Options: See Table 12

Table 12. OpenSHMEM barrier benchmark options

Option	Description
`-h`	Displays the help page
`-i INTEGER[K]`	Outer iterations (default 1)

7.8.4 OpenSHMEM reduce benchmark

`shmem-reduce`: reduce benchmark

- Usage: `shmem-reduce [options]`

  Options: See Table 13

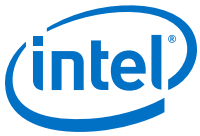


Table 13. OpenSHMEM reduce benchmark options

Option	Description
`-b INTEGER`	Number of barriers between reduces (default 0)
`-h`	Displays the help page
`-i INTEGER[K]`	Outer iterations (default 1)
`-r INTEGER`	Inner iterations (default 10000)

8.0 Virtual Fabric Support in PSM2

Performance Scaled Messaging for Intel® Omni-Path (PSM2) provides support for full Virtual Fabric (vFabric) integration, allowing users to specify IB Service Level (SL) and Partition Key (PKey), or to provide a configured Service ID (SID) to target a vFabric. Support for using IB path record queries to the Intel® Omni-Path Fabric Suite Fabric Manager (FM) during connection setup is also available, enabling alternative switch topologies.

All PSM2-enabled Message Passing Interfaces (MPIs) can leverage these capabilities transparently use of environment variables to leverage these capabilities. With MPI applications, the environment variables need to be propagated across all nodes/ processes and not just the node from where the job is submitted/run. The mechanisms to do this are MPI specific, but for two common MPIs the following may be helpful:

- **Open MPI**: Use `-x ENV_VAR=ENV_VAL` in the mpirun command line.
- Example:

```
mpirun -np 2 -machinefile machinefile -x PSM2_ENV_VAR=PSM_ENV_VAL prog
prog_args
```

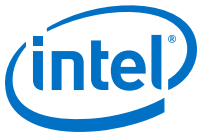
- **MVAPICH2**: Use `mpirun_rsh` to perform job launch. Do not use `mpiexec` or `mpirun` alone. Specify the environment variable and value in the `mpirun_rsh` command line before the program argument.
- Example:

```
mpirun_rsh -np 2 -hostfile machinefile PSM2_ENV_VAR=PSM_ENV_VAL prog prog_args
```

8.1 Virtual Fabric Support

Virtual Fabric (vFabric) in PSM2 is supported with the Fabric Manager (FM). The latest version of the Fabric Manager contains a sample file with pre-configured vFabrics for PSM2, at `/etc/sysconfig/opafm.xml` . Sixteen unique Service IDs have been allocated for PSM2 enabled MPI vFabrics to ease their testing however any Service ID can be used. Refer to the *Intel® Omni-Path Fabric Suite Fabric Manager User Guide* on how to configure vFabrics.

There are two ways to use vFabric with PSM2. The “legacy” method requires the user to specify the appropriate SL and Pkey for the vFabric in question. For complete integration with vFabrics, users can now specify a Service ID (SID) that identifies the vFabric to be used. PSM2 will automatically obtain the SL and Pkey to use for the vFabric from the Fabric Manager via path record queries.



9.0 PSM2 Multi-Rail

Multi-rail means that a process can use multiple network interface cards to transfer messages. With modern computer servers supporting multiple HFIs, multi-rail improves network performance for applications. For Intel® Omni-Path, implementation of PSM2 Multi-rail requires multiple HFIs.

Prior to supporting multi-rail, PSM could use multiple cards/ports for a single job, but for a particular process in the job, only one port could be used to transfer a message. All the ports had to be on the same fabric in order for the application to use them.

The PSM2 multi-rail feature can be applied to a single fabric with multiple ports (multiple HFIs), or multiple fabrics. It does not change the PSM2 API application, and it is binary-compatible with previous PSM versions. The main goal is to use multiple HFIs to transfer messages to improve the message bandwidth.

Note: `PSM2_MUTIRAIL` is the PSM2 environment variable that is used to enable this feature in PSM2. If `PSM2_MULTIRAIL` is not enabled, it is not supported.

9.1 User Base

The system administrator sets up a PSM2 multi-rail system using multiple Intel® Omni-Path HFIs per node. If multiple fabrics are desired, the system administrator connects the HFIs to multiple fabrics, and configures each fabric with different subnet IDs.

PSMs, by default, uses the single-rail configuration, where each process only uses a single context/sub-context, which maps to a single port, to communicate to other processes. The user must tell PSM2 to use multiple rail communication on systems with multiple cards per node.

On a multi-fabric system, if multi-rail is not turned on, the user must set the `HFI_UNIT` environment variable (from 0) to tell the PSM2 job which HFI to use. The HFIs have to be on the same fabric, otherwise, the same job might try to use HFIs from different fabrics and cause the job to hang because there is no path between fabrics. If multi-rail is turned on, PSM2 can reorder and automatically match the HFIs by using the subnet ID. That is why different subnet IDs are required for different fabrics.

9.2 Environment Variables

The following are environment variables that can be set:

PSM2_MULTIRAIL=*n* (–***n*** can be any value up to a maximum of 4). If this environment variable is set to a non-zero value, PSM2 tries to setup multiple rails, to a maximum of 4. Otherwise, multi-rail is turned off. How multi-rails are set up and how many rails are set up depends on whether the environment variable `PSM2_MULTIRAIL_MAP` is set.

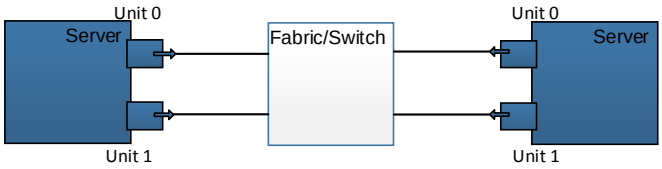
PSM2_MULTIRAIL_MAP=*unit:port,unit:port,unit:port,...* – ***unit*** is from 0, and because Intel® Omni-Path Host Fabric Interfaces are single port devices, ***port*** is always 1. This environment variable tells PSM2 which unit/port pair is used to set up a rail. Multiple specifications are separated by a comma. If only one rail is specified, it is equivalent to a single-rail case; the Unit/Port is specified instead of using Unit/Port assigned by the hfi driver. PSM2 scans the above pattern until a violation or error is encountered, and uses the information it has gathered.

9.3 Examples of Single- and Multi-rail

The following are a few examples of single- and multi-rail configurations for both single- and multi-fabrics.

Single fabric, each node has two HFIs, Unit 0 has one port, Unit 1 has one port

The following figure shows an example of a single fabric with each node having two cards. Unit 0 (hfi_0) has one port and Unit 1 (hfi_1) has one port.

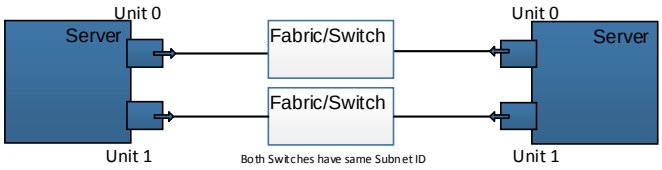


Example Environment Variables

- `PSM2_MULTIRAIL` is not set. PSM2 is using single-rail, the Unit/Port/context selection is from the assignment of the hfi1 driver . `HFI_UNIT` and `HFI_PORT` are set by the user to specify the Unit/Port to use.
- `PSM2_MULTIRAIL` is set. PSM2 checks that there are two units in the system. The first available port is Port 1 for Unit 0. The next available port is Port 1 for Unit 1. PSM2, by default, will use a `PSM2_MULTIRAIL_MAP` of 0:1,1:1. Since this a single fabric, all of the ports have the same subnet ID. PSM2 sets up the first (master) connection over 0:1, and sets up the second slave connection over 1:1
- `PSM2_MULTIRAIL=1` and `PSM2_MULTIRAIL_MAP=1:1,0:1` The user specifies the map, how to use the Unit/Port, and PSM2 uses the given pairs. PSM2 sets up the master connection over Unit 1 Port 1 and sets up the slave connection over Unit 0 Port 1.

Multi-fabrics, with same subnet ID

The following figure shows an example of multiple fabrics with the same subnet ID.

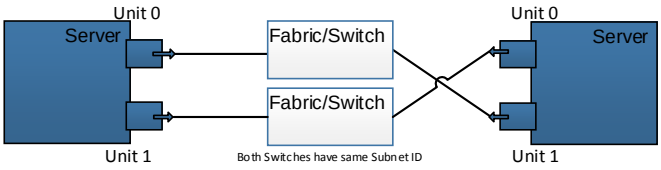


Example Environment Variables

- PSM2_MULTIRAIL is not set. There are multiple fabrics, therefore PSM2 will not work if multi-rail is not turned on. If one process on a different node chooses Unit 0 Port 1 and another process chooses Unit 1 Port 1, there is no path between these two processes and the MPI job will fail to start.
- PSM2_MULTIRAIL is set. The two fabrics have the same subnet ID and PSM2 does not know which ports are in the same fabric. PSM2 does not work in this case.

Multi-fabrics, single subnet ID, abnormal wiring.

The following figure shows an example of multiple fabrics with a single subnet ID, and abnormal wiring.

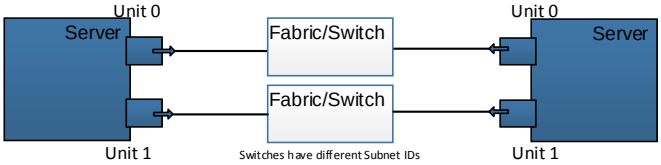


Example Environment Variables

- PSM2_MULTIRAIL is not set. PSM2 does not work since there are multiple fabrics.
- PSM2_MULTIRAIL=1. The two fabrics have the same subnet ID, PSM2 does not know which ports are in the same fabric. PSM2 does not work in this case.

Multi-fabrics, different subnet IDs

The following figure shows an example of multiple fabrics with different subnet IDs.

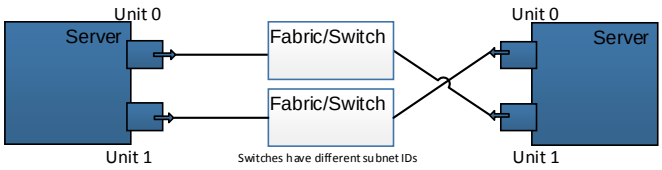


Example Environment Variables

- PSM2_MULTIRAIL is not set. PSM2 does not work because there are multiple fabrics. Unit 0/Port 1 on first node has no connection to Unit 1/Port 1 on second node.
- PSM2_MULTIRAIL=1 automatic selection. Both nodes get Unit/Port pairs 0:1,1:1 first, after the PSM2 reordering based on subnet ID, the node on the left side will get 0:1,1:1 and the node on the right side gets 0:1,1:1. The PSM2 makes the master rail on 0:1 of left node with 0:1 on right node. The slave rail is setup on 1:1 of the left node with 1:1 of the right node. PSM2 works in this configuration/ setting.
- PSM2_MULTIRAIL=1 and **PSM2_MULTIRAIL_MAP=1:1,0:1**. The user specifies the Unit/Port pairs. PSM2 does not reorder them. Both nodes use 1:1 to make the connection on the second fabric as the master rail, and set up the second rail over 0:1 on both sides. PSM2 works fine in this configuration.

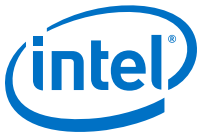
Multi-fabrics, different subnet IDs, abnormal wiring.

The following figure shows an example of multiple fabrics with different subnet IDs and abnormal wiring.



Example Environment Variables

- `PSM2_MULTIRAIL` is not set. PSM2 does not work because there are multiple fabrics.
- `PSM2_MULTIRAIL=1` automatic selection. Both nodes get Unit/Port pairs 0:1,1:1 first, after PSM2 reordering based on the subnet ID, the node on the left side will get 0:1,1:1 again and the node on the right side gets 1:1,0:1. The PSM2 makes the master rail on 0:1 of the left node with 1:1 on the right node. The slave rail is setup on 1:1 of left with 0:1 of right. PSM2 works in this configuration/setting.
- `PSM2_MULTIRAIL=1` and **PSM2_MULTIRAIL_MAP=1:1,0:1**. The user specifies the Unit/Port pairs. PSM2 does not reorder them. Both nodes use 1:1 to make a connection, it fails because there is no path between them. PSM2 does not work in this case.



10.0 Routing

Intel® Omni-Path features two methods for routing messages through the fabric, Dispersive Routing and Adaptive Routing. The default routing method is Dispersive routing. When Adaptive routing is enabled, Dispersive Routing is used until the Intel® Omni-Path Fabric Suite Fabric Manager detects congestion on one or more ports in the fabric. Then the Adaptive Routing algorithm reroutes a subset of traffic at the congested port(s) over less congested to the switches involved so the congestion can be alleviated.

10.1 Dispersive Routing

Intel® Omni-Path architecture uses deterministic routing that is keyed from the Destination LID (DLID) of a port. The Fabric Manager (FM) programs the forwarding tables in a switch to determine the egress port a packet takes based on the DLID.

Deterministic routing can create hotspots even in full bisection bandwidth (FBB) fabrics for certain communication patterns if the communicating node pairs map onto a common upstream link, based on the forwarding tables. Since routing is based on DLIDs, the Intel® Omni-Path fabric provides the ability to assign multiple LIDs to a physical port using a feature called Lid Mask Control (LMC). The total number of DLIDs assigned to a physical port is 2^LMC with the LIDS being assigned in a sequential manner. The common Intel® Omni-Path fabric uses a LMC of 0, meaning each port has 1 LID assigned to it. With non-zero LMC fabrics, this results in multiple potential paths through the fabric to reach the same physical port. For example, multiple DLID entries in the port forwarding table that could map to different egress ports.

Dispersive routing, as implemented in PSM2, attempts to avoid congestion hotspots, described above, by "spraying" messages across these paths. A congested path will not bottleneck messages flowing down the alternate paths that are not congested. The current implementation of PSM2 supports fabrics with a maximum LMC of 3 (8 LIDs assigned per port). This can result in a maximum of 64 possible paths between a SLID, DLID pair ([SLID, DLID],[SLID, DLID_1], [SLID,DLID_2]…..[SLID,DLID_7], [SLID_1, DLID],[SLID_1, DLID_1]…..[SLID_7, DLID_7]). Keeping state associated with these many paths requires large amount of memory resources, with empirical data showing not much gain in performance beyond utilizing a small set of multiple paths. Therefore, PSM2 reduces the number of paths actually used in the above case to 8, where the following paths are the only ones considered for transmission: [SLID, DLID], [SLID _ 1, DLID _ 1], [SLID _ 2, DLID _ 2] ….. [SLID _ N, DLID _ N]. This makes the resource requirements manageable while providing most of the benefits of dispersive routing (congestion avoidance by utilizing multiple paths).

Internally, PSM2 utilizes dispersive routing differently for small and large messages. Large messages are any messages greater-than or equal-to 64K. For large messages, the message is split into message fragments of 128K by default (called a window). Each of these message windows is sprayed across a distinct path between ports. All packets belonging to a window utilize the same path; however, the windows themselves can take a different path through the fabric. PSM2 assembles the windows that make up an MPI message before delivering it to the application. This allows limited out of order semantics through the fabrics to be maintained with little

overhead. Small messages, on the other hand, always utilize a single path when communicating to a remote node; however, different processes executing on a node can utilize different paths for their communication between the nodes. For example, for two nodes A and B each with 8 processors per node and assuming the fabric is configured for a LMC of 3. In this example, PSM2 constructs 8 paths through the fabric as described above and a 16 process MPI application that spans these nodes (8 process per node). Then:

- Each MPI process is automatically bound to a given CPU core numbered between 0-7. PSM2 does this at startup to get improved cache hit rates and other benefits.
- Each process is assigned an HFI Receive context number
- Small Messages sent to a remote process that will use a fixed path N, where N is selected with a round-robin algorithm.

  Note: Only path N will be used by this process for all communications to any process on the remote node.
- For a large message, each process will utilize all of the 8 paths and spray the windowed messages across it.

The above example highlights the default path selection policy that is active in PSM2 when running on non-zero LMC configured fabrics. There are 3 other path selection policies that determine how to select the path (or path index from the set of available paths) used by a process when communicating with a remote node. The above path policy is called adaptive. The 3 remaining path policies are static policies that assign a static path on job startup for both small and large message transfers.

- **Static_Src**: Only one path per process is used for all remote communications. The path index is based on the context number the process is running.

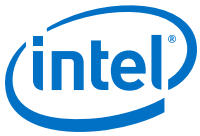
  Note: Multiple paths are still used in the fabric if multiple processes (each in a different context) are communicating.
- **Static_Dest**: The path selection is based on the context index of the destination process. Multiple paths can be used if data transfer is to different remote processes within a node. If multiple processes from Node A send a message to a single process on Node B, only one path will be used across all processes.
- **Static_Base**: The only path that is used is the base path [SLID,DLID] between nodes regardless of the LMC of the fabric or the number of paths available.

  Note: A fabric configured with LMC of 0 even with the default adaptive policy enabled operates as the Static_Base policy as there only exists a single path between any pairs of port.

10.2 Adaptive Routing

Adaptive routing works congruently with the Intel® Omni-Path Congestion Control feature. When congestion is detected in the fabric, the Fabric Manager will change the routing algorithm from the default Dispersive Routing to Adaptive Routing, wherein messages are redirected from congested ports and routes and rerouted to less trafficed routes, while introducing no additional hops. This allows the Fabric Manager to monitor the fabric and dynamically adjust routing maps on the switches, to maintain maximum throughput.

Refer to *Intel® Omni-Path Fabric Suite Fabric Manager User Guide* for detailed information regarding Adaptive Routing function and configuration.



11.0 Integration with a Batch Queuing System

Most cluster systems use some kind of batch queuing system as an orderly way to provide users with access to the resources they need to meet their job's performance requirements. One task of the cluster administrator is to allow users to submit MPI jobs through these batch queuing systems.

For Open MPI, there are resources at openmpi.org that document how to use the MPI with three batch queuing systems. The links to the Frequently Asked Questions (FAQs) for each of the three batch queuing system are as follows:

- Torque / PBS Pro: http://www.open-mpi.org/faq/?category=tm
- SLURM: http://www.open-mpi.org/faq/?category=slurm
- Bproc: http://www.open-mpi.org/faq/?category=bproc

There are two topics which deal with process and file clean-up after batch MPI/PSM2 jobs have completed:

- Clean Termination of MPI Processes
- Clean-up PSM2 Shared Memory Files

11.1 Clean Termination of MPI Processes

The Intel® Omni-Path Fabric Host Software normally ensures clean termination of all Message Passing Interface (MPI) programs when a job ends, but in some rare circumstances an MPI process may remain alive, and potentially interfere with future MPI jobs. To avoid this problem, run a script before and after each batch job that kills all unwanted processes. Intel does not provide such a script, but it is useful to know how to find out which processes on a node are using the Intel interconnect. The easiest way to do this is with the `fuser` command, which is normally installed in `/sbin`.

Run these commands as a root user to ensure that all processes are reported.

```
/sbin/fuser -v /dev/hfi1
/dev/ipath:    22648m 22651m
```

In this example, processes 22648 and 22651 are using the Intel interconnect. It is also possible to use this command (as a root user):

```
lsof /dev/hfi1
```

This command displays a list of processes using Intel® Omni-Path. Additionally, to get all processes, including stats programs, SMA, diags, and others, run the program in the following manner:

```
/sbin/fuser -v /dev/hfi1*
```

lsof can also take the same form:

```
lsof /dev/hfi1*
```

The following command terminates all processes using the Intel interconnect:

```
/sbin/fuser -k /dev/hfi1
```

For more information, see the man pages for fuser(1) and lsof(8).

Note: Hard and explicit program termination, such as kill -9 on the mpirun Process ID (PID), may result in Open MPI being unable to guarantee that the /dev/shm shared memory file is properly removed. As many stale files accumulate on each node, an error message can appear at startup:

```
node023:6.Error creating shared memory object in shm_open(/dev/shm may have stale
shm files that need to be removed):
```

If this occurs, refer to Clean-up PSM2 Shared Memory Files for information.

11.2 Clean-up PSM2 Shared Memory Files

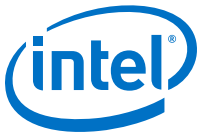
In some cases if a PSM2 job terminates abnormally, such as with a segmentation fault, there could be POSIX shared memory files leftover in the /dev/shm directory. The file is owned by the user and they have permission (-rwx------) to remove the file either by the user or by root.

PSM2 relies on the Message Passing Interface (MPI) implementation to cleanup after abnormal job termination. In cases where this does not occur there may be leftover shared memory files. To clean up the system, create, save, and run the following PSM2 SHM cleanup script as root on each node. Either logon to the node, or run remote using pdsh/ssh.

```
#!/bin/sh
files=`/bin/ls /dev/shm/psm2_shm.* 2> /dev/null`;
for file in $files;
do
/sbin/fuser $file > /dev/null 2>&1;
if [ $? -ne 0 ];
then
/bin/rm $file > /dev/null 2>&1;
fi;
done;
```

When the system is idle, the administrators can remove all of the shared memory files, including stale files, by using the following command:

```
# rm -rf /dev/shm/psm2_shm.*
```



12.0 Benchmark Programs

Several Message Passing Interface (MPI) performance measurement programs are installed by default with the MPIs you choose to install (such as Open MPI or MVAPICH2). This appendix describes a few of these benchmarks and how to run them. Several of these programs are based on code from the group of Dr. Dhabaleswar K. Panda at the Network-Based Computing Laboratory at the Ohio State University. For more information, see: http://mvapich.cse.ohio-state.edu/

These programs allow you to measure MPI latency, bandwidth, and message rate between two or more nodes in your cluster. The executables are installed by default under `/usr/mpi/compiler/mpi/tests/osu_benchmarks-X.X.X`. The remainder of this chapter will assume that the gcc-compiled version of Open MPI was installed in the default location of `/usr/mpi/gcc/openmpi-X.X.X-hfi` and that mpi-selector is used to choose this Open MPI version as the MPI to be used.

Note: The following examples are intended to show only the syntax for invoking these programs and the meaning of the output. They are not representations of actual Intel® Omni-Path performance characteristics.

For additional MPI sample applications, refer to the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*.

12.1 Benchmark 1: Measuring MPI Latency Between Two Nodes

In the MPI community, latency for a message of given size is the time difference between a node program calling MPI_Send and the time that the corresponding MPI_Recv in the receiving node program returns. The term latency alone, without a qualifying message size, indicates the latency for a message of size zero. This latency represents the minimum overhead for sending messages, due to both software overhead and delays in the electronics of the fabric. To simplify the timing measurement, latencies are usually measured with a ping-pong method, timing a round-trip and dividing by two.

The program `osu_latency`, from The Ohio State University, measures the latency for a range of messages sizes from 0 bytes to 4 megabytes. It uses a ping-pong method, where the rank zero process initiates a series of sends and the rank one process echoes them back, using the blocking MPI send and receive calls for all operations. Half the time interval observed by the rank zero process for each exchange is a measure of the latency for messages of that size, as previously defined. The program uses a loop, executing many such exchanges for each message size, to get an average. The program defers the timing until the message has been sent and received a number of times, to be sure that all the caches in the pipeline have been filled.

This benchmark always involves two node programs. It can be run with the command:

```
$ mpirun -H host1,host2 /usr/mpi/gcc/openmpi-X.X.X-hfi/tests/osu_benchmarks-X.X.X/
osu_latency
```

-H (or --hosts) allows the specification of the host list on the command line instead of using a host file (with the -m or -machinefile option). Since only two hosts are listed in this example, only two host programs would be started (as if -np 2 were specified).

Note: This example shows the syntax of the command and the format of the output. The output of the program will depend on your particular configuration.

```
# OSU MPI Latency Test vX.X.X)
# Size          Latency (us)
0                 -
.                 -
.                 -
.                 -
4194304           -
```

The first column displays the message size in bytes. The second column displays the average (one-way) latency in microseconds.

12.2 Benchmark 2: Measuring MPI Bandwidth Between Two Nodes

The osu_bw benchmark measures the maximum rate that you can move data between two nodes. This benchmark also uses a ping-pong mechanism, similar to the osu_latency code. But in this case the originator of the messages pumps a number of them (64 in the installed version) in succession using the non-blocking MPI_Isend function, while the receiving node consumes them as quickly as it can using the non-blocking MPI_Irecv function and then returns a zero-length acknowledgment when all of the sent data has been received.

You can run this program by typing:

```
$ mpirun -H host1,host2 /usr/mpi/gcc/openmpi-X.X.X-hfi/tests/osu_benchmarks-X.X.X/
osu_bw
```

Note: This example shows the syntax of the command and the format of the output. The output of the program will depend on your particular configuration.

```
# OSU MPI Bandwidth Test vX.X.X
# Size        Bandwidth (MB/s)
1                     -
.                     -
.                     -
.                     -
4194304               -
```

You will see an increase in measured bandwidth with the message size due to the contribution of each packet's overhead to the measured time becoming relatively smaller.

12.3 Benchmark 3: Messaging Rate Microbenchmarks A.3.1 OSU Multiple Bandwidth / Message Rate Test (osu_mbw_mr)

`osu_mbw_mr` is a multi-pair bandwidth and message rate test that evaluates the aggregate uni-directional bandwidth and message rate between multiple pairs of processes. Each of the sending processes sends a fixed number of messages (the window size) back-to-back to the paired receiving process before waiting for a reply from the receiver. This process is repeated for several iterations. The objective of this benchmark is to determine the achieved bandwidth and message rate from one node to another node with a configurable number of processes running on each node. You can run this program as follows:

```
$ mpirun -H host1,host2 -npernode 12 /usr/mpi/gcc/openmpi-X.X.X-hfi/tests/
osu_benchmarks-X.X.X/osu_mbw_mr
```

This sample was run on 12-core compute nodes, so we used Open MPI's `-npernode 12` option to place 12 MPI processes on each node (for a total of 24) to maximize message rate. Note that the output below indicates that there are 12 pairs of communicating processes.

Note: This example shows the syntax of the command and the format of the output. The output of the program will depend on your particular configuration.

```
OSU MPI Multiple Bandwidth / Message Rate Test vX.X.X
[ pairs: 12 ] [ window size: 64 ]
Size                    MB/s          Messages/s
1                          -                   -
.                          -                   -
.                          -                   -
.                          -                   -
4194304                    -                   -
```

12.3.1 An Enhanced Multiple Bandwidth / Message Rate Test (mpi_multibw)

`mpi_multibw` is a version of osu_mbw_mr which has been enhanced by Intel to optionally run in a bidirectional mode and to scale better on the larger multi-core nodes available today. This benchmark is a modified form of the OSU Network-Based Computing Lab's `osu_mbw_mr` benchmark (as shown in the previous example). It has been enhanced with the following additional functionality:

- N/2 is dynamically calculated at the end of the run.
- You can use the `-b` option to get a bidirectional message rate and bandwidth results.
- Scalability has been improved for larger core-count nodes.

The benchmark has been updated with code to dynamically determine what processes are on which host.

Note: The values returned by the test will depend on your particular configuration.

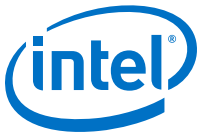
```
$ mpirun -H host1,host2 -npernode 12  /usr/mpi/gcc/openmpi-X.X.X-hfi/tests/intel/
mpi_multibw
PathScale Modified OSU MPI Bandwidth Test
(OSU Version X.X, PathScale $Revision: X.X.X.X $)
Running on 12 procs per node (uni-directional traffic for each process pair)

Size          Aggregate Bandwidth (MB/s)      Messages/s
1                   -                              -
.                   -                              -
.                   -                              -
.                   -                              -
4194304             -                              -
Searching for N/2 bandwidth.  Maximum Bandwidth of - MB/s...
Found N/2 bandwidth of - MB/s at size 121 bytes
```

You will see improved message rates at small message sizes of ~- million compared to the rate measured with `osu_mbw_mr`. Note that it only takes a message of size - bytes to generate half of the peak uni-directional bandwidth.

The following is an example output when running with the bidirectional option (`-b`):

Note: This example shows the syntax of the command and the format of the output. The output of the program will depend on your particular configuration.

```
$ mpirun -H host1,host2 -np 24  \
  /usr/mpi/gcc/openmpi-X.X.X-hfi/tests/intel/mpi_multibw -b
PathScale Modified OSU MPI Bandwidth Test
(OSU Version X.X, PathScale $Revision: X.X.X.X $)
Running on 12 procs per node (bi-directional traffic for each process pair)
Size          Aggregate Bandwidth (MB/s)      Messages/s
1                   -                              -
.                   -                              -
.                   -                              -
.                   -                              -
4194304             -                              -
Searching for N/2 bandwidth.   Maximum Bandwidth of - MB/s...
Found N/2 bandwidth of - MB/s at size - bytes
```



13.0 Troubleshooting

This chapter describes some of the tools you can use to diagnose and fix problems. The following topics are discussed

Troubleshooting information for hardware installation is found in the *Intel® Omni-Path Host Fabric Interface Installation Guide* and software installation is found in the *Intel® Omni-Path Fabric Software Installation Guide*.

13.1 Using the LED to Check the State of the HFI

The LED on the Intel® Omni-Path Host Fabric Interfacefunctions as link state and data rate indicator once the Intel® Omni-Path software has been installed, the driver has been loaded, and the fabric is being actively managed by a subnet manager.

The LED functions are as follows:

- • Link Down = Off
- • Link Up Initialized = On solid green
- • Link Up Active (No Traffic) = On solid green
- • Link Up: Active (Slow Packet rate <10K/S) = BLINK: 384ms On, 384ms Off
- • Link Up: Active (Fast Packet rate >10K/S) = BLINK: 128ms On, 128ms Off

13.2 BIOS Settings

Please see the *Intel® Omni-Path Fabric Performance Tuning User Guide* for information relating to checking, setting, and changing BIOS settings.

13.3 Kernel and Initialization Issues

Issues that may prevent the system from coming up properly are described in the following sections:

- Intel® Omni-Path HFI Initialization Failure on page 87
- MPI Job Failures Due to Initialization Problems on page 88

13.3.1 Driver Load Fails Due to Unsupported Kernel

If you try to load the Omni-Path driver on a kernel that Omni-Path software does not support, the load fails. Error messages will point to `hfi1.ko`

To correct this problem, install one of the appropriate supported Linux kernel versions, then reload the driver.

13.3.2 Rebuild or Reinstall Drivers if Different Kernel Installed

If you upgrade the kernel, you must reboot and then rebuild or reinstall the Intel® Omni-Path kernel modules (drivers). Intel recommends using the IFS Software Installation Textual User Interface (TUI) to preform this rebuild or reinstall. Refer to the *Intel® Omni-Path Fabric Software Installation Guide* for more information.

13.3.3 Intel® Omni-Path Interrupts Not Working

The driver cannot configure the Intel® Omni-Path link to a usable state unless interrupts are working. Check for this problem with the command:

```
$ grep hfi1 /proc/interrupts
```

Note: The output you see may vary depending on board type, distribution, or update level, and the number of CPUs in the system.

If there is no output at all, the driver initialization failed. For more information on driver problems, see Driver Load Fails Due to Unsupported Kernel on page 86 or Intel® Omni-Path HFI Initialization Failure on page 87.

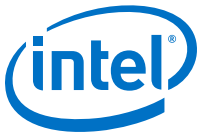
If the output is similar to one of these lines, then interrupts are not being delivered to the driver.

```
-MSI-edge    hfi1_0 sdma6
177:     0    0    0    PCI-MSI-edge    hfi1_0 sdma7
178:     0    0    0    PCI-MSI-edge    hfi1_0 sdma8
179:     0    0    0    PCI-MSI-edge    hfi1_0 sdma9
180:     0    0    0    PCI-MSI-edge    hfi1_0 sdma10
181:     0    0    0    PCI-MSI-edge    hfi1_0 sdma11
182:     0    0    0    PCI-MSI-edge    hfi1_0 sdma12
183:     0    0    0    PCI-MSI-edge    hfi1_0 sdma13
184:     0    0    0    PCI-MSI-edge    hfi1_0 sdma14
185:     0    0    0    PCI-MSI-edge    hfi1_0 sdma15
186:    39    0    0    PCI-MSI-edge    hfi1_0 kctxt0
187:     1   77    0    PCI-MSI-edge    hfi1_0 kctxt1
188:     0    0    0    PCI-MSI-edge    hfi1_0 kctxt2
```

A zero count in all CPU columns means that no Intel® Omni-Path interrupts have been delivered to the processor.

The possible causes of this problem are:

- Booting the Linux kernel with ACPI disabled on either the boot command line or in the BIOS configuration



- Other Intel® Omni-Path initialization failures

To check if the kernel was booted with the `noacpi` or `pci=noacpi` option, use this command:

```
$ grep -i acpi /proc/cmdline
```

If output is displayed, fix the kernel boot command line so that ACPI is enabled. This command line can be set in various ways, depending on your distribution. If no output is displayed, check that ACPI is enabled in your BIOS settings.

To track down other initialization failures, see Intel® Omni-Path HFI Initialization Failure on page 87.

13.3.4 OpenFabrics Load Errors if HFI Driver Load Fails

When the HFI driver fails to load, the other OpenFabrics drivers/modules will load and be shown by `lsmod`. But commands like `ibv_devinfo` will fail as follows:

-
- `ibv_devinfo`

  `libibverbs: Fatal: couldn't read uverbs ABI version.`

  `No Omni-Path devices found`

13.3.5 Intel® Omni-Path HFI Initialization Failure

There may be cases where the HFI driver was not properly initialized. Symptoms of this may show up in error messages from an Message Passing Interface (MPI) job or another program. Here is a sample command and error message:

```
$ mpirun -np 2 -m ~/tmp/mbu13 osu_latency
<nodename>:hfi_userinit: assign_port command failed: Network is down
<nodename>:can't open /dev/hfi1, network down
```

This will be followed by messages of this type after 60 seconds:

```
MPIRUN<node_where_started>: 1 rank has not yet exited 60 seconds after rank 0
(node
<nodename>) exited without reaching MPI_Finalize().
MPIRUN<node_where_started>:Waiting at most another 60 seconds for the remaining
ranks to do a clean shutdown before terminating 1 node processes.
```

If this error appears, check to see if the Intel® Omni-Path HFI driver is loaded by typing:

```
$ lsmod | grep hfi
```

If no output is displayed, the driver did not load for some reason. In this case, try the following commands (as root):

```
modprobe -v hfi1
lsmod | grep hfi1
dmesg | grep -i hfi1 | tail -25
```

The output will indicate whether the driver has loaded. Printing out messages using `dmesg` may help to locate any problems with the HFI driver.

If the driver loaded, but MPI or other programs are not working, check to see if problems were detected during the driver and Intel hardware initialization with the command:

```
$ dmesg | grep -i hfi1
```

This command may generate more than one screen of output.

Also, check the link status with the command:

```
$ hfi1_control -iv
```

13.3.6 MPI Job Failures Due to Initialization Problems

If one or more nodes do not have the interconnect in a usable state, messages similar to the following appear when the MPI program is started:

```
userinit: userinit ioctl failed: Network is down [1]: device init failed
userinit: userinit ioctl failed: Fatal Error in keypriv.c(520): device init failed
```

These messages may indicate that a cable is not connected, the switch is down, SM is not running, or that a hardware error occurred.

13.4 OpenFabrics and Intel® Omni-Path Issues

The following sections cover issues related to OpenFabrics (including Subnet Managers) and Intel® Omni-Path.

13.4.1 Stop Services Before Stopping/Restarting Intel® Omni-Path

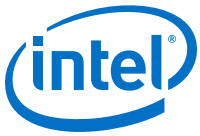
The Fabric Manager must be stopped before stopping/starting/restarting Intel® Omni-Pathsoftware.

User the systemctl command to stop or start the

```
# systemctl [start|stop|restart] opafm
```

To verify the status of the Fabric Manager run the following command:

```
# systemctl status opafm
```



13.5 System Administration Troubleshooting

The following section provides details on locating problems related to system administration.

13.5.1 Broken Intermediate Link

Sometimes message traffic passes through the fabric while other traffic appears to be blocked. In this case, MPI jobs fail to run.

In large cluster configurations, switches may be attached to other switches to supply the necessary inter-node connectivity. Problems with these inter-switch (or intermediate) links are sometimes more difficult to diagnose than failure of the final link between a switch and a node. The failure of an intermediate link may allow some traffic to pass through the fabric while other traffic is blocked or degraded.

If you notice this behavior in a multi-layer fabric, check that all switch cable connections are correct. Statistics for managed switches are available on a per-port basis, and may help with debugging. See your switch vendor for more information.

Intel recommends using FastFabric to help diagnose this problem.

13.6 Performance Issues

Optimizing Intel® Omni-Path Fabric performance and performance issues are discussed in the *Intel® Omni-Path Fabric Performance Tuning User Guide*.

14.0 Recommended Reading

Reference material for further reading is listed in the following sections.

- References for MPI , below
- Books for Learning MPI Programming, below
- References and Source for SLURM, below
- OpenFabrics
- Clusters
  - Networking on page 91
  - Other Software Packages on page 91

14.1 References for MPI

The MPI Standard specification documents are located at:

http://www.mpi-forum.org/docs

The MPICH implementation of MPI and its documentation are located at:

http://www-unix.mcs.anl.gov/mpi/mpich/

The ROMIO distribution and its documentation are located at:

http://www.mcs.anl.gov/romio

14.2 Books for Learning MPI Programming

Gropp, William, Ewing Lusk, and Anthony Skjellum, *Using MPI*, Second Edition, 1999, MIT Press, ISBN 0-262-57134-X

Gropp, William, Ewing Lusk, and Anthony Skjellum, *Using MPI-2*, Second Edition, 1999, MIT Press, ISBN 0-262-57133-1

Pacheco, *Parallel Programming with MPI,* 1997, Morgan Kaufman Publishers, ISBN 1-55860

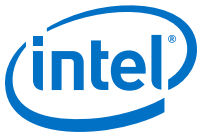
14.3 Reference and Source for SLURM

The open-source resource manager designed for Linux clusters is located at:

http://www.llnl.gov/linux/slurm/

14.4 OpenFabrics

Information about the OpenFabrics Alliance (OFA) is located at:



http://www.openfabrics.org

14.5 Clusters

Gropp, William, Ewing Lusk, and Thomas Sterling, *Beowulf Cluster Computing with Linux*, Second Edition, 2003, MIT Press, ISBN 0-262-69292-9

14.5.1 Networking

The Internet Frequently Asked Questions (FAQ) archives contain an extensive Request for Command (RFC) section. Numerous documents on networking and configuration can be found at:

http://www.faqs.org/rfcs/index.html

14.5.2 Other Software Packages

Environment Modules is a popular package to maintain multiple concurrent versions of software packages and is available from:

http://modules.sourceforge.net/