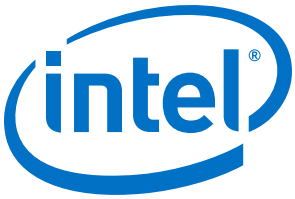


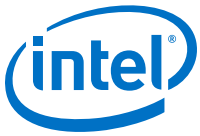
Intel® Omni-Path Fabric Host Software

User Guide

Rev. 5.0

December 2016

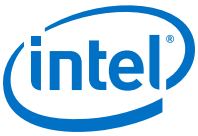
Order No.: H76470-5.0



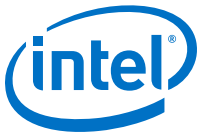
Revision History

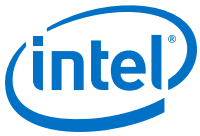
For the latest documentation, go to: http://www.intel.com/omnipath/FabricSoftwarePublications.

Date	Revision	Description
December 2016	5.0	Updates to this document include: • Added sections Intel® Omni-Path Architecture Overview and Host Software Stack. • Multi-Rail Support in PSM2: clarified behavior when `PSM2_MULTIRAIL` is not set in multiple fabrics. Added Multi-Rail Overview section. • Added hfidiags User Guide (moved from *Software Installation Guide* to this document). • Globally, updated the following filepaths: — from `/opt/opa` to `/usr/lib/opa` — from `/var/opt/opa` to `/var/usr/lib/opa` — from `/opt/opafm` to `/usr/lib/opa-fm` — from `/var/opt/opafm` to `/var/usr/lib/opa-fm` • Added Cluster Configurator for Intel® Omni-Path Fabric to Preface.
August 2016	4.0	Updates to this document include: • Added Setting up Open MPI with SLURM. • Removed pre-built PGI* MPIs from Installing SHMEM. The PGI* compiler is supported, however, pre-built PGI* MPIs are not included in the software package. • Added Intel® Omni-Path Routing Features and Innovations.
May 2016	3.0	Global updates to document.
February 2016	2.0	Global updates to document.
November 2015	1.0	Global updates to document.

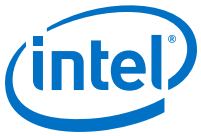


Contents





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 Intel® Omni-Path Host Fabric Interfaces (HFIs), Intel® Omni-Path 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.

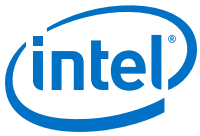
Intended Audience

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

Documentation Set

The complete end user publications set for the Intel® Omni-Path product includes the following items.

- 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*
- 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*
- *Intel® Omni-Path Storage Router Design Guide*
- *Building Lustre* Servers with Intel® Omni-Path Architecture Application Note*
- *Intel® Omni-Path Fabric Staging Guide*

Documents are available at the following URLs:

- Intel® Omni-Path Switches Installation, User, and Reference Guides

  http://www.intel.com/omnipath/SwitchPublications
- Intel® Omni-Path Host Fabric Interface Installation, User, and Reference Guides (includes software documents)

  http://www.intel.com/omnipath/FabricSoftwarePublications
- Drivers and Software (including Release Notes)

  http://www.intel.com/omnipath/Downloads

Cluster Configurator for Intel® Omni-Path Fabric

The Cluster Configurator for Intel® Omni-Path Fabric is available at: http://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-configurator.html.

This tool generates sample cluster configurations based on key cluster attributes, including a side-by-side comparison of up to four cluster configurations. The tool also generates parts lists and cluster diagrams.

Documentation Conventions

The following conventions are standard for Intel® Omni-Path documentation:

- *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 websites are also shown in blue. For example:

  See License Agreements on page 12 for more information.

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

Click the **Start** button, point to **Programs**, point to **Accessories**, and then click **Command Prompt**.

Press **CTRL+P** and then press the **UP ARROW** key.

- Text in `Courier` font indicates a file name, directory path, or command line text. For example:

  Enter the following command: `sh ./install.bin`

- Text in *italics* indicates terms, emphasis, variables, or document titles. For example:

  Refer to *Intel® Omni-Path Fabric Software Installation Guide* for details.

  In this document, the term *chassis* refers to a managed switch.

Procedures and information may be marked with one of the following qualifications:

- **(Linux)** – Tasks are only applicable when Linux* is being used.
- **(Host)** – Tasks are only applicable when Intel® Omni-Path Fabric Host Software or Intel® Omni-Path Fabric Suite is being used on the hosts.
- **(Switch)** – Tasks are applicable only when Intel® Omni-Path Switches or Chassis are being used.
- Tasks that are generally applicable to all environments are not marked.

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

This document, the *Intel® Omni-Path Fabric Host Software User Guide*, shows you how to set up and administer the fabric after the software has been installed. The audience for this document includes both cluster administrators and Message Passing Interface (MPI) application programmers, who have different but overlapping interests in the details of the technology.

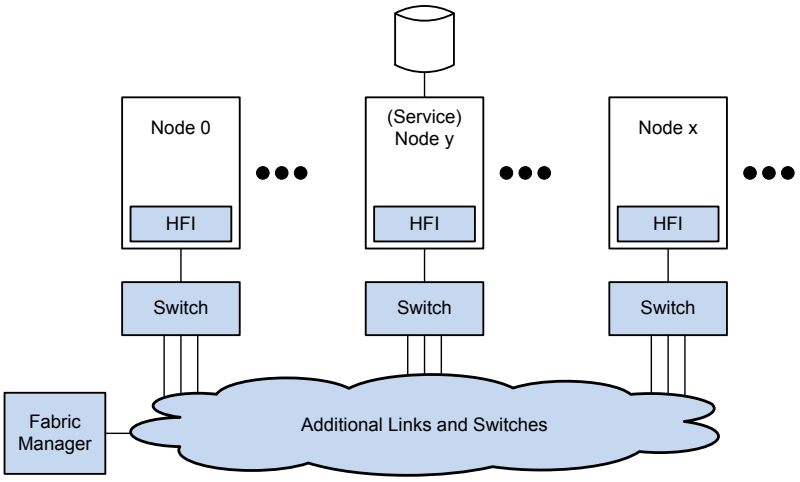
For installation details, see the following documents:

- *Intel® Omni-Path Fabric Software Installation Guide*
- *Intel® Omni-Path Fabric Switches Hardware Installation Guide*
- *Intel® Omni-Path Host Fabric Interface Installation Guide*

1.1 Intel® Omni-Path Architecture Overview

Intel® Omni-Path Architecture (Intel® OPA) is an end-to-end solution consisting of Intel® Omni-Path Host Fabric Interfaces (HFIs), Intel® Omni-Path switches, and fabric management and development tools. These building blocks are shown in the following figure.

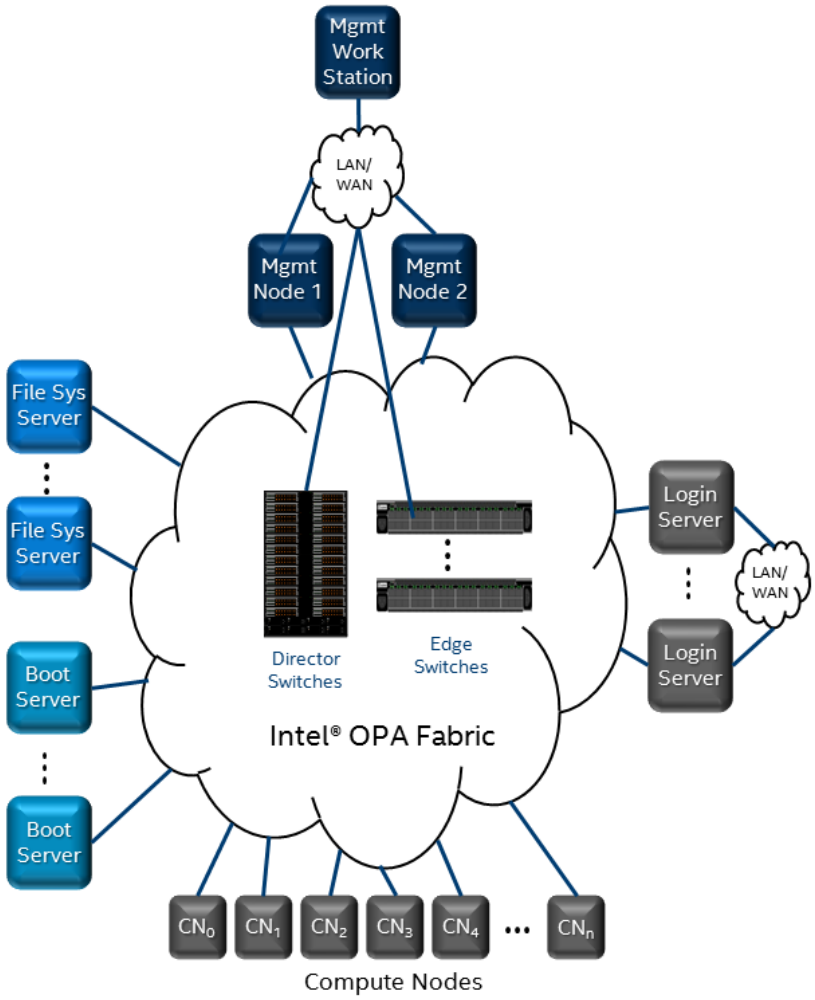
Figure 1. Intel® OPA Building Blocks



For software applications, Intel® OPA maintains consistency and compatibility with existing Intel® True Scale Fabric and InfiniBand* APIs utilizing the open source OpenFabrics Alliance* (OFA) software stack on Linux* distribution releases.

The following figure shows a sample Intel® OPA-based fabric, consisting of different types of nodes and servers.

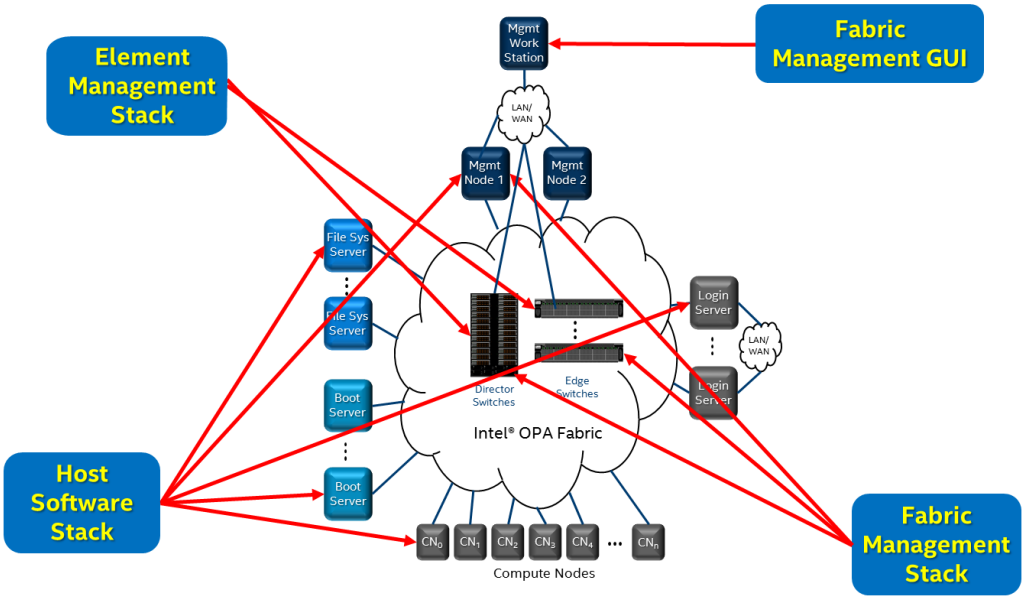
Figure 2. Intel® OPA Fabric



Software Components

The key software components and their usage models are shown in the following figure and described in the following paragraphs.

Figure 3. Intel® OPA Fabric and Software Components



Software Component Descriptions
Element Management Stack • Runs on an embedded Intel processor included in managed Intel® OP Edge Switch 100 Series and Intel® Omni-Path Director Class Switch 100 Series switches. • Provides system management capabilities, including signal integrity, thermal monitoring, and voltage monitoring, among others. • Accessed via Ethernet* port using command line interface (CLI) or graphical user interface (GUI). User documents: • *Intel® Omni-Path Fabric Switches GUI User Guide* • *Intel® Omni-Path Fabric Switches Command Line Interface Reference Guide*
Host Software Stack • Runs on all Intel® OPA-connected host nodes and supports compute, management, and I/O nodes. • Provides high performance, highly scalable MPI implementation via PSM2 and extensive set of upper layer protocols. • Includes Boot over Fabric mechanism for configuring a server to boot over Intel® Omni-Path using the Intel® OP HFI Unified Extensible Firmware Interface (UEFI) firmware. User documents: • *Intel® Omni-Path Fabric Host Software User Guide*

continued...

Software Component Descriptions
• *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide*
Fabric Management Stack • Runs on Intel® OPA-connected management nodes or embedded Intel processor on the switch. • Initializes, configures, and monitors the fabric routing, QoS, security, and performance. • Includes a toolkit for configuration, monitoring, diagnostics, and repair. User documents: • *Intel® Omni-Path Fabric Suite Fabric Manager User Guide* • *Intel® Omni-Path Fabric Suite FastFabric User Guide* • *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*
Fabric Management GUI • Runs on workstation with a local screen and keyboard. • Provides interactive GUI access to Fabric Management features such as configuration, monitoring, diagnostics, and element management drill down. User documents: • *Intel® Omni-Path Fabric Suite Fabric Manager GUI Online Help* • *Intel® Omni-Path Fabric Suite Fabric Manager GUI User Guide*

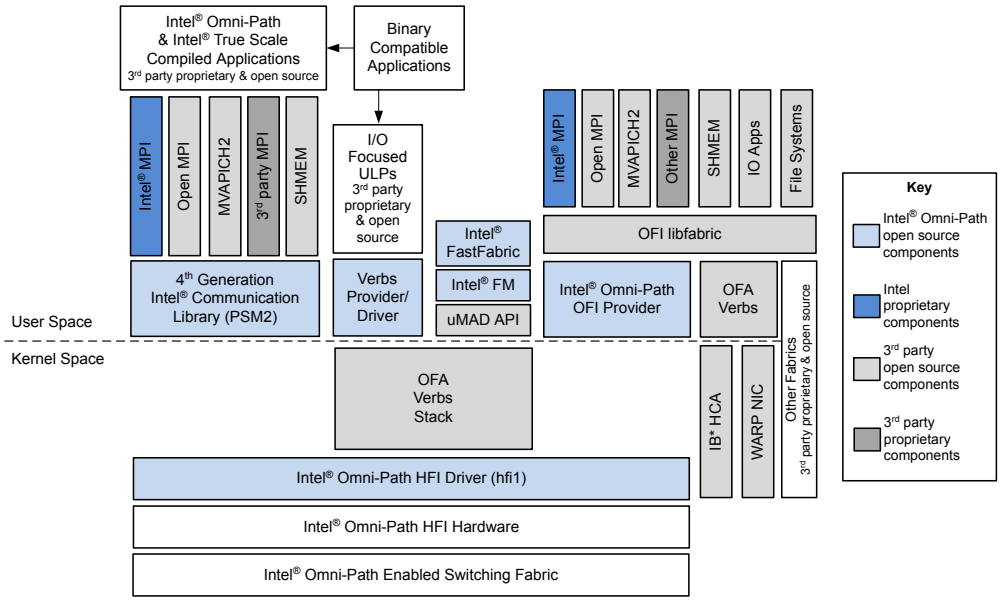
1.2 Host Software Stack

Host Software Stack Components

The design of Intel® OPA leverages the existing OpenFabrics Alliance* (OFA), which provides an extensive set of mature upper layer protocols (ULPs). OFA integrates fourth-generation proven, scalable PSM capability for HPC. The OpenFabrics Interface (OFI) API is aligned with application requirements.

Key elements are open source accessible, including the host software stack (via OFA*) and the Intel® Omni-Path Fabric Suite FastFabric Tools, Fabric Manager, and GUI. Intel® Omni-Path Architecture support is included in standard Linux* distributions and delta distribution of OFA stack atop Linux* distributions is provided as needed.

Figure 4. Host Software Stack Components

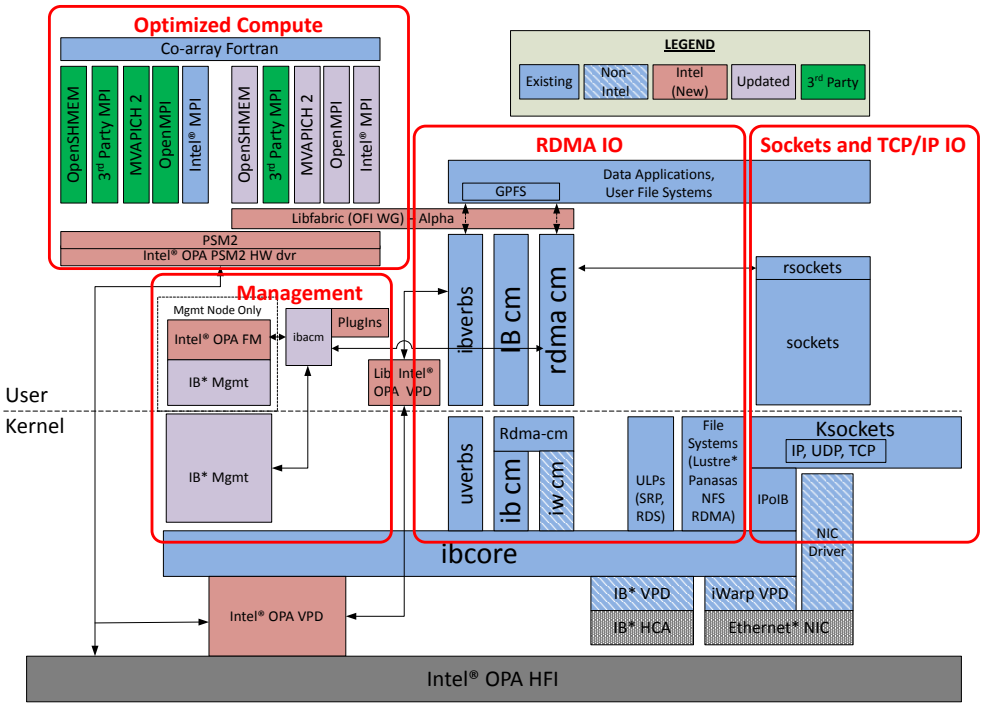


The Linux* kernel contains an RDMA networking stack that supports both InfiniBand* and iWarp transports. User space applications use interfaces provided by the OpenFabrics Alliance* that work in conjunction with the RDMA networking stack contained within the kernel.

The following figure illustrates the relationships between Intel® Omni-Path, the kernel-level RDMA networking stack, and the OpenFabrics Alliance* user space components. In the figure, items labelled as "Intel (new)" are new components written to use and manage the Intel® Omni-Path host fabric interface. Items marked as "non-Intel" are existing components that are neither used nor altered by the addition of Intel® Omni-Path to the Linux* environment.

Note: The following figure illustrates how Intel® Omni-Path components fit within the Linux* architecture, however, the figure does not show actual implementation or layering.

Figure 5. Host Software Stack - Detailed View



Most of the software stack is open sourced, with the exception of middleware that is not open sourced such as Intel® MPI Library and third-party software. The blue components show the existing open fabric components applicable for fabric designs. In the management grouping, the Fabric Manager and FastFabric tools are open source, 100% full feature capability. A new driver and a corresponding user space library were introduced for the Intel® Omni-Path hardware for the verbs path.

Intel is working with the OpenFabrics Alliance* (OFA) community, and has upstreamed new features to support Intel® Omni-Path management capabilities and introduced extensions around address resolution. Intel is also participating in the OFA libfabric effort which is a multi-vendor standard API that takes advantage of available hardware features and supports multiple product generations.

Host Software APIs

Application Programming Interfaces (APIs) provide a set of common interfaces for software-based applications and services to communicate with each other, and to access and use services within the software architecture. The following table provides a brief overview of the APIs supported by the Intel® Omni-Path Host Software.

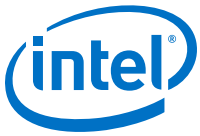


Table 1. APIs Supported by Host Software

API	Description
Sockets	Enables applications to communicate with each other using standard interfaces locally, or over a distributed network. Sockets are supported via standard Ethernet* NIC network device interfaces and IPoIB network device interfaces.
RDMA verbs	Provides access to RDMA capable devices, such as Intel® OPA, InfiniBand* (IB), iWarp, and RDMA over Converged Ethernet* (RoCE).
IB Connection Manager (CM)	Establishes connections between two verbs queue pairs on one or two nodes.
RDMA Connection Manager (CM)	Provides a sockets-like method for connection establishment that works in Intel® OPA, InfiniBand*, and iWarp environments.
Performance Scaled Messaging 2 (PSM2)	Provides a user-level communications interface for Intel® Omni-Path products. PSM2 provides HPC-middleware, such as MPI stacks, with mechanisms necessary to implement higher level communications interfaces in parallel environments.
OpenFabrics Interface (OFI) libfabric	Extends the OpenFabrics APIs to increase application support and improve hardware abstraction. Framework features include connection and addressing services, message queues, RDMA transfers, and others, as well as support for the existing verbs capabilities. OFI is an open source project that is actively evolving APIs, sample providers, sample applications, and infrastructure implementation.
NetDev	In Linux*, NetDev interfaces provide access to network devices, their features, and their state. For Intel® Omni-Path, NetDev interfaces are created for IP communications over Intel® OPA via IPoIB.
Linux* Virtual File System	Provides the abstraction for the traditional system calls of open, close, read, and write.
SCSI low-level driver	Implements the delivery of SCSI commands to a SCSI device.
Message Passing Interface (MPI)	Standardized API for implementing HPC computational applications. Supported implementations include: Intel® MPI Library, Open MPI, MVAPICH2, and others. See the *Intel® Omni-Path Fabric Software Release Notes* for the complete list.
OpenSHMEM	Provides one-sided communication, synchronization, collectives, locks, atomics, and memory management using a cluster-wide shared address space between processing elements (PEs). For Intel® Omni-Path, OpenSHMEM uses the Active Messaging APIs within PSM2.
Unified Extensible Firmware Interface (UEFI) Specification	Defines the specification for future BIOS work, including the multi-core processor (MCP) packaging of the HFI. The specification details booting, including booting over fabric. Intel® OPA includes a UEFI BIOS extension to enable PXE boot of notes over the Intel® OPA fabric.

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

Prerequisites

Make sure that hardware installation has been completed according to the instructions in the following documents:

- *Intel® Omni-Path Host Fabric Interface Installation Guide*
- *Intel® Omni-Path Fabric Switches Hardware Installation Guide*

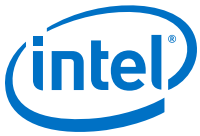
Make sure that software installation and driver configuration has been completed according to the instructions in the *Intel® Omni-Path Fabric Software Installation Guide*.

To minimize management problems, Intel recommends that the compute nodes of the cluster have similar hardware configurations and identical software installations.

Cluster Setup

Perform the following tasks when setting up the cluster:

1. Check that the BIOS is set properly according to the information provided in the *Intel® Omni-Path Fabric Performance Tuning User Guide*.
2. **Optional:** Set up the Distributed Subnet Administration Provider (DSAP) to correctly synchronize your virtual fabrics. See Intel Distributed Subnet Administration on page 27.
3. Set up automatic Node Description name configuration. See HFI Node Description Assignment.
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` using one of the following methods:
   - Use the `opasetupssh` CLI command. See the man pages or the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide* for details.
   - Use the FastFabric textual user interface (TUI) to set up `ssh`. See the *Intel® Omni-Path Fabric Suite FastFabric User Guide* for details.
6. Verify the cluster setup using the `opainfo` CLI command. See the man pages or the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*.



2.2 Using MPI

The instructions in this section use Open MPI as an example. Other MPIs, such as MVAPICH2 and Intel® MPI Library, may be used instead.

Prerequisites

Before you continue, the following tasks must be completed:

1. Verify that the Intel hardware and software has been installed on all the nodes.
2. Set up the host environment to use `ssh` on your cluster, using one of the following methods:

   Use the `opasetupssh` CLI command. See the man pages or the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide* for details.

   Use the FastFabric textual user interface (TUI) to set up `ssh`. See the *Intel® Omni-Path Fabric Suite FastFabric User Guide* for details.

Using MPI

Perform the following tasks:

1. Set up Open MPI. See Setting up Open MPI.
2. Compile Open MPI applications. See Compiling Open MPI Applications.
3. Create an `mpihosts` file that lists the nodes where your programs will run. See Creating the mpi_hosts File.
4. Run Open MPI applications. See Running Open MPI Applications.
5. Configure MPI programs for Open MPI. See Configuring MPI Programs for Open MPI.
6. To test using other MPIs that run over PSM2, such as MVAPICH2, and Intel® MPI Library, see Using Other MPIs.
7. Use the MPI Selector Utility to switch between multiple versions of MPI. See Managing MPI Versions with the MPI Selector Utility.
8. Refer to Intel® Omni-Path Cluster Setup and Administration, and the document *Intel® Omni-Path Fabric Performance Tuning User Guide* for information regarding fabric performance tuning.
9. Refer to Using Other MPIs 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 Installation Packages Overview

The following software installation packages are available for an Intel® Omni-Path Fabric:

- Intel® Omni-Path Fabric Host Software – This is the basic installation package that installs the Intel® Omni-Path Fabric Host Software components needed to set up compute, I/O, and Service nodes with drivers, stacks, and basic tools for local configuration and monitoring.
- Intel® Omni-Path Fabric Suite (IFS) Software – This installation package provides special features and includes the Intel® Omni-Path Fabric Host Software package, along with the Intel® Omni-Path Fabric Suite FastFabric Toolset (FastFabric) and the Intel® Omni-Path Fabric Suite Fabric Manager (Fabric Manager).
- Intel® Omni-Path Fabric Suite Fabric Manager GUI (Fabric Manager GUI) – This installation package provides a set of features for viewing and monitoring the fabric or multiple fabrics, and is installed on a computer outside of the fabric.

3.2 Installed Layout

As described in the previous section, there are several installation packages. Refer to the *Intel® Omni-Path Fabric Software Installation Guide* for complete instructions.

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

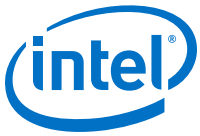
Table 2. Installed Files and Locations

File Type	Location
Intel-supplied Open MPI and MVAPICH2 RPMs	Compiler-specific directories using the following format: `/usr/mpi/<compiler>/<mpi>-<mpi_version>-hfi` For example: `/usr/mpi/gcc/openmpi-X.X.X-hfi`

Utility	`/usr/sbin`
	`/usr/lib/opa/*`
	`/usr/bin`
	`/usr/lib/opa-fm`

Documentation	`/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. See Documentation Set on page 10 for URLs.

continued...



File Type	Location
Configuration	/etc/sysconfig /etc/sysconfig/opa /etc/rdma
Initialization	/etc/init.d
Intel® Omni-Path Fabric Driver Modules	/lib/modules/<kernel name>/updates/drivers/infiniband/hw/hfi1
Other Modules	/usr/lib/opa /lib/modules/*<kernel name>*/updates/kernel/drivers/net /lib/modules/*<kernel name>*/updates/kernel/net/rds

3.3 Intel® Omni-Path Fabric and OFA Driver Overview

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

A list of optional configurable OFA components and their default settings follows:

- 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. You must use the Intel® Omni-Path Fabric Suite Fabric Manager which may be installed on a host or may be run inside selected switch models, for smaller fabrics.

- 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 also be configured and enabled, as described in Configuring IPoIB Network Interface.

3.4 Configuring IPoIB Network Interface

The following instructions show you how to manually configure your OpenFabrics Alliance* (OFA) IPoIB network interface. Intel recommends using the Intel® Omni-Path Software Installation package for installation of the software, including setting up IPoIB, or using the `opaconfig` tool if the Intel® Omni-Path Software has already been installed on your management node. For larger clusters, Intel® Omni-Path Fabric Suite FastFabric can be used to automate installation and configuration of many nodes. These tools automate the configuration of the IPoIB network interface.

This example assumes the following:

- Shell is either `sh` or `bash`.
- All required Intel® Omni-Path and OFA RPMs are installed.
- Your startup scripts have been run, either manually or at system boot.

- The IPoIB network is 10.1.17.0, which is one of the networks reserved for private use, and thus not routeable on the Internet. The network has a /8 host portion. In this case, the netmask must be specified.
- The host to be configured has the IP address 10.1.17.3, no hosts files exist, and DHCP is not used.

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

Perform the following steps:

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

   ```
   ip addr add 10.1.17.3/255.255.255.0 dev ib0
   ```

   Note: You can configure/reconfigure the IPoIB network interface from the TUI using the `opaconfig` command.

2. Bring up the link with the command:

   ```
   ifup ib0
   ```

3. To verify the configuration, use the command:

   ```
   ip addr show ib0
   ```

   The output from this command is similar to:

   ```
   ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 2044 qdisc pfifo_fast state UP qlen
   256
   link/infiniband 80:00:00:02:fe:80:00:00:00:00:00:00:00:11:75:01:01:6a:36:83
   brd
   00:ff:ff:ff:ff:12:40:1b:80:01:00:00:00:00:00:00:ff:ff:ff:ff
   inet 10.1.17.3/24 brd 10.1.17.255 scope global ib0
   valid_lft forever preferred_lft forever
   inet6 fe80::211:7501:16a:3683/64 scope link
   valid_lft forever preferred_lft forever
   ```

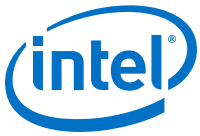
4. Enter the command:

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

   The output of the `ping` command is 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
   64 bytes from 10.1.17.7: icmp_seq=0 ttl=64 time=0.073 ms
   ```

   The IPoIB network interface is now configured.



Note: Connected Mode is enabled by default if you are using the `INSTALL` script to create `ifcfg` files via the setup IPoIB menus/prompts, or using the Intel® Omni-Path Fabric Installer TUI to install other nodes in the fabric. In such cases, the setting in `/etc/sysconfig/network-scripts/ ifcfg-ib0` is `CONNECTED_MODE=yes`. However, if loading rpms directly via a provisioning system or using stock distro mechanisms, it defaults to datagram mode. In such cases, the setting is `CONNECTED_MODE=no`. In addition, the `Connected Mode` setting can also be changed when asked during initial installation via `./INSTALL` or using the `FF_IPOIB_CONNECTED` option in `opafastfabric.conf` when using FastFabric to install other nodes in the fabric.

3.5 Configuring IPoIB Driver

Intel recommends using the FastFabric TUI or the `opaconfig` command to configure the IPoIB driver for autostart. Refer to the *Intel® Omni-Path Fabric Suite FastFabric User Guide* for more information on using the 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. An example of the `ifcfg-NAME` file follows:

```
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
CONNECTED_MODE=yes
```

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

   In the `ifcfg` file, the following options are listed by default:

   - `ONBOOT=yes` This is a standard network option that tells the system to activate the device at boot time.
   - `NM_CONTROLLED=no` This option prohibits the Network Manager (NM) daemon from controlling the IPoIB interface state.
   - `CONNECTED_MODE=yes` This option controls IPoIB interface transport mode.

     `yes` [default] sets Reliable Connection (RC) mode and `no` sets Unreliable Datagram (UD) mode.

2. Bring up the IPoIB interface with the following command:

```
ifup <interface name>
```

   For example:

```
ifup ib0
```

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, a server reboot is required.

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 `yes` in `/etc/sysconfig/boot`:

```
RUN_PARALLEL="yes"
```

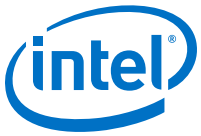
3.6.2 Verifying IB Bonding Configuration

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 processes. When a MPI job is started across the entire fabric, each process needs to collect path records for every other node in the fabric. For this example, 4 processes x 1000 nodes = 4000 processes. Each of those processes tries to contact all other processes to get the path records from its own node to the nodes for all other processes. That is, each process needs to get the path records from its own node to the 3999 other processes. This amounts to a total of nearly 16 million path queries just to start the job. Each process queries the subnet manager for these path records at roughly the same time.

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` path record caching 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 most beneficial on nodes running OpenSHMEM* 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.

Any application that uses `rdmacm` also uses the DSAP plugin for path record caching, assuming the FM is configured with the `rdmacm` SID range correctly. For details, see Service ID (SID).

Other applications must be modified specifically to take advantage of it.

If you plan to use the DSAP plugin, refer to the header file `/usr/include/Infiniband/opasadb_path.h` for information. 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

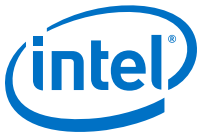
The Distributed Subnet Administration Provider (DSAP) configuration file is `/etc/rdma/dsap.conf` and it contains a brief description of each setting. Typically administrators only need to deal with several of the options included in the file.

Refer to the following subsections for details on the recommended settings.

3.7.2.1 Service ID (SID)

The SID is the primary configuration setting for the DSAP. SIDs identify applications that 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 is used to identify an application. 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 OPA libraries treat zero values as unspecified.

3.7.2.2 ScanFrequency

Periodically, the DSAP completely re-synchronizes 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

Most DSAP events are logged into `/var/log/ibacm.log`, and the message output is controlled by the `log_level` parameter in `/etc/rdma/ibacm_opts.cfg`. When this parameter is given, the I/O path events are redirected to the specified log file. Error events in the I/O path library, `oibutils/libibumad`, are printed to the screen by default.

```
LogFile=/var/log/ibacm.log
```

The `LogFile` and `Dbg` parameters are used primarily for debugging purposes.

3.7.2.4 Dbg

This parameter controls whether the DSAP records a normal level or a debugging level of logging information.

To change the `Dbg` setting for DSAP, find the line in `dsap.conf` that reads `Dbg=n`, and change the parameter to the desired value.

Valid options for `Dbg` include:

- `Dbg = 1-6` Normal Logging

  Errors, warnings, events, status and operation information are logged. Any value between 1 and 6, inclusive, has the same effect.

- `Dbg = 7` Debug Logging

  Debug logging should only be turned if requested by Intel Support, because it generates so much information that system operation is impacted.

The `LogFile` and `Dbg` parameters are used primarily for debugging purposes.

3.7.3 Virtual Fabrics and the Distributed SA Provider

Open Fabrics Alliance* 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 only stores one copy of each path record within a Virtual Fabric. SIDs that match more than one Virtual Fabric are associated with a single Virtual Fabric. The method for handling cases of multiple VFs matching the same SID is discussed later in this document. The Virtual Fabrics that do not match SIDs in the DSAP's database are ignored.

3.7.3.1 Configuring the DSAP

To minimize the number of queries made by the DSAP, it is important to configure it correctly, both matching the configuration of the Fabric Manager (FM) and excluding portions of the fabric not used by applications using the DSAP.

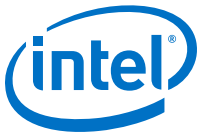
If you want to include an application in DSAP, you must do both of the following steps:

- include the Service ID (SID) of the application in `etc/rdma/dsap.conf`
- include the application in the FM configuration file `/etc/sysconfig/opafm.xml`

DSAP Configuration Example

The default `dsap.conf` file's SID section looks like this sample:

```
#PSM MPI SID range.
# PSM_Control
      SID=0x1000117500000000
      SID=0x1000117500000001
      SID=0x1000117500000002
      SID=0x1000117500000003
      SID=0x1000117500000004
      SID=0x1000117500000005
      SID=0x1000117500000006
      SID=0x1000117500000007

# PSM_Data
      SID=0x1000117500000008
      SID=0x1000117500000009
      SID=0x100011750000000a
      SID=0x100011750000000b
      SID=0x100011750000000c
      SID=0x100011750000000d
      SID=0x100011750000000e
      SID=0x100011750000000f

#PSM MPI SID range #2
# PSM_Control
```



```
      SID=0x1
      SID=0x2
      SID=0x3
      SID=0x4
      SID=0x5
      SID=0x6
      SID=0x7
# PSM_Data
      SID=0x9
      SID=0xa
      SID=0xb
      SID=0xc
      SID=0xd
      SID=0xe
      SID=0xf
```

By default, DSAP enables the PSM application by including the SIDs for PSM. However, the FM configuration file **does not** enable the `PSM_Compute` virtual fabric by default, as shown in this sample:

```
<VirtualFabric>
      <Name>PSM_Compute</Name>
      <Enable>0</Enable>
                        <Security>1</Security>
      <QOS>1</QOS>
      <Bandwidth>70%</Bandwidth>
      <!-- <Member>compute_nodes</Member> -->
      <Member>All</Member>    <!-- can be reduced in scope if desired -->
      <Application>PSM</Application>
</VirtualFabric>
```

In this example, the application PSM contains all the SIDs defined by `dsap.conf`:

```
<Application>
      <Name>PSM</Name>
      <ServiceIDRange>0x1000117500000000-0x100011750000000f</ServiceIDRange>
      <!-- for simplicity, when serviceId is needed on mpirun command line -->
      <!-- we also allow serviceId 0x1 to 0xf -->
      <ServiceIDRange>0x1-0xf</ServiceIDRange>
    </Application>
```

Therefore, to skip an application in DSAP, simply remove the SIDs of the application from `dsap.conf`. The application SIDs are generally defined by FM in the `/etc/sysconfig/opafm.xml` file.

DSAP Configuration BKMs

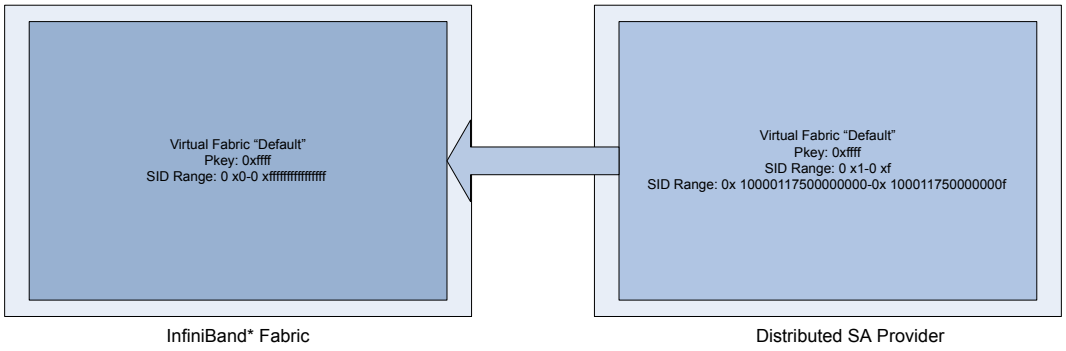
The following items are some Best Known Methods (BKMs) for configuring DSAP:

- When using PSM without the PSM arguments to use DSAP, you can remove PSM in the `dsap.conf` file.
- Applications that rarely establish new connections such as SRP can also be removed from the `dsap.conf` file.

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. 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 6. 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, you do not need to modify the DSAP configuration file. Be aware that the DSAP has restricted the range of applicable SIDs to those defined in its configuration file. Attempts to get path records using other SIDs do not work, even if those other SIDs are valid for the fabric. When using this default configuration, MPI applications can only be run using one of these 31 SIDs.

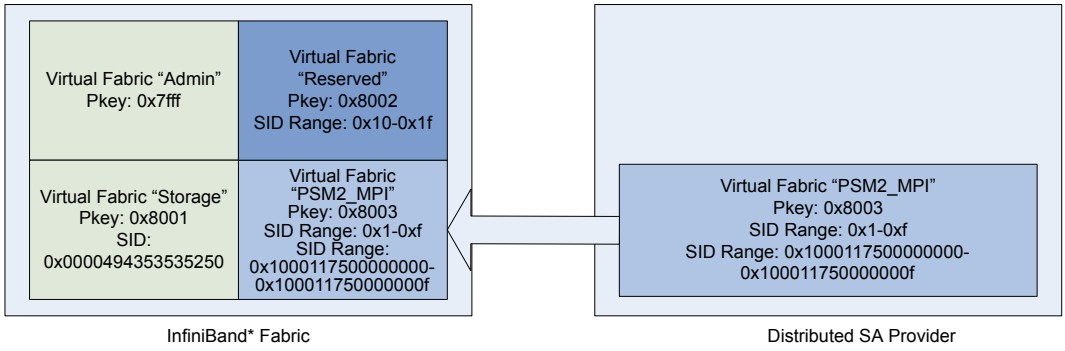
3.7.3.3 Multiple Virtual Fabrics Example

If you are configuring the physical fabric, you may want to add other Virtual Fabrics to the default configuration.

In the following figure, the administrator has divided the physical fabric into four virtual fabrics:

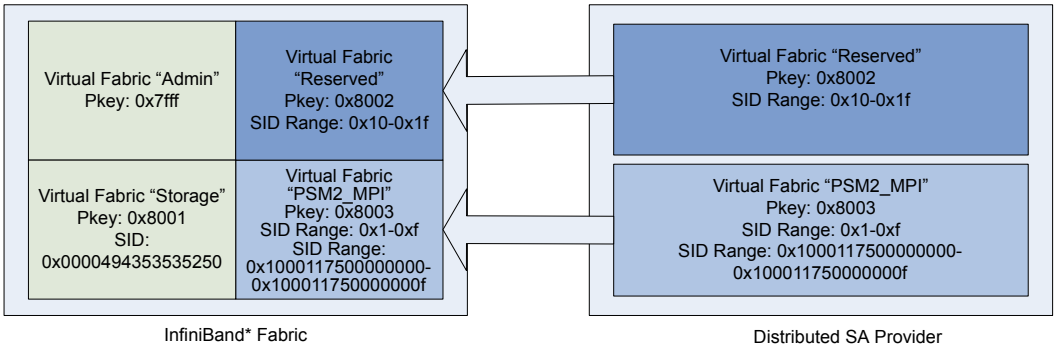
- Admin: communicates with the Fabric Manager
- Storage: used by SRP
- PSM2_MPI: used by regular MPI jobs
- Reserved: for special high-priority jobs

Figure 7. Distributed SA Provider Multiple Virtual Fabrics Example



In this example, the DSAP was not configured to include the SID Range 0x10 through 0x1f, therefore it has ignored the Reserved virtual fabric. To resolve this issue, add those SIDs to the `dsap.conf` file as shown in the following figure.

Figure 8. Distributed SA Provider Multiple Virtual Fabrics Configured Example

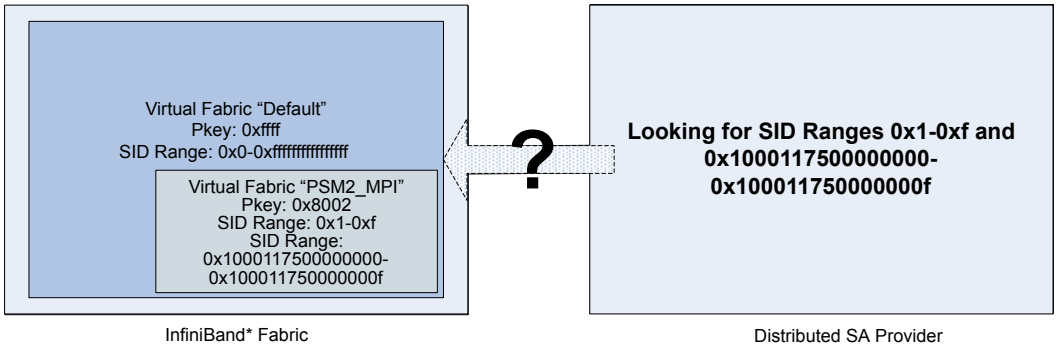


3.7.3.4 Virtual Fabrics with Overlapping Definitions

This section describes some examples of overlapping definitions which can occur when SIDs are used in more than one virtual fabric.

In the following example, 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.

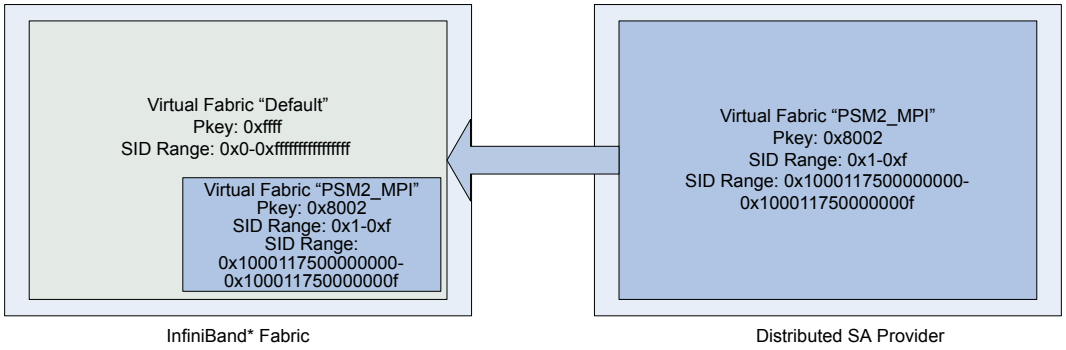
Figure 9. Virtual Fabrics with Overlapping Definition for PSM2_MPI



When a path query is received, the DSAP deals with this conflict as follows:

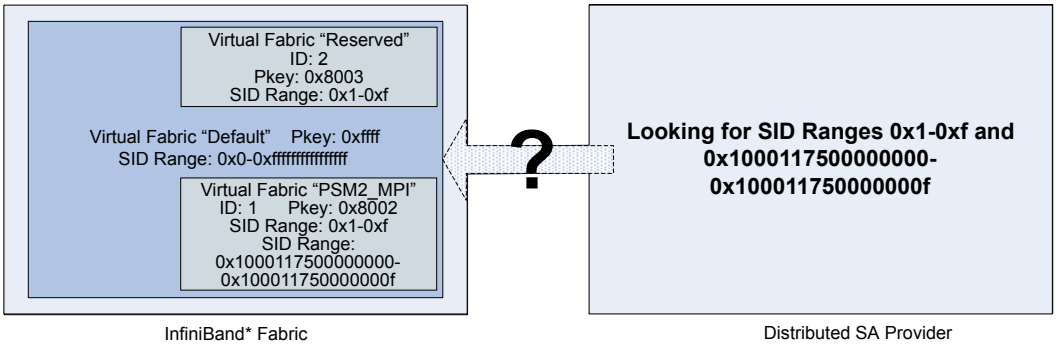
Any Virtual Fabric with a pkey of 0xffff or 0x7fff is considered to be an Admin or Default Virtual Fabric. This Admin or Default Virtual Fabric is treated as a special case by the DSAP and is used only as a last resort. Stored SIDs are only mapped to the default Virtual Fabric if they do not match any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

Figure 10. Virtual Fabrics with all SIDs assigned to PSM2_MPI Virtual Fabric



In the following example, the fabric administrator created two different Virtual Fabrics without turning off the Default and two of the new fabrics have overlapping SID ranges.

Figure 11. Virtual Fabrics with PSM2_MPI Virtual Fabric Enabled



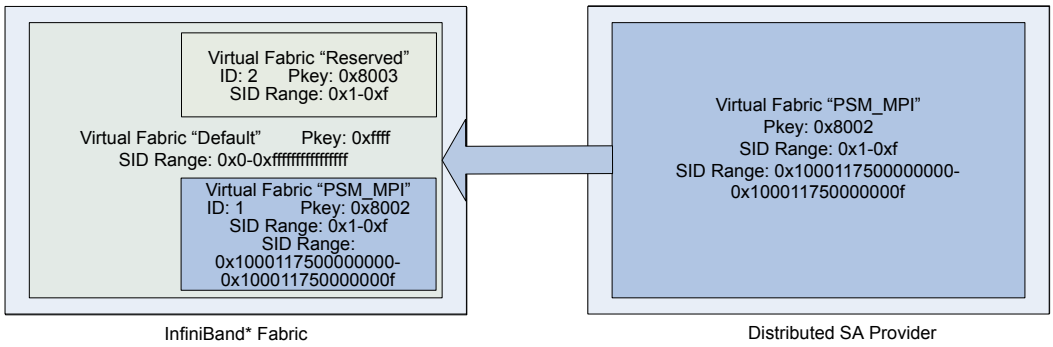
When a path query is received, the DSAP deals with this conflict as follows:

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 that the Virtual Fabrics appear in the configuration file. These indexes can be seen by using the `opasaquery -o vfinfo` command. The DSAP always assigns a SID to the Virtual Fabric with the lowest index. This ensures that all copies of the DSAP in the fabric 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.

In the following figure, 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 is ever a member of two Virtual Fabrics that service the same SID.

Figure 12. Virtual Fabrics with Unique Numeric Indexes



3.7.3.5 Configuring DSAP for AutoStart

When using Virtual Fabrics in conjunction with Intel HFIs and PSM2 with PathRecord query enabled, Intel recommends that you enable the DSAP for autostart on all the compute nodes to simplify the operation of MPI jobs.

You can use any of the following methods to enable DSAP for autostart:

- Enable OpenFabrics Alliance* IBACM (`ibacm`) in the Intel OPA Autostart Menu when installing, upgrading, or running the `opaconfig` command.
- Add `-E dist_sa` to the `FF_INSTALL_OPTIONS` and `FF_UPGRADE_OPTIONS` in `opafastfabric.conf` when using FastFabric to install or upgrade all the compute nodes.
- Use the `opacmdall` command or any other distributed shell to perform a `opaconfig -E dist_sa` operation on all nodes.

3.8 HFI Node Description Assignment

Node Description names can be configured in many ways. For Intel® Omni-Path, Intel recommends the use of the `rdma-ndd` daemon to keep the Node Description up to date with the *hostname* of the node.

The Intel® Omni-Path Fabric Suite includes the `rdma-ndd` daemon in the infiniband-diags package for convenience. For details on `rdma-ndd`, see the man page.

3.9 MTU Size

Intel® OP Software provides central configuration for the Maximum Transfer Unit (MTU) based on individual *virtual fabrics*, or vFabrics. Refer to the *Intel® Omni-Path Fabric Suite Fabric Manager User Guide* for information on configuring vFabrics, including setting MTU size to some value other than the default. The values may be set to one of the following: 2048 bytes, 4096 bytes, and 8192 bytes.

Note: Intel recommends that you use the default MTU size of 8K for the best overall fabric performance.

Applications that use `librdmacm`, for example, PSM2 and RDMA, automatically obtain the MTU information from the Fabric Manager. For other applications, configuration of MTU varies.

3.10 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 typically 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`.

Typically, 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.

Existing configuration files are not overwritten during an upgrade operation.

3.10.1 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 driver runs, for example, at system startup. The file system is unmounted when the driver stops running, for example, at system shutdown.

A sample layout of a system with two cards is shown in the following example:

```
/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` contains counters for the device. These counters include interrupts received, bytes and packets in and out, and others.
- `port2counters` contains counters for the device. These counters include interrupts received, bytes and packets in and out, and others.
- `flash` 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.

Tthe file `driver_stats_names` provides the names for the stats in the binary `driver_stats` files.

3.10.2 More Information on Configuring and Loading Drivers

See the `modprobe(8)`, `modprobe.conf(5)`, and `lsmod(8)` man pages for more information.

Also refer to the `/usr/share/doc/initscripts-*/sysconfig.txt` file for 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 an Intel® True Scale adapter card to an Intel® Omni-Path switch. Likewise, you cannot connect an Intel® Omni-Path adapter card to an 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.

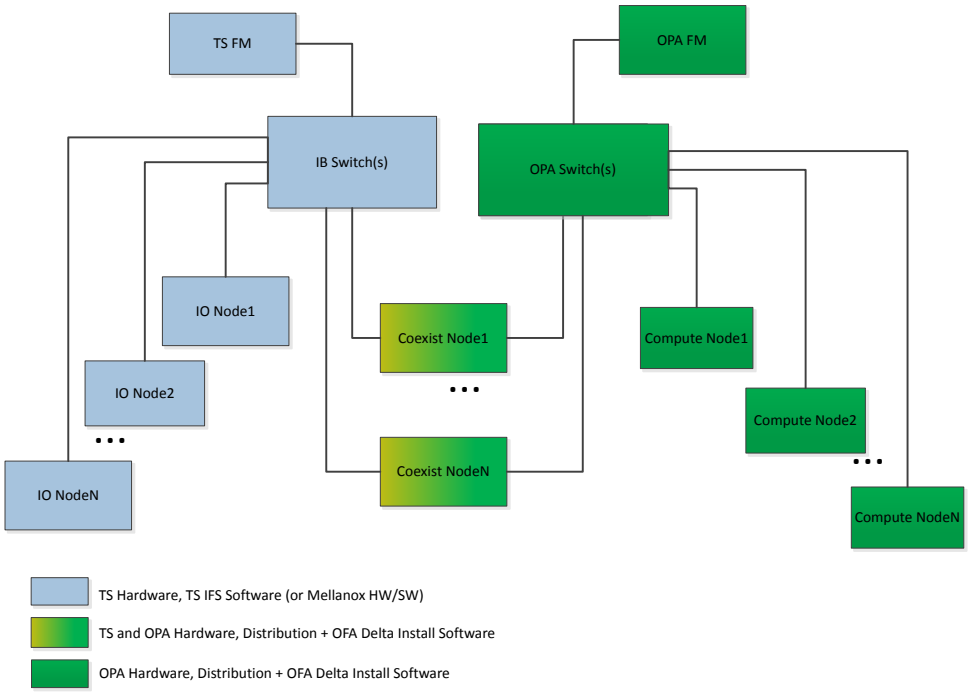
4.1 Coexist Nodes

In a mixed fabric (Intel® True Scale and Intel® Omni-Path coexisting), certain nodes may 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

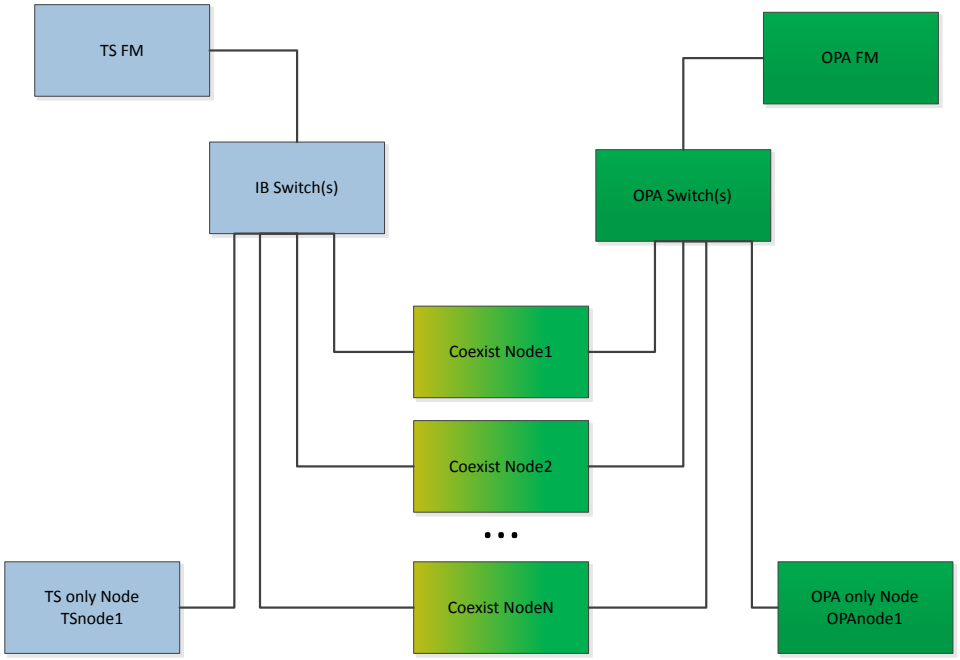
There are two supported scenarios for creating Intel® True Scale/Intel® Omni-Path coexistence configurations. The first is where the Intel® True Scale hardware serves as an InfiniBand* storage network, and the Intel® Omni-Path hardware is used for computing. This configuration allows for the continued 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 13. 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 14. Intel® True Scale/Omni-Path Rolling Upgrade

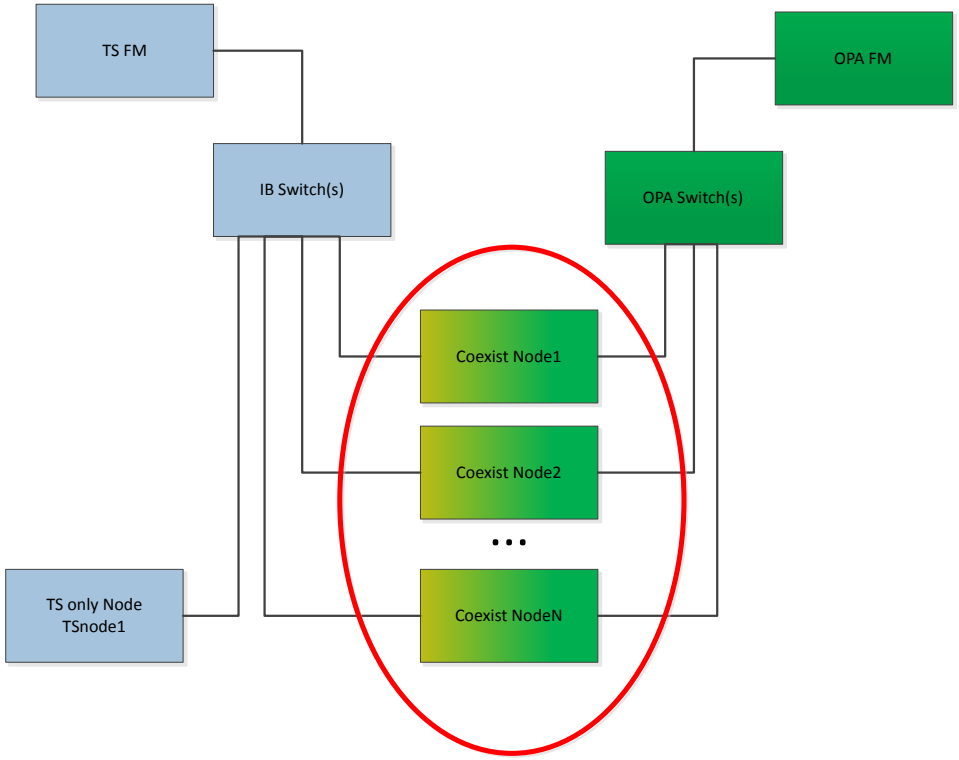


4.3 Coexist Node Details

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

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

Figure 15. Coexist Node Details

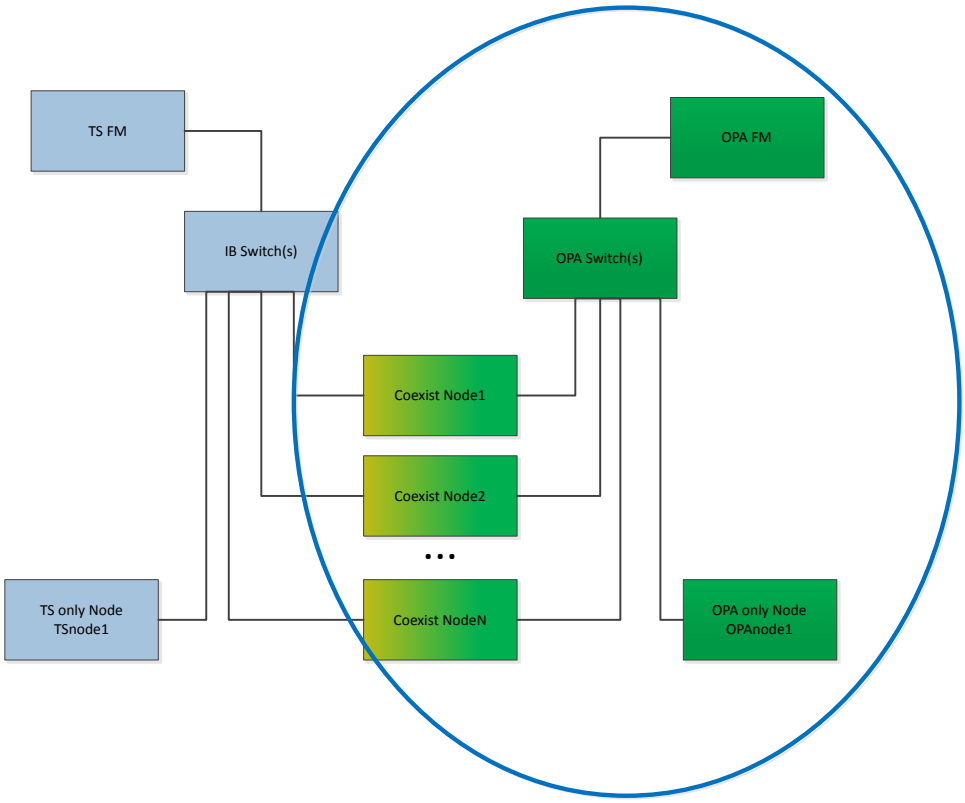


4.4 Intel® Omni-Path Node Details

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

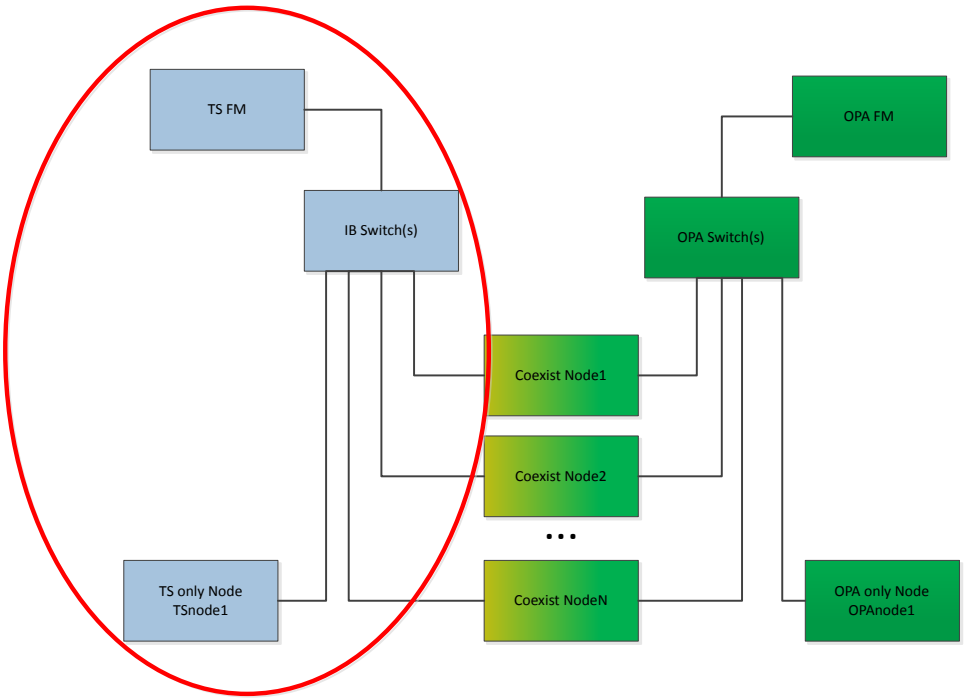
Figure 16. Intel® Omni-Path Node Details



4.5 Intel® True Scale Node Details

Full support for Intel® True Scale Fabric Suite (IFS) software is available on the Intel® True Scale Fabric management node. This includes support for OFA and all Intel® True Scale Fabric Suite FastFabric 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 17. Intel® True Scale Node Details

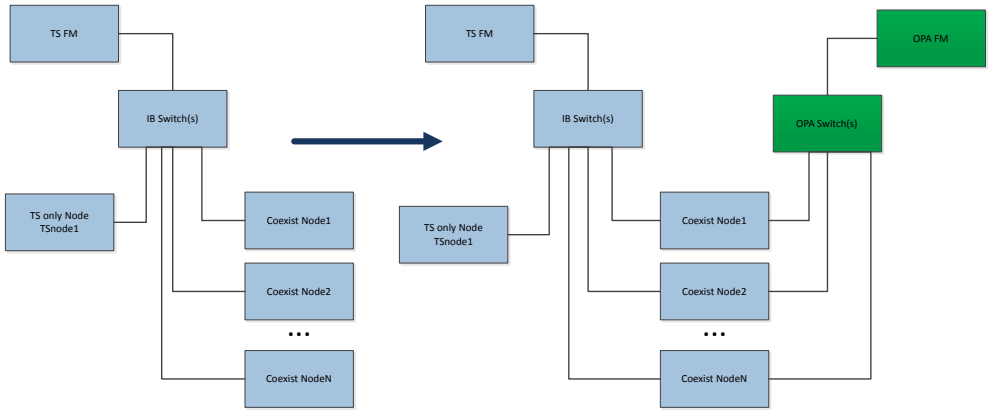


4.6 Installing on an Existing Intel® True Scale Cluster

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

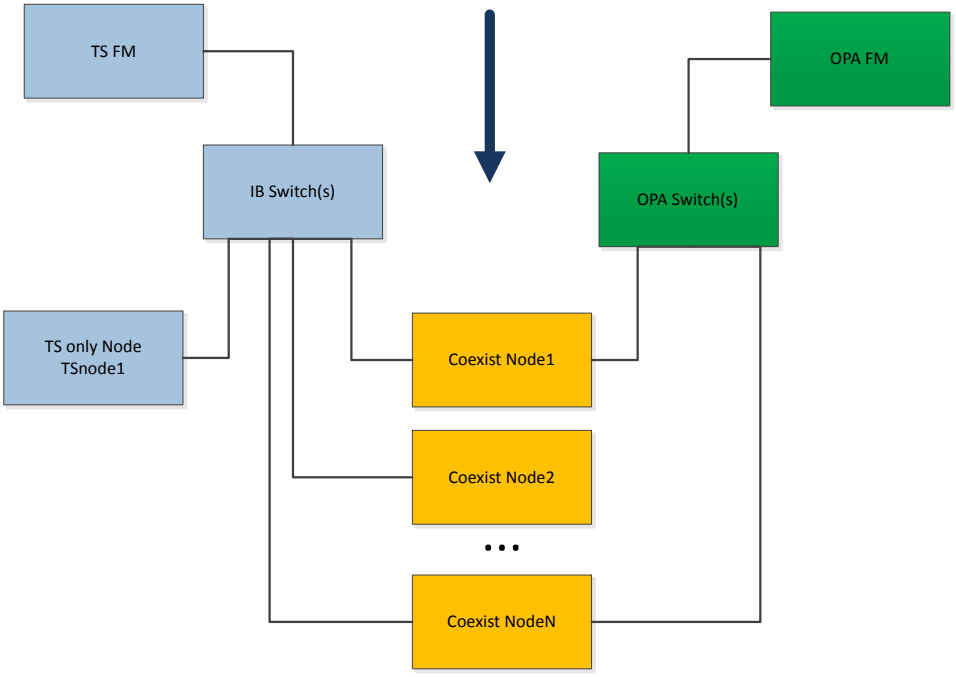
Add an Intel® Omni-Path Fabric management node, Intel® Omni-Path switches, and an Intel® Omni-Path HFI to the Intel® True Scale nodes that are to operate as dual mode nodes. With the hardware in place and connected, install the standard Intel® Omni-Path Fabric Suite software on the management node. Follow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

Figure 18. Adding Intel® Omni-Path Hardware and Software



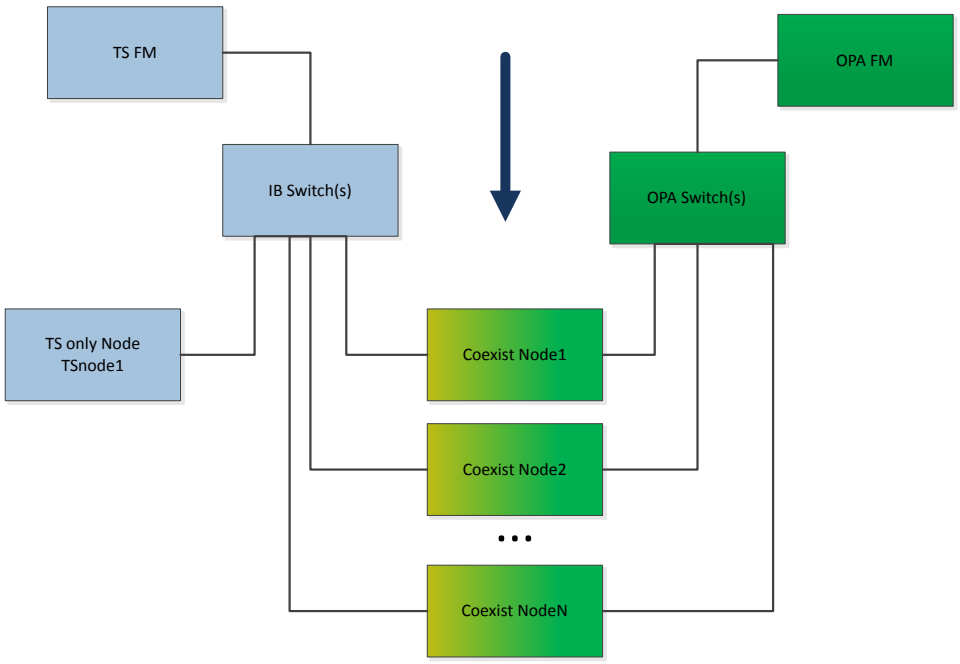
Next, install the Base Distro (RHEL* or SLES*) onto the coexist nodes. Note that Intel® True Scale IFS should *not* be installed on the coexist nodes.

Figure 19. Adding the Base Distro



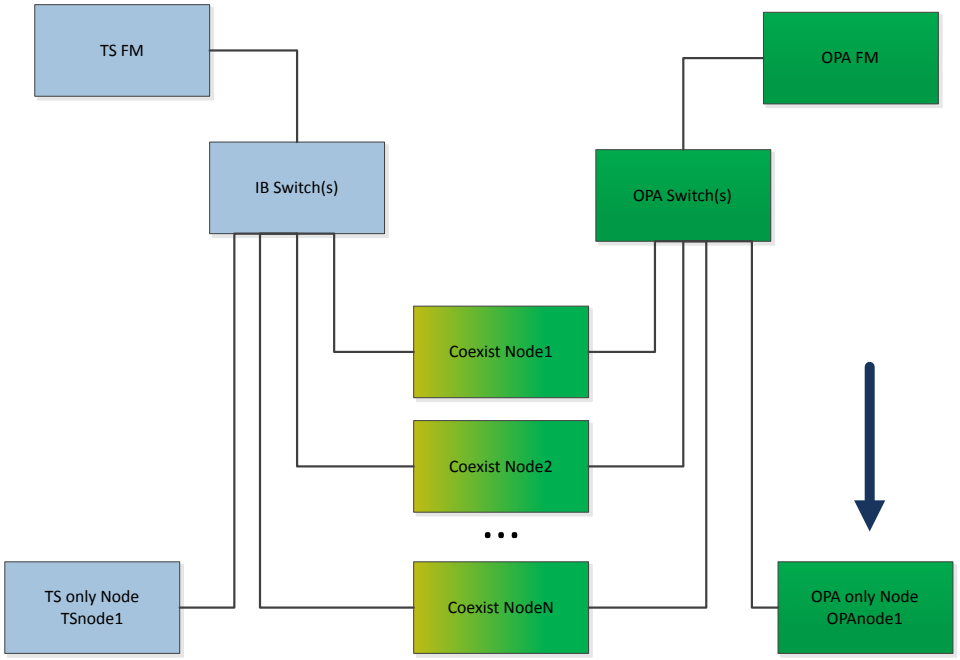
Install Intel® Omni-Path Basic software on the coexist nodes. Follow the standard Intel® Omni-Path installation procedures, as described in the *Intel® Omni-Path Fabric Switches Hardware Installation Guide*, the *Intel® Omni-Path Host Fabric Interface Installation Guide*, and the *Intel® Omni-Path Fabric Software Installation Guide*.

Figure 20. Adding Basic Software



Add Intel® Omni-Path-only nodes to the Intel® Omni-Path Fabric. Follow the standard Intel® Omni-Path installation procedures, as described in the *Intel® Omni-Path Fabric Switches Hardware Installation Guide*, the *Intel® Omni-Path Host Fabric Interface Installation Guide*, and the *Intel® Omni-Path Fabric Software Installation Guide*.

Figure 21. Adding Nodes to Intel® Omni-Path Fabric



4.7 PSM2 Compatibility

For enhanced compatibility, Open MPI can be recompiled to run on the same node at the same time on both Intel® True Scale and Intel® Omni-Path hardware. Note that updating Intel® MPI Library to version 5.1 Gold allows Intel® True Scale and Intel® Omni-Path hardware to operate simultaneously without using the `compat` package.

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

Table 3. PSM and PSM2 Compatibility Matrix

Library	PSM (Intel® True Scale Hardware)	PSM-Compat (Intel® Omni-Path Hardware)	PSM2 (Intel® Omni-Path Hardware)
Recompiled for PSM2			
Open MPI	X		X
MVAPICH2			X
Intel® MPI Library 5.1 Gold	X		X
Existing version			
Open MPI	X	X	
MVAPICH2	X	X	
Intel® MPI Library	X	X	

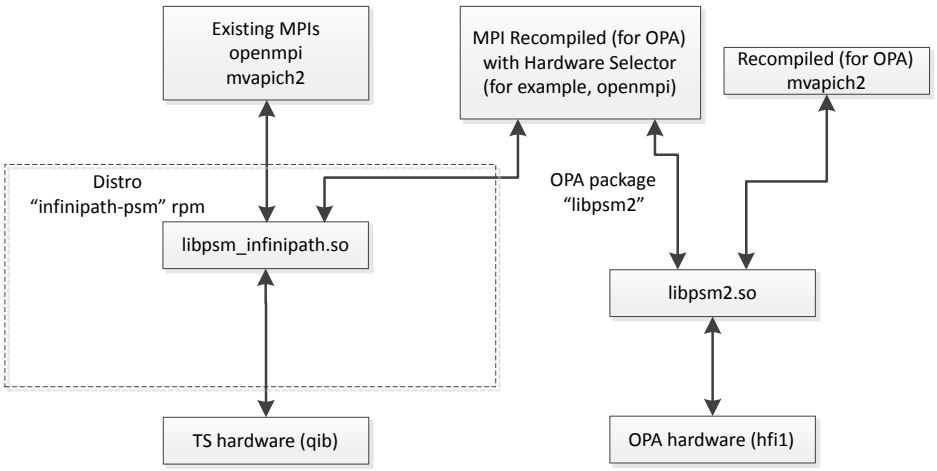
4.7.1 PSM2 Standard Configuration

Existing MPIs that were compiled for Intel® True Scale can be linked to an optionally-installed `compat` RPM using an `LD_LIBRARY_PATH` directive. The `compat` RPM contains a library that has a binary-compatible API for linking old MPIs into PSM2.

Intel® Omni-Path Fabric Suite defaults to the `compat` RPM library, including new MPIs that are compiled against the new PSM2 library. MPIs such as OpenMPI, which have runtime selectors of the underlying hardware, can be used to select the hardware using their mechanisms. 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 Intel® True Scale hardware is selected.

Figure 22. PSM2 Standard Configuration



4.7.2 Using the PSM2 Interface on Intel® Omni-Path Hardware

The `libpsm2-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 user's shell environment.

To use PSM2 with existing OpenMPI, enter the command:

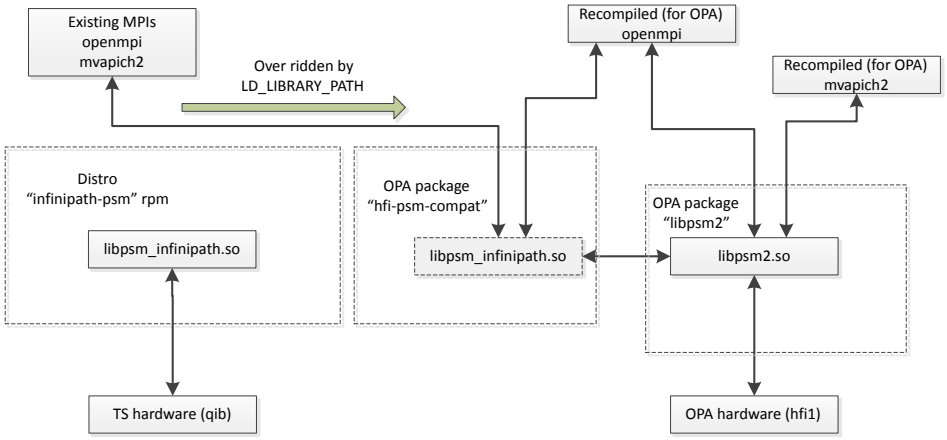
```
mpirun -np 2 -x LD_LIBRARY_PATH=/usr/lib64/psm2-compat:$LD_LIBRARY_PATH
```

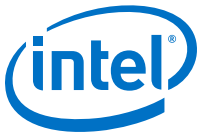
To use PSM2 with existing Intel® MPI Library or MVAPICH2, enter the command:

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

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

Figure 23. Overriding LD_LIBRARY_PATH to Run Existing MPIs on Intel® Omni-Path Hardware





5.0 Running MPI on Intel® Omni-Path Host Fabric Interfaces

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® Omni-Path Basic installation package include Open MPI and MVAPICH2. These MPIs are offered in versions built with the Intel® Omni-Path Performance Scaled Messaging 2 (PSM2) interface, and versions built to run over IB Verbs.

Also, there are commercial MPIs that are not packaged with Intel® Omni-Path Basic installation package that use the PSM2 application programming interface (API) and run over IB Verbs, including Intel® MPI Library.

For more information on other MPIs, see Using Other MPIs.

5.2 Intel® MPI Library

Intel® Omni-Path supports the 64-bit version of Intel® MPI Library. The 32-bit version is not supported. The Intel® MPI Library is not included in the Intel® Omni-Path software, but is available separately. Go to www.intel.com for more information.

5.2.1 Intel® MPI Library Installation and Setup

Download the Intel® MPI Library from www.intel.com and follow the installation instructions. The following subsections provide setup instructions for the Intel® MPI Library.

5.2.1.1 Setting Up the Intel® MPI Library

Intel® Omni-Path supports only the 64-bit version of the Intel® MPI Library.

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
```

5.2.1.2 Compiling Intel® MPI Library Applications

Intel recommends that you use the included wrapper scripts that invoke the underlying compiler. The default underlying compiler is GCC*, including gfortran.

Note: The Intel® MPI Library includes more wrapper scripts than what is listed in the following table. See the Intel® MPI Library documentation for the complete list of wrapper scripts.

Table 4. Intel® MPI Library 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, enter the command:

```
$ 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.

5.2.2 Running Intel® MPI Library Applications

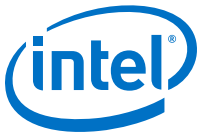
Here is an example of a simple `mpirun` command running with four processes:

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

For more information, follow the Intel® MPI Library instructions for using `mpirun`, which is a wrapper script that invokes the `mpiexec.hydra` command.

Pass the following option to `mpirun` to select Tag Matching Interface (TMI):

```
-genv I_MPI_FABRICS tmi
```



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

```
-genv TMI_DEBUG 1
```

5.3 Open MPI

Open MPI is an open source MPI implementation from the Open MPI Project. Precompiled versions of Open MPI that run over PSM2 and are built with the GCC* and Intel compilers are available with the Intel download. Open MPI that runs over verbs is also available, however, Intel strongly recommends that you do not use MPI on verbs for Intel® Omni-Path.

Open MPI can be managed with the mpi-selector utility, as described in Managing MPI Versions with the MPI Selector Utility on page 66.

5.3.1 Installing Open MPI

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

5.3.2 Setting up Open MPI

Intel recommends that you use the `mpi-selector` tool, because it performs the necessary `$PATH` and `$LD_LIBRARY_PATH` setup.

If the `mpi-selector` tool is not used, you must do the following:

- Put the Open MPI installation directory in the appropriate path by adding the following to `PATH`:

  ```
  $mpi_home/bin
  ```

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

- Set `$LD_LIBRARY_PATH` appropriately.

5.3.3 Setting up Open MPI with SLURM

To allow launching Open MPI applications using SLURM, you must set the Open MPI environment variable `OMPI_MCA_orte_precondition_transports` in every node running the job. The format is 16 digit hexadecimal characters separated by a dash. For example:

```
OMPI_MCA_orte_precondition_transports=13241234acffedeb-abcdefabcdef1233
```

This key is used by the PSM2 library to uniquely identify each different job end point used on the fabric. If two MPI jobs are running on the same node sharing the same HFI and using PSM2, each one must have a different key.

5.3.4 Compiling Open MPI Applications

Intel recommends that you use the included wrapper scripts that invoke the underlying compiler 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.

The following table lists the included wrapper scripts.

Table 5. 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, enter the following:

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

All of the wrapper scripts provide the command line options listed in the following table.

The wrapper scripts pass most options on to the underlying compiler. Use the documentation for the underlying compiler to determine which options to use for your application.

Table 6. 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.

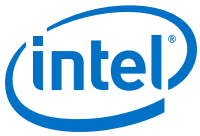
5.3.5 Creating the mpi_hosts File

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`.

For more details, see Using the mpi_hosts File on page 58.

5.3.6 Running Open MPI Applications

The `mpi-selector --list` command invokes 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
- openmpi_gcc-X.X.X
- openmpi_gcc_hfi-X.X.X
- openmpi_intel_hfi-X.X.X

Intel strongly recommends using the choices that contain the _hfi string, because they run using PSM2 which is the default PSM for Intel® Omni-Path. The choices without the _hfi string use verbs by default.

For example, if you chose openmpi_gcc_hfi-*X.X.X*, 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*, enter this command:

```
$ 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
```

Note: In the previous 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>
```

In this example, 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`.

5.3.7 Configuring MPI Programs for Open MPI

When configuring an MPI program, for example, 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 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 you specify the linker 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:

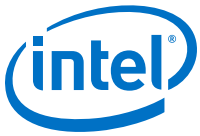
```
LD=mpif90
```

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.

5.3.8 Using Another Compiler

Open MPI and all other Message Passing Interfaces (MPIs) that run on Intel® Omni-Path support multiple compilers, including:

- GNU* Compiler Collection (GCC*, including gcc, g__ and gfortran) versions 3.3 and later
- Intel compiler versions 9.*x*, 10.1, 11.*x*, and 12.*x*



- PGI* compiler versions 8.0 through 11.9

  Note: The PGI* compiler is supported, however, pre-built PGI* MPIs are not included in the Intel® Omni-Path Fabric software package.

The easiest way to use other compilers with any MPI that comes with Intel® Omni-Path Fabric software is to use `mpi-selector` to change the selected MPI/compiler combination. For details, see Managing MPI Versions with the MPI Selector Utility on page 66.

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

The following table shows 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 7. 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 .....`

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

If you are not using the provided scripts for linking, you can link a sample program using the `-show` option as a test to see what libraries to add to your link line.

5.3.8.1 Compiler and Linker Variables

When you use environment variables to select the compiler to use, the scripts also set the matching linker variable if it is not already set. For example, if you use the `$MPICH_CC` variable, the matching linker variable `$MPICH_CLINKER` is also set.

If both the environment variable and command line options are used, the command line option takes precedence.

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

5.3.9 Allocating Processes

MPI ranks are processes that communicate through the PSM2 library for best performance. These MPI ranks are called PSM2 processes.

Typically, MPI jobs are run with each rank associated with a dedicated 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, therefore 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 that can be run per HFI when software context sharing is enabled (default mode).

The number of hardware contexts available can be set by the driver, and depends on the number of processor cores in the system, including hyper-threaded cores.

The default hardware context/CPU mappings can be changed on the 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-Path Hardware Contexts in a Batch Environment.

Errors that may occur with context sharing are covered in Reviewing Context Sharing Error Messages on page 57.

There are multiple ways of specifying how processes are allocated, including:

- `mpi_hosts` file
- `-np` and `-ppn` options with `mpirun`
- `MPI_NPROCS` and `PSM2_MAX_CONTEXTS_PER_JOB` (formerly called `PSM2_SHAREDCONTEXTS_MAX`) environment variables

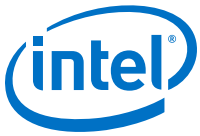
These methods are described later in this document.

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

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_MAX_CONTEXTS_PER_JOB` environment variable. This option may be required for resource sharing between multiple jobs in batch systems.

Note: Before enabling hardware context sharing, first ensure that the maximum number of hardware contexts are enabled and used.

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_MAX_CONTEXTS_PER_JOB` can be set to 72 for each job. Both of the jobs that want to share a node would have to set `PSM2_MAX_CONTEXTS_PER_JOB=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 automatically propagates PSM2 environment variables.

Setting `PSM2_MAX_CONTEXTS_PER_JOB=16` as a cluster-wide default unnecessarily penalizes 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 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. Typically, you use the same value for all nodes in that job.

Note: Either `PSM2_RANKS_PER_CONTEXT` or `PSM2_MAX_CONTEXTS_PER_JOB` is used in a particular job, but not both. If both are used and the settings are incompatible, PSM2 reports an error and the job fails to start.

See the *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide* for details.

5.3.9.2 Reviewing 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.

5.3.9.3 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 `hostfile`. For example, if the file is named `onehost` and it is in the working directory, enter the following:

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

Enabling shared memory mode uses a feature of Open MPI host files to list the number of slots, which is the number of possible MPI processes (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` functions identically.

You can use the `hostfile` for the following operations:

- To measure MPI latency between two cores on the same host using shared memory, run:

```
$ mpirun -np=2 -hostfile onehost osu_latency
```

- To measure MPI unidirectional bandwidth using shared memory, run:

```
$ mpirun -np=2 -hostfile onehost osu_bw
```

5.3.10 Using the mpi_hosts File

As noted in Creating the mpi_hosts File on page 52, a `hostfile` (also called *machines file*, *nodefile*, or *hostsfile*) is created in your current working directory during software installation. This file names the nodes that the node programs may run.

The two supported formats for the `hostfile` are:

```
hostname1
hostname2
...
```

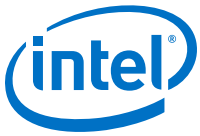
or

```
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 are repeated (in order) as many times as necessary for the requested number of node programs. Also, 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 are 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.

Intel recommends that you use the second format and various command line options to schedule the placement of processes to nodes and cores. For example, use the `mpirun` option `-npernode` to specify how many processes should be scheduled on each node on each pass through the `hostfile`. (The `-npernode` option is similar to the Intel® MPI Library option `-ppn`.) 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 assigns a maximum of 8 processes to the node and there can be no over-subscription of the 8 cores.

There are several ways of specifying the `hostfile`:

- Use the command line option `-hostfile` as shown in the following example:

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

  In this case, if the named file cannot be opened, the MPI job fails.

  Also, `-machinefile` is a synonym for `-hostfile`.

- Use the `-H`, `-hosts`, or `--host` command line option, 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
  ```

- Use 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. For more details, see the website:

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

5.3.11 Using the Open MPI mpirun script

The `mpirun` script 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 `./`, as shown in this 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, because 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.

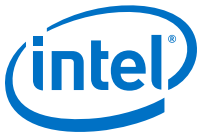
There are many more `mpirun` options for scheduling where the processes get assigned to nodes. See `man mpirun` for details.

`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`.

5.3.12 Using 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 the `mpirun` 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`, as shown in the following example:

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

In this example, only the `MPI_COMM_WORLD` rank `0` process receives the stream from `my_input` on stdin. The stdin on all the other nodes is tied to `/dev/null`. However, the stdout from all nodes is collected into the `my_output` file.

5.3.13 Process Environment for mpirun

See the Open MPI documentation for additional details on the `mpirun` command, specifically these sections:

- Remote Execution:
  https://www.open-mpi.org/doc/v2.0/man1/mpirun.1.php#sect18
- Exported Environment Variables:
  https://www.open-mpi.org/doc/v2.0/man1/mpirun.1.php#sect19
- Setting MCA Parameters:
  https://www.open-mpi.org/doc/v2.0/man1/mpirun.1.php#sect20

5.3.14 Environment Variables

The *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide* lists environment variables that are relevant to any PSM, including Open MPI. Refer to that document for complete details.

PSM2-specific environment variables include, but are not limited to, the following:

- PSM2_DEVICES
- PSM2_IB_SERVICE_ID
- PSM2_MAX_CONTEXTS_PER_JOB
- PSM2_MEMORY
- PSM2_MQ_RECVREQS_MAX
- PSM2_MQ_RNDV_HFI_THRESH
- PSM2_MQ_RNDV_SHM_THRESH
- PSM2_MQ_SENDREQS_MAX
- PSM2_MTU
- PSM2_PATH_REC
- PSM2_PATH_SELECTION

- PSM2_RANKS_PER_CONTEXT
- PSM2_RCVTHREAD
- PSM2_SHAREDCONTEXTS
- PSM2_SHAREDCONTEXTS_MAX
- PSM2_TID
- PSM2_TRACEMASK

Open MPI provides its own environment variables that may be more relevant, 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, specifically:

- MPI Environment Variables:

  https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
- Setting mpirun -mca Parameters:

  https://www.open-mpi.org/doc/v1.6/man1/mpirun.1.php#sect19

5.3.15 Further Information on Open MPI

For more information about Open MPI, see:

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

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

5.4 Open MPI and Hybrid MPI/OpenMP Applications

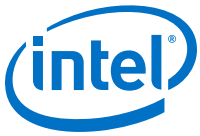
Open MPI supports hybrid MPI/OpenMP applications, provided that MPI routines are called only by the master OpenMP thread. This application is called the *funneled thread model*. Instead of `MPI_Init/MPI_INIT` (for C/C__ and Fortran respectively), the program can call `MPI_Init_thread/MPI_INIT_THREAD`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

MPI routines can be called only by the master OpenMP thread. The hybrid executable is executed using `mpirun`, but typically only one MPI process is run per node and the OpenMP library creates additional threads to use all CPUs on that node. If there are sufficient CPUs on a node, you may run multiple MPI processes and multiple OpenMP threads per node.

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

The number of OpenMP threads is on a per-node basis and is 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 is set to match the number of CPUs per node.



An example case is a node with four CPUs, running one MPI process and four OpenMP threads. In this case, `OMP_NUM_THREADS` is set to four.

See Process Environment for mpirun on page 61 for information on setting environment variables.

Note: With Open MPI, and other PSM2-enabled MPIs, you typically 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)
mpirun -np 2 -H host1,host2 -x HFI_NO_CPUAFFINITY=1 ./hybrid_app
```

In this case, 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 the load is distributed to 8 different CPU cores.

The `OMP_NUM_THREADS` and `HFI_NO_CPUAFFINITY` variables can be set in `.bashrc` or on the command line after `-x` options.

5.5 Debugging MPI Programs

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

5.5.1 MPI Errors

Almost all MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error code. It is returned 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 terminates the MPI job. Therefore, you can get information about MPI exceptions in your code by providing your own handler for `MPI_ERRORS_RETURN`. For details, see the `MPI_Errhandler_set` man page: https://www.open-mpi.org/doc/v2.0/man3/MPI_Errhandler_set.3.php

For details on MPI error codes, see the man page: https://www.open-mpi.org/doc/v2.0/man3/MPI.3.php

5.5.2 Using Debuggers

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

The TotalView* debugger can be used with the Open MPI supplied in this release. For more information, see: http://www.open-mpi.org/faq/?category=running#run-with-tv

6.0 Using Other MPIs

This section provides information on using Message Passing Interface (MPI) implementations other than Open MPI and Intel® MPI Library, which are discussed in Open MPI and Intel® MPI Library, respectively. This section also compares the MPIs available and discusses how to choose between MPIs.

6.1 Introduction

Intel® Omni-Path supports multiple high-performance MPI implementations. Most implementations run over both PSM2 and OpenFabrics Verbs, as shown in 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 66.

Table 8. Other Supported MPI Implementations

MPI Implementation	Runs Over	Compiled With	Comments
Open MPI	PSM2 Verbs	GCC* Intel	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	PSM2 Verbs	GCC* Intel	Provides MPI-2 functionality. Can be managed by `mpi-selector`.
Intel® MPI Library	PSM2 Verbs	GCC* Intel	Provides MPI-1 and MPI-2 functionality. Available for purchase from Intel.

Note: The MPI implementations run on multiple interconnects and have their own mechanisms for selecting the relevant interconnect. This section contains basic information about using the MPIs. For details, see the MPI-specific documentation.

6.2 Installed Layout

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

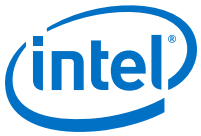
`/usr/mpi/$compiler/$mpi-mpi_version`

Note: See documentation for the Intel® MPI Library for information on the default installation directory.

The Intel-supplied MPIs are precompiled with GCC* and Intel compilers. They also have `-hfi` 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 examples in this section assume that the default path for each MPI implementation to `mpirun` is:

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

If a prefixed installation location is used, `/usr` may be replaced by *`$prefix`*. This path is sometimes referred to as *`$mpi_home`*`/bin/mpirun` in the following sections.

6.3 MVAPICH2

Precompiled versions of MVAPICH2 that run over PSM2 and are built with GCC* and Intel compilers are included with the Intel download. A precompiled version of MVAPICH2 that runs over Verbs and is built with the GCC* 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 66.

6.3.1 Compiling MVAPICH2 Applications

Intel recommends that you use the included wrapper scripts that invoke the underlying compiler, as shown in the following table.

Table 9. MVAPICH2 Wrapper Scripts

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

To compile your program in C, use the command:

```
$ 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/mvapich2.conf
```

6.3.2 Running MVAPICH2 Applications

The `mpi-selector --list` command invokes the MPI Selector and lists the MPI options, including the following MVAPICH2 choices:

- `mvapich2_gcc-X.X`
- `mvapich2_gcc_hfi-X.X`
- `mvapich2_intel_hfi-X.X`

Intel strongly recommends using the choices that contain the `_hfi` string, because they run using PSM2 which is the default PSM for Intel® Omni-Path. The choices without the `_hfi` string use verbs by default.

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

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

6.3.3 Further Information on MVAPICH2

For more information about MVAPICH2, refer to:

http://mvapich.cse.ohio-state.edu/support/

6.4 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 installed as a part of Intel® Omni-Path Fabric Host Software and it includes the following basic functions:

- 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

Here 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
openmpi_gcc-X.X.X
openmpi_gcc_hfi-X.X.X
openmpi_intel_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.

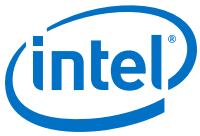
Each MPI registers itself with the MPI Selector, and provides shell scripts `mpivar.sh` and `mpivars.sh` scripts that can be found in `$prefix/mpi/<COMPILER>/<MPI>/bin` directories.

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.

Additional details can be found at:

- Process Environment for mpirun
- Environment Variables
- Compiler and Linker Variables

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



7.0 SHMEM Description

SHMEM (Symmetric Hierarchical Memory) is a programing paradigm for one-sided operations. The OpenSHMEM* community defines the standard, which includes an Application Programming Interface (API). The OpenSHMEM* API provides global distributed shared memory across a network of hosts.

The Intel® Omni-Path Fabric software package includes a copy of the University of Houston SHMEM, which is based on OpenSHMEM*.

SHMEM is quite distinct from local shared memory, which may be abbreviated as *shm*. 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

The SHMEM library distributed with the Intel® Omni-Path Fabric Suite depends on the GASNet layer, which in turn depends on the Intel Performance Scaled Messaging (PSM2) protocol layer, implemented as a user-space library.

7.2 Installing SHMEM

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 must be running Red Hat* Enterprise Linux* (RHEL*) operating system.

An Message Passing Interface (MPI) implementation with Performance Scaled Messaging (PSM2) support must be available to run SHMEM. The following MPI implementations are included in IFS:

- Open MPI configured to include PSM2 support:
  - `/usr/mpi/gcc/openmpi-X.X.X-hfi`
  - `/usr/mpi/intel/openmpi-X.X.X-hfi`

  The `-hfi` suffix denotes that this is the Intel PSM2 version.
- MVAPICH2 compiled for PSM2:
  - `/usr/mpi/gcc/mvapich2-X.X-hfi`
  - `/usr/mpi/intel/mvapich2-X.X-hfi`

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

Intel recommends 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 you 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 C compiler in 64-bit mode
- PGI* compiler in 64-bit mode

  Note: The PGI* compiler is supported, however, pre-built PGI* MPIs are not included in the Intel® Omni-Path Fabric software package.

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

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, use the command:

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

This is necessary because PSM2 cannot be used by both MPI and SHMEM at the same time due to a limitation in the number of endpoints per process used by PSM2. The above environment variable prevents Open MPI from using PSM2. Note that SHMEM only uses MPI for job launch and initialization.

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

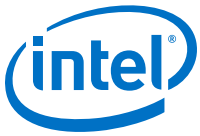
```
export PATH=/usr/shmem/gcc/openshmem-$VERSION-hfi/bin:/usr/shmem/gcc/
gasnet-$VERSION-openmpi-hfi/bin:/usr/mpi/gcc/openmpi-$VERSION/bin:$PATH
```

```
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
```

Note that some MPI builds may also contain their own SHMEM implementation.

7.5 Integrating SHMEM with slurm

SHMEM relies on an MPI implementation to provide a runtime environment for jobs. This includes job startup, stdin/stdout/stderr routing, and other low performance control mechanisms. SHMEM 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.

SLURM Workload Manager (Simple Linux Utility for Resource Management or slurm) is a free and open-source job scheduler for Linux* and Unix*-like operating systems. Integration of SHMEM with slurm is provided by the MPI implementation. For details, see: http://slurm.schedmd.com/slurm.html

The following sections describe methods for integrating SHMEM and slurm:

- Full Integration on page 69
- Two-Step Integration on page 69
- No Integration on page 70

7.5.1 Full Integration

This approach fully integrates SHMEM startup 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 relies on `slurm` providing a process management interface (PMI) library and the MPI implementation using that library so that each MPI process can hook into `slurm`.

You must set up the environment appropriately, including adding the SHMEM library directory to `LD_LIBRARY_PATH`. See Compiling and Running SHMEM Programs on page 68 for more information on the environment setup.

7.5.2 Two-Step Integration

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

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

The `salloc` command 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. Because `shmemrun` is used in this approach, you do not need to set up the environment.

For additional details, see Setting up Open MPI with SLURM on page 51.

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 method for this approach is:

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

In this example, *`shmemrun_wrapper`* is a user-provided wrapper script that:

- Creates a hosts file based on the current `slurm` allocation.
- Invokes `mpirun` with the hosts file and other appropriate options.

Note: In this case, ssh is used for starting processes, not `slurm`.

7.6 Sizing Global Shared Memory

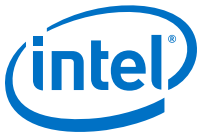
SHMEM provides a symmetric heap that determines the amount of global shared memory per Processing Element (PE) that is available to the application. This is an important resource and this section discusses the mechanisms available to size it. The SHMEM library pre-allocates room in the virtual address 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 it fails if there is no room in the virtual memory space of the process following the global shared memory segment. 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 `opatop`, due to the large maximum size setting. The `RES` field of `opatop` indicates the actual amount of memory that is resident and in 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.

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. Because 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

Full documentation on the OpenSHMEM* Application Programming Interface can be found at: http://openshmem.org/site/Specification

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 benchmarks must be run with an even number of processes. They are typically run on exactly two hosts with the processes divided equally 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 the following table.

Table 10. SHMEM Micro-Benchmarks Command Line 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	Display the help page.
-l INT	Set minimum message size. Default = 2
-m INT	Set the maximum message size. Default = 4194304

Additional SHMEM micro-benchmark programs are described in the following sections of this document:

- SHMEM Random Access Benchmark on page 72
- SHMEM All-to-All Benchmark on page 73
- SHMEM Barrier Benchmark on page 74
- SHMEM Reduce Benchmark on page 74

7.8.1 SHMEM Random Access Benchmark

Name: shmem-rand

Description: randomized put/get benchmark. This is a hybrid SHMEM/MPI code, so a binary is provided per supported MPI implementation.

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

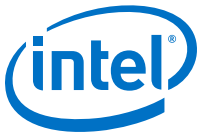
Message sizes are specified in bytes. Default = 8.

Options are listed in the following table.

Table 11. SHMEM 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 = scaled
-h	Display the help page.
-l	Enable communication to local ranks.
-m INTEGER[K]	Memory size in MB. Can also specify KB using a K suffix. Default = 8 MB
-n	Use non-pipelined mode for NB ops. Default = pipelined
-o OP	Operation. Values include: get getnb put

continued...



Option	Description
	putnb
-p	For blocking puts, no quiet every window. This is the default operation.
-q	For blocking puts, use quiet every window.
-r	Use ring pattern. Default = random
-s	Enable communication to self.
-t FLOAT	If the loop count is not given, run the test for this many seconds. Default = 10 seconds
-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 SHMEM All-to-All Benchmark

Name: shmem-alltoall

Description: all-to-all put benchmark. This is a hybrid SHMEM/MPI code, so a binary is provided per supported MPI implementation.

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

Message sizes are specified in bytes. Default = 8.

Options are listed in the following table.

Table 12. SHMEM 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 = scaled
-h	Display the help page.
-l	Enable communication to local ranks.
-m INTEGER[K]	Memory size in MB. Can also specify KB using a K suffix. Default = 8 MB
-n	Use non-pipelined mode for NB ops. Default = pipelined
-o OP	Operation. Values include: put putnb
-p INTEGER	Offset for all-to-all schedule. Default = 1. Typically set to ppn
-r	Randomize all-to-all schedule.
-s	Enable communication to self.

continued...

Option	Description
-t FLOAT	If the loop count is not given, run the test for this many seconds. Default = 10 seconds
-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 SHMEM Barrier Benchmark

Name: shmem-barrier

Description: barrier benchmark.

Usage: shmem-barrier [options]

Options are listed in the following table.

Table 13. SHMEM Barrier Benchmark Options

Option	Description
-h	Display the help page.
-i INTEGER[K]	Outer iterations. Default = 1

7.8.4 SHMEM Reduce Benchmark

Name: shmem-reduce

Description: reduce benchmark.

Usage: shmem-reduce [options]

Options are listed in the following table.

Table 14. SHMEM Reduce Benchmark Options

Option	Description
-b INTEGER	Number of barriers between reduces. Default = 0
-h	Display the help page.
-i INTEGER[K]	Outer iterations. Default = 1
-r INTEGER	Inner iterations. Default = 10000

8.0 Virtual Fabric Support in PSM2

Intel® Omni-Path supports full Virtual Fabric (vFabric) integration using Performance Scaled Messaging (PSM2). To target a vFabric, you can specify IB Service Level (SL) and Partition Key (PKey) or provide a configured Service ID (SID). You can also use path record queries to the Intel® Omni-Path Fabric Suite Fabric Manager (FM) during connection setup.

All PSM2-enabled MPIs can leverage these capabilities transparently using environment variables. PSM2 environment variables are documented in the *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide*.

With MPI applications, the environment variables must be propagated across all nodes and processes and not just the node from where the job is submitted and run. The mechanisms to do this are MPI-specific, as shown in the following examples:

- **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
  ```

- **Intel® MPI Library**: Use `-genv` as shown below. Additional examples are shown at: https://software.intel.com/en-us/node/535596

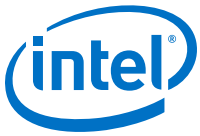
  Example:

  ```
  mpirun -np 2 -machinefile machinefile -genv PSM2_ENV_VAR PSM_ENV_VAL prog prog_args
  ```

  Additional options include:

  `-genvall` Enables propagation of all environment variables to all MPI processes.

  `-genvlist <list of genv var names>` Passes a list of environment variables with their current values, where *<list of genv var names>* is a comma-separated list of environment variables to be sent to all MPI processes.



8.1 Virtual Fabric Support using Fabric Manager

Virtual Fabric (vFabric) in PSM2 is supported with the Fabric Manager (FM). 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 simplify testing, however, any Service ID can be used.

There are two ways to use vFabric with PSM2:

- Specify the appropriate SL and PKey for the vFabric in question.
- Specify a Service ID (SID) that identifies the vFabric to be used. PSM2 automatically obtains the SL and PKey to use for the vFabric from the Fabric Manager via path record queries.

Refer to the *Intel® Omni-Path Fabric Suite Fabric Manager User Guide* for more details about vFabrics.

9.0 Multi-Rail Support in PSM2

Multi-rail means that a process can use multiple network interface cards to transfer messages. This chapter defines terminology, explains user scenarios, and describes implementation details for MPI application programmers.

9.1 Multi-Rail Overview

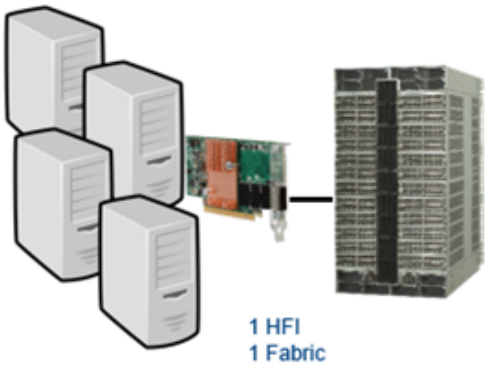
A multi-rail configuration provides load balancing and failover capabilities, adding a higher degree of fabric redundancy. If one HFI or an entire subnet fails, traffic can be moved to the remaining switches.

The multi-rail feature can be applied to a single subnet or multiple subnets. By enabling multi-rail, a process can use multiple network interface cards (HFIs) to transfer messages.

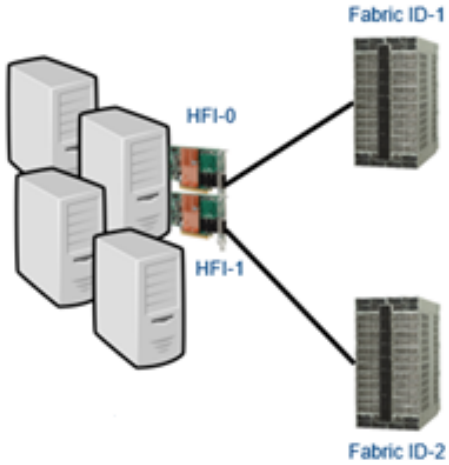
Note: *Subnets* can also be referred to as *planes* or *fabrics*. *Rails* are also referred to as *HFIs*.

Three basic scenarios include:

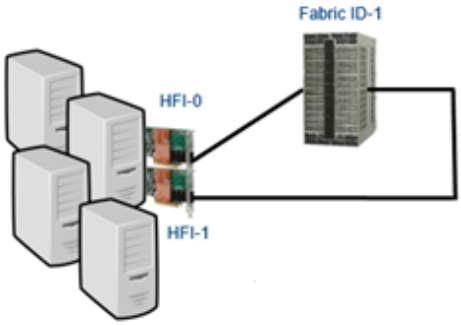
- Single-rail in a single subnet: This scenario, shown in the following figure, consists of one HFI in a server connected to one subnet. This is the default configuration during installation. This configuration provides the performance required by most applications in use today.



- Dual-rail in dual subnets: This scenario, shown in the following figure, consists of two HFIs in the same server connected to separate subnets. This configuration provides improved MPI message rate, latency, and bandwidth to the node as well as flexibility for configuring failover and load-balancing.



- Dual-rail in a single subnet: This scenario, shown in the following figure, consists of two HFIs in the same server connected to the same subnet. This configuration also provides improved MPI message rate, latency, and bandwidth to the node, but only basic HFI failover capabilities if configured.

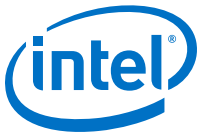


Note: Other multi-rail scenarios can be configured up to the supported number of FM instances. These scenarios are out of scope at this time.

9.2 Multi-Rail Users

Multi-rail can be used by both system administrators and MPI application programmers.

The system administrator sets up a multi-rail system using multiple Intel® Omni-Path HFIs per node. If desired, the system administrator connects the HFIs to multiple subnets, and configures each subnet with different subnet IDs. This is done by editing the `opafm.xml` file, as described in the *Intel® Omni-Path Fabric Software Installation Guide*.



MPI application programmers can run their application over a multi-rail system to improve performance or increase the number of hardware resources available for jobs. By default, PSM2 selects HFIs in round-robin fashion, in an attempt to evenly distribute the use of hardware resources and provide performance benefits.

The default behavior is single-rail configuration, where each process uses a single context/sub-context that maps to a single HFI to communicate to other processes. Intel recommends that you enable PSM2 to use multiple rail communication on systems with multiple HFIs per node.

On a multi-subnet system, if multi-rail is not turned on, Intel recommends that you set the `HFI_UNIT` environment variable (from 0) to notify the PSM2 job which HFI to use. The HFIs should be on the same subnet, otherwise, the same job might try to use HFIs from different subnets and cause the job to hang because there is no path between subnets. In this case, some jobs may run across different subnets, but this behavior cannot be guaranteed. However, if multi-rail is turned on, PSM2 can reorder and automatically match the HFIs by using the subnet ID. That is why unique subnet IDs are required for each subnet.

9.3 Environment Variables

The following environment variables can be set:

- `PSM2_MULTIRAIL` = `n` where `n` can be any value up to a maximum of 4.

  If set to a non-zero value, PSM2 sets up 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,...`

  Specifies the HFI/port pair(s) to set up a rail. Multiple specifications are separated by a comma. Values include:

  `unit` starts from 0. The term *unit* refers to an HFI.

  `port` is always 1 because Intel® Omni-Path Host Fabric Interfaces are single port devices.

  If only one rail is specified, it is equivalent to a single-rail case. The `unit/port` is specified instead of using the `unit/port` values assigned by the `hfi1` driver. PSM2 scans the above pattern until a violation or error is encountered, and uses the information it has gathered.

9.4 Multi-Rail Configuration Examples

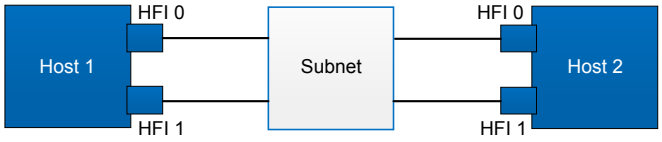
This section contains examples of multi-rail used in a single subnet and in multiple subnets.

Terminology:

- Subnets can also be referred to as *planes* or *fabrics*.
- Hosts can also be referred to as *nodes*.
- HFIs can also be referred to as *rails*.

Single subnet, each host has two HFIs

The following figure shows an example of a single subnet with each host having two HFIs. HFI 0 has one port and HFI 1 has one port.

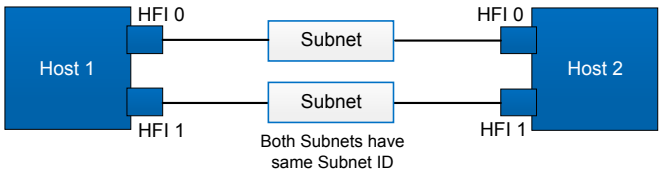


Example Environment Variables

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

Multi-subnets, with same subnet ID

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

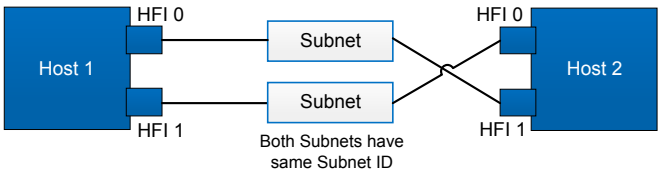


Example Environment Variables

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

Multi-subnets, single subnet ID, atypical wiring

The following figure shows an example of multiple subnets with a single subnet ID and atypical wiring.

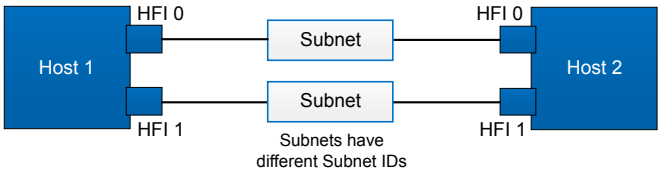


Example Environment Variables

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

Multi-subnets, different subnet IDs

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

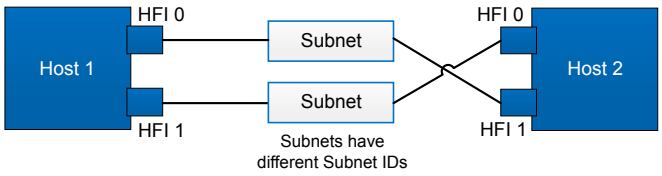


Example Environment Variables

- `PSM2_MULTIRAIL` is not set. PSM2 may not work because there are multiple subnets. HFI 0/Port 1 on first host has no connection to HFI 1/Port 1 on second host.
- `PSM2_MULTIRAIL=1` automatic selection. Both hosts get HFI/Port pairs 0:1,1:1 first. After the PSM2 reordering based on subnet ID, the host on the left side gets 0:1,1:1 and the host on the right side gets 0:1,1:1. The PSM2 makes the master rail on 0:1 of the left host with 0:1 on the right host. The standby rail is set up on 1:1 of the left host with 1:1 of the right host. PSM2 works in this configuration/ setting.
- `PSM2_MULTIRAIL=1` and `PSM2_MULTIRAIL_MAP=1:1,0:1`. You must specify the HFI/port pairs. PSM2 does not reorder them. Both hosts use 1:1 to make the connection on the second subnet as the master rail, and set up the second rail over 0:1 on both sides. PSM2 works fine in this configuration.

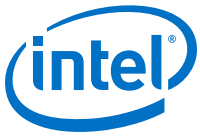
Multi-subnets, different subnet IDs, atypical wiring

The following figure shows an example of multiple subnets with different subnet IDs and atypical wiring.



Example Environment Variables

- `PSM2_MULTIRAIL` is not set. PSM2 may not work because there are multiple subnets.
- `PSM2_MULTIRAIL=1` automatic selection. Both hosts get HFI/port pairs 0:1,1:1 first. After PSM2 reordering based on the subnet ID, the host on the left side gets 0:1,1:1 and the host on the right side gets 1:1,0:1. The PSM2 makes the master rail on 0:1 of the left host with 1:1 on the right host. The standby rail is set up on 1:1 of the left host with 0:1 on the right host. PSM2 works in this configuration/ setting.
- `PSM2_MULTIRAIL=1` and `PSM2_MULTIRAIL_MAP=1:1,0:1`. You must specify the HFI/port pairs. PSM2 does not reorder them. Both hosts 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

This chapter discusses the following topics:

- Intel® Omni-Path Routing Features and Innovations
- Dispersive Routing

Refer to the *Intel® Omni-Path Fabric Suite Fabric Manager User Guide* for information about Fabric Manager features that can improve application performance, including:

- Adaptive routing - improves bandwidth and congestion.
- Congestion control - helps with many to one congestion handling.
- Traffic flow optimization (TFO) - decreases latency for important jobs, using Fabric Manager `DeviceGroup`-based routing algorithms.

10.1 Intel® Omni-Path Routing Features and Innovations

Intel® Omni-Path Architecture (Intel® OPA) supports clusters of all sizes with optimization for HPC applications at both the host and fabric levels for benefits that are not possible with the standard InfiniBand*-based designs. This section describes some of the routing features.

Adaptive Routing

Adaptive Routing monitors the performance of the possible paths between fabric end-points and periodically rebalances the routes to reduce congestion and achieve a more balanced packet load. While other technologies also support adaptive routing, the implementation is vital. Intel's implementation is based on cooperation between the Fabric Manager and the switch ASICs. The Fabric Manager—with a global view of the topology—initializes the switch ASICs with several egress ports per destination, updating these options as the fundamental fabric changes when links are added or removed. Once the switch egress ports are set, the Fabric Manager monitors the fabric state, and the switch ASICs dynamically monitor and react to the congestion sensed on individual links. This approach enables Adaptive Routing to scale as fabrics grow larger and more complex.

Refer to the *Intel® Omni-Path Fabric Suite Fabric Manager User Guide* for additional information.

Dispersive Routing

One of the critical roles of fabric management is the initialization and configuration of routes through the fabric between pairs of nodes. Intel® Omni-Path supports a variety of routing methods, including defining alternate routes that disperse traffic flows for redundancy, performance, and load balancing. Instead of sending all packets from a source to a destination via a single path, Dispersive Routing allows the sending host to distribute traffic across multiple paths. Once received, the destination host processes packets in their proper order for rapid, efficient processing. By leveraging more of the fabric to deliver maximum communications performance for all jobs, Dispersive Routing promotes optimal fabric efficiency.

Traffic Flow Optimization

Traffic Flow Optimization optimizes the quality of service beyond traditional per-VL packet level scheduling of packets to be sent on a given link. At the Intel® Omni-Path Architecture link level, low priority packets may be preempted and a higher priority packet may be placed on the link. Once the higher priority packet(s) have completed, the lower priority preempted packet is resumed.

The key benefit is that Traffic Flow Optimization reduces the variation in latency seen through the network by high priority traffic in the presence of lower priority traffic. It addresses a traditional weakness of both Ethernet and InfiniBand* in which a packet must be transmitted to completion once the link starts even if higher priority packets become available.

Packet Integrity Protection

Intel® Omni-Path specifies a link-level correction mechanism called Packet Integrity Protection that corrects bit errors, but does not add latency, making it a much more suitable scheme for HPC environments. (Prior schemes used forward error correction as the link-level correction mechanism, which had the side-effect of adding significant latency to every hop.) Packet Integrity Protection is capable of handling bit error rates of up to 1e-07 without performance impacts. Intel specifies a worst case bit error rate of 1e-08, which allows up to 1000 corrected link integrity errors per second per link without impacting performance.

Packet Integrity Protection allows for rapid and transparent recovery of transmission errors between a sender and a receiver on an Intel® Omni-Path Architecture link. Given the very high Intel® OPA signaling rate (25.78125G per lane) and the goal of supporting large scale systems of a hundred thousand or more links, transient bit errors must be tolerated while ensuring that the performance impact is insignificant. Packet Integrity Protection enables recovery of transient errors whether it is between a host and switch or between switches. This eliminates the need for transport level timeouts and end-to-end retries. This is done without the heavy latency penalty associated with alternate error recovery approaches.

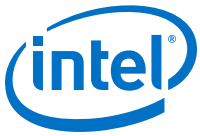
Dynamic Lane Scaling

Dynamic Lane Scaling allows an operation to continue even if one or more lanes of a 4x link fail, saving the need to restart or go to a previous checkpoint to keep the application running. The job can then run to completion before taking action to resolve the issue. Currently, InfiniBand* typically drops the whole 4x link if any of its lanes drops, costing time and productivity.

10.2 Dispersive Routing

By default, Intel® Omni-Path architecture uses deterministic routing that is chosen based on the Destination LID (DLID) of a given packet.

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. Because routing is based on DLIDs, the Intel® Omni-Path fabric provides the ability to assign multiple LIDs to an HFI port using a feature called Lid Mask Control (LMC). The total number of DLIDs assigned to an HFI port is 2^LMC with the LIDS being assigned contiguously. The default Intel® Omni-Path fabric



configuration uses a LMC of 0, meaning each port has 1 LID assigned to it. With non-zero LMC fabrics, there can be multiple potential paths through the fabric to reach the same destination HFI port.

Dispersive routing, as implemented in PSM2, attempts to avoid congestion hotspots, described above, by *spraying* messages across these paths. This can result in better traffic balance and therefore avoid congestion in the fabric. The current implementation of PSM2 supports fabrics with a maximum LMC of 3, with 8 LIDs assigned per port. When more than 1 DLID is assigned per HFI port, PSM2 takes advantage of the DLIDs by using the following style of paths for packets: [SLID, DLID], [SLID _ 1, DLID _ 1], [SLID _ 2, DLID _ 2] ….. [SLID _ N, DLID _ N].

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 different path between HFI ports. All packets belonging to a window use 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. Small messages, on the other hand, always use a single path when communicating to a remote node; however, different processes executing on a node can use different paths for their communication between the nodes.

The default PSM2 path selection policy is *adaptive*, which behaves as described in the following example. Assume two nodes A and B each with 8 CPU cores per node and assume the fabric is configured for a LMC of 3. In this case, PSM2 constructs 8 paths through the fabric as described above and a 16 process MPI application that spans these nodes (8 processes per node). Then:

- Small Messages are sent to a remote process that use a fixed path *X*, where *X* is selected with a round-robin algorithm.

  Note: Only path *X* is used by this process for all communications to any process on the remote node.

- For a large message, each process uses all of the 8 paths and sprays the windowed message across all 8 paths.

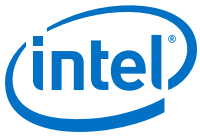
There are other path selection policies that determine how to select the path or the path index from the set of available paths. The policies are used by a process when communicating with a remote node. These are static policies that assign a static path on job startup for both small and large message transfers. They include:

- **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 on a given node are sending packets.

- **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 destination node. If multiple processes from Node A send a message to a single process on Node B, only one path is 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 because only a single path exists between any port pairs.

For more information, see the *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide*, specifically the PSM2_PATH_SELECTION environment variable.

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 different batch queuing systems, located at the following links:

- 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
- LSF: https://www.open-mpi.org/faq/?category=building#build-rte-lsf

11.1 Clean Termination of MPI Processes

The Intel® Omni-Path Fabric Host Software typically ensures clean termination of all Message Passing Interface (MPI) programs when a job ends. In some rare circumstances an MPI process may remain alive, and potentially interfere with future MPI jobs. To avoid this problem, Intel recommends you run a script before and after each batch job to kill all unwanted processes. Intel does not provide such a script, however, you can find out which processes on a node are using the Intel interconnect with the `fuser` command, which is typically installed in the `/sbin` directory.

Run the following 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.

Another example using the `lsof` command:

```
lsof /dev/hfi1
```

This command displays a list of processes using Intel® Omni-Path.

To get all processes, including stats programs, SMA, diags, and others, run the following command:

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

`lsof` can also take the same form:

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

Run the following command to terminate 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. If 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 error occurs, refer to Clean Up PSM2 Shared Memory Files.

11.2 Clean Up PSM2 Shared Memory Files

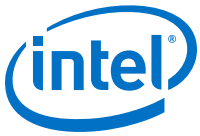
If a PSM2 job terminates abnormally, such as with a segmentation fault, there could be POSIX shared memory files left over in the `/dev/shm` directory. The files are owned by the user and can be deleted either by the user or by root.

To clean up the system, create, save, and run the following PSM2 SHM cleanup script as root on each node. Either log on to the node, or run remotely using `pdsh` or `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, you can remove all of the shared memory files, including stale files, with 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 during installation. This section describes a few of these benchmarks and how to run them. For additional details on running these and other MPI sample applications, refer to the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*.

The remainder of this section assumes 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.

12.1 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. Because 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 depends 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.

Alternative Method

An alternative method to run this program uses the commands:

```
# cd /usr/lib/opa/src/mpi_apps
# ./run_lat
```

For details, see the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*, MPI Sample Applications section.

12.2 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. In this case, the originator of the messages pumps 64 of them 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.

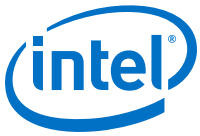
You can run this program by typing:

```
$ 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 depends 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.



Alternative Method

An alternative method to run this program uses the commands:

```
# cd /usr/lib/opa/src/mpi_apps
# ./run_bw
```

For details, see the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*, MPI Sample Applications section.

12.3 Multiple Bandwidth / Message Rate Test

`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 the Open MPI `-npernode 12` option was used 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 depends 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                    -                   -
```

Alternative Method

An alternative method to run this program uses the commands:

```
# cd /usr/lib/opa/src/mpi_apps
# ./run_mbw_mr3 NP

where NP is the number of processes to run or all
```

For details, see the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*, MPI Sample Applications section.

12.4 Enhanced Multiple Bandwidth / Message Rate Test (mpi_multibw)

mpi_multibw is a version of osu_mbw_mr that 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 depends on your particular configuration.

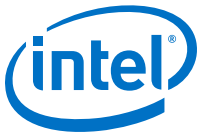
```
$ mpirun -H host1,host2 -npernode 12  /usr/mpi/gcc/openmpi-X.X.X-hfi/tests/intel/
mpi_multibw
PathScale Modified OSU MPI Bandwidth Test
(OSU Version X.X, PathScale $Revision: X.X.X.X $)
Running on 12 procs per node (uni-directional traffic for each process pair)

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 depends 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
```



Alternative Method

An alternative method to run this program uses the commands:

```
# cd /usr/lib/opa/src/mpi_apps
# ./run_multibw processes

where processes is the number of processes on which to run the test.
All indicates the test should be run for every process in the mpi_hosts file.
```

For details, see the *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*, MPI Sample Applications section.

13.0 Troubleshooting

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

- Using the LED to Check the State of the HFI
- BIOS Settings
- Kernel and Initialization Issues
- OpenFabrics and Intel® Omni-Path Issues
- System Administration Troubleshooting
- Performance Issues

Additional troubleshooting information can be found in:

- *Intel® Omni-Path Host Fabric Interface Installation Guide*
- *Intel® Omni-Path Fabric Switches Hardware Installation Guide*
- *Intel® Omni-Path Fabric Software Installation Guide*

Intel® Omni-Path user documentation can be found on the Intel web site. See Documentation Set on page 10 for URLs.

13.1 Using the LED to Check the State of the HFI

The LED on the Intel® Omni-Path Host Fabric Interface functions 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

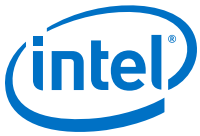
13.2 BIOS Settings

Refer to 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:

- Driver Load Fails Due to Unsupported Kernel



- Rebuild or Reinstall Drivers if Different Kernel Installed
- Intel® Omni-Path Interrupts Not Working
- OpenFabrics Load Errors if HFI Driver Load Fails
- Intel® Omni-Path HFI Initialization Failure
- MPI Job Failures Due to Initialization Problems

13.3.1 Driver Load Fails Due to Unsupported Kernel

If you try to load the Intel® Omni-Path driver on a kernel that the Intel® Omni-Path software does not support, the load fails with error messages that 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 that you use the Textual User Interface (TUI) to perform 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 or Intel® Omni-Path HFI Initialization Failure.

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 OS 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.

13.3.4 OpenFabrics Load Errors if HFI Driver Load Fails

When the HFI driver fails to load, the other OpenFabrics drivers/modules are loaded and shown by `lsmod`. However, commands such as `ibv_devinfo` fail if the HFI driver fails to load, as shown in the following example:

```
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 MPI job or another program.

Here is a sample command and error message:

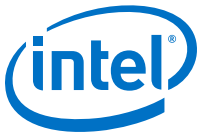
```
$ mpirun -np 2 -m ~/tmp/mbu13 osu_latency
<nodename>:hfi_userinit: assign_port command failed: Network is down
<nodename>:can't open /dev/hfi1, network down
```

This is 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 with the command:

```
$ 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 indicates whether the driver has loaded or not. 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 section covers 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, or restarting Intel® Omni-Path software.

Use the `systemctl` command to stop or start the Fabric Manager:

```
# 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.

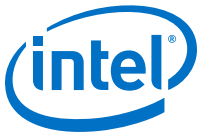
In large cluster configurations, switches may be attached to other switches to supply the necessary inter-node connectivity. Problems with these inter-switch (or intermediate) links are sometimes more difficult to diagnose than failure of the final link between a switch and a node. The failure of an 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. For details, see the *Intel® Omni-Path Fabric Suite FastFabric User Guide*.

13.6 Performance Issues

See the *Intel® Omni-Path Fabric Performance Tuning User Guide* for details about Intel® Omni-Path Fabric optimizing performance and handling performance issues.



14.0 Recommended Reading

This section contains lists of reference material for further reading.

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/

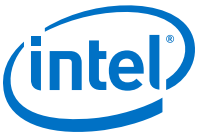
14.4 OpenFabrics Alliance*

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



14.6 Networking

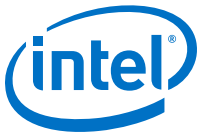
The Internet Frequently Asked Questions (FAQ) archives contain an extensive Request for Command (RFC) section. Numerous documents on networking and configuration can be found at:

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

14.7 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/



Appendix A hfidiags User Guide

`hfidiags` is a Python* tool that allows users with the appropriate permissions to open the HFI driver's user interface to examine and modify Control and Status Registers (CSRs) of the Intel® Omni-Path HFI Silicon 100 Series hardware.

A.1 Key Features

The key features for `hfidiags` include:

- CSR name tab completion
- Hardware state capture and export functionality
- Support for parsing/decoding of data structures and headers
- Command operations on previously saved states
- Extendable functionality through additions of new commands

A.2 Usage

The `hfidiags` tool requires that you have sufficient permissions to open the device's UI interface (provided by the hfi device driver). Typically, the system superuser has these permissions.

A.2.1 Command Line

The default mode of operation for the `hfidiags` tool is in the shell-like interactive mode. When started, the tool presents a command prompt and accepts typed-in commands. However, it is also possible to run the tool in *scripted* mode. In this mode, the shell-like command interface is not started and the tool processes commands passed in via a script. This can be accomplished in the following ways:

1. Using the `-e` command line option, you can pass in a single command to be executed by the tool. The output of the command is the same as if it were typed into the command line interface.
2. Using the `-s` command line option, you can pass in a filename containing a list of commands (one per line), which the tool reads in and executes sequentially. This option also accepts reading the commands from the standard input if the value for the filename argument is set to – as shown in the following example:

   ```
   $ echo -e "help\nunit 1" | hfidiags -s -
   ```

   The example executes two commands, `help` and `unit 1`, and then exits. Note that the new line character `\n` is also present between the commands because of the `-s` option and – option.

Both methods above accept all commands and command options supported by the command line interface.

By default, the tool attempts to open the hardware device interface on startup. However, it can be directed to skip opening the device interface using command line options. This can be used to examine a previously saved hardware state.

A.2.2 Interactive Interface

The `hfidiags` tool provides some default functionality as a standard part of the tool. Most of these commands work regardless of the target (hardware or saved state). Some commands do not work as they may not make sense nor do anything useful. An example of such a command is `write` when used with a previously saved state; a previously saved state cannot be altered.

Summary of standard commands (see Command Descriptions for details):

- read – Reads hardware CSR(s) or memory
- write – Writes hardware CSR(s) or memory
- decode – Decodes a hexadecimal value as a CSR
- info – Describes a hardware CSR
- unit – Switches HFI unit
- state – Captures, saves, and loads hardware state
- config – Sets/views tool configuration
- help – Shows command and other help screens

All commands that operate on hardware CSRs by accepting names as operands use a common CSR name format (see CSR Addressing).

All standard commands include help screens describing their usage and operation. While it is highly recommended and desired, commands that extend the capabilities of the tool may not provide such help text.

A.2.2.1 Command Aliasing

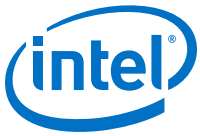
The interactive command interface supports aliasing for command names. Some commonly used commands have a preset alias name (usually, the first letter of the command) that remains valid regardless of other available commands.

Commands that do not have a preset alias name are automatically aliased by the tool to the shortest string that uniquely identifies the commands. This can be used in combination with the command auto-completion support to execute commands in a more convenient manner.

Note: It is important to note that automatic aliases are not static. These aliases can change depending on the availability of other commands, which may change the string required to uniquely identify a particular command. Therefore, Intel recommends that you use the full command names in `hfidiags` scripts.

A.2.3 CSR Addressing

All tool commands that accept CSR names as operands accept and use the following CSR specification. The information below is also provided as help text within the tool.



The CSR specification format can be used to reference hardware CSRs by their symbolic names instead of address offsets. The CSR names are defined to match the hardware specification documents.

There are three types of hardware CSRs:

- **Scalar CSRs** are CSRs that have only one instance defined in the hardware.
- **Context-index CSRs** are CSRs that have one instance per send and/or receive context (depending on the section where they are defined).
- **Context- and array-index CSRs** are CSRs that have an array of instances for every send and/or receive context (depending on the section where the CSR is defined).

The CSR specification format defines a common way to reference any of the CSRs, independent of the type of CSR.

A.2.3.1 Index Formatting

CSRs that support indexing (context-indexed and context- and array-indexed) can be a single index or an index range. A single index references only the CSR or array instance indexed. A range references all CSRs or array instances between the starting and ending index, inclusively.

Both context indexes and array indexes support the above described format.

A.2.3.2 Scalar CSR Specification

Scalar CSRs are CSRs that have only one instance defined in the hardware. These CSRs do not support any type of indexing. If indexing is used, an error is reported. To reference a scalar CSR, only the name of that CSR is required.

Example:

```
read CceRevision
```

A.2.3.3 Context-Indexed or Array-Only-Index CSR Specification

Context-index CSR are CSRs that have one instance per send or receive context.

Array-only-index CSRs are CSRs that have multiple instances defined, however, these instances are global for the entire chip. Examples of such CSRs are the counter CSRs.

To reference the CSR index for a particular context or a particular array element, a standard one-dimensional array indexing syntax is used. For example:

```
<CSR>'['<index>']'
```

The index is optional. If the index is omitted, the command performs its action on every CSR in the CSR set. For example, the command `read RcvCtxtCtrl` reads and displays the value of the `RcvCtxtCtrl` CSR for each of the receive contexts defined by the hardware.

Examples:

```
read SendCtxtCtrl[3]
read RcvCtxtCtrl[1-4]
```

A.2.3.4 Context- and Array-Indexed CSR Specification

Context- and array-index CSRs are CSRs that have a number of instances defined per send or receive context. If the hardware defines *N* contexts of a certain type (send or receive) and *M* instances per context, then the CSR has a total of *NxM* instances.

To reference such CSRs, a standard two-dimensional array indexing syntax is used:

```
<CSR>'['<index>']''['<index>']'
```

Both the context index and the array index are optional. However, if only one index is present, it is assumed that it is the context index. In other words, there is no support for short-circuiting the context index of the CSR.

As with the context-indexed CSR specification, if any of the indexes are missing, the operation is carried out on the entire range.

Examples:

```
read RcvTidFlowTable[3][0]
read RcvTidFlowTable[4-6][3]
read RcvTidFlowTable[4-6][0-3]
```

A.2.3.5 Bit Fields

All three types of CSR specifications support the use of bit fields. When bit fields are used, the commands are limited to performing their action only on the specified bit field. The rest of the CSR is ignored.

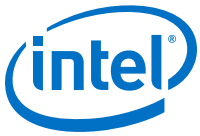
Currently, only one bit field at a time can be specified. If multiple bit fields within a CSR must be operated on, multiple commands must be issued—one for each bit field.

Use the following syntax to specify a CSR bit field:

```
<CSR>'['<index>']''['<index>']'
```

where *<CSR>* is one of the three CSR types. The following table lists the CSR types and the appropriate format:

CSR Type	Bit Field Addressing Format
Scalar CSRs	*<CSR name>*'.'*<bit field>*
Context-indexed or Array-only-index CSRs	*<CSR name>*'['*<index>*']''.'<bit field>
Context- and Array-indexed CSRs	*<CSR name>*'['*<index>*']''['*<index>*']''.'*<bit field>*



A.2.3.6 Symbolic CSR Indexes

Some Array-only-index CSRs defined in the hardware specification include symbolic references to their indexes. Examples of such CSRs are the counter CSRs in the send block. Though the tool supports use of these symbolic references, it does not do any special processing on the symbolic references such as verification and reference-CSR matching (that is, whether the symbolic reference applies to the CSR). Therefore, these symbolic references are treated just like names for numeric indexes. This means that they could be used to index any single- or multi-dimensional CSR, provided that the CSR has sufficient depth in the dimension in question.

A.2.3.7 Globals/Wildcards

The `hfidiags` user interface supports several regular-expression style wildcards that can be used to address multiple CSRs and/or bit fields.

Wildcard	Description
`*`	Matches any (zero or more) alpha-numeric set of characters.
`+`	Matches one or more alpha-numeric characters.
`?`	Matches exactly one alpha-numeric character.
`{n}`	Matches exactly `n` alpha-numeric characters.
`{n,m}`	Matches between `n` and `m` alpha-numeric characters.

When using the wildcards, all other CSR addressing options are still supported. CSR ranges (both context and array indexes) can still be used as well as bit fields. Furthermore, the wildcards can be used on the CSR bit field names.

Wildcard characters can be used multiple times within the same scope (CSR name or bit field name). For example, this could be used to get all CSR values for specific class of CSRs.

When the wildcard characters are used for both CSR names and CSR bit fields, only CSRs that match both criteria are returned. This means that from the set of CSRs that match the CSR name pattern, only those that contain fields matching the bit field pattern are returned.

While the wildcards above are similar to those used in most regular expressions, other special character wildcards are not supported. This is especially true for `.` (period) and `[]` (square brackets) due to their special meaning in CSR addressing. Another important consideration is that, while the actual wildcard characters are supported, non-greedy matching combinations (supported by some regular expression syntaxes) are not supported. Therefore, a matching character combination of `+?`, for example, does not produce the desired result.

A.3 Command Descriptions

This section describes the `hfidiags` tool's default commands and custom commands, which allow you to extend the tool's functionality.

Note: All command names are abbreviated to their first letter if that first letter is unique. If two commands start with the same letter, the abbreviation is not present.

A.3.1 Default Command Descriptions

This section describes all the default commands that are distributed with the `hfidiags` tool.

A.3.1.1 config Command

Syntax

```
config [<option name>] [<new value>]
```

Description

The `config` command allows you to configure the look, feel, and behavior of the UI. UI options are not persistent.

The following configuration options are supported:

Option Name	Type	Default	Description
`ignore_debug_registers`	Boolean	True	Do not process CSRs marked as DEBUG CSRs.
`last_on_enter`	Boolean	True	Repeat last command when **Enter** is pressed on an empty command line.
`output_paging`	Boolean	True	Use command output paging. When enabled, command output that is longer than the screen height is paused when it fills up the screen.
`show_64bit_fields`	Boolean	False	Show CSR bit fields as 64-bit value. Otherwise, only use enough bits to cover the width of the field.
`show_fields`	Boolean	True	Show CSR bit fields in the output of any command.
`show_register_bitfields`	Boolean	True	Decode CSR bit fields. When set to *`False`*, only the CSR value is shown.
`show_zero_values`	Boolean	True	Show CSRs that have a value of 0. This is useful when using the `watch` command to monitor multiple CSRs.
`show_zero_bitfield_values`	Boolean	True	When showing decoded CSR bit fields, show bit fields that have values of 0.

The `config` command is used to display the current values of configuration options or to set new values for an option.

To display the current values for all configuration options, issue the `config` command without any arguments:

```
diags >> config

UI Config
------------------------------------------------------------
ignore_debug_CSRs:                                      True
last_on_enter:                                          True
output_paging:                                          True
show_64bit_fields:                                     False
show_CSR_bitfields:                                     True
```

```
show_zero_values:                                       True

diags >>
```

To display the current value of a single option, issue the `config` command with the option name as the only argument:

```
diags >> config last_on_error
```

To change the value of a UI configuration option, issue the `config` command with the option name and the new value as arguments:

```
diags >> config last_on_error false
```

Configuration options do not support strict type-checking. However, the UI converts the input value to the type of the configuration option. As an example, Boolean type configuration options accept a number of values when being changed. The values `true`, `True`, `TRUE`, `yes`, `Yes`, `YES`, and `1` are all evaluated to *`True`*. Similarly, the values `false`, `False`, `FALSE`, `no`, `No`, `NO`, and `0` are all evaluated to *`False`*.

A.3.1.2 decode Command

Syntax

```
decode <CSR> <hex value>
```

Description

Most output related to the Intel® Omni-Path HFI Silicon 100 Series displays CSR values as 64-bit hexadecimal values that represent the value of the entire CSR. However in most cases, each CSR encodes multiple pieces of information in bit fields.

The `decode` command allows you to parse the 64-bit CSR value into the values of the individual bit fields encoded by the CSR. It does so by only considering the bits of the 64-bit CSR value that cover each bit field and shifting the masked value down.

The best way to describe the `decode` command is by using an example. Assume that the Intel® Omni-Path HFI Silicon 100 Series defines a CSR named `ExampleCSR` at address 0x1000000, which encodes the following bit fields:

<table>
<tr><th colspan="8">Byte 3</th><th colspan="8">Byte 2</th><th colspan="8">Byte 1</th><th colspan="8">Byte 0</th><th rowspan="2">Byte Offset</th></tr>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="3">Reserved1</td><td colspan="5">BitField3</td><td colspan="8">BitField2</td><td colspan="8">BitField1</td><td>0</td></tr>
<tr><td colspan="8">Reserved2</td><td colspan="8">BitField5</td><td colspan="8">BitField4</td><td>4</td></tr>
</table>

In addition, assume that when reading the address of the CSR, the value returned is 0x12345678. Separating the values of the individual bit fields from the value of the 64-bit CSR could be difficult, especially when the bit fields are not aligned on byte boundaries.

By using the `decode` command, you can easily get the values of the individual bit fields:

```
diags >> decode ExampleCSR 0x12345678

ExampleCSR @ 0x1000000                0x0000000012345678
-----------------------------------------------------------
ExampleCSR.Reserved2     (63-54):                0x000000
ExampleCSR.BitField5     (53-42):                     0x0
ExampleCSR.BitField4     (41-32):                     0x0
ExampleCSR.Reserved1     (31-28):                     0x0
ExampleCSR.BitFeild3     (27-20):                   0x123
ExampleCSR.BitField2     (19-6):                     0x59
ExampleCSR.BitField1     (5-0):                      0x18

diags >>
```

Each bit field and its value, as well as the bit range that the bit field occupies, are displayed separately. By default, the width of the bit field values is adjusted to cover just the bit range of the bit field. However, the tool can display full 64-bit wide bit field values by using the `show_64bit_fields` UI configuration option. See the config Command on page 106 for details.

A.3.1.3 help Command

Syntax

```
help [<topic>]
```

Description

The `help` command allows you to retrieve help text for all of the tool's commands from the UI. Without any arguments, the `help` command displays general help text, the list of documented commands, and any miscellaneous help topics. You can use the command or topic as an argument to the `help` command to get additional information.

A.3.1.4 info Command

Syntax

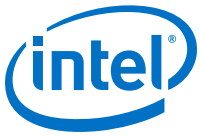
```
info <CSR>
```

Description

The `info` command shows you a description of a CSR. The output lists the address of the CSR in the CSR memory space and all its defined bit fields. Each defined bit field shows the values of its `ACCESS`, `SHIFT`, `MASK`, and `SMASK` attributes, where the attributes are defined as:

Value	Description
`ACCESS`	The value of the access attribute for the bit field.

continued...



Value	Description
	The hardware supports several types of access attributes that define the hardware behavior when accessing the bit field. Consult the Intel® Omni-Path HFI Silicon 100 Series Specification document for more information.
SHIFT	The starting bit index of the bit field. Hardware CSRs are defined as little-endian CSRs. Therefore, the starting bit index is the least significant bit of the bit field.
MASK	The bit mask indicating the width of the field.
SMASK	The bit mask defined by the MASK attribute shifted right by the value of the SHIFT attribute.

In cases where the queried CSR has multiple instances, is an array, or both, the address shown is the starting address. That is, the address in the CSR address map of the first instance or element.

While the `info` command accepts all supported CSR addressing formats, it only displays the information for the first instance or element of the CSR. The largest impact of this is on the CSR address shown, as described in the previous paragraph.

A.3.1.5 read Command

Syntax

```
read <CSR> | <address> [<size>]
```

Description

The `read` command allows you to read the values of hardware CSRs or memory addresses. Two formats are supported using either the CSR Addressing formats or a memory address-length combination.

The first form of the command (`read <CSR>`) reads and displays the value of a single or multiple CSRs. It accepts all CSR addressing formats and methods described in the CSR Addressing on page 102 section.

By default, the output of the `read` command in its first form is the name of the read CSR and its address in the CSR memory map along with the full 64-bit value of the CSR, followed by the fully decoded values of all the CSR's defined bit fields:

```
diags >> read CceRevision

CceRevision @ 0x0                          0x0000000003020700
-----------------------------------------------------------
CceRevision.Reserved_63_40     (63-40):          0x000000
CceRevision.BoardIDUpperNibble (39-36):               0x0
CceRevision.BoardIDLowerNibble (35-32):               0x0
CceRevision.SW                 (31-24):              0x03
CceRevision.Arch               (23-16):              0x02
CceRevision.ChipRevMajor       (15-8):               0x07
CceRevision.ChipRevMinor       (7-0):                0x00
```

The decoding of the CSR bit fields can be controlled with the `show_CSR_bitfields` configuration option (see config Command on page 106).

By default, the `read` command shows all defined bit fields. However, bit fields with values of 0 can be omitted from the output by using the `show_zero_values` configuration option.

The second form of the command (`read <address> [<size>]`) reads entire sections of the memory space. With this form of the command, the *<address>* argument is the starting address of the section to be read and the *<size>* argument is the length (in bytes) to be read. The *<size>* argument is optional, and if not present, the size read defaults to eight bytes. Both the address and size values are a multiple of eight bytes.

When using the second form of the `read` command, no CSR decoding is performed. The output of the command is the raw memory values read from the memory map:

```
diags >> read 0x0 40

0x0: 0x0000000003020700 0x0000000018030200
0x0000000000000000 0x0000000000000000
0x20: 0x0000000000000000
```

The output is in the form of lines. Each line is prefaced by the starting memory address of the values to come, followed by up to 32 bytes (in four groups of eight bytes) of data.

A.3.1.6 state Command

Syntax

```
state <subcommand> [<subcommand arguments>]
    subcommand syntax:
        capture
        load <filename>
        save [<state>] <filename>
        diff <state1> <state2>
        switch <state>
        show
```

Description

The `state` command allows you to capture, save, and/or use the complete hardware state of the entire chip. The state captured includes the entire memory mapped chip area that includes all memory mapped CSRs.

The hardware state is stored internally and can be saved to a file or loaded from a file.

The internal memory object supports two loaded memory states and a single captured state. Each state capture overwrites the previously captured state. States loaded from files overwrites previously loaded states in a round-robin fashion once both internal slots have been filled.

There are several sub-commands that can be used with the `state` command:

Command	Description
`capture`	Captures the current hardware state to internal memory.

continued...

Command	Description
	This capture does not overwrite any loaded states. Successive captures overwrite each other.
`load`	Loads a saved state from a file on disk into one of the internal memory slots. Once loaded, it can be made the active state and examined. The loaded memory state is not made active by default.
`save`	Saves an internally captured state to a file. This file can later be loaded and used to examine the state of the hardware. By specifying which state to be saved, you can save any previously loaded states or save a newly captured state. When saving the current hardware state (instead of a loaded one), a new capture is done prior to the save.
`diff`	Examines both hardware states and shows any differences in CSR values. The *<stateN>* arguments can be any of the following: • `saved1` – This is the state in the first internal slot. • `saved2` – This is the state in the second internal slot. • `hw` – This is the current hardware state. Using this state causes the tool to capture a new hardware state prior to the comparison.
`switch`	Switches the active state to `<state>`. Values possible for `<state>` are the same as with the `diff` argument. When the state is switched to one of the internal states, the `read` command gets CSR values from that saved state.
`show`	Displays information about all hardware states currently in memory (loaded and captured).

It is possible for the CSR definitions used by the `hfidiags` tool to change from time to time. However, for saved states to be valid at a later time, they need to match the CSR definitions. To ensure a correct patch between CSR definitions and saved states, the tool uses a version value.

Each state that is saved to a file includes a version value in its binary data. This version value is also present in the CSR definition file. When a saved state is loaded into the tool, both of the version values are compared and if they do not match, the load halts and the loaded data is not made available.

A.3.1.7 unit Command

Syntax

```
unit <unit>
```

Description

The `unit` command allows you to switch between active units on the same node. This command is useful only if the node on which the `hfidiags` tool is running contains multiple HFIs. Using this command, you can switch between any of the available units without having to restart the tool.

A.3.1.8 write Command

Syntax

```
write <CSR> <hex value> | <address> <length> <hex value>
```

Description

The `write` command allows you to write to hardware CSRs. Two formats are supported using either the CSR addressing formats or a memory address-length combination.

The CSR addressing format writes the specified value to the CSRs given on the command line. The address-length formation writes the specified value starting at address *<address>* and continuing for *<length>* bytes.

In both formats, at least eight bytes are written to a memory location in the hardware memory map. When the CSR addressing formats are used and multiple CSRs are specified on the command line or when the *<length>* argument is larger than eight bytes, the same value is written to all CSRs or eight byte memory increments.

Since the CSR addressing format is supported by the `write` command, it is possible to write only to CSR bit fields. However, because the hardware addressing does not support writing to bit fields natively, modifying a CSR bit field results in a read-modify-write operation.

Writing CSR bit fields is subject to the access type of the CSR bit field. The Intel® Omni-Path host adapter defines several types of access rights for CSR bit fields. Only bit fields with access type of RW (Read/Write) can be modified. This restriction is only applicable to writing CSR bit fields.

Note: It is possible for CSRs to include bit fields of different access types. If a CSR includes a RW and a WO (Write Only) bit fields, the read-modify-write operation results in a potentially unwanted side effect. Reading the WO bit fields returns values of 0. When the modified value is written back, the 0 values could overwrite any other values already stored in the bit fields.

A.3.2 Custom Command Descriptions

This section describes the custom commands that are distributed with the `hfidiags` tool.

A.3.2.1 header Command

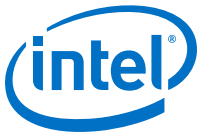
Syntax

```
header <header/structure name>["["<index>[-<index>]"]"]
<hex value>[<hex value>]
```

Description

The `header` command is similar to the `decode` command except that instead of operating on CSR and CSR bit fields, it operates on pre-defined headers and memory structures.

The `header` command decodes entire or partial headers or memory structures. The command splits the input byte stream into double words (DWords) and can process each DWord value separately. To accomplish this, the command supports header and structure indexing on the DWord level. The header/structure indexing uses the same format as the Context-Indexed or Array-Only-Index CSR Specification on page 103 format.



The following headers/structure are defined and decoded by the command:

Name	Type	Endianness	Size (in DWords)
PBC	Structure	Little Endian	2
LRH	Header	Big Endian	2
BTH	Header	Big Endian	3
BTH31	Header	Big Endian	3
KDETH	Header	Little Endian	2 (See Note.)
RHF	Structure	Little Endian	2
SDMADESC	Structure	Little Endian	8
Note: The size of the KDETH header listed here is only the size that is visible to the hardware. The actual size of the KDETH header could differ due to it being software defined after the first two DWords.			

The hexadecimal byte stream input values can be specified as individual DWords (one or more) or a continuous hexadecimal string covering multiple DWords.

If the values specified do not cover the entire size of the header or structure, the indexing format can be used to decode particular fields. If the indexing format is not used, the command assumes that the byte stream represents the start of the header or structure. (It assigns the first byte of the input values to the first byte of the header or structure, taking endianness into account.)

A.3.2.2 mempeek Command

Syntax

```
mempeek <type> [<pid>] <address> [<length>]
```

Description

The `mempeek` command allows you to read data from host physical memory by using either a physical address or a process virtual address. Due to security and reliability concerns, you cannot write to host memory.

The *<type>* argument indicates the type of address given on the command line. It can have two values, `phys` (physical memory) and `virt` (virtual memory). For virtual memory, the *<pid>* argument is required to convert the virtual address to a physical one based on the process' virtual address space.

Note: This command cannot swap in a page that may have been swapped out of main memory.

A.3.2.3 pci Command

Syntax

```
pci read|write <PCI CSR> [<hex value>]
```

Description

The `pci` command allows you to read and write the CSR within the PCIe configuration space of the Intel® Omni-Path HFI Silicon 100 Series hardware. Unlike the general `read` and `write` commands, this command only works on PCIe configuration space registers. Therefore, tab completion actions on the *<PCI CSR>* argument only operate on the set of PCIe configuration space CSRs defined by the hardware.

When writing to PCIe configuration space registers, the *<hex value>* argument is required and is a 32-bit value to be written to the CSR.

While the `read` and `write` commands differ from the `pci` command in the set of CSRs operated on, the `decode` and `info` commands can be used with all CSRs, including PCIe configuration space CSRs.

A.3.2.4 watch Command

Syntax

```
watch <register spec> | <addr> [<register spec> | <addr>] [<timeout>]
```

Description

This command allows you to continuously monitor CSRs for changes. It keeps reading the values of all CSRs or addresses specified on the command line and displays the current values.

The command accepts both symbolic register specifications (see CSR Addressing on page 102) and hardware memory addresses.

Multiple CSRs or addresses can be monitored at the same time by specifying them on the command line. The last command line option is a floating point number representing the interval at which to read the values. Default interval is 1 second.

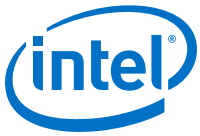
Note: Since there is no reason to monitor CSRs from a saved state, the current active state is ignored and the command monitors only hardware CSRs.

A.4 Extending the Interface

The `hfidiags` tool and framework is designed to be extendable through additions of commands to the user interface. The framework provides a set of common modules, classes, and functions to be used for accessing the hardware, processing input, and displaying output to the terminal. Beyond that, custom commands are free to perform any action needed.

Custom commands are written in the Python* programming language (http://www.python.org) or any other language that can use Python* modules.

All custom command files are stored in the `commands/` directory. At start-up, the tool scans that directory and processes all files in it. If any of the files contains custom commands, those commands are added to the tool's UI.



A file in the `commands/` directory can contain multiple custom commands. When processing a file in the `commands/` directory, the tool expects a specific variable to be defined in the global scope of the file containing a list of all commands provided by the file. The variable that needs to be defined is "`__commands__`". Its value is a list of four or five-element tuples, each of which has the following format:

```
("<name>", <alias>, <action callback>, <tab completion callback>, <help callback>)
```

where:

<name> is the command name to be used in the UI.

<alias> is optional. When present, the element is a string assigning an alias to the command. The alias is registered with the UI and is treated as a normal command.

The rest of the elements are command callbacks (see Command Callbacks on page 115). Only the *<action callback>* is required. If the custom command does not provide the other callbacks, their values can be set to `None`.

If a file in the `commands/` directory does not define the "`__commands__`" variable, it is skipped by the tool even if it does define custom command callbacks.

A.4.1 Command Callbacks

Each custom command provides three callbacks:

1. A callback that executes the command's actions.
2. A callback that performs the command's tab-completion actions.
3. A callback that provides the command's help text.

The callback types are defined in the following sections.

A.4.1.1 Action Callback

The action callback is called when you execute the custom command using the name defined in the tuple. The action callback has the following structure:

```
action_callback(ui_object, command_line, output)
```

where the arguments are:

Argument	Description
ui_object	UI instance. The primary use of this argument is to get access to the hardware and its CSRs.
command_line	String containing the complete command line input with the exception of the first command word. The first command word is not present since it is the one that triggers the call of this callback and is already known.
output	A list passed to the callback where the callback can append text to be displayed. Upon return from the command, if the list is not empty, the UI formats and displays the output to the screen.

The action callback returns `False` in normal conditions even if the actual command action did not result in an error. A `True` return stops the UI from accepting any further commands and the application exits. Command action failures are displayed through one of the error notification methods provided by the UI object.

A.4.1.2 Tab-Completion Callback

The tab-completion callback is called when you attempt a tab-completion after you have already input the name of the command. Tab-completion of actual command names is handled by the UI object. This callback provides tab-completion for any command arguments. The tab-completion callback structure is:

```
tab_completion_callback(ui_object, text, cmd_line, start_index, end_index)
```

where the arguments are:

Argument	Description
ui_object	UI instance. The primary use of this argument is to get access to the hardware and its CSRs.
text	The text that is being completed.
line	The complete command line up to this point.
start_index	Beginning index of the readline tab-completion scope.
end_index	Ending index of the readline tab-completion scope.

This callback returns a list of all possible completions for *text* (if *text* is not an empty string) or all possible next tokens.

A.4.1.3 Help Callback

The help callback is called when you request help for a command through the built-in `help <command/topic>` command.

This callback is called without any arguments. It returns a Python* dictionary of help text broken into sections. The following sections are supported:

- Synopsis
- Description
- Arguments
- Returns
- Known issues
- See also

Intel recommends that custom commands provide Synopsis and Description sections, at a minimum.

The value for each section in the Python* dictionary is a string containing all the text for that section. The help string can contain newline (`\n`) and tab (`\t`) characters to help with formatting.