Intel® True Scale Fabric OFED+ Host Software

User Guide

January 2015

Doc. Number: G91902 Revision: 005US

Contents

4.0 Running MPI on Intel HCAs 53

5.0 Using Other MPIs 73

Figures

Tables

Revision History

Date	Revision	Description
May, 2013	001US	Initial Intel® release
Sept. 2013	002US	Added information in Troubleshooting section for HPL Residual Error Failure
December 2013	003US	Updated Table 10, "Environment Variables Relevant for any PSM" in Section 4.2.14, "Environment Variables" on page 68
July, 2014	004US	Updated the Support link in Section 1.8, "Technical Support" on page 14.
January, 2015	005US	• Updated Table 10, "Environment Variables Relevant for any PSM" in Section 4.2.14, "Environment Variables" on page 68 • Updated the section "AMD CPU Systems" on page 38 • Updated the section "Typical tuning for recent Intel CPUs" on page 39

1.0 Introduction

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

For specific instructions about installing the Intel QLE7340, QLE7342, QMH7342, and QME7342 PCI Express* (PCIe*) adapters see the *Intel® True Scale Fabric Adapter Hardware Installation Guide*, and the initial installation of the Fabric Software, see the *Intel® True Scale Fabric Software Installation Guide*.

1.1 Overview

The material in this documentation pertains to an Intel® True Scale Fabric OFED+ Host Software cluster. A cluster is defined as a collection of nodes, each attached to a fabric through the Intel interconnect.

The Intel® True Scale Fabric Host Channel Adapters (HCA) are True Scale 4X adapters. The quad data rate (QDR) adapters (QLE7300 and QMH7300 series) have a raw data rate of 40Gbps (data rate of 32Gbps). The QLE7300 and QMH7300 series adapters can also run in DDR or SDR mode.

The Intel HCA utilize standard, off-the-shelf InfiniBand* 4X switches and cabling. The Intel interconnect is designed to work with all InfiniBand*-compliant switches.

Note: If you are using the QLE7300 series HCAs in QDR mode, a QDR switch must be used.

Intel OFED+ software is interoperable with other vendors' IBTA InfiniBand*-compliant adapters running compatible OFED releases. There are several options for subnet management in your cluster:

- An embedded subnet manager can be used in one or more managed switches. Intel offers the Embedded Fabric Manager (EFM) for both DDR and QDR switch product lines supplied by your True Scale switch vendor.
- A host-based subnet manager can be used. Intel provides the Intel® True Scale Suite Fabric Manager (FM), as a part of the Intel® True Scale Fabric Suite (IFS).

1.2 Interoperability

Intel OFED+ participates in the standard IB subnet management protocols for configuration and monitoring. Note that:

- Intel OFED+, including Internet Protocol over InfiniBand* (IPoIB), is interoperable with other vendors' InfiniBand*-compliant adapters running compatible OFED releases.
- In addition to supporting running MPI over verbs, Intel provides a high-performance InfiniBand*-compliant vendor-specific protocol, known as PSM. MPIs run over PSM will not inter-operate with other adapters.

Note: See the OpenFabrics web site at www.openfabrics.org for more information on the OpenFabrics Alliance.

1.3 Intended Audience

This guide is intended for end users responsible for administration of a cluster network as well as for end users who want to use that cluster.

This guide assumes that all users are familiar with cluster computing, that the cluster administrator is familiar with Linux* administration, and that the application programmer is familiar with MPI, vFabrics, and Distributed SA.

1.4 How this Guide is Organized

The *Intel® True Scale Fabric OFED+ Host Software User Guide* is organized into these sections:

- Chapter 1.0, "Introduction," provides an overview and describes interoperability.
- Chapter 2.0, "Step-by-Step Cluster Setup and MPI Usage Checklists," describes how to setup your cluster to run high-performance MPI jobs.
- Chapter 3.0, "True Scale Cluster Setup and Administration," describes the lower levels of the supplied Intel OFED+ Host software. This section is of interest to a True Scale cluster administrator.
- Chapter 4.0, "Running MPI on Intel HCAs," helps the Message Passing Interface (MPI) programmer make the best use of the Open MPI implementation. Examples are provided for compiling and running MPI programs.
- Chapter 5.0, "Using Other MPIs," gives examples for compiling and running MPI programs with other MPI implementations.
- Chapter 7.0, "Virtual Fabric support in PSM," describes Intel Performance Scaled Messaging (PSM) that provides support for full Virtual Fabric (vFabric) integration, allowing users to specify InfiniBand* Service Level (SL) and Partition Key (PKey), or to provide a configured Service ID (SID) to target a vFabric.
- Chapter 9.0, "Dispersive Routing," describes dispersive routing in the True Scale fabric to avoid congestion hotspots by "sraying" messages across the multiple potential paths.
- Chapter 10.0, "gPXE," describes open-source Preboot Execution Environment (gPXE) boot including installation and setup.
- Appendix A, "Benchmark Programs," describes how to run Intel's performance measurement programs.
- Appendix B, "Integration with a Batch Queuing System," describes two methods the administrator can use to allow users to submit MPI jobs through batch queuing systems.
- Appendix C, "Troubleshooting," provides information for troubleshooting installation, cluster administration, and MPI.
- Appendix D, "Write Combining," provides instructions for checking write combining and for using the Page Attribute Table (PAT) and Memory Type Range Registers (MTRR).
- Appendix E, "Commands and Files," contains useful programs and files for debugging, as well as commands for common tasks.
- Appendix F, "Recommended Reading," contains a list of useful web sites and documents for a further understanding of the True Scale Fabric, and related information.

In addition, the *Intel® True Scale Fabric Adapter Hardware Installation Guide* contains information on Intel hardware installation and the *Intel® True Scale Fabric Software Installation Guide* contains information on Intel software installation.

1.5 Related Materials

- *Intel® True Scale Fabric Adapter Hardware Installation Guide*
- *Intel® True Scale Fabric Software Installation Guide*
- Release Notes

1.6 Documentation Conventions

This guide uses the following documentation conventions:

- *Note:* provides additional information.
- *Caution:* indicates the presence of a hazard that has the potential of causing damage to data or equipment.
- *Warning:* indicates the presence of a hazard that has the potential of causing personal injury.
- Text in blue font indicates a hyperlink (jump) to a figure, table, or section in this guide, and links to Web sites are also shown in blue. For example:
  - Table 2 lists problems related to the user interface and remote agent.
  - See "Installation Checklist" on page 6.
  - For more information, visit www.intel.com.
- Text in **bold** font indicates user interface elements such as menu items, buttons, check boxes, or column headings. For example:
  - Click the **Start** button, point to **Programs**, point to **Accessories**, and then click **Command Prompt**.
  - Under **Notification Options**, select the **Warning Alarms** check box.
- Text in `Courier` font indicates a file name, directory path, or command line text. For example:
  - To return to the root directory from anywhere in the file structure: Type `cd /root` and press **Enter**.
  - Enter the following command: `sh ./install.bin`
- Key names and key strokes are indicated with **uppercase**:
  - Press **ctrl+P.**
  - Press the **up arrow** key.
- Text in *italics* indicates terms, emphasis, variables, or document titles. For example:
  - For a complete listing of license agreements, refer to the *Intel Software End User License Agreement*.
  - What are *shortcut keys*?
  - To enter the date type *mm/dd/yyyy* (where *mm* is the month, *dd* is the day, and *yyyy* is the year).
- Topic titles between quotation marks identify related topics either within this manual or in the online help throughout this document.

1.7 License Agreements

Refer to the *Intel Software End User License Agreement* for a complete listing of all license agreements affecting this product.

1.8 Technical Support

Intel True Scale Technical Support for products under warranty is available during local standard working hours excluding Intel Observed Holidays. For customers with extended service, consult your plan for available hours. For Support information, see the Support link at www.intel.com/truescale.

§ §

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

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

2.1 Cluster Setup

Perform the following tasks when setting up the cluster. These include BIOS, adapter, and system settings.

1. Make sure that hardware installation has been completed according to the instructions in the *Intel® True Scale Fabric Adapter Hardware Installation Guide* and software installation and driver configuration has been completed according to the instructions in the *Intel® True Scale Fabric Software Installation Guide*. To minimize management problems, the compute nodes of the cluster must have very similar hardware configurations and identical software installations. See "Homogeneous Nodes" on page 46 for more information.
2. Check that the BIOS is set properly according to the instructions in the *Intel® True Scale Fabric Adapter Hardware Installation Guide*.
3. Set up the Distributed Subnet Administration (SA) to correctly synchronize your virtual fabrics. See "Intel Distributed Subnet Administration" on page 27
4. Adjust settings, including setting the appropriate MTU size. See "Adapter and Other Settings" on page 46.
5. Remove unneeded services. See "Remove Unneeded Services" on page 47.
6. Disable powersaving features. See "Host Environment Setup for MPI" on page 48.
7. Check other performance tuning settings. See "Performance Settings and Management Tips" on page 36.
8. Set up the host environment to use `ssh`. Two methods are discussed in "Host Environment Setup for MPI" on page 48.
9. Verify the cluster setup. See "Checking Cluster and Software Status" on page 51.

2.2 Using MPI

1. Verify that the Intel hardware and software has been installed on all the nodes you will be using, and that `ssh` is set up on your cluster (see all the steps in the Cluster Setup checklist).
2. Setup Open MPI. See "Setup" on page 58.
3. Compile Open MPI applications. See "Compiling Open MPI Applications" on page 58
4. Create an `mpihosts` file that lists the nodes where your programs will run. See "Create the `mpihosts` File" on page 59.
5. Run Open MPI applications. See "Running Open MPI Applications" on page 59.

6. Configure MPI programs for Open MPI. See "Configuring MPI Programs for Open MPI" on page 60
7. To test using other MPIs that run over PSM, such as MVAPICH, MVAPICH2, Platform MPI, and Intel MPI, see Section 5.0, "Using Other MPIs" on page 77.
8. To switch between multiple versions of MVAPICH, use the `mpi-selector`. See "Managing MVAPICH, and MVAPICH2 with the `mpi-selector` Utility" on page 80.
9. Refer to "Performance Tuning" on page 36 to read more about runtime performance tuning.
10. Refer to Section 5.0, "Using Other MPIs" on page 77 to learn about using other MPI implementations.

3.0 True Scale Cluster Setup and Administration

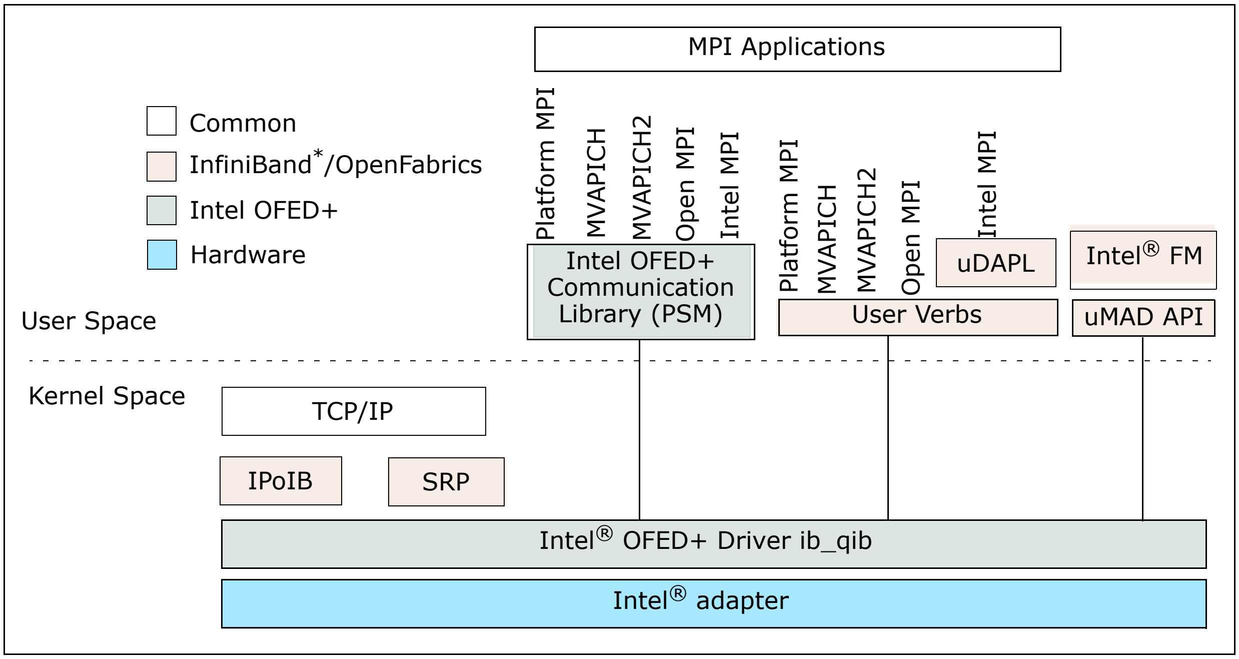
This section describes what the cluster administrator needs to know about the Intel OFED+ software and system administration.

3.1 Introduction

The True Scale driver ib_qib, Intel Performance Scaled Messaging (PSM), accelerated Message-Passing Interface (MPI) stack, the protocol and MPI support libraries, and other modules are components of the Intel OFED+ software. This software provides the foundation that supports the MPI implementation.

Figure 3-1 illustrates these relationships. Note that HP-MPI, Platform MPI, Intel MPI, MVAPICH, MVAPICH2, and Open MPI can run either over PSM or OpenFabrics* User Verbs.

Figure 3-1. Intel OFED+ Software Structure



3.2 Installed Layout

This section describes the default installed layout for the Intel OFED+ software and Intel-supplied MPIs.

Intel-supplied Open MPI, MVAPICH, and MVAPICH2 RPMs with PSM support and compiled with GCC, PGI, and the Intel compilers are installed in directories using the following format:

`/usr/mpi/<compiler>/<mpi>-<mpi_version>-qlc`

For example: `/usr/mpi/gcc/openmpi-1.8.1-qlc`

Intel OFED+ utility programs, are installed in:

`/usr/bin`

`/sbin`

`/opt/iba/*`

Documentation is found in:

`/usr/share/man`

Intel OFED+ Host Software user documentation can be found on the Intel web site on the software download page for your distribution.

Configuration files are found in:

`/etc/sysconfig`

Init scripts are found in:

`/etc/init.d`

The True Scale driver modules in this release are installed in:

`/lib/modules/$(uname -r)/ updates/kernel/drivers/infiniband/hw/qib`

Most of the other OFED modules are installed under the `infiniband` subdirectory. Other modules are installed under:

`/lib/modules/$(uname -r)/updates/kernel/drivers/net`

The RDS modules are installed under:

`/lib/modules/$(uname -r)/updates/kernel/net/rds`

3.3 True Scale and OpenFabrics Driver Overview

The True Scale `ib_qib` module provides low-level Intel hardware support, and is the base driver for both MPI/PSM programs and general OpenFabrics protocols such as IPoIB and sockets direct protocol (SDP). The driver also supplies the Subnet Management Agent (SMA) component.

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

- IPoIB network interface. This component is required for TCP/IP networking for running IP traffic over the True Scale link. It is not running until it is configured.
- OpenSM. This component is disabled at startup. Intel recommends using the Intel® True Scale Suite Fabric Manager (FM), which is included with the IFS or optionally available within the Intel switches. The FM or OpenSM can be installed on one or more nodes with only one node being the master SM.
- SRP (OFED modules). SRP is not running until the module is loaded and the SRP devices on the fabric have been discovered.

- MPI over uDAPL (can be used by Intel MPI). IPoIB must be configured before MPI over uDAPL can be set up.

Other optional drivers can now be configured and enabled, as described in "IPoIB Network Interface Configuration" on page 19.

Complete information about starting, stopping, and restarting the Intel OFED+ services are in "Managing the True Scale Driver" on page 33.

3.4 IPoIB Network Interface Configuration

The following instructions show you how to manually configure your OpenFabrics IPoIB network interface. Intel recommends using the Intel OFED+ Host Software Installation package or the iba_config tool. For larger clusters, Intel® True Scale Fabric Suite FastFabric (FF) can be used to automate installation and configuration of many nodes. These tools automate the configuration of the IPoIB network interface. This example assumes that you are using `sh` or `bash` as your shell, all required Intel OFED+ and OpenFabric's RPMs are installed, and your startup scripts have been run (either manually or at system boot).

For this example, the IPoIB network is 10.1.17.0 (one of the networks reserved for private use, and thus not routable on the Internet), with a /8 host portion. In this case, the netmask must be specified.

This example assumes that no hosts files exist, the host being configured has the IP address 10.1.17.3, and DHCP is not used.

Note: Instructions are only for this static IP address case. Configuration methods for using DHCP will be supplied in a later release.

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

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

2. To verify the configuration, type:

```
ifconfig ib0

ifconfig ib1
```

The output from this command will be similar to:

```
ib0   Link encap:InfiniBand HWaddr

00:00:00:02:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00

inet addr:10.1.17.3  Bcast:10.1.17.255 Mask:255.255.255.0

UP BROADCAST RUNNING MULTICAST  MTU:4096  Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:128

RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

3. Type:

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

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

```
WARNING: pinging broadcast address

PING 10.1.17.255 (10.1.17.255) 517(84) bytes of data.

174 bytes from 10.1.17.3: icmp_seq=0 ttl=174 time=0.022 ms

64 bytes from 10.1.17.1: icmp_seq=0 ttl=64 time=0.070 ms (DUP!)

64 bytes from 10.1.17.7: icmp_seq=0 ttl=64 time=0.073 ms (DUP!)
```

The IPoIB network interface is now configured.

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

```
/etc/init.d/openibd restart
```

Note: The configuration must be repeated each time the system is rebooted.

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

3.5 IPoIB Administration

3.5.1 Stop, Start and Restart the IPoIB Driver

Intel recommends using the Intel® Fabric Installer TUI or `iba_config` command to enable autostart for the IPoIB driver. Refer to the *Intel® True Scale Fabric Software Installation Guide* for more information. For using the command line to stop, start, and restart the IPoIB driver use the following commands.

To stop the IPoIB driver, use the following command:

```
/etc/init.d/openibd stop
```

To start the IPoIB driver, use the following command:

```
/etc/init.d/openibd start
```

To restart the IPoIB driver, use the following command:

```
/etc/init.d/openibd restart
```

3.5.2 Configure IPoIB

Intel recommends using the Intel® Fabric Installer TUI, `fastfabric` command, or `iba_config` command to configure the boot time and autostart of the IPoIB driver. Refer to the *Intel® True Scale Fabric Software Installation Guide* for more information on using the Intel® Fabric Installer TUI. Refer to the *Intel® True Scale Fabric Suite FastFabric User Guide* for more information on using FF. For using the command line to configure the IPoIB driver, edit the IPoIB configuration file as follows:

1. For each IP Link Layer interface, create an interface configuration file, `/etc/sysconfig/network/ifcfg-`*`NAME`*, where *`NAME`* is the value of the *`NAME`* field specified in the `CREATE` block. The following is an example of the `ifcfg-`*`NAME`* file:

```
DEVICE=ib1

BOOTPROTO=static

BROADCAST=192.168.18.255

IPADDR=192.168.18.120

NETMASK=255.255.255.0

ONBOOT=yes

NM_CONTROLLED=no
```

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

2. After modifying the `/etc/sysconfig/ipoib.cfg` file, restart the IPoIB driver with the following:

```
/etc/init.d/openibd restart
```

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 should not be used. Intel supports bonding across HCA ports and bonding port 1 and port 2 on the same HCA.

3.6.1 Interface Configuration Scripts

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

Note: For all of the following OS configuration script examples that set MTU, `MTU=65520` is valid only if all IPoIB slaves operate in connected mode and are configured with the same value. For IPoIB slaves that work in datagram mode, use MTU=2044. If the MTU is not set correctly or the MTU is not set at all (set to the default value), performance of the interface may be lower.

3.6.1.1 Red Hat Enterprise Linux*

The following is an example for `bond0` (master). The file is named `/etc/sysconfig/network-scripts/ifcfg-bond0`:

```
DEVICE=bond0

IPADDR=192.168.1.1

NETMASK=255.255.255.0
```

```
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
MTU=65520
BONDING_OPTS="primary=ib0 updelay=0 downdelay=0"
```

The following is an example for `ib0` (slave). The file is named `/etc/sysconfig/network-scripts/ifcfg-ib0`:

```
DEVICE=ib0
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
TYPE=InfiniBand
PRIMARY=yes
```

The following is an example for `ib1` (slave 2). The file is named `/etc/sysconfig/network-scripts/ifcfg-ib1`:

```
DEVICE=ib1
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
TYPE=InfiniBand
```

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

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

3.6.1.2 SuSE Linux* Enterprise Server (SLES)

The following is an example for `bond0` (master). The file is named `/etc/sysconfig/network-scripts/ifcfg-bond0`:

```
DEVICE="bond0"

TYPE="Bonding"

IPADDR="192.168.1.1"

NETMASK="255.255.255.0"

NETWORK="192.168.1.0"

BROADCAST="192.168.1.255"

BOOTPROTO="static"

USERCTL="no"

STARTMODE="onboot"

BONDING_MASTER="yes"

BONDING_MODULE_OPTS="mode=active-backup miimon=100 primary=ib0
updelay=0 downdelay=0"

BONDING_SLAVE0=ib0

BONDING_SLAVE1=ib1

MTU=65520
```

The following is an example for `ib0` (slave). The file is named `/etc/sysconfig/network-scripts/ifcfg-ib0`:

```
DEVICE='ib0'

BOOTPROTO='none'

STARTMODE='off'

WIRELESS='no'

ETHTOOL_OPTIONS=''

NAME=''

USERCONTROL='no'

IPOIB_MODE='connected'
```

The following is an example for `ib1` (slave 2). The file is named `/etc/sysconfig/network-scripts/ifcfg-ib1`:

```
DEVICE='ib1'

BOOTPROTO='none'

STARTMODE='off'

WIRELESS='no'

ETHTOOL_OPTIONS=''

NAME=''

USERCONTROL='no'

IPOIB_MODE='connected'
```

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

```
RUN_PARALLEL="yes"
```

3.6.2 Verify IB Bonding is Configured

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

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

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

```
# cat /proc/net/bonding/bond0

Ethernet Channel Bonding Driver: v3.2.3 (December 6, 2007)



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 Subnet Manager Configuration

Intel recommends using the Intel® True Scale Suite Fabric Manager (FM) to manage your fabric. Refer to the *Intel® True Scale Fabric Suite Fabric Manager User Guide* for information on configuring the FM.

OpenSM is a component of the OpenFabrics project that provides a Subnet Manager (SM) for the fabric. This package can optionally be installed on any machine, but only needs to be enabled on the machine in the cluster that will act as a subnet manager. You cannot use OpenSM if any of your fabric switches provide a subnet manager, or if you are running a host-based SM, for example the FM.

Warning: Don’t run OpenSM with FM in the same fabric.

If you are using the Installer tool, you can set the OpenSM default behavior at the time of installation.

OpenSM only needs to be enabled on the node that acts as the subnet manager.Toenable OpenSM the `iba_config` command can be used or the `chkconfig` command (as a root user) can be used on the node where it will be run. The `chkconfig` command to enable the OpenSM is:

```
chkconfig opensmd on
```

The `chkconfig` command to disable it on reboot is:

```
chkconfig opensmd off
```

You can start `opensmd` without rebooting your machine by typing:

```
/etc/init.d/opensmd start
```

You can stop `opensmd` by typing:

```
/etc/init.d/opensmd stop
```

If you want to pass any arguments to the OpenSM program, modify the following file, and add the arguments to the `OPTIONS` variable:

```
/etc/init.d/opensmd
```

For example:

```
Use the UPDN algorithm instead of the Min Hop algorithm.
OPTIONS="-R updn"
```

For more information on OpenSM, see the OpenSM man pages, or look on the OpenFabrics web site.

3.8 Intel Distributed Subnet Administration

As True Scale clusters are scaled into the Petaflop range and beyond, a more efficient method for handling queries to the FM is required. One of the issues is that while the FM can configure and operate that many nodes, under certain conditions it can become overloaded with queries from those same nodes.

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

In the past, MPI implementations have side-stepped this problem by hand crafting path records themselves, but this solution cannot be used if advanced fabric management techniques such as virtual fabrics and mesh/torus 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 Distributed Subnet Administration (SA) solves this problem by allowing each node to locally replicate the path records needed to reach the other nodes on the fabric. At boot time, each Distributed SA 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 Distributed SA'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 Distributed SA updates the affected portions of the database. The Distributed SA can be installed and run on any node in the fabric. It is only needed on nodes running SHMEM and MPI applications.

3.8.1 Applications that use Distributed SA

The Intel PSM Library has been extended to take advantage of the Distributed SA. Therefore, all MPIs that use the Intel PSM library can take advantage of the Distributed SA. Other applications must be modified specifically to take advantage of it. For developers writing applications that use the Distributed SA, refer to the header file `/usr/include/Infiniband/ofedplus_path.h` for information on using Distributed SA APIs. This file can be found on any node where the Distributed SA is installed. For further assistance please contact Intel Support.

3.8.2 Virtual Fabrics and the Distributed SA

The IBTA standard states that applications can be identified by a Service ID (SID). The FM uses SIDs to identify applications. One or more applications can be associated with a Virtual Fabric using the SID. The Distributed SA is designed to be aware of Virtual Fabrics, but to only store records for those Virtual Fabrics that match the SIDs in the Distributed SA's configuration file. The Distributed SA recognizes when multiple SIDs match the same Virtual Fabric and will only store one copy of each path record within a Virtual Fabric. SIDs that match more than one Virtual Fabric will be associated with a single Virtual Fabric. The Virtual Fabrics that do not match SIDs in the Distributed SA's database will be ignored.

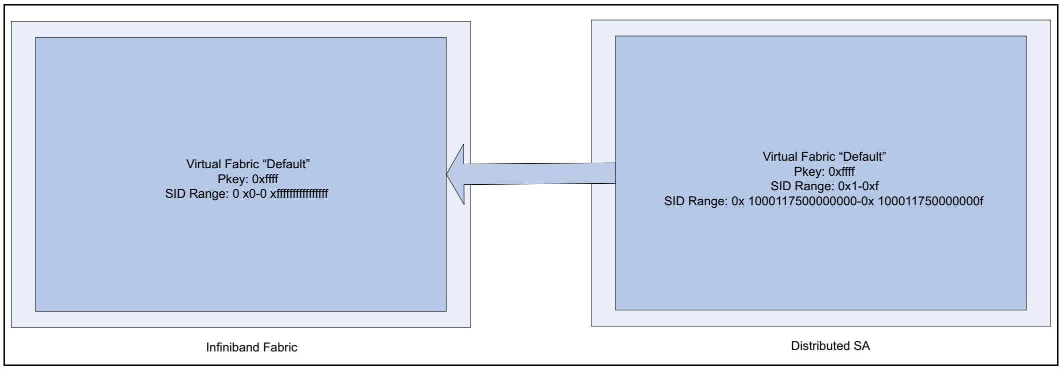
3.8.3 Configuring the Distributed SA

In order to absolutely minimize the number of queries made by the Distributed SA, it is important to configure it correctly, both to match the configuration of the FM and to exclude those portions of the fabric that will not be used by applications using the Distributed SA. The configuration file for the Distributed SA is named `/etc/sysconfig/iba/dist_sa.conf`.

3.8.4 Default Configuration

As shipped, the FM creates a single virtual fabric, called “Default” and maps all nodes and Service IDs to it, and the Distributed SA ships with a configuration that lists a set of thirty-one SIDs, 0x1000117500000000 through 0x100011750000000f and 0x1 through 0xf. This results in an arrangement like the one shown in Figure 3-2

Figure 3-2. Distributed SA Default Configuration



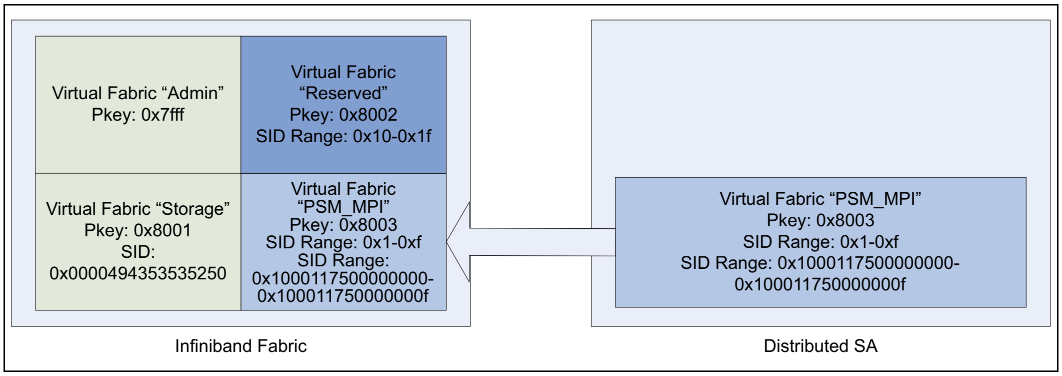
If you are using the FM in its default configuration, and you are using the standard Intel PSM SIDs, this arrangement will work fine and you will not need to modify the Distributed SA's configuration file - but notice that the Distributed SA has restricted the range of SIDs it cares about to those that were defined in its configuration file. Attempts to get path records using other SIDs will not work, even if those other SIDs are valid for the fabric. When using this default configuration it is necessary that MPI applications only be run using one of these 32 SIDs.

3.8.5 Multiple Virtual Fabrics Example

A person configuring the physical fabric may want to limit how much bandwidth MPI applications are permitted to consume. In that case, they may re-configure the FM, turning off the “Default” Virtual Fabric and replacing it with several other Virtual Fabrics.

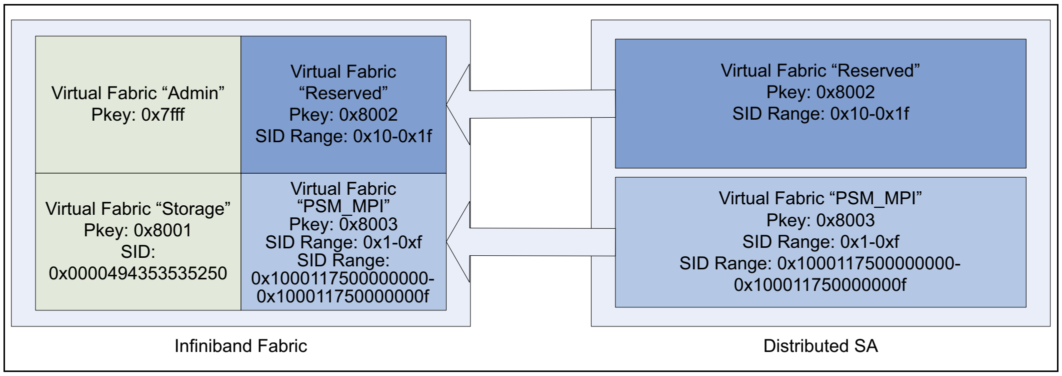
In Figure 3-3, the administrator has divided the physical fabric into four virtual fabrics: “Admin” (used to communicate with the FM), “Storage” (used by SRP), “PSM_MPI” (used by regular MPI jobs) and a special “Reserved” fabric for special high-priority jobs.

Figure 3-3. Distributed SA Multiple Virtual Fabrics Example



Due to the fact that the Distributed SA was not configured to include the SID Range 0x10 through 0x1f, it has simply ignored the "Reserved" VF. Adding those SIDs to the intel_sa.conf file solves the problem as shown in Figure 3-4.

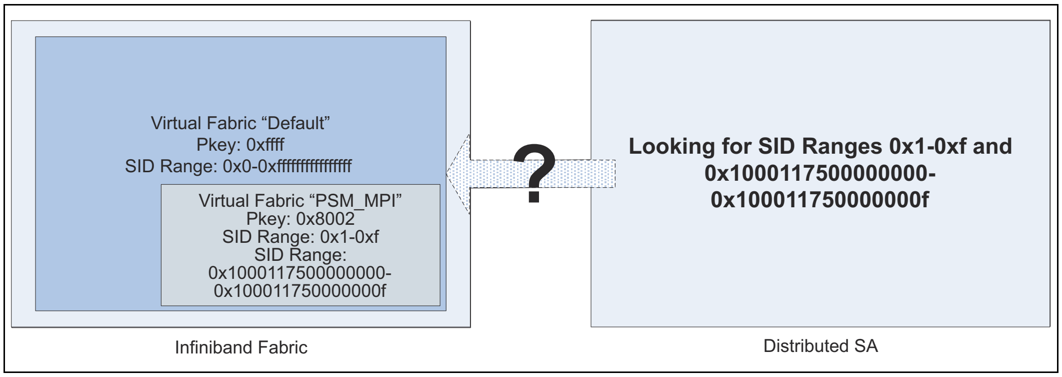
Figure 3-4. Distributed SA Multiple Virtual Fabrics Configured Example



3.8.6 Virtual Fabrics with Overlapping Definitions

As defined, SIDs should never be shared between Virtual Fabrics. Unfortunately, it is very easy to accidentally create such overlaps. Figure 3-5 shows an example with overlapping definitions.

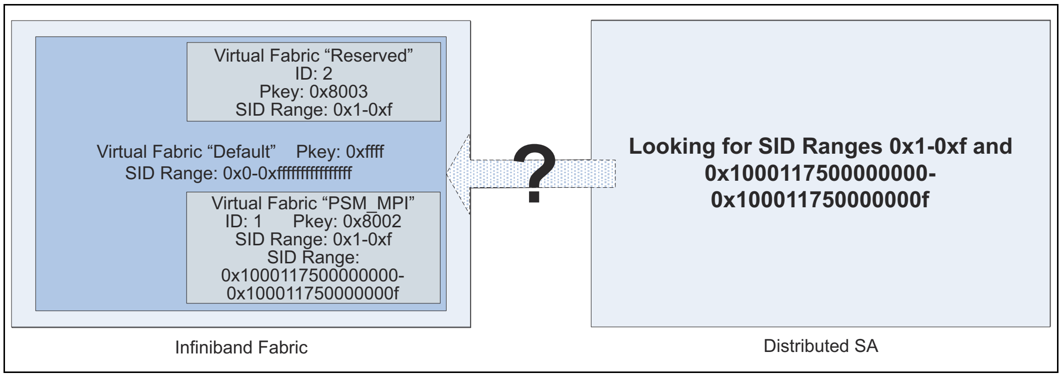
Figure 3-5. Virtual Fabrics with Overlapping Definitions



In Figure 3-5, the fabric administrator enabled the "PSM_MPI" Virtual Fabric without modifying the "Default" Virtual Fabric. As a result, the Distributed SA sees two different virtual fabrics that match its configuration file.

In Figure 3-6, the person administering the fabric has created two different Virtual Fabrics without turning off the Default - and two of the new fabrics have overlapping SID ranges.

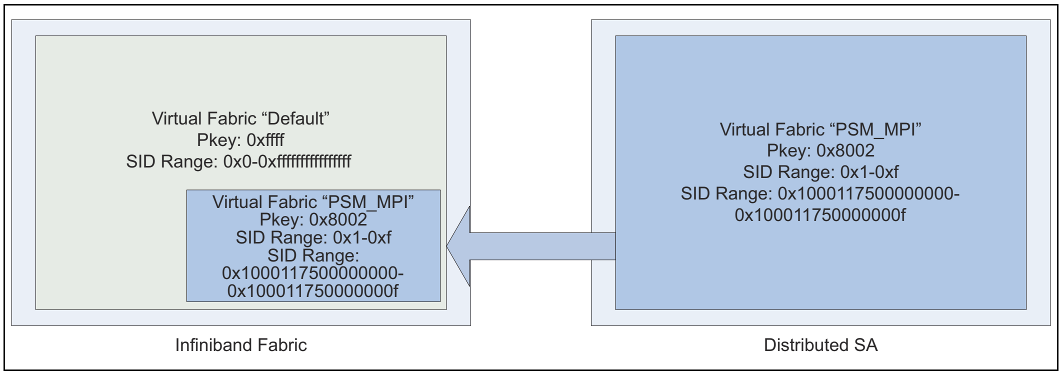
Figure 3-6. Virtual Fabrics with PSM_MPI Virtual Fabric Enabled



In Figure 3-6, the administrator enabled the "PSM_MPI" fabric, and then added a new "Reserved" fabric that uses one of the SID ranges that "PSM_MPI" uses. When a path query has been received, the Distributed SA deals with these conflicts as follows:

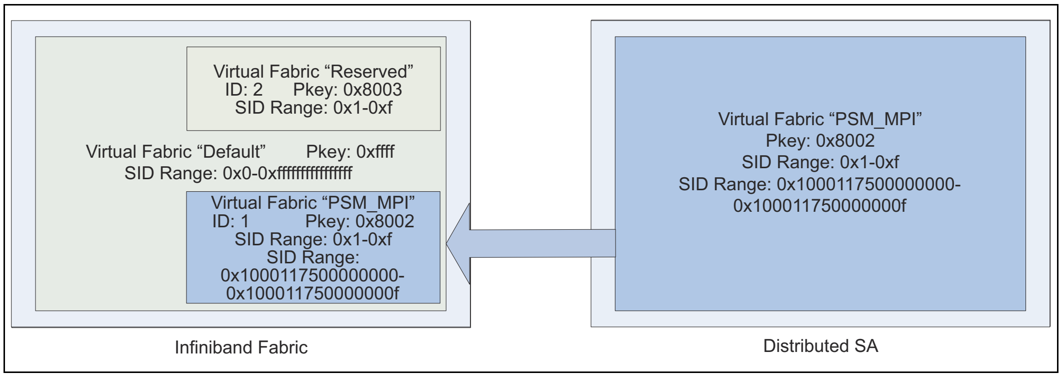
First, any virtual fabric with a pkey of 0xffff or 0x7fff is considered to be an Admin or Default virtual fabric. This Admin or Default virtual fabric is treated as a special case by the Distributed SA and is used only as a last resort. Stored SIDs are only mapped to the default virtual fabric if they do not match any other Virtual Fabrics. Thus, in the first example, Figure 3-6, the Distributed SA will assign all the SIDs in its configuration file to the "PSM_MPI" Virtual Fabric as shown in Figure 3-7.

Figure 3-7. Virtual Fabrics with all SIDs assigned to PSM_MPI Virtual Fabric



Second, the Distributed SA handles overlaps by taking advantage of the fact that Virtual Fabrics have unique numeric indexes. These indexes are assigned by the FM in the order which the Virtual Fabrics appear in the configuration file. These indexes can be seen by using the command `iba_saquery -o vfinfo` command. The Distributed SA will always assign a SID to the Virtual Fabric with the lowest index, as shown in Figure 3-8. This ensures that all copies of the Distributed SA in the fabric will make the same decisions about assigning SIDs. However, it also means that the behavior of your fabric can be affected by the order you configured the virtual fabrics.

Figure 3-8. Virtual Fabrics with Unique Numeric Indexes



In Figure 3-8, the Distributed SA assigns all overlapping SIDs to the “PSM_MPI” fabric because it has the lowest Index

Note: The Distributed SA makes these assignments not because they are right, but because they allow the fabric to work even though there are configuration ambiguities. The correct solution in these cases is to redefine the fabric so that no node will ever be a member of two Virtual Fabrics that service the same SID.

3.8.7 Distributed SA Configuration File

The Distributed SA configuration file is `/etc/sysconfig/iba/intel_sa.conf`. It has several settings, but normally administrators will only need to deal with two or three of them.

3.8.7.1 SID

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

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

```
SID=0x0a1

SID=0x0a2

SID=0x0a3

SID=0x1a1

SID=0x1a2

SID=0x1a3

SID=0x2a1

SID=0x2a2
```

```
SID=0x2a3
```

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

3.8.7.2 ScanFrequency

Periodically, the Distributed SA will completely re synchronize its database. This also occurs if the 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 can 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 intel_sa.conf file:

```
ScanFrequency=900
```

3.8.7.3 LogFile

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

```
LogFile=/var/log/SAReplica.log
```

3.8.7.4 Dbg

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

- Dbg=1 or Dbg=2: Alerts and Critical Errors
  Only errors that will cause the Distributed SA to terminate will be reported.
- Dbg=3: Errors
  Errors will be reported, but nothing else. (Includes Dbg=1 and Dbg=2)
- Dbg=4: Warnings
  Errors and warnings will be reported. (Includes Dbg=3)
- Dbg=5: Normal
  Some normal events will be reported along with errors and warnings. (Includes Dbg=4)
- Dbg=6: Informational Messages
  In addition to the normal logging, Distributed SA will report detailed information about its status and operation. Generally, this will produce too much information for normal use. (Includes Dbg=5)
- Dbg=7: Debugging
  This should only be turned on at the request of Intel Support. This will generate so much information that system operation will be impacted. (Includes Dbg=6)

3.8.7.5 Other Settings

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

3.9 Changing the MTU Size

The Maximum Transfer Unit (MTU) size enabled by the True Scale HCA and set by the driver is 4KB. To see the current MTU size, and the maximum supported by the HCA, type the command:

```
$ ibv_devinfo
```

If the switches are set at 2K MTU size, then the HCA will automatically use this as the active MTU size, there is no need to change any file on the hosts.

To ensure that the driver on this host uses 2K MTU, add the following options line (as a root user) in to the configuration file:

```
options ib_qib ibmtu=4
```

Table 3-1 shows the value of each ibmtu number designation.

Table 3-1. ibmtu Values

Number Designation	Value in Bytes
1	256
2	512
3	1024
4	2048
5	4096

The following is a list of the configuration file locations for each OS:

- For SLES use file: `/etc/modprobe.conf.local`
- For RHEL use file:`/etc/modprobe.d/ib_qib.conf`

Restart the driver as described in Managing the True Scale Driver.

Note: To use 4K MTU, set the switch to have the same 4K default. If you are using Intel® 12000 Series Switches, refer to the *Intel® True Scale Fabric Suite FastFabric User Guide* for externally managed switches, and to the *Intel® True Scale Fabric Suite FastFabric Command Line Interface Reference Guide* for the internally managed switches.

Note: For other switches, see the vendors' documentation.

3.10 Managing the True Scale Driver

The startup script for `ib_qib` is installed automatically as part of the software installation, and normally does not need to be changed. It runs as a system service.

The primary configuration file for the True Scale driver `ib_qib` and other modules and associated daemons is `/etc/infiniband/openib.conf`.

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

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

See the `ib_qib` man page for more details.

3.10.1 Configure the True Scale Driver State

Use the following commands to check or configure the state. These methods will not reboot the system.

To check the configuration state, use this command. You do not need to be a root user:

```
$ chkconfig --list openibd
```

To enable the driver, use the following command (as a root user):

```
# chkconfig openibd on 2345
```

To disable the driver on the next system boot, use the following command (as a root user):

```
# chkconfig openibd off
```

Note: This command does not stop and unload the driver if the driver is already loaded nor will it start the driver.

3.10.2 Start, Stop, or Restart True Scale Driver

Restart the software if you install a new Intel OFED+ Host Software release, change driver options, or do manual testing.

Intel recommends using `/etc/init.d/openibd` to stop, stat and restart the ib_qib driver. For using the command line to stop, start, and restart (as a root user) the True Scale driver use the following syntex:

```
# /etc/init.d/openibd [start | stop | restart]
```

Warning: If the FM, or OpenSM is configured and running on the node, it must be stopped before using the `openibd stop` command, and may be started after using the `openibd start` command.

This method will not reboot the system. The following set of commands shows how to use this script.

When you need to determine which True Scale and OpenFabrics modules are running, use the following command. You do not need to be a root user.

```
$ lsmod | egrep ’ipath_|ib_|rdma_|findex’
```

You can check to see if `opensmd` is configured to autostart by using the following command (as a root user); if there is no output, `opensmd` is not configured to autostart:

```
# /sbin/chkconfig --list opensmd | grep -w on
```

3.10.3 Unload the Driver/Modules Manually

You can also unload the driver/modules manually without using `/etc/init.d/openibd`. Use the following series of commands (as a root user):

```
# umount /ipathfs

# fuser -k /dev/ipath* /dev/infiniband/*

# lsmod | egrep '^ib_|^rdma_|^iw_' | xargs modprobe -r
```

3.10.4 True Scale Driver Filesystem

The True Scale driver supplies a filesystem for exporting certain binary statistics to user applications. By default, this filesystem is mounted in the `/ipathfs` directory when the True Scale script is invoked with the `start` option (e.g. at system startup). The filesystem is unmounted when the True Scale script is invoked with the `stop` option (for example, at system shutdown).

Here is a sample layout of a system with two cards:

```
/ipathfs/0/flash

/ipathfs/0/port2counters

/ipathfs/0/port1counters

/ipathfs/0/portcounter_names

/ipathfs/0/counter_names

/ipathfs/0/counters

/ipathfs/driver_stats_names

/ipathfs/driver_stats

/ipathfs/1/flash

/ipathfs/1/port2counters

/ipathfs/1/port1counters

/ipathfs/1/portcounter_names

/ipathfs/1/counter_names

/ipathfs/1/counters
```

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

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

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

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

3.11 More Information on Configuring and Loading Drivers

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

3.12 Performance Settings and Management Tips

The following sections provide suggestions for improving performance and simplifying cluster management. Many of these settings will be done by the system administrator.

3.12.1 Performance Tuning

Tuning compute or storage (client or server) nodes with True Scale HCAs for MPI and verbs performance can be accomplished in several ways:

- Run the `ipath_perf_tuning` script in automatic mode (See "Performance Tuning using `ipath_perf_tuning` Tool" on page 44) (easiest method)
- Run the `ipath_perf_tuning` script in interactive mode (See "Performance Tuning using `ipath_perf_tuning` Tool" on page 44 or see `man ipath_perf_tuning`). This interactive mode allows more control, and should be used for tuning storage (client or server) nodes.
- Make changes to ib_qib driver parameter files, the BIOS or system services using the information provided in the following sections

Note: The `modprobe.conf` file name will be used in this section for the ib_qib module configuration file, which has various paths and names in the different Linux distributions as shown in the following list:

- For SLES or RHEL use file `/etc/modprobe.d/ib_qib.conf`

3.12.1.1 Systems in General (With Either Intel or AMD CPUs)

For best performance on dual-port HCAs on which only the first port is connected and active, the module parameter line in the modprobe.conf file should include the following:

```
options ib_qib singleport=1
```

Note: The option `singleport=1` assigns all of the hardware contexts to the only active port, enhancing the performance of that port.

3.12.1.1.1 Services

Turn off the specified daemons using one of the following commands according to which OS is being used:

- For RHEL or similar systems use:

```
/sbin/chkconfig --level 12345 cpuspeed off
```

- For SLES systems use:

```
/sbin/chkconfig --level 12345 powersaved off
```

If `cpuspeed` or `powersaved` are being used as part of implementing Turbo modes to increase CPU speed, then they can be left on. With these daemons left on, micro-benchmark performance results may be more variable from run-to-run.

For compute nodes, set the default runlevel to 3 to reduce overheads due to unneeded processes, such as the X Windows system and GUIs that use the overhead. Reboot the system for this change to take effect.

3.12.1.1.2 Default Parameter Settings

The qib driver makes certain settings by default based on a check of which CPUs are in the system. Since these are done by default, no user- or ipath_perf_tuning-generated changes need to be made in the modprobe configuration file. It doesn't hurt anything if these settings are in the file, but they are not necessary.

On all systems, the qib driver behaves as if the following parameters were set:

```
rcvhdrcnt=4096
```

If you run a script, such as the following:

```
for x in /sys/module/ib_qib/parameters/*; do echo $(basename $x)
$(cat $x); done
```

Then in the list of qib parameters, you should see the following parameter being discussed:

```
. . .

rcvhdrcnt 0
```

The `0` means the driver automatically sets these parameters. Therefore, neither the user nor the `ipath_perf_tuning` script should modify these parameters.

3.12.1.1.3 Compute-only Node (Not part of a parallel file system cluster)

No tuning is required, other than what is in Section 3.12.1.1, “Systems in General (With Either Intel or AMD CPUs)” on page 36.

For more details on settings that are specific to either Intel or AMD CPUs, refer to the following sections for details on systems with those types of CPUs.

3.12.1.1.4 Storage Node (for example, Lustre/GPFS client or server node)

Although termed a “Storage Node” this information includes nodes that are primarily compute nodes, but also act as clients of a parallel file server.

Increasing the number of kernel receive queues allows more CPU cores to be involved in the processing of verbs traffic. This is important when using parallel file systems such as Lustre or IBM's GPFS (General Parallel File System). The module parameter that sets this number is `krcvqs`. Each additional kernel receive queue (beyond the one default queue for each port) takes user contexts away from PSM and from the support of MPI or compute traffic. The formula which illustrates this trade-off is:

```
PSM Contexts = 16 - (krcvqs-1)x num_ports
```

Where *`num_ports`* is the number of ports on the HCA
For example, on a single-port card with `krcvqs=4` set in `modprobe.conf`:

```
PSM Contexts = 16 - (4-1)x 1 = 16 - 3 = 13
```

If this were a 12-core node, then 13 is more than enough PSM contexts to run an MPI process on each core without making use of context-sharing. An example, `ib_qib options` line in the `modprobe.conf` file, for this 12-core node case is:

```
options ib_qib singleport=1 krcvqs=4
```

Table 3-2 can be used as a guide for setting the `krcvqs` parameter for the number of cores in the system supporting PSM processes and the number of ports in the HCA. Table 3-2 applies most readily to nodes with 1 HCA being used to support PSM (for example, MPI or SHMEM) processes. For nodes with multiple HCAs that are being used for PSM, the table decide the maximum number of cores that will be assigned on each HCA to support PSM (MPI or SHMEM) processes, then apply the table to each HCA in turn.

Table 3-2. `krcvqs` Parameter Settings

Cores per Node (to be used for MPI/PSM on 1 HCA):	1-port, Set krcvqs=	2 active ports in the HCA, Set krcvqs=
61-64	1†	1†
57-60	2	1†
53-56	3	2,1 (2 for Port 1, 1 for Port 2)
12-52	4	2
8-11	3	2,1 (2 for Port 1, 1 for Port 2)
4-7	2	1†
1-3	1†	1†

†. 1 is the default setting, so if the table recommends '1', `krcvqs` does not need to be set.

In the rare case that the node has more than 64 cores, and it is desired to run MPI on more than 64 cores, then two HCAs are required and settings can be made, using the rules in Table 3-2 on page 38, as though half the cores were assigned to each HCA.

3.12.1.1.5 Parallel Filesystem/Lustre Notes

The best performance for Lustre is with the following "modprobe.conf" statements:

```
options lnet networks="o2ib(ib0)"

options ko2iblnd map_on_demand=32
```

These parameter settings should be set in `/etc/modprobe.d/lustre.conf` for RHEL and SLES.

3.12.1.2 AMD CPU Systems

To improve performance on AMD CPU systems, Intel recommends setting **`pcie_caps=0x51 cache_bypass_copy=1`** as modprobe configuration parameters. For example, the module parameter line in the modprobe configuration file should include the following for AMD Opteron CPUs:

```
options ib_qib pcie_caps=0x51 cache_bypass_copy=1
```

On AMD systems, the pcie_caps=0x51 setting will typically result in a line of the 'lspci -vv' output associated with the HCA reading in the "DevCtl" section:

MaxPayload 128 bytes, MaxReadReq 4096 bytes.

On AMD Opteron 6300 Series servers with a PCIe bridge, the pcie_caps=0x51 setting will have no effect and the MaxReadReq value may still show as 512 bytes in the 'lspci -vv' output.

3.12.1.3 AMD Interlagos CPU Systems

With AMD Interlagos (Opteron 6200 Series) CPU systems, better performance will be obtained if, on single-HCA systems, the HCA is put in a PCIe slot closest to Socket number 1. You can typically find out which slots these are by looking at the schematics in the manual for your motherboard. (There is currently a BIOS or kernel problem which implies that no NUMA topology information is available from the kernel.)

To obtain top “Turbo boosts” of up to 1GHz in clock rate, when running on half the cores of a node, AMD recommends enabling the C6 C-state in the BIOS. Some applications (but certainly not all) run better when running on one-half of the cores or a Interlagos node (on every other core, one per Bulldozer module). Intel recommends enabling this C-state in the BIOS.

3.12.1.4 Intel CPU Systems

3.12.1.4.1 Typical tuning for recent Intel CPUs

For recent Intel CPUs (with core architecture code-named Ivy Bridge or Haswell) no special tuning is required for C-states or Intel Hyper-Threading technology, and cpuspeed can be enabled to allow Turbo mode to be in effect.

For older Intel CPUs (code-named Sandy Bridge, Westmere or Nehalem), set the following BIOS parameters (if available in your BIOS):

- Disable all C-States.
- Disable Intel Hyper-Threading technology

For setting all C-States to 0 where there is no BIOS support:

1. Add kernel boot option using the following command:

```
processor.max_cstate=0
```

2. Reboot the system.

If the node uses a single-port HCA, and is not part of a parallel file system cluster, there is no need for performance tuning changes to a modprobe configuration file. The driver will automatically set parameters appropriately for the node's Intel CPU, in a conservative manner.

For Intel® Xeon® systems with Intel® Xeon® 5500 Series (Nehalem) through Intel® Xeon® E5-2600 v2 (Ivy Bridge) CPUs, the following settings are default:

```
pcie_caps=0
```

On Intel systems with Intel® Xeon® 5500 through E5-2600 v2 Series or newer CPUs, the lspci output will typically read:

```
MaxPayload 256 bytes, MaxReadReq 4096 bytes
```

The default is:

```
pcie_caps=0
```

On Intel systems with Intel® Xeon® E5-2600 v3 Series (Haswell) CPUs, the lspci output will typically read:

```
MaxPayload 256 bytes, MaxReadReq 512 bytes
```

If you want to increase bandwidth slightly on Haswell-based systems, and to increase the MaxReadReq PCIe parameter, you may set pcie_caps=0x51 in the ib_qib modprobe parameter file. This will result in the lspci output will typically read:

```
MaxPayload 256 bytes, MaxReadReq 4096 bytes.
```

This tuning has some risk associated with it. If your system has problems following this setting, refer to the section "High Risk Tuning for Intel Harpertown CPUs" on page 40 for details on removing this tuning.

If you run a script, such as the following:

```
for x in /sys/module/ib_qib/parameters/*; do echo $(basename $x)
$(cat $x); done
```

Then in the list of qib parameters, you should see the following for the two parameters being discussed:

```
. . .

rcvhdrcnt 0

. . .

pcie_caps 0
```

The `0` means the driver automatically sets these parameters. Therefore, neither the user nor the `ipath_perf_tuning` script should modify these parameters.

3.12.1.4.2 Intel Nehalem or Westmere CPU Systems (DIMM Configuration)

Compute node memory bandwidth is important for high-performance computing (HPC) application performance and for storage node performance. On Intel CPUs code named Nehalem or Westmere (Intel® Xeon® 5500 series or 5600 series) it is important to have an equal number of dual in-line memory modules (DIMMs) on each of the three memory channels for each CPU. On the common dual CPU systems, you should use a multiple of six DIMMs for best performance.

3.12.1.5 High Risk Tuning for Intel Harpertown CPUs

For tuning the Harpertown generation of Intel® Xeon® CPUs that entails a higher risk factor, but includes a bandwidth benefit, the following can be applied:

For nodes with Intel Harpertown, Intel® Xeon® 54xx CPUs, you can add `pcie_caps=0x51` and `pcie_coalesce=1` to the `modprobe.conf` file. For example:

```
options ib_qib pcie_caps=0x51 pcie_coalesce=1
```

If the following problem is reported by syslog, a typical diagnostic can be performed, which is described in the following paragraphs:

```
[PCIe Poisoned TLP][Send DMA memory read]
```

Another potential issue is that after starting `openibd`, messages such as the following appear on the console:

```
Message from syslogd@st2019 at Nov 14 16:55:02 ...

kernel:Uhhuh. NMI received for unknown reason 3d on CPU 0
```

After this happens, you may also see the following message in the syslog:

```
Mth dd hh:mm:ss st2019 kernel: ib_qib 0000:0a:00.0: infinipath0:

Fatal Hardware Error, no longer usable, SN AIB1013A43727
```

These problems typically occur on the first run of an MPI program running over the PSM transport or immediately after the link becomes active. The adapter will be unusable after this situation until the system is rebooted. To resolve this issue try the following solutions in order:

- Remove `pcie_coalesce=1`
- Restart `openibd` and try the MPI program again
- Remove both `pcie_caps=0x51` and `pcie_coalesce=1` options from the `ib_qib` line in `modprobe.conf` file and reboot the system

Note: Removing both options will technically avoid the problem but can result in an unnecessary performance decrease. If the system has already failed with the above diagnostic it will need to be rebooted. Note that in modprobe.conf file all options for a particular kernel module must be on the same line and not on repeated options ib_qib lines.

3.12.1.6 Additional Driver Module Parameter Tunings Available

3.12.1.6.1 Setting driver module parameters on Per-unit or Per-port basis

The `ib_qib` driver allows the setting of different driver parameter values for the individual HCAs and ports. This allows the user to specify different values for each port on a HCA or different values for each HCA in the system. This feature is used when there is a need to tune one HCA or port for a particular type of traffic, and a different HCA or port for another type of traffic, for example, compute versus storage traffic.

Not all driver parameters support per-unit or per-port values. The driver parameters which can be used with the new syntax are listed below:

Per-unit parameters:

- `singleport` – Use only port 1; more per-port buffer space
- `cfgctxts` – Set max number of contexts to use
- `pcie_caps` – Max PCIe tuning: MaxPayload, MaxReadReq

Per-port parameters:

- `ibmtu` – Set max IB MTU
- `krcvqs` – number of kernel receive queues
- `num_vls` – Set number of Virtual Lanes to use

Specifying individual unit/port values is done by using a specific module parameter syntax:

```
param name=[default,][unit[:port]=value]
```

Where:

- *param name* is the driver module parameter name (listed above)
- *default* is the default value for that parameter. This value will be used for all remaining units/port which have not had individual values set. If no individual unit/port values have been specified, the default value will be used for all units/ports
- *unit* is the index of the HCA unit (as seen by the driver). This value is 0-based (index of first unit is '0').
- *port* is the port number on that HCA. This value is 1-based (number of first port is '1').
- *value* is the parameter value for the particular unit or port.

The fields in the square brackets are options; however, either a default or a per-unit/per-port value is required.

Example usage:

To set the default IB MTU to 1K for all ports on all units:

```
ibmtu=3
```

To set the IB MTU to 256-bytes for unit 0/port 1 and 4096-bytes for unit 0/port 2:

```
ibmtu=0:1=1,0:2=5
```

To set the default IB MTU to 2K for all ports but specify 4K for unit 0/port 1:

```
ibmtu=4,0:1=5
```

To set singleport to OFF as default and turn it ON for unit 1:

```
singleport=0,1=1
```

To set number of configured contexts to 10 on unit 0 and 16 on unit 1:

```
cfgctxts=0=10,1=16
```

A user can identify HCAs and correlate them to system unit numbers by using the `-b` option (beacon mode option) to the `ipath_control` script. Issuing the following command (as root):

```
ipath_control -u unit -b on
```

Where:

unit is the system unit number will cause that HCA to start blinking the LEDs on the face of the board in an alternating pattern.

Once the board has been identified, the user can return the LEDs to normal mode of operation with the following command (as root):

```
ipath_control -u unit -b off
```

3.12.1.6.2 numa_aware

The Non-Uniform Memory Access (NUMA) awareness (`numa_aware`) module parameter enables driver memory allocations in the same memory domain or NUMA node of the HCA. This improves the overall system efficiency with CPUs on the same NUMA node having faster access times and higher bandwidths to memory.

The default is:

```
option ib_qib numa_aware=10
```

This command lets the driver automatically decide on the allocation behavior and disables this feature on platforms with AMD and Intel Westmere-or-earlier CPUs, while enabling it on newer Intel CPUs.

Tunable options:

```
option ib_qib numa_aware=0
```

This command disables the NUMA awareness when allocating memory within the driver. The memory allocation requests will be satisfied on the node's CPU that executes the request.

```
option ib_qib numa_aware=1
```

This command enables this feature with the driver allocating memory on the NUMA node closest to the HCA.

3.12.1.6.3 recv_queue_size, Tuning Related to NAKs

The Receiver Not Ready Negative Acknowledgement (RNR NAKs) can slow IPoIB down significantly. InfiniBand* is fast enough to overrun IPoIB's receive queue before the post receives can occur.

The counter to look for on the sending side in this file is `RC RNR NAKs` as shown in the following example:

```
# cat /sys/class/infiniband/qib0/stats

Port 1:

RC timeouts 0

RC resends  0

RC QACKs    0

RC SEQ NAKs 0

RC RDMA seq 0

RC RNR NAKs 151 <---------

RC OTH NAKs 0

. . .

Ctx:npkts 0:170642806
```

Check the RC RNR NAKs before and after running the IPoIB test to see if that counter is increasing. If so, then increasing IPoIB's `recv_queue_size` to 512 in the `ib_ipoib.conf` file should eliminate RNR NAKs.

For example:

```
# cat /etc/modprobe.d/ib_ipoib.conf

alias ib0 ib_ipoib

alias ib1 ib_ipoib

options ib_ipoib recv_queue_size=512
```

3.12.2 Performance Tuning using `ipath_perf_tuning` Tool

The `ipath_perf_tuning` tool is intended to adjust parameters to the True Scale driver to optimize the InfiniBand* and application performance. The tool is designed to be run once per installation, however it can be re-run if changes to the configuration need to be made. Changes are made to the appropriate modprobe file depending on Linux distribution (see Section 3.12.2.3, “Affected Files” on page 46).

The tool takes into account the type of the node being configured and can be run in one of two modes - automatic (the default) and interactive. In automatic mode, the tool will make the parameter adjustments without the need for any user input. Interactive mode will prompt the user for input on some of the settings and actions.

Table 3-3 list the checks the tool performs on the system on which it is run.

Table 3-3. Checks Performed by `ipath_perf_tuning` Tool

Check Type	Description
pcie_caps	Adjust PCIe tuning for max payload and read request size. The result of this test depends on the CPU type of the node.
singleport	Determine whether to run the HCA in single port mode increasing the internal HCA resources for that port. This setting depends on the user’s input and is only performed in interactive mode.
krcvqs	Determine the number of kernel receive context to allocate. Normally, the driver allocates one context per physical port. However, more kernel receive contexts can be allocated to improve Verbs performance.
pcie_coalesce	Enable PCIe coalescing. PCIe coalescing is only needed or enabled on some systems with Intel Harpertown CPUs.
cache_bypass_copy	Enable the use of cache bypass copies. This option is enabled on AMD processors.
numa_aware	Enable NUMA-aware memory allocations.
cstates	Check whether (and which) C-States are enabled. C-States should be turned off for best performance.
services	Check whether certain system services (daemons) are enabled. These services should be turned off for best performance.

The values picked for the various checks and tests may depend on the type of node being configured. The tool is aware of two types of nodes<V_Variable>—compute and storage nodes.

3.12.2.0.1 Compute Nodes

Compute nodes are nodes which should be optimized for faster computation and communication with other compute nodes.

3.12.2.0.2 Storage (Client or Server) Nodes

Storage nodes are nodes which serve as clients or servers in a parallel filesystem network. Storage nodes (especially clients) are typically performing computation and using MPI, in addition to sending and receiving storage network traffic. The objective is to improve IB verbs communications while maintaining good MPI performance.

3.12.2.1 OPTIONS

Table 3-4 list the options for the ipath_perf_tuning tool and describes each option.

Table 3-4. `ipath_perf_tuning` Tool Options

Option	Description
-h	Display a short multi-line help message
-T test	This option is used to limit the list of tests/check which the tool performs to only those specified by the option. Multiple tests can be specified as a comma-separated list.
-I	Run the tool in interactive mode. In this mode, the tool will prompt the user for input on certain tests.

3.12.2.2 AUTOMATIC vs. INTERACTIVE MODE

The tool performs different functions when running in automatic mode compared to running in the interactive mode. The differences include the node type selection, test execution, and applying the results of the executed tests.

3.12.2.2.1 Node Type Selection

The tool is capable of configuring compute nodes or storage nodes (see Section 3.12.2.0.1, “Compute Nodes” on page 45 and Section 3.12.2.0.2, “Storage (Client or Server) Nodes” on page 45). When the tool is executed in interactive mode, it will query the user for the type of node. When the tool is running in automatic mode, it assumes that the node being configured is a compute node.

3.12.2.2.2 Test Execution

The main difference between the two test modes is that some of the tests are effectively skipped when the tool is in automatic mode. This is done, due to the fact, that these test do not provide a guaranteed universal performance gain and therefore, changing driver parameters associated with them requires user approval. Other tests, where the tool can make a safe determination, are performed in both modes without any user interaction. Table 3-5 list the test and describe the mode(s) for each.

Table 3-5. Test Execution Modes

Test	Mode
pcie_caps	Test is performed in both modes without any user interaction.
singleport	Test is only performed in interactive mode. The user is queried whether to enable singleport mode.
krcvqs	Test is performed in both modes without any user interaction.
pci_coalesce	Test is performed only in interactive mode. The user is queried whether to enable PCIe coalescing.

Table 3-5. Test Execution Modes (Continued)

Test	Mode
cache_bypass_copy	Test is performed in both modes without any user interaction.
num_aware	Test is performed in both modes without any user interaction.
cstates	Test is performed in both modes but the user is only notified of a potential issue if the tool is in interactive mode. In that case, the tool displays a warning and a suggestion on how to fix the issue.
services	Test is performed in both modes but the user is notified of running services only if the tool is in interactive mode. In that case, the user is queried whether to turn the services off.

3.12.2.2.3 Applying the Results

Automatic mode versus interactive mode also has an effect when the tool is committing the changes to the system. Along with the necessary driver parameters, the script also writes a comment line in the appropriate file which serves as a marker. This marker contains the version of the script which is making the changes. If the version recorded matches the version of the script currently being run, the changes are only committed if the tool is in interactive mode. The assumption is that the script is being re-run by the user to make adjustments.

3.12.2.3 Affected Files

The following lists the distribution and the file that is modified by the ipath_perf_tuning tool:

- For SLESor RHEL`<V_Variable>`— `/etc/modprobe.d/ib_qib.conf`

3.12.3 Homogeneous Nodes

To minimize management problems, the compute nodes of the cluster should have very similar hardware configurations and identical software installations. A mismatch between the True Scale software versions can also cause problems. Old and new libraries must not be run within the same job. It may also be useful to distinguish between the True Scale-specific drivers and those that are associated with kernel.org, OpenFabrics, or are distribution-built. The most useful tools are:

- `ident` (see “`ident`” on page 183)
- `ipathbug-helper` (see “`ipath_checkout`” on page 184)
- `ipath_checkout` (see “`ipath_checkout`” on page 184)
- `ipath_control` (see “`ipath_control`” on page 185)
- `mpirun` (see “`mpirun`” on page 188)
- `rpm` (see “`rpm`” on page 189)
- `strings` (see “`strings`” on page 189)

Note: Run these tools to gather information before reporting problems and requesting support.

3.12.4 Adapter and Other Settings

Note: For the most current information on performance tuning refer to the *Intel OFED+ Host Software Release Notes*.

The adapter and other settings can be adjusted for better performance using the information provided in the following paragraphs.

- **Use an IB MTU of 4096 bytes instead of 2048 bytes, if available, with the QLE7340, and QLE7342.** 4K MTU is enabled in the True Scale driver by default. To change this setting for the driver, see "Changing the MTU Size" on page 33.
- **Make sure that write combining is enabled.** The x86 Page Attribute Table (PAT) mechanism that allocates Write Combining (WC) mappings for the PIO buffers has been added and is now the default. If PAT is unavailable or PAT initialization fails for some reason, the code will generate a message in the log and fall back to the MTRR mechanism. See Appendix D Write Combining for more information.
- **Check the PCIe bus width.** If slots have a smaller electrical width than mechanical width, lower than expected performance may occur. Use this command to check PCIe Bus width:

```
$ ipath_control -iv
```

This command also shows the link speed.

- **Experiment with non-default CPU affinity while running single-process-per-node latency or bandwidth benchmarks.** Latency may be slightly lower when using different CPUs (cores) from the default. On some chipsets, bandwidth may be higher when run from a non-default CPU or core. For the MPI being used, look at its documentation to see how to force a benchmark to run with a different CPU affinity than the default. With OFED micro benchmarks such as from the qperf or perftest suites, taskset will work for setting CPU affinity.

3.12.5 Remove Unneeded Services

The cluster administrator can enhance application performance by minimizing the set of system services running on the compute nodes. Since these are presumed to be specialized computing appliances, they do not need many of the service daemons normally running on a general Linux computer.

Following are several groups constituting a minimal necessary set of services. These are all services controlled by `chkconfig`. To see the list of services that are enabled, use the command:

```
$ /sbin/chkconfig --list | grep -w on
```

Basic network services are:

- `network`
- `ntpd`
- `syslog`
- `xinetd`
- `sshd`

For system housekeeping, use:

- `anacron`
- `atd`
- `crond`

If you are using Network File System (NFS) or yellow pages (yp) passwords:

- `rpcidmapd`
- `ypbind`
- `portmap`

- `nfs`
- `nfslock`
- `autofs`

To watch for disk problems, use:

- `smartd`
- `readahead`

The service comprising the True Scale driver and SMA is:

- `openibd`

Other services may be required by your batch queuing system or user community.

If your system is running the daemon `irqbalance`, Intel recommends turning it off. Disabling `irqbalance` will enable more consistent performance with programs that use interrupts. Use this command:

```
#  /sbin/chkconfig irqbalance off
```

See Section C.6.2, “Erratic Performance” on page 151 for more information.

3.13 Host Environment Setup for MPI

After the Intel OFED+ Host software and the GNU* (GCC*) compilers have been installed on all the nodes, the host environment can be set up for running MPI programs.

3.13.1 Configuring for ssh

Running MPI programs with the command `mpirun` on a True Scale cluster depends, by default, on secure shell `ssh` to launch node programs on the nodes.

To use `ssh`, you must have generated Rivest, Shamir, Adleman (RSA) or Digital Signal Algorithm (DSA) keys, public and private. The public keys must be distributed and stored on all the compute nodes so that connections to the remote machines can be established without supplying a password.

You or your administrator must set up the `ssh` keys and associated files on the cluster. There are two methods for setting up ssh on your cluster. The first method, the `shosts.equiv` mechanism, is typically set up by the cluster administrator. The second method, using `ssh-agent`, is more easily accomplished by an individual user.

Note: `rsh` can be used instead of `ssh`. To use `rsh`, set the environment variable `MPI_SHELL=rsh`. See Section 4.2.14, “Environment Variables” on page 72 for information on setting environment variables.

Note: `rsh` has a limit on the number of concurrent connections it can have, typically 255, which may limit its use on larger clusters.

3.13.1.1 Configuring `ssh` and `sshd` Using `shosts.equiv`

This section describes how the cluster administrator can set up `ssh` and `sshd` through the `shosts.equiv` mechanism. This method is recommended, provided that your cluster is behind a firewall and accessible only to trusted users.

"Configuring for `ssh` Using `ssh-agent`" on page 50 shows how an individual user can accomplish the same thing using `ssh-agent`.

The example in this section assumes the following:

- Both the cluster nodes and the front end system are running the `openssh` package as distributed in current Linux systems.
- All cluster end users have accounts with the same account name on the front end and on each node, by using Network Information Service (NIS) or another means of distributing the password file.
- The front end used in this example is called `ip-fe`.
- Root or superuser access is required on `ip-fe` and on each node to configure `ssh`.
- `ssh`, including the host's key, has already been configured on the system `ip-fe`. See the `sshd` and `ssh-keygen` man pages for more information.

To use `shosts.equiv` to configure `ssg` and `sshd`:

1. On the system `ip-fe` (the front end node), change the `/etc/ssh/ssh_config` file to allow host-based authentication. Specifically, this file must contain the following four lines, all set to `yes`. If the lines are already there but commented out (with an initial #), remove the #.

```
RhostsAuthentication yes

RhostsRSAAuthentication yes

HostbasedAuthentication yes

EnableSSHKeysign yes
```

2. On each of the True Scale node systems, create or edit the file `/etc/ssh/shosts.equiv`, adding the name of the front end system. Add the line:

```
ip-fe
```

Change the file to mode 600 when you are finished editing.

3. On each of the True Scale node systems, create or edit the file `/etc/ssh/ssh_known_hosts`. You will need to copy the contents of the file `/etc/ssh/ssh_host_dsa_key.pub` from `ip-fe` to this file (as a single line), and then edit that line to insert `ip-fe ssh-dss` at the beginning of the line. This is very similar to the standard `known_hosts` file for `ssh`. An example line might look like this (displayed as multiple lines, but a single line in the file):

```
ip-fe ssh-dss
AAzAB3NzaC1kc3MAAACBAPoyES6+Akk+z3RfCkEHCkmYuYzqL2+1nwo4LeTVWpCD1
QsvrYRmpsfwpzYLXiSJdZSA8hfePWmMfrkvAAk4ueN8L3ZT4QfCTwqvHVvSctpibf
8n
aUmzloovBndOX9TIHyP/Ljfzzep4wL17+5hr1AHXldzrmgeEKp6ect1wxAAAAFQDR
56dAKFA4WgAiRmUJailtLFp8swAAAIBB1yrhF5P0jO+vpSnZrvrHa0Ok+Y9apeJp3
sessee30NlqKbJqWj5DOoRejr2VfTxZROf8LKuOY8tD6I59I0vlcQ812E5iw1GCZf
NefBmWbegWVKFwGlNbqBnZK7kDRLSOKQtuhYbGPcrVlSjuVpsfWEju64FTqKEetA8
l8QEgAAAIBNtPDDwdmXRvDyc0gvAm6lPOIsRLmgmdgKXTGOZUZ0zwxSL7GP1nEyFk
9wAxCrXv3xPKxQaezQKs+KL95FouJvJ4qrSxxHdd1NYNR0DavEBVQgCaspgWvWQ8c
L 0aUQmTbggLrtD9zETVU5PCgRlQL6I3Y5sCCHuO7/UvTH9nneCg==
```

Change the file to mode 600 when you are finished editing.

4. On each node, the system file `/etc/ssh/sshd_config` must be edited, so that the following four lines are uncommented (no # at the start of the line) and set to yes. (These lines are usually there, but are commented out and set to no by default.)

```
RhostsAuthentication yes

RhostsRSAAuthentication yes

HostbasedAuthentication yes

PAMAuthenticationViaKbdInt yes
```

5. After creating or editing the three files in Steps 2, 3, and 4, `sshd` must be restarted on each system. If you are already logged in via `ssh` (or any other user is logged in via `ssh`), their sessions or programs will be terminated, so restart only on idle nodes. Type the following (as root) to notify `sshd` to use the new configuration files:

```
# killall -HUP sshd
```

Note: This command terminates all `ssh` sessions into that system. Run from the console, or have a way to log into the console in case of any problem.

At this point, any end user should be able to login to the `ip-fe` front end system and use `ssh` to login to any True Scale node without being prompted for a password or pass phrase.

3.13.1.2 Configuring for `ssh` Using `ssh-agent`

The `ssh-agent`, a daemon that caches decrypted private keys, can be used to store the keys. Use `ssh-add` to add your private keys to `ssh-agent`'s cache. When `ssh` establishes a new connection, it communicates with `ssh-agent` to acquire these keys, rather than prompting you for a passphrase.

The process is described in the following steps:

1. Create a key pair. Use the default file name, and be sure to enter a passphrase.

```
$ ssh-keygen -t rsa
```

2. Enter a passphrase for your key pair when prompted. Note that the key agent does not survive X11 logout or system reboot:

```
$ ssh-add
```

3. The following command tells `ssh` that your key pair should let you in:

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Edit the `~/.ssh/config` file so that it reads like the following:

```
Host*

ForwardAgent   yes

ForwardX11   yes

CheckHostIP   no
```

```
StrictHostKeyChecking   no
```

This file forwards the key agent requests back to your desktop. When you log into a front end node, you can use `ssh` to compute nodes without passwords.

4. Follow your administrator's cluster policy for setting up `ssh-agent` on the machine where you will be running `ssh` commands. Alternatively, you can start the `ssh-agent` by adding the following line to your `~/.bash_profile` (or equivalent in another shell):

```
eval ‘ssh-agent‘
```

Use back quotes rather than single quotes. Programs started in your login shell can then locate the `ssh-agent` and query it for keys.

5. Finally, test by logging into the front end node, and from the front end node to a compute node, as follows:

```
$ ssh frontend_node_name

$ ssh compute_node_name
```

For more information, see the man pages for `ssh(1)`, `ssh-keygen(1)`, `ssh-add(1)`, and `ssh-agent(1)`.

3.13.2 Process Limitation with `ssh`

Process limitation with `ssh` is primarily an issue when using the `mpirun` option `-distributed=off`. The default setting is now `-distributed=on`; therefore, in most cases, `ssh` process limitations will not be encountered. This limitation for the `-distributed=off` case is described in the following paragraph.

MPI jobs that use more than 10 processes per node may encounter an `ssh` throttling mechanism that limits the amount of concurrent per-node connections to 10. If you need to use more processes, you or your system administrator must increase the value of `MaxStartups` in your `/etc/ssh/sshd_config` file.

3.14 Checking Cluster and Software Status

3.14.1 `ipath_control`

IB status, link speed, and PCIe bus width can be checked by running the program `ipath_control`. Sample usage and output are as follows:

```
$ ipath_control -iv

Intel OFED.VERSION yyyy_mm_dd.hh_mm_ss

0: Version: ChipABI VERSION, InfiniPath_QLE7340, InfiniPath1
VERSION, SW Compat 2

0: Serial: RIB0935M31511 LocalBus: PCIe,5000MHz,x8

0,1: Status: 0xe1 Initted Present IB_link_up IB_configured

0,1: LID=0x23 GUID=0011:7500:005a:6ad0

0,1: HRTBT:Auto LINK:40 Gb/sec (4X QDR)
```

3.14.2 iba_opp_query

`iba_opp_query` is used to check the operation of the Distributed SA. You can run it from any node where the Distributed SA is installed and running, to verify that the replica on that node is working correctly. See "`iba_opp_query`" on page 161 for detailed usage information.

```
# iba_opp_query --slid 0x31 --dlid 0x75 --sid 0x107

Query Parameters:

    resv1                   0x0000000000000107

    dgid                    ::

    sgid                    ::

    dlid                    0x75

    slid                    0x31

    hop                     0x0

    flow                    0x0

    tclass                  0x0

    num_path                0x0

    pkey                    0x0

    qos_class               0x0

    sl                      0x0

    mtu                     0x0

    rate                    0x0

    pkt_life                0x0

    preference              0x0

    resv2                   0x0

    resv3                   0x0

Using HCA qib0

Result:

    resv1                   0x0000000000000107

    dgid                    fe80::11:7500:79:e54a
    sgid                    fe80::11:7500:79:e416
    dlid                    0x75
    slid                    0x31
    hop                     0x0
    flow                    0x0
    tclass                  0x0
    num_path                0x0
    pkey                    0xffff
    qos_class               0x0
    sl                      0x1
    mtu                     0x4
    rate                    0x6
    pkt_life                0x10
    preference              0x0
    resv2                   0x0
    resv3                   0x0
```

3.14.3 ibstatus

Another useful program is `ibstatus` that reports on the status of the local HCAs. Sample usage and output are as follows:

```
$ ibstatus

Infiniband device 'qib0' port 1 status:

        default gid:     fe80:0000:0000:0000:0011:7500:005a:6ad0

        base lid:        0x23

        sm lid:          0x108

        state:           4: ACTIVE

        phys state:      5: LinkUp

        rate:            40 Gb/sec (4X QDR)

        link_layer:      IB
```

3.14.4 ibv_devinfo

`ibv_devinfo` queries RDMA devices. Use the `-v` option to see more information. Sample usage:

```
$ ibv_devinfo

hca_id: qib0

        fw_ver:                         0.0.0

        node_guid:                      0011:7500:00ff:89a6

        sys_image_guid:                 0011:7500:00ff:89a6

        vendor_id:                      0x1175

        vendor_part_id:                 29216

        hw_ver:                         0x2

        board_id:                       InfiniPath_QLE7280

        phys_port_cnt:                  1

                port:   1

                        state:                  PORT_ACTIVE (4)

                        max_mtu:                4096 (5)

                        active_mtu:             4096 (5)

                        sm_lid:                 1

                        port_lid:               31

                        port_lmc:               0x00
```

3.14.5 ipath_checkout

`ipath_checkout` is a `bash` script that verifies that the installation is correct and that all the nodes of the network are functioning and mutually connected by the True Scale fabric. It must be run on a front end node, and requires specification of a nodefile. For example:

```
$ ipath_checkout [options] nodefile
```

The `nodefile` lists the hostnames of the nodes of the cluster, one hostname per line. The format of `nodefile` is as follows:

```
hostname1

hostname2
```

. . .

For more information on these programs, see “`ipath_control`” on page 185, “`ibstatus`” on page 181, and “`ipath_checkout`” on page 184.

§ §

4.0 Running MPI on Intel HCAs

This section provides information on using the Message-Passing Interface (MPI) on Intel HCAs. Examples are provided for setting up the user environment, and for compiling and running MPI programs.

4.1 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.

4.1.1 MPIs Packaged with Intel OFED+

The high-performance open-source MPIs packaged with Intel OFED+ include: Open MPI version 1.8.1, Ohio State University MVAPICH version 1.2, and MVAPICH2 version 1.8.1. These MPIs are offered in versions built with the high-performance Performance Scaled Messaging (PSM) interface and versions built run over IB Verbs. There are also the commercial MPIs which are not packaged with Intel OFED+, Intel MPI and Platform MPI, which both make use of the PSM application programming interface (API) and can both run over IB Verbs or over user direct access programming library (uDAPL), which uses IB Verbs. For more information on other MPIs, see Section 5.0, “Using Other MPIs” on page 77.

4.2 Open MPI

Open MPI is an open source MPI-2 implementation from the Open MPI Project. Pre-compiled versions of Open MPI version 1.8.1 that run over PSM and are built with the GCC, PGI, and Intel compilers are available with the Intel download. Open MPI that runs over Verbs is also available.

Open MPI can be managed with the mpi-selector utility, as described in “Managing MVAPICH, and MVAPICH2 with the mpi-selector Utility” on page 80.

4.2.1 Installation

Follow the instructions in the *Intel® True Scale Fabric Software Installation Guide* for installing Open MPI.

Newer versions of Open MPI released after this Intel OFED+ release will not be supported (refer to the *OFED+ Host Software Release Notes* for version numbers). Intel does not recommend installing any newer versions of Open MPI. If a newer version is required it can be found on the Open MPI web site (http://www.open-mpi.org/) and installed after Intel OFED+ has been installed.

4.2.2 Setup

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

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

```
$mpi_home/bin
```

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

4.2.3 Compiling Open MPI Applications

Intel recommends that you use the included wrapper scripts that invoke the underlying compiler (see Table 4-1).

Table 4-1. Open MPI Wrapper Scripts

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

To compile your program in C, type the following:

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

These scripts all provide the command line options listed in Table 4-2.

Table 4-2. Command Line Options for Scripts

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

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

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

4.2.4 Create the `mpihosts` 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 `mpihosts`. The contents can be in the following format:

More details on the `mpihosts` file can be found in "`mpihosts` File Details" on page 68.

4.2.5 Running Open MPI Applications

The Open MPI choices available from `mpi-selector --list` are:

- openmpi_gcc-1.8.1
- openmpi_gcc_qlc-1.8.1
- openmpi_intel_qlc-1.8.1
- openmpi_pgi_qlc-1.8.1.

The first choice will use verbs by default, and any with the `_qlc` string will use PSM by default. If you chose `openmpi_gcc_qlc`-1.8.1, for example, then the following simple mpirun command would run using PSM:

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

To run over IB Verbs instead of the default PSM transport in `openmpi_gcc_qlc-1.8.1`, use this `mpirun` command line:

```
$ mpirun -np 4 -machinefile mpihosts --mca btl sm --mca btl
openib,self --mca mtl ^psm 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 PSM transport:

```
--mca mtl ^psm
```

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

PSM 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 mpihosts --mca btl sm --mca btl
tcp,self --mca btl_tcp_if_exclude eth0 --mca btl_tcp_if_include
ib0 --mca mtl ^psm mpi_app_name
```

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

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

4.2.6 Further Information on Open MPI

For more information about Open MPI, see:

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

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

4.2.7 Configuring MPI Programs for Open MPI

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

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

```
$ export CC=mpicc

$ export CXX=mpicxx

$ export F77=mpif77

$ export F90=mpif90
```

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

```
$ setenv CC mpicc
```

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

```
$ ./configure -cc=mpicc -fc=mpif77 -c++=mpicxx -c++linker=mpicxx
```

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

```
CC=mpicc

F77=mpif77

F90=mpif90

CXX=mpicxx
```

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

```
LD=mpif90
```

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

4.2.8 To Use Another Compiler

Open MPI and all other MPIs that run on True Scale, support a number of compilers, in addition to the default GNU Compiler Collection (GCC, including gcc, g++ and gfortran) versions 3.3 and later. These include the PGI 8.0, through 11.9; and Intel 9.*x*, 10.1, 11.*x*, and 12.*x*.

The easiest way to use other compilers with any MPI that comes with Intel OFED+ is to use `mpi-selector` to change the selected MPI/compiler combination, see "Managing MVAPICH, and MVAPICH2 with the mpi-selector Utility" on page 80.

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

Table 4-3 and Table 4-4 show the options for each of the compilers.

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

Table 4-3. Intel Compilers

Compiler	Command
C	**`$ mpicc -cc=icc .....`**
C++	`$ mpicc -CC=icpc`
Fortran 77	`$ mpif77 -fc=ifort .....`

Fortran 90/95	`$ mpif90 -f90=ifort .....`
	`$ mpif95 -f95=ifort .....`

Table 4-4. Portland Group (PGI) Compilers

Compiler	Command
C	`mpicc -cc=pgcc .....`
C++	`mpicc -CC=pgCC`
Fortran 77	`mpif77 -fc=pgf77 .....`

Fortran 90/95	`mpif90 -f90=pgf90 .....`
	`mpif95 -f95=pgf95 .....`

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

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

For Fortran 90 programs:

```
$ mpif90 -f90=pgf90 -show pi3f90.f90 -o pi3f90

pgf90 -I/usr/include/mpich/pgi5/x86_64 -c -I/usr/include

pi3f90.f90 -c
```

```
pgf90 pi3f90.o -o pi3f90 -lmpichf90 -lmpich -lmpichabiglue_pgi5
```

Fortran 95 programs will be similar to the above.

For C programs:

```
$ mpicc -cc=pgcc -show cpi.c

pgcc -c cpi.c

pgcc cpi.o -lmpich -lpgftnrtl -lmpichabiglue_pgi5
```

4.2.8.1 Compiler and Linker Variables

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

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

4.2.9 Process Allocation

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

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

For the QLE7342 and QLE7340 adapters, the maximum number of contexts available is:

- 16 user hardware contexts available per HCA
- 64 MPI ranks (processes or node programs) that can be run per HCA when the Software Context Sharing is Enabled (default mode)

The default hardware context/CPU mappings can be changed on the True Scale HCAs. See "True Scale Hardware Contexts on the HCAs" on page 63 for more details.

Context sharing is enabled by default. How the system behaves when context sharing is enabled or disabled is described in "Enabling and Disabling Software Context Sharing" on page 66.

Achieving optimal performance by ensuring that the PSM process affinity is assigned to the CPU of the Non-Uniform Memory Access (NUMA) node local to the HCA that it is operating on as described in "Optimal Assignment of PSM Processes to HCAs" on page 63.

When running a job in a batch system environment where multiple jobs may be running simultaneously, it is useful to restrict the number of True Scale contexts that are made available on each node of an MPI. See "Restricting True Scale Hardware Contexts in a Batch Environment" on page 66.

Errors that may occur with context sharing are covered in "Context Sharing Error Messages" on page 67.

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

4.2.9.1 True Scale Hardware Contexts on the HCAs

On the QLE7340 and QLE7342 HCAs, the receive resources are statically partitioned across the True Scale contexts according to the number of True Scale contexts enabled. The following defaults are automatically set according to the number of online CPUs in the node:

For four or less CPUs: 6 (4 + 2)

For five to eight CPUs: 10 (8 + 2)

For nine or more CPUs: 18 (16 + 2)

The one additional context on HCAs are to support the kernel on each port.

Performance can be improved in some cases by disabling True Scale hardware contexts when they are not required so that the resources can be partitioned more effectively.

To disable this behavior, explicitly configure for the number you want to use with the `cfgctxts` module parameter in the modprobe configuration file (see "Affected Files" on page 46 for exact file name and location).

The maximum that can be set is 18 on HCAs.

The driver must be restarted if this default is changed. See "Managing the True Scale Driver" on page 33.

Note: In rare cases, setting contexts automatically on HCAs can lead to sub-optimal performance where one or more True Scale hardware contexts have been disabled and a job is run that requires software context sharing. Since the algorithm ensures that there is at least one True Scale context per online CPU, this case occurs only if the CPUs are over-subscribed with processes (which is not normally recommended). In this case, it is best to override the default to use as many True Scale contexts as are available, which minimizes the amount of software context sharing required.

4.2.9.2 Optimal Assignment of PSM Processes to HCAs

The optimal assignment of PSM processes to HCAs enhancement automatically assigns processes to NUMA nodes and HCAs to automatically improve MPI/PSM performance to the greatest extent in systems where the OS and CPUs support NUMA (Non-Uniform Memory Architecture) and NUMA node to I/O device binding, and where two HCAs connect to different PCIe root complexes which, in turn, connect to different NUMA nodes.

For the best performance, PSM processes running on an NUMA node should use an HCA that is closest to that NUMA node. PSM processes assigned to an HCA should use a NUMA node that is closest to that HCA. In a dual rail environment, it may be non-trivial to determine what the optimal assignments should be, but the PSM code will automatically assign the optimal NUMA node and HCA for a given PSM process.

4.2.9.2.1 Background and Definitions

Non-Uniform Memory Access (NUMA) causes unequal latency relative to the distance of the memory from a CPU. This is due to the fact that some regions of memory are on physically different busses from other regions. NUMA also introduces the concept of local and remote memory.

A NUMA node is a block of memory with the CPU cores, caches, and so on physically on the same bus as the memory. With most Intel Xeon CPUs since the Nehalem generation, a single CPU chip (which might have four, six or eight cores) together with the memory and bus attached to it, constitute a NUMA node. The same holds true with systems built with AMD's, original Opteron CPUs. The recent AMD Opteron 6100 Series (known as the Magny Cours) and 6200 Series (known as Interlagos) CPUs, are really 2 CPU chips on each multi-chip package that plugs into a socket. Each of these CPU chips has six or eight cores and its own memory bus to connect to its local memory block. Therefore, a two-socket system with these 6100/6200 Series CPUs consists of four NUMA nodes. NUMA nodes are connected using high speed system interconnect links (known as QPI or HT links by Intel and AMD, respectively).

One NUMA node connects to the HCA by a PCIe root-complex and a PCIe bus. To make a process running on a CPU core perform the best, the objective is to place data that is needed frequently in the memory local to that core (on the same NUMA node as that core) and to use an HCA which is closest to that CPU core in terms of system interconnect hops (QPI or HT links).

MPI ranks are processes which communicate through the PSM (for best performance on Intel True Scale) library to get access to the HCA. We refer to these MPI ranks as PSM processes.

4.2.9.2.2 Overview

For each PSM process needing access to the HCA, the PSM library requests that the qib driver allocate an HCA and a CPU core for it to use for computation and communications. This enhancement optimizes performance by automatically allocating the PSM process to a CPU core and to an HCA that are close to each other, and by allocating the driver's send buffer registers and user contexts to the NUMA node that includes that CPU core.

By "automatic," we mean without the need for any configuration by the user. Prior to this automatic process, the user could select an HCA by the means of a user-level environment variable (IPATH_UNIT), along with a user-level command utility (taskset), to bind the PSM process to a given CPU and subsequently its memories on the respective NUMA node. This configuration was complicated since it required the user to have detailed knowledge of the system architecture.

Most MPIs set affinity by default, and PSM will honor the MPI's affinity settings. If you want PSM to assign processes to cores, turn off the MPI's affinity placement (for details refer to Section 4.2.9.2.3)

Note: When using this optimal assignment of PSM processes to HCAs with mvapich-1.2.0-qlc MPI, the variable **VIADEV_USE_AFFINITY** must be set to 0 in order to ensure that the optimal CPU affinity and HCA are chosen. This can be done by specifying **VIADEV_USE_AFFINITY=0** on the **mpirun_rsh** command line or in the `ofed.mvapich.params` file.

4.2.9.2.3 Configuration

The optimal assignment of PSM processes to HCAs is enabled by default.

In the single rail case this optimization ensures that the PSM processes are running on the CPU cores of the NUMA node local to the HCA when possible, or on the closest available NUMA node to the HCA.

In the dual-HCA per node case, this optimization ensures that, whenever possible, the PSM process affinity is assigned to the CPU of the NUMA node local to the HCA that it is operating on. Besides the dual-HCA case, this optimization also helps with one HCA or more than two HCAs.

MPIs offer a wide range of process affinity policies, and there are good reasons for using them. However, if you believe that placement to optimize communications over the True Scale HCAs is paramount, the affinity settings of the MPI should be disabled so that PSM can take over this responsibility. The settings that accomplish this with our supported MPIs are:

- Intel MPI: set mpirun option: -binding pin=0
- Open MPI: affinity is off by default
- MVAPICH2: set environment variable MV2_ENABLE_AFFINITY=0
- MVAPICH: set environment variable VIADEV_USE_AFFINITY=0
- IBM/Platform MPI: use mpirun option: -aff=none

Although the first optimization in the following list is the default and typically would serve most users' needs, three possible optimizations are described for a dual-socket, dual HCA,16-core Sandy Bridge (Xeon E5-2600 Series CPU) platform:

- IPATH_HCA_SELECTION_ALG="Round Robin" -- Default for PSM (no need to set this variable)
    - When running up to 16 processes, each process will have a dedicated CPU core. Processes will be assigned to each HCA alternately. Therefore, if running 16 processes, 8 will be assigned to each HCA.
    - When running more than 16 processes, the 17th process onward will share a CPU, with up to 32 total processes. Context-sharing will not be needed.

  This configuration typically has very good performance, gets benefits from the multiple HCAs with any process count of 2 or greater, requires no special configuration, and is the default configuration.
- IPATH_HCA_SELECTION_ALG=Packed
    - cfgctxts=10 (implies 8 user contexts per HCA)
    - When running up to 16 processes, each will have a CPU core dedicated per process. Eight processes will be assigned to each HCA.
    - When running with more than 16 processes, the 17th through 32nd processes share a CPU allocated by the OS scheduler (and share an HCA context).

  This configuration requires cfgctxts=10 to be set in the modprobe configuration file for ib_qib. This configuration has a possible advantage for applications that perform better if adjacent MPI ranks are running (mostly) on adjacent CPU cores, to share cache and ring resources for faster inter-process communications.
- IPATH_HCA_SELECTION_ALG=Packed
    - cfgctxts=18 (the default, implies 16 user contexts per HCA)
    - Up to 16 processes will have a CPU dedicated per process with contexts from the first HCA only.
    - When running more than 16 processes, up to 32 processes will have a CPU dedicated per process. The first 16 processes will be assigned contexts from the first HCA. The remaining 16 processes will be assigned contexts from the second HCA but will share a CPU with one of the processes and are handled by the system scheduler.

4.2.9.2.4 Benefits

Using default settings, the user should observe lower latencies and higher message rates. The default latency results should reflect what formerly was observed when a process is pinned to a CPU on the NUMA node local to the HCA. There should be benefits for 1, 2, or more HCAs on the system, since the contexts are now allocated on the NUMA node closest (fewest hops through chips) to the HCA.

4.2.9.3 Enabling and Disabling Software Context Sharing

By default, context sharing is enabled; it can also be specifically disabled.

Context Sharing Enabled: The MPI library provides PSM the local process layout so that True Scale contexts available on each node can be shared if necessary; for example, when running more node programs than contexts. All PSM jobs assume that they can make use of all available True Scale contexts to satisfy the job requirement and try to give a context to each process.

When context sharing is enabled on a system with multiple Intel HCAs and the `IPATH_UNIT` environment variable is set, the number of True Scale contexts made available to MPI jobs is restricted to the number of contexts available on that HCA. When multiple True Scale devices are present, it restricts the use to a specific HCA. By default, all configured HCAs are used in round robin order.

Context Sharing Disabled: Each node program tries to obtain exclusive access to an True Scale hardware context. If no hardware contexts are available, the job aborts.

To explicitly disable context sharing, set this environment variable in one of the two following ways:

```
PSM_SHAREDCONTEXTS=0
```

```
PSM_SHAREDCONTEXTS=NO
```

The default value of `PSM_SHAREDCONTEXTS` is `1` (enabled).

4.2.9.4 Restricting True Scale Hardware Contexts in a Batch Environment

If required for resource sharing between multiple jobs in batch systems, you can restrict the number of True Scale hardware contexts that are made available on each node of an MPI job by setting that number in the `PSM_SHAREDCONTEXTS_MAX` or `PSM_RANKS_PER_CONTEXT` environment variables.

For example, if you are running two different jobs on nodes using Intel® HCAs, set `PSM_SHAREDCONTEXTS_MAX` to 8 instead of the default 16. Each job would then have at most 8 of the 16 available hardware contexts. Both of the jobs that want to share a node would have to set `PSM_SHAREDCONTEXTS_MAX=8`.

Note: MPIs use different methods for propagating environment variables to the nodes used for the job; See Section 7.0, "Virtual Fabric support in PSM" on page 109 for examples. Open MPI will automatically propagate PSM environment variables.

Setting `PSM_SHAREDCONTEXTS_MAX=8` as a clusterwide default would unnecessarily penalize nodes that are dedicated to running single jobs. Intel recommends that a per-node setting, or some level of coordination with the job scheduler with setting the environment variable should be used.

The number of contexts can be explicitly configured with the `cfgctxts` module parameter. This will override the default settings based on the number of CPUs present on each node. See "True Scale Hardware Contexts on the HCAs" on page 63.

`PSM_RANKS_PER_CONTEXT` provides an alternate way of specifying how PSM should use contexts. The variable is the number of ranks that will share each hardware context. The supported values are 1, 2, 3 and 4, where 1 is no context sharing, 2 is 2-way context sharing, 3 is 3-way context sharing and 4 is the maximum 4-way context sharing. The same value of `PSM_RANKS_PER_CONTEXT` must be used for all ranks on a node, and typically, you would use the same value for all nodes in that job. Either `PSM_RANKS_PER_CONTEXT` or `PSM_SHAREDCONTEXTS_MAX` would be used in a particular job, but not both. If both are used and the settings are incompatible, then PSM will report an error and the job will fail to start up.

4.2.9.5 Context Sharing Error Messages

The error message when the context limit is exceeded is:

```
No free InfiniPath contexts available on /dev/ipath
```

This message appears when the application starts.

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

```
PSM found 0 available contexts on InfiniPath device
```

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

4.2.9.6 Running in Shared Memory Mode

Open MPI supports running exclusively in shared memory mode; no Intel HCA 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-1.8.1-qlc/tests/osu_benchmarks-3.1.1` to your PATH (or if you installed the MPI in another location: add *$MPI_HOME*`/tests/osu_benchmarks-3.1.1` to your PATH).

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

```
$ cat /tmp/onehost

idev-64 slots=8
```

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

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

4.2.10 `mpihosts` File Details

As noted in "Create the `mpihosts` File" on page 59, a `hostfile` (also called *machines file, nodefile*, or *hostsfile*) has been created in your current working directory. This file names the nodes that the node programs may run.

The two supported formats for the `hostfile` are:

```
hostname1

hostname2

...
```

or

```
hostname1 slots=process_count

hostname2 slots=process_count

...
```

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

Also in the first format, if the `-np` count is less than the number of lines in the machine file, `mpirun` still processes the entire file and tries to pack processes to use as few hosts as possible in the hostfile. This is a different behavior than MVAPICH or the no-longer-supported Intel MPI.

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

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

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

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

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

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

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

```
host-01, or

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

- In the absence of the `-hostfile` option, the `-H` option, `mpirun` uses the file `./mpihosts`, 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 `mpihosts` file. More details about how to schedule processes to nodes with Open MPI refer to the Open MPI website:
  http://www.open-mpi.org/faq/?category=running#mpirun-scheduling

4.2.11 Using Open MPI's `mpirun`

The script `mpirun` is a front end program that starts a parallel MPI job on a set of nodes in an True Scale 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 True Scale cluster machines to be used in a job.

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

```
The general syntax is:

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

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

```
./program-name
```

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

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

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

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 program.

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

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

This option overrides the `slots=`*process_count* specifications, if any, in the lines of the `mpihosts` file. As a general rule, `mpirun` distributes the *n* node programs 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 `mpihosts` file.

```
Typically, the number of node programs 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 program is started as a process on one node. While a node program 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 programs 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`.

4.2.12 Console I/O in Open MPI Programs

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

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

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

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

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

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

4.2.13 Environment for Node Programs

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

4.2.13.1 Remote Execution

Open MPI requires that the PATH environment variable be set to find executables on remote nodes (this is typically only necessary in rsh- or ssh-based environments -- batch/scheduled environments typically copy the current environment to the execution of remote jobs, so if the current environment has PATH and/or LD_LIBRARY_PATH set properly, the remote nodes will also have it set properly). If Open MPI was compiled with shared library support, it may also be necessary to have the LD_LIBRARY_PATH environment variable set on remote nodes as well (especially to find the shared libraries required to run user MPI applications).

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

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

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

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

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

If the following command line is used:

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

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

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

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

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

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

is equivalent to

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

4.2.13.2 Exported Environment Variables

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

4.2.13.3 Setting MCA Parameters

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

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

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

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

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

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

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

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

```
OMPI_MCA_key=value
```

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

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

4.2.14 Environment Variables

Table 4-5 contains a summary of the environment variables that are relevant to any PSM including Open MPI. Table 4-6 is more relevant for the MPI programmer or script writer, because these variables are only active after the mpirun command has been issued and while the MPI processes are active. Open MPI provides the environmental variables shown in Table 4-6 that will be defined on every MPI process. Open MPI guarantees that these variables will remain stable throughout future releases.

Table 4-5. Environment Variables Relevant for any PSM

Name	Description
PSM_TID	Setting to 0 will turn off TID receive. This is the counterpart of SDMA for the receive side. Turning it off will reduce performance, but, again, it can be useful for diagnosing problems. Default: 1
PSM_TID_SENDSESSIONS_MAX	Max tid transfer sessions a process can do in parallel, since a process can do tid transfer with many other processes, this could be more than the tidflows a process can have. Default value depends on the memory mode, 256-4096 (min-max) in normal memory mode; 512-8192 in large memory mode. 1 for mini memory mode.
PSM_SHAREDCONTEXTS	If set, turn on PSM context sharing. The default is 1 (ON). Maximum of 4 processes are able to share a context. Default: 1
PSM_SHAREDCONTEXTS_MAX	Use to set the max-way to share contexts. PSM supports a maximum of 4-way context sharing. PSM_RANKS_PER_CONTEXT is usually a simpler way to control context sharing behavior than this variable. Either variable can be used to more easily allow multiple jobs to share cores on one node. Default: 16
PSM_DEVICES	The order of these items are important for determining which devices PSM used for connections between pairs of processes. [For MPSS 3.2 and earlier, setting to self,ipath,shm was needed when Intel® Xeon Phi™ cards needed to be paired with an HCA and communicate over the fabric, rather than using on-node paths. That is no longer necessary with MPSS 3.4 and after with PSM's symmetric mode support was added for Intel® Xeon Phi™ and True Scale.] Default: self,shm,ipath
PSM_MULTIRAIL	Set =1 in a multi-HCA-per-node environment to turn on striping of large messages across multiple HCAs. Also this can be used to connect a node to multiple fabrics, aka multi-plane fabrics. Default: 0
PSM_MULTIRAIL_MAP	This environment variable tells PSM which unit/port pair is used to set up a "rail". Multiple specifications are separated by a comma. If only one rail is specified, it is eqivalent to the single-rail case: the unit/port specified will be used instead of the unit/port assigned by the qib driver. Default: 0:1,1:1
PSM_MQ_RNDV_IPATH_THRESH	This is the # of bytes above which, PSM uses the Rendezvous protocol, i.e. SDMA and TID receive apply. It is also the point at which, when PSM_MULTIRAIL is set that larger messages are striped over two HCAs. Default: 64000
PSM_MQ_SENDREQS_MAX	Max num of isend requests in-flight. Default: 1048576
PSM_MQ_RECVREQS_MAX	Max num of irec requests in-flight. Default: 1048576
PSM_RANKS_PER_CONTEXT	Sets the number of ranks that will share each hardware context. The same value of PSM_RANKS_PER_CONTEXT must be used for all ranks on a node, and typically, all nodes in the job would use the same setting. PSM_RANKS_PER_CONTEXT and PSM_SHAREDCONTEXTS_MAX are incompatible. PSM will report an error and the job will fail to start up if both variables are set. Default: 1

Table 4-5. Environment Variables Relevant for any PSM (Continued)

Name	Description
PSM_SDMA	Setting to 0 will turn off Send DMA. This uses PIO send only and will limit peak unidirectional bandwidth to 2-2.5 GB/s, but can be useful for diagnosing problems. Default: 1
PSM_IDENTIFY	Will print at MPI_init/PSM init time, which PSM library was used in (typically) the mpirun command just used. It is also useful to verify that PSM was used, as opposed to verbs. Default: 0
IPATH_NO_CPUAFFINITY	When set to 1, the PSM library will skip trying to set processor affinity. This is also skipped if the processor affinity mask is set to a list smaller than the number of processors prior to MPI_Init() being called. Otherwise the initialization code sets cpu affinity in a way that optimizes cpu and memory locality and load. Default: Unset
IPATH_PORT	Specifies the port to use for the job, 1 or 2. Specifying 0 will autoselect IPATH_PORT. Default: 0
IPATH_UNIT	Selects which HCA to use. Setting =0 will cause the PSM process to use the qib0 HCA (aka unit). Setting =1 will cause the process to use qib1. When multiple True Scale devices are present and this variable is unset, then this process can be assigned to either HCA, or use both if PSM_MULTIRAIL is enabled. Default: Unset
IPATH_HCA_SELECTION_ALG	This variable provides user-level support to specify HCA/port selection algorithm through the environment variable. The default option is a "Round Robin" that allocates MPI processes to the HCAs in an alternating or round robin fashion. The alternate value is "Packed" will assign MPI processes to the first HCA until all contexts for that HCA are used up (max of 16), then MPI processes will be assigned to the 2nd HCA. Default: Round Robin

Table 4-6. Environment Variables Relevant for Open MPI

Name	Description
OMPI_COMM_WORLD_SIZE	This environment variable selects the number of processes in this process' MPI Comm_World
OMPI_COMM_WORLD_RANK	This variable is used to select the MPI rank of this process
OMPI_COMM_WORLD_LOCAL_RANK	This environment variable selects the relative rank of this process on this node within it job. For example, if four processes in a job share a node, they will each be given a local rank ranging from 0 to 3.
OMPI_UNIVERSE_SIZE	This environment variable selects the number of process slots allocated to this job. Note that this may be different than the number of processes in the job.

4.2.15 Job Blocking in Case of Temporary Link Failures

By default, as controlled by `mpirun’s` quiescence parameter `-q`, an MPI job is killed for quiescence in the event of an link failure (or unplugged cable). This quiescence timeout occurs under one of the following conditions:

- A remote rank’s process cannot reply to out-of-band process checks.
- MPI is inactive on the link for more than 15 minutes.

To keep remote process checks but disable triggering quiescence for temporary link failures, use the `-disable-mpi-progress-check` option with a nonzero `-q` option. To disable quiescence triggering altogether, use `-q 0`. No matter how these options are used, link failures (temporary or other) are always logged to `syslog`.

If the link is down when the job starts and you want the job to continue blocking until the link comes up, use the `-t -1` option.

4.3 Open MPI and Hybrid MPI/OpenMP Applications

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

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

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

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

See "Environment for Node Programs" on page 70 for information on setting environment variables.

Note: With Open MPI, and other PSM-enabled MPIs, you will typically want to turn off PSM'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 `IPATH_NO_CPUAFFINITY=1` setting as follows:

```
OMP_NUM_THREADS=8  (typically set in the ~/.bashrc file)
```

```
mprun -np 2 -H host1,host2 -x IPATH_NO_CPUAFFINITY=1 ./hybrid_app
```

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

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

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

4.4 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.

4.4.1 MPI Errors

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

See the standard MPI documentation referenced in Appendix F, "Recommended Reading" for details on the MPI error codes.

4.4.2 Using Debuggers

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

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

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

§ §

5.0 Using Other MPIs

This section provides information on using other MPI implementations. Detailed information on using Open MPI is provided in Section 4.0, "Running MPI on Intel HCAs" on page 57, and will be covered in this Section in the context of choosing among multiple MPIs or in tables which compare the multiple MPIs available.

5.1 Introduction

Support for multiple high-performance MPI implementations has been added. Most implementations run over both PSM and OpenFabrics Verbs (see Table 5-1). To choose which MPI to use, use the `mpi-selector-menu` command, as described in "Managing MVAPICH, and MVAPICH2 with the `mpi-selector` Utility" on page 80.

Table 5-1. Other Supported MPI Implementations

MPI Implementation	Runs Over	Compiled With	Comments
Open MPI 1.8.1	PSM Verbs	GCC, Intel, PGI	Provides some MPI-2 functionality (one-sided operations and dynamic processes). Available as part of the Intel download. Can be managed by `mpi-selector`.
MVAPICH version 1.2	PSM Verbs	GCC, Intel, PGI	Provides MPI-1 functionality. Available as part of the Intel download. Can be managed by `mpi-selector`.
MVAPICH2 version 1.8.1	PSM Verbs	GCC, Intel, PGI	Provides MPI-2 Functionality. Can be managed by `MPI-Selector`.
Platform MPI 8	PSM Verbs	GCC (default)	Provides some MPI-2 functionality (one-sided operations). Available for purchase from Platform Computing (an IBM Company).
Intel MPI version 4.0	TMI/PSM, uDAPL	GCC (default)	Provides MPI-1 and MPI-2 functionality. Available for purchase from Intel.

† MVAPICH and Open MPI have been have been compiled for PSM to support the following versions of the compilers:
(GNU) gcc 4.1.0
(PGI) pgcc 9.0
(Intel) icc 11.1

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

5.2 Installed Layout

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

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

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

For example:

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

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

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

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

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

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

5.3 Open MPI

Open MPI is an open source MPI-2 implementation from the Open MPI Project. Pre-compiled versions of Open MPI version 1.8.1 that run over PSM and are built with the GCC, PGI, and Intel compilers are available with the Intel download.

Details on Open MPI operation are provided in Section 4.0, "Running MPI on Intel HCAs" on page 57.

5.4 MVAPICH

Pre-compiled versions of MVAPICH 1.2 built with the GNU, PGI, and Intel compilers, and that run over PSM, are available with the Intel download.

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

MVAPICH can be managed with the `mpi-selector` utility, as described in "Managing MVAPICH, and MVAPICH2 with the `mpi-selector` Utility" on page 80.

5.4.1 Compiling MVAPICH Applications

As with Open MPI, Intel recommends that you use the included wrapper scripts that invoke the underlying compiler (see Table 5-2).

Table 5-2. MVAPICH Wrapper Scripts

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

To compile your program in C, type:

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

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

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

5.4.2 Running MVAPICH Applications

By default, the MVAPICH shipped with the Intel OFED+ and IFS (IFS), runs over PSM once it is installed.

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

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

Password-less `ssh` is used unless the `-rsh` option is added to the command line above.

5.4.3 Further Information on MVAPICH

For more information about MVAPICH, see:
http://mvapich.cse.ohio-state.edu/

5.5 MVAPICH2

Pre-compiled versions of MVAPICH2 1.8.1 built with the GNU, PGI, and Intel compilers, and that run over PSM, are available with the Intel download.

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

MVAPICH2 can be managed with the `mpi-selector` utility, as described in "Managing MVAPICH, and MVAPICH2 with the `mpi-selector` Utility" on page 80.

5.5.1 Compiling MVAPICH2 Applications

As with Open MPI, Intel recommends that you use the included wrapper scripts that invoke the underlying compiler (see Table 5-3).

Table 5-3. MVAPICH Wrapper Scripts

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

To compile your program in C, type:

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

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

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

5.5.2 Running MVAPICH2 Applications

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

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

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

5.5.3 Further Information on MVAPICH2

For more information about MVAPICH2, see:
http://mvapich.cse.ohio-state.edu/support/mvapich-1.8.1-quick-start.html

or for more detail:
http://mvapich.cse.ohio-state.edu/support/mvapich-1.8.1rc2_user_guide.pdf

5.6 Managing MVAPICH, and MVAPICH2 with the `mpi-selector` Utility

When multiple MPI implementations have been installed on the cluster, you can use the `mpi-selector` to switch between them. The MPIs that can be managed with the `mpi-selector` are:

- MVAPICH
- MVAPICH2

The `mpi-selector` is an OFED utility that is installed as a part of OFED+ Host Software. Its basic functions include:

- Listing available MPI implementations
- 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 in use

Following is an example for listing and selecting an MPI:

```
$ mpi-selector --list

  mpi-1.2.3

  mpi-3.4.5

$ mpi-selector --set mpi-3.4.5
```

The new default takes effect in the next shell that is started. See the `mpi-selector` man page for more information.

The example shell scripts `mpivars.sh` and `mpivars.csh`, for registering with `mpi-selector`, are provided as part of the `mpi-devel` RPM in `$prefix/share/mpich/mpi-selector-{intel, gnu, pgi}` directories.

For all non-GNU compilers that are installed outside standard Linux search paths, set up the paths so that compiler binaries and runtime libraries can be resolved. For example, set `LD_LIBRARY_PATH`, both in your local environment and in an rc file (such as `.mpirunrc`, `.bashrc`, or `.cshrc`), are invoked on remote nodes. See

"Environment for Node Programs" on page 70 and "Compiler and Linker Variables" on page 62 for information on setting up the environment for information on setting the run-time library path.

Note: The Intel-compiled versions require that the Intel compiler be installed and that paths to the Intel compiler runtime libraries be resolvable from the user's environment. The version used is Intel 10.1.012.

5.7 Platform MPI 8

Platform MPI 8 (formerly HP–MPI) is a high performance, production–quality implementation of the Message Passing Interface (MPI), with full MPI-2 funcionality. Platform MPI 8 is distributed by over 30 commercial software vendors, so you may need to use it if you use certain HPC applications, even if you don't purchase the MPI separately.

5.7.1 Installation

Follow the instructions for downloading and installing Platform MPI 8 from the Platform Computing web site.

5.7.2 Setup

Edit two lines in the `hpmpi.conf` file as follows:

Change,

```
MPI_ICMOD_PSM__PSM_MAIN = "^ib_ipath"
```

to,

```
MPI_ICMOD_PSM__PSM_MAIN = "^"
```

Change,

```
MPI_ICMOD_PSM__PSM_PATH = "^ib_ipath"
```

to,

```
MPI_ICMOD_PSM__PSM_PATH = "^"
```

5.7.3 Compiling Platform MPI 8 Applications

As with Open MPI, Intel recommends that you use the included wrapper scripts that invoke the underlying compiler (see Table 5-4).

Table 5-4. Platform MPI 8 Wrapper Scripts

Wrapper Script Name	Language
`mpicc`	C
`mpiCC`	C
`mpi77`	Fortran 77
`mpif90`	Fortran 90

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

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

5.7.4 Running Platform MPI 8 Applications

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

```
$ mpirun -np 4 -hostfile mpihosts -PSM mpi_app_name
```

To run over IB Verbs, type:

```
$ mpirun -np 4 -hostfile mpihosts -IBV mpi_app_name
```

To run over TCP (which could be IPoIB if the hostfile is setup for IPoIB interfaces), type:

```
$ mpirun -np 4 -hostfile mpihosts -TCP mpi_app_name
```

5.7.5 More Information on Platform MPI 8

For more information on Platform MPI 8, see the Platform Computing web site

5.8 Intel MPI

Intel MPI version 4.0 is supported with this release.

5.8.1 Installation

Follow the instructions for download and installation of Intel MPI from the Intel web site.

5.8.2 Setup

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

The setup for Intel MPI is described in the following steps:

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

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

```
tmi-1.0-1
```

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

```
psm X.X libtmip_psm.so " " # Comments OK
```

Intel MPI can also be run over uDAPL, which uses IB Verbs. uDAPL is the user mode version of the Direct Access Provider Library (DAPL), and is provided as a part of the OFED packages. You will also have to have IPoIB configured.

The setup for Intel MPI is described in the following steps:

1. Make sure that DAPL 1.2 or 2.0 is installed on every node and all nodes have the same version installed. In this release they are called `compat-dapl`. Both versions are supplied with the OpenFabrics RPMs and are included in the Intel OFED+ Host Software package. They can be installed either with the Intel OFED+ Host Software installation or using the `rpm files` after the Intel OFED+ Host Software tar file has been unpacked. For example:

   Using DAPL 1.2.

```
$ rpm -qa | grep compat-dapl

compat-dapl-1.2.12-1.x86_64.rpm
compat-dapl-debuginfo-1.2.12-1.x86_64.rpm
compat-dapl-devel-1.2.12-1.x86_64.rpm
compat-dapl-devel-static-1.2.12-1.x86_64.rpm
compat-dapl-utils-1.2.12-1.x86_64.rpm
```

   Using DAPL 2.0.

```
$ rpm -qa | grep dapl

dapl-devel-static-2.0.19-1

compat-dapl-1.2.14-1

dapl-2.0.19-1

dapl-debuginfo-2.0.19-1

compat-dapl-devel-static-1.2.14-1

dapl-utils-2.0.19-1

compat-dapl-devel-1.2.14-1

dapl-devel-2.0.19-1
```

2. Verify that there is a `/etc/dat.conf` file. It should be installed by the `dapl-` RPM. The file `dat.conf` contains a list of interface adapters supported by uDAPL service providers. In particular, it must contain mapping entries for OpenIB-cma for `dapl 1.2.x` and `ofa-v2-ib` for `dapl 2.0.x`, in a form similar to this (each on one line):

```
OpenIB-cma u1.2 nonthreadsafe default libdaplcma.so.1 dapl.1.2
"ib0 0" ""
```

   and

```
ofa-v2-ib0 u2.0 nonthreadsafe default libdaplofa.so.2 dapl.2.0
"ib0 0" ""
```

3. On every node, type the following command (as a root user):

```
# modprobe rdma_ucm
```

   To ensure that the module is loaded when the driver is loaded, add `RDMA_UCM_LOAD=yes` to the `/etc/infiniband/openib.conf` file. (Note that `rdma_cm` is also used, but it is loaded automatically.)

4. Bring up an IPoIB interface on every node, for example, `ib0`. See "Configure IPoIB" on page 20 for more details on configuring IPoIB.

Intel MPI has different bin directories for 32-bit (`bin`) and 64-bit (`bin64`); 64-bit is the most commonly used.

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
```

Substitute `bin` if using 32-bit.

5.8.3 Compiling Intel MPI Applications

As with Open MPI, Intel recommended that you use the included wrapper scripts that invoke the underlying compiler. The default underlying compiler is GCC, including gfortran. Note that there are more compiler drivers (wrapper scripts) with Intel MPI than are listed here (see Table 5-5); check the Intel documentation for more information.

Table 5-5. Intel MPI Wrapper Scripts

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

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

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

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

5.8.4 Running Intel MPI Applications

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

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

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

Pass the following option to `mpirun` to select TMI:

```
-genv I_MPI_FABRICS tmi
```

Pass the following option to `mpirun` to select uDAPL:

uDAPL 1.2:

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

uDAPL 2.0:

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

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

TMI:

```
-genv TMI_DEBUG 1
```

uDAPL:

```
-genv I_MPI_DEBUG 2
```

5.8.5 Further Information on Intel MPI

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

5.9 Improving Performance of Other MPIs Over IB Verbs

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

Note: No manual tuning is necessary for PSM-based MPIs, since the PSM layer determines the largest possible IB MTU for each source/destination path.

The maximum supported MTU size of HCAs is 4K.

Support for 4K IB MTU requires switch support for 4K MTU. The method to set the IB MTU size varies by MPI implementation:

- Open MPI defaults to the lower of either the IB MTU size or switch MTU size.
- MVAPICH defaults to an IB MTU size of 1024 bytes. This can be over-ridden by setting an environment variable:

```
$ export VIADEV_DEFAULT_MTU=MTU2048
```

  Valid values are `MTU256`, `MTU512`, `MTU1024`, `MTU2048` and `MTU4096`. This environment variable must be set for all processes in the MPI job. To do so, use `~/.bashrc` or use of `/usr/bin/env`.

- MVAPICH2 defaults to an IB MTU size of 2048 bytes, which should be sufficient for most applications.
- Platform MPI over IB Verbs automatically determines the IB MTU size.
- Intel MPI over uDAPL (which uses IB Verbs) automatically determines the IB MTU size.

§ §

6.0 SHMEM Description and Configuration

6.1 Overview

Intel SHMEM is a user-level communications library for one-sided operations. It implements the SHMEM Application Programming Interface (API) and runs on the Intel® True Scale Fabric Stack. The SHMEM API provides global distributed shared memory across a network of hosts. Details of the API implementation are included in an appendix.

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

6.2 Interoperability

Intel SHMEM depends on the Performance Scaled Messaging (PSM) protocol layer, implemented as a user-space library. Intel SHMEM is only available to run with Intel HCAs.

6.3 Installation

Note: Refer to the *Intel® True Scale Fabric OFED+ Host Software Release Notes* for the latest supported OS, MPI, and MVAPICH releases.

SHMEM is packaged with the Intel IFS or Intel OFED+ Host software.Every node in the cluster must have a Intel HCA and be running RedHat Enterprise Linux* (RHEL) OS. One or more Message Passing Interface (MPI) implementations are required and Performance Scaled Messaging (PSM) support must be enabled within the MPI. The following MPI Implementations are supported:

- Open MPI `<VERSION>` configured to include PSM support. This is provided by Intel IFS and can be found in the following directories:

```
/usr/mpi/gcc/openmpi-<VERSION>-qlc

/usr/mpi/intel/openmpi-<VERSION>-qlc

/usr/mpi/pgi/openmpi-<VERSION>-qlc
```

  The `-qlc` suffix denotes that this is the Intel PSM version.

- MVAPICH `<VERSION>` compiled for PSM. This is provided by Intel IFS and can be found in the following directories:

```
/usr/mpi/gcc/mvapich-<VERSION>-qlc
```

`/usr/mpi/intel/mvapich-<VERSION>-qlc`

`/usr/mpi/pgi/mvapich-<VERSION>-qlc`

The `-qlc` suffix denotes that this is the Intel PSM version.

- MVAPICH2 `<VERSION>` compiled for PSM. This is provided by Intel IFS and can be found in the following directory:

`/usr/mpi/gcc/mvapich2-<VERSION>-qlc`

`/usr/mpi/intel/mvapich2-<VERSION>-qlc`

`/usr/mpi/pgi/mvapich2-<VERSION>-qlc`

The `-qlc` suffix denotes that this is the Intel PSM version.

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

`/usr/mpi/intel/openmpi-<VERSION>-qlc`

The following C compilers are supported:

- gcc (as provided by distro) in 64-bit mode
- Intel `<VERSION>` C compiler in 64-bit mode
- PGI `<VERSION>` C compiler in 64-bit mode

For more information or to perform and installation with SHMEM enabled refer to Section 4 of the *Intel® True Scale Fabric Software Installation Guide*.

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

`/usr/shmem/intel`

`/usr/shmem/intel/bin`

`/usr/shmem/intel/bin/mvapich`

`/usr/shmem/intel/bin/mvapich2`

`/usr/shmem/intel/bin/openmpi`

`/usr/shmem/intel/lib64`

`/usr/shmem/intel/lib64/mvapich`

`/usr/shmem/intel/lib64/mvapich2`

`/usr/shmem/intel/lib64/openmpi`

`/usr/shmem/intel/include`

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

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

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

6.4 SHMEM Programs

6.4.1 Basic SHMEM Program

Following is an example of a basic SHMEM program:

```
% cat shmem_world.c

#include <shmem.h>

#include <stdio.h>



int main ()

{

  shmem_init();

  printf("Hello from PE %d out of %d\n", my_pe(), num_pes());

  return 0;

  }
```

Note: These instructions assume a standard SHMEM installation and that `/usr/shmem/Intel/bin` has been added to the $PATH.

The `%` character in the previous example is used to indicate the shell prompt and is followed by a command. The program can be compiled and linked using the `shmemcc` wrapper script:

```
% shmemcc shmem_world.c -o shmem_world
```

The program can be run using the `shmemrun` wrapper script:

```
% shmemrun -m hosts -np 2 ./shmem_world

  Hello from PE 1 out of 2

  Hello from PE 0 out of 2
```

This script assumes a `hosts` file is available, containing the host names on which the program is run. The `-np` option is used to specify the number of processing elements (PEs) to be run (for example, 2).

6.4.2 Compiling SHMEM Programs

The `shmemcc` script is a wrapper script for the compilation of the SHMEM C programs. The main purpose of the script is to call the C compiler with additional options to specify the SHMEM include directory, the SHMEM library directory, and to appropriately link in the SHMEM library. The `shmemcc` script automatically determines the correct directories by finding them relative to its own location. The standard directory layout of the Intel SHMEM software is assumed.

The default C compiler is gcc, and can be overridden by specifying a compiler with the `$SHMEM_CC` environment variable.

If the option `-show` is added to the `shmemcc` command, it displays the command line that would be used to invoke the C compiler, but the C compiler will not be invoked. All other arguments to `shmemcc` are passed through to the C compiler without modification.

The C compiler can be used directly without using `shmemcc`. In that case the user must add the following to the command line:

For compilations add the following option:

```
-I $SHMEM_DIR/include
```

For linkages add the following options:

```
-Wl,--export-dynamic,--allow-shlib-undefined
```

```
-L $SHMEM_DIR/lib64/default
```

```
-lintel_shmem
```

Where *`$SHMEM_DIR`* in both of the options denotes the top-level directory of the SHMEM installation, typically the directory is `/usr/shmem/intel`.

The `-L` option uses the default version of the SHMEM libraries. The default is actually a symbolic link to libraries built for a specific MPI implementation. However, this choice does not constrain the SHMEM binary, and it can be run over any of the supported MPIs.

Note: If the SHMEM RPM is installed with `--prefix=usr` then the `-I` option is not necessary since the header files are in system default locations. All of the linkage options are still required.

The rationale for the `-Wl, --export-dynamic, --allow-shlib-undefined` options are to prevent other library and symbol dependencies in the SHMEM library from percolating up into the application binaries. These symbols include those from the underlying MPI implementation. There is no need to couple the application binary to a particular MPI, and these symbols will be correctly resolved at run-time. The advantage of this approach is that SHMEM application binaries will be portable across different implementations of the Intel SHMEM library, including portability over different underlying MPIs.

6.4.3 Running SHMEM Programs

6.4.3.1 Using shmemrun

The `shmemrun` script is a wrapper script for running SHMEM programs using `mpirun`. The main purpose of the script is to call `mpirun` with additional options to specify the SHMEM library directory so that its dynamic libraries can be resolved. The script detects which `mpirun` is being used and remaps some common `mpirun` options to present a convenient and consistent interface to SHMEM users. Additionally, it enables PSM support in the underlying `mpirun` if required, and auto-propagates PSM, IPATH and SHMEM environment variables to the MPI processes. The `shmemrun` script automatically determines the correct directories by finding them relative to its own location. The `shmemrun` script can only automatically determine the correct directories if the standard directory layout of the Intel SHMEM software has not been changed.

By default `mpirun` is picked up from the path and is assumed to be called `mpirun`. Alternatively, the pathname of `mpirun` can be specified with the `$SHMEM_MPIRUN` environment variable. There is also support for integration with slurm (see "Slurm Integration" on page 92). The following mpirun commands are supported:

- Open MPI: `mpirun`
- MVAPICH: `mpirun` and `mpirun_rsh`
- MVAPICH2: `mpirun` and `mpirun_rsh`

If the `shmemrun` script is run with `–show` option, it shows that the command line was used to invoke `mpirun`, but will not invoke it. Options that specify the number of processes and the hosts file are mapped by `shmemrun` to options that are accepted by the underlying `mpirun`. The contents of the host file can be parsed and regenerated if necessary and options to propagate environment variables are provided. The rationale for this script is to allow you to use the familiar options from the `mpirun` chosen and the options will automatically be remapped as required for the actual `mpirun`. This makes it possible to write scripts that call `shmemrun` without exposing these details of the underlying `mpirun` command.

If the `shmemrun` script finds the special -- option while processing the option list, that option is deleted and subsequent options and command line arguments are passed through without any modification. Using this option is useful to prevent `shmemrun` from modifying options of the program that are being run.

6.4.3.2 Running programs without using shmemrun

If you do not wish to use this wrapper script, then you must arrange for the SHMEM libraries to be found at run time using `$LD_LIBRARY_PATH` or an equivalent mechanism, and ensure that PSM support is enabled in your MPI implementation. The libraries can be found at:

`$SHMEM_DIR``/lib64/`*`$MPI`*

Where *`$SHMEM_DIR`* denotes the top-level directory of the SHMEM installation, typically `/usr/shmem/intel`, and ***$MPI*** is your choice of MPI (one of `mvapich`, `mvapich2`, or `openmpi`).

Additionally, the PSM receive thread and back-trace must be disabled using the following commands:

```
export PSM_RCVTHREAD=0

export IPATH_NO_BACKTRACE=1
```

6.5 Intel SHMEM Relationship with MPI

Intel SHMEM requires the Intel PSM layer to provide the network transport function and this runs exclusively on Intel HCAs. It also requires a compatible MPI implementation (also running over PSM) to provide program start up and other miscellaneous services. The one-sided operations in Intel SHMEM are not layered on top of MPI, however, and go directly to PSM to give low-latency, high-performance access to the HCA architecture.

Typical SHMEM programs are written using calls to the SHMEM API and do not use MPI calls. In this case the program binary generated by `shmemcc` contains references to the SHMEM dynamic library and no references at all to MPI libraries. These binaries are portable across all MPI implementations supported by Intel SHMEM. This is true of the `get/put` micro-benchmarks provided by Intel SHMEM. The desired MPI can be selected at run time simply by placing the desired `mpirun` on `$PATH`, or by using the `$SHMEM_MPIRUN` environment variable.

Alternatively, it is possible to write hybrid SHMEM/MPI programs that use features from both the SHMEM and MPI libraries. These programs must call `shmem_init()` to initialize the SHMEM library state. They may also use `MPI_Init()` and `MPI_Finalize()` if needed. There will be a direct one-to-one correspondence between the SHMEM and `MPI_COMM_WORLD` rank assignments:

`shmem_my_pe()` will match `MPI_Comm_rank()` on `MPI_COMM_WORLD`

```
shmem_n_pes() will match MPI_Comm_size() on MPI_COMM_WORLD
```

Hybrid SHMEM/MPI programs must be linked against SHMEM libraries and the correct MPI libraries. It is recommended that the implementation of the MPI wrapper script(s) (`mpicc`) is used for compilation and that additional options are specified to find the SHMEM include and library files. One approach is to set up the `shmemcc` wrapper script to use `mpicc` as its compiler using the environment variable setting:

```
export SHMEM_CC=mpicc
```

This setting needs to be adjusted if `mpicc` is not already on the $PATH. The generated binary has references to both SHMEM and MPI libraries and is specific to that MPI implementation. Intel recommends that `shmemrun` is used to run the program. The user must ensure that the correct `mpirun` is picked up from `$PATH` or using the `$SHMEM_MPIRUN` environment variable.

6.6 Slurm Integration

Intel SHMEM relies on an MPI implementation to provide a run-time environment for jobs. This includes job start-up, stdin/stdout/stderr routing, and other low performance control mechanisms. Intel 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.

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

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

Below are various options for integration of the Intel SHMEM and slurm.

6.6.1 Full Integration

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

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

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

The user is responsible for setting up the environment appropriately. This includes adding Intel SHMEM's library directory to LD_LIBRARY_PATH. See "Running SHMEM Programs" on page 90 for more information on the environment setup.

6.6.2 Two-step Integration

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

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

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

6.6.3 No Integration

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

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

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

6.7 Sizing Global Shared Memory

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

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

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

- Accessing global shared memory on a PE on a different host: This is achieved by sending put, get, and atomic requests across the interconnect.

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

The Intel 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 will fail if there is no room in the virtual memory space of the process following the global shared memory segment. Upon failure the call to `shmalloc` or `shrealloc` returns NULL. The only down-side of using a large maximum size is occupancy of virtual address space (48 bits for 64-bit processes is very plentiful), and set-up of page table entries by the OS. A reasonable limit is 4GB per process. One side-effect of this approach is that SHMEM programs consume a large amount of virtual memory when viewed with the "top" program. This is due to the large maximum size setting. The `RES` field of `top` indicates the actual amount of memory that is resident in memory (for example, in actual use).

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

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

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

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

- Check for low `ulimits` on memory:

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

ulimit -v : max virtual memory
```

- Check the contents of these *sysctl* variables:

```
sysctl kernel.shmmax ; maximum size of a single shm allocation in
bytes

sysctl kernel.shmall ; maximum size of all shm allocations in
“pages”

sysctl kernel.shmnmi ; maximum number of shm segments
```

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

```
df /dev/shm
```

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

```
ls /dev/shm
```

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

6.8 Progress Model

Intel SHMEM supports active and passive progress models. Active progress means that the PE must actively call into SHMEM for progress to be made on SHMEM one-sided operations. Passive progress means that progress on SHMEM one-sided operations can occur without the application needing to call into SHMEM. Active progress is the default mode of operation for Intel SHMEM. Passive progress can be selected using an environment variable where required.

6.8.1 Active Progress

In the active progress mode SHMEM progress is achieved when the application calls into the SHMEM library. This approach is well matched to applications that call into SHMEM frequently, for example, to have a fine grained mix of SHMEM operations and computation. This mix is typical of many SHMEM applications. Applications that spend large amount of contiguous time in computation without calling SHMEM routines will cause SHMEM progress to be delayed for that period of time. Additionally, applications must not poll on locations waiting for puts to arrive without calling SHMEM, since progress will not occur and the program will hang. Instead, SHMEM applications should use one of the wait synchronization primitives provided by SHMEM. In active progress mode Intel SHMEM will achieve full performance.

6.8.2 Passive Progress

In the passive progress mode SHMEM progress will continue to occur when the application calls into SHMEM, but can additionally occur in the background when the application is not calling into SHMEM. This is achieved using an additional progress thread per PE. The progress thread is provided by PSM and is scheduled at a relatively low frequency, typically 10 to 100 times a second. This thread will cause independent SHMEM progress where required, both on the initiator side and the target side of SHMEM operations. In this mode applications can poll on locations waiting for puts to arrive without calling SHMEM. Progress will be achieved in this case by the progress thread, though it will incur the scheduling latency for the progress thread which may have a significant impact on overall performance if this idiom is used frequently. The scheduling frequency of the PSM progress thread can be tuned as described in “Environment Variables” on page 96.

Other performance effects of using passive progress include the following:

- The progress thread consumes some CPU cycles, though this is low because the progress thread runs infrequently.
- The SHMEM library uses additional locks in its implementation to protect its data structures against concurrent updates from the PE thread and the progress thread. There is a slight additional cost in the performance critical path because of this locking. This cost is minimal because contention on the lock is very low (the progress thread runs infrequently) and because each progress thread runs on the same CPU core as the corresponding PE thread (giving good cache locality for the lock).
- SHMEM's long message protocol is disabled. This is because the long message protocol implementation does not support passive progress. The effect of disabling this is to reduce long message bandwidth to that which can be achieved with the short message protocol. There is no effect on the bandwidth for message sizes below the long message break-point, which is set to 16KB by default.

6.8.3 Active versus Passive Progress

It is expected that most applications will be run with Intel SHMEM's active progress mode since this gives full performance. The passive progress mode will typically be used in the following circumstances:

- For applications that use a polling idiom that is incompatible with the active progress mode, and where the application programmer is unable or unwilling to recode to use the appropriate SHMEM wait primitive.
- For compliance to a SHMEM standard that has a passive progress requirement.

6.9 Environment Variables

Table 6-1 list the environment variables that are currently provided by the SHMEM run time library.

Note: The set of supported environment variables and their defaults may vary.from release to release.

Table 6-1. SHMEM Run Time Library Environment Variables

Environment Variable	Default	Description
`$SHMEM_SHMALLOC_INIT_SIZE`	16MB	Initial size of the global shared memory segment.
`$SHMEM_SHMALLOC_MAX_SIZE`	4GB	Maximum size of the global shared memory segment.
`$SHMEM_SHMALLOC_CHECK`	on	Shared memory consistency checks set for 0 to disable and 1 to enable. These are good checks for correctness but degrade the performance of shmalloc() and shfree(). These routines are usually not important for benchmark performance, so for now the checks are turned on to catch bugs early.
`$SHMEM_IDENTIFY`		If set, each SHMEM process will print out the SHMEM identity string and the path to the SHMEM library file.
`$SHMEM_GET_REQ_LIMIT`	64	Maximum number of outstanding short get requests for this end-point for the short get protocol (0 means unlimited). Each short get request can be up to 2KB.

Table 6-1. SHMEM Run Time Library Environment Variables (Continued)

Environment Variable	Default	Description
`$SHMEM_GET_LONG_REQ_LIMIT`	16	Maximum number of outstanding get requests for this end-point for the long get protocol (0 means unlimited).
`$SHMEM_PUT_FRAG_LIMIT`	4096	Maximum number of outstanding put fragments for this end-point for the short put protocol (0 means unlimited). Each short put fragment can be up to 2KB.
`$SHMEM_PUT_LONG_FRAG_LIMIT`	128	Maximum number of outstanding put fragment requests for this end-point for the long get protocol (0 means unlimited).
`$SHMEM_GET_LONG_SIZE`	8KB for non-blocking `gets` 32KB for blocking `gets`	Gets of this size and larger use the SHMEM long get message protocol. Note that the parameter only allows the size to be changed in unison for both non-blocking and blocking gets.
`$SHMEM_PUT_LONG_SIZE`	8KB for non-blocking puts 16KB for blocking puts	Puts of this size and larger use the SHMEM long put message protocol. Note that the parameter only allows the size to be changed in unison for both non-blocking and blocking puts.
`$SHMEM_PUT_REPLY_COMBINING_COUNT`	8	Number of consecutive put replies on a flow to combine together into a single reply.

The command `shmemrun` automatically propagates `SHMEM*` environment variables from its own environment to all the SHMEM processes. This means that the environment variables can be simply setup in the front-end shell used to invoke `shmemrun`. The command `shmemrun` also has its own environment variables that are listed in Table 6-2.

Table 6-2. `shmemrun` Environment Variables

Environment Variable	Default	Description
`$SHMEM_MPIRUN`	`mpirun` from the PATH	Specifies where to find `mpirun`.
`$SHMEMRUN_VERBOSE`		Enables verbose output for `shmemrun`.
`$SHMEMRUN_SLEEP`		Specifies a sleep time (in seconds) after the job completes. This variable is intended for testing use.
`$SHMEMRUN_TIMEOUT`		Specifies a time-out value (in seconds). When the timeout value is reached, the `mpirun` is killed. This variable is intended for testing use.

6.10 Implementation Behavior

Some SHMEM properties are not fully specified by the SHMEM API specification. This section discusses the behavior for the Intel SHMEM implementation.

For a put operation, these descriptions use the terms "local completion" and “remote completion”. Once a put is locally complete, the source buffer on the initiating PE is available for reuse. Until a put is locally complete the source buffer must not be modified since that buffer is in use for the put operation. A blocking put is locally complete immediately upon return from the put. A non-blocking put is not locally complete upon return from the put<V_Variable>—different mechanisms are used for detecting local completion using either an explicit handle (use `shmem_test_nb()` or `shmem_wait_nb()`) or a NULL handle (use `shmem_quiet()`). Once a put is remotely complete the destination buffer on the target PE is fully written and available for use. The mechanism provided by SHMEM for detecting remote completion are described below.

- `shmem_fence()` - This function ensures that all puts issued by this PE prior to the fence will become remotely visible before any puts issued by this PE after the fence. The call does not necessarily imply that any of the prior puts are actually remotely visible at the point of the fence, only that this ordering is guaranteed.
- `shmem_quiet()` - This function waits for remote completion of all puts issued by this PE prior to the quiet operation. Therefore, once the quiet operation returns, it is guaranteed that all those puts will be remotely visible to other PEs. This guarantee of remote completion applies to all puts<V_Variable>—blocking puts, non-blocking puts with handles, and non-blocking puts with NULL handles. Additionally, this function additionally waits for local completion of non-blocking puts and non-blocking gets that were issued with a NULL handle.
- `shmem_test_nb()` and `shmem_wait_nb()` can be used to test and wait for local completion of a `non_blocking` operation. For a non-blocking put, this does not indicate whether remote completion has occurred.

Additional properties of the Intel SHMEM implementation are:

- The Intel SHMEM implementation makes no guarantees as to the ordering in which the bytes of a put operation are delivered into the remote memory. It is *not* a safe assumption to poll or read certain bytes of the put destination buffer (for example, the last 8 bytes) to look for a change in value and then infer that the entirety of the put has arrived. The correct mechanism for this is to use the `shmem_quiet()` operation to force remote completion, or to use the following type of sequence:
  - — Initiator side:
- Issue a batch of puts all unordered with respect to each other
- `shmem_fence()`
- 8 byte put to a sync location
  - — Target side:
- Wait for the sync location to be written
- Now it is safe to make observations on all puts prior to fence
- `shmem_int_wait()`, `shmem_long_wait()`, `shmem_longlong_wait()`, `shmem_short_wait()`, `shmem_wait()`, `shmem_int_wait_until()`, `shmem_long_wait_until()`, `shmem_longlong_wait_until()`, `shmem_short_wait_until()`, `shmem_wait_until()` - These SHMEM operations are provided for waiting for a variable in local symmetric memory to change value due to an incoming put. In the active progress mode SHMEM applications must use these routines for this purpose, and not implement their own polling loop without SHMEM library calls. In the passive progress mode SHMEM application may use a polling loop without a SHMEM library call. However, performance will typically be substantially improved by using the SHMEM wait operation instead.
- `shmem_stack()` is implemented as a no-op since this is a distributed memory cluster architecture.
- `shmem_ptr`(void *target, int pe) returns the provided address, if the PE is my PE, otherwise NULL. This implementation is sufficient to conform to the SHMEM API and is appropriate for a distributed memory cluster architecture.
- `shmem_clear_cache_inv()`, `shmem_clear_cache_line_inv()`, `shmem_set_cache_inv()`, `shmem_set_cache_line_inv()`, `shmem_udcflush()`, and `shmem_udcflush_line()` are each implemented as a no-op since there is no global memory caching in this implementation.

- This SHMEM implementation allows remote access to variables that are in the symmetric heap and static data/read-only data sections only. It does not support static data sections in dynamically loaded libraries.

6.11 Application Programming Interface

Table 6-3 lists the provided SHMEM Application Programming Interface (API) calls and details any restrictions.

Table 6-3. SHMEM Application Programming Interface Calls

Operation	Calls
General Operations	shmem_init
	start_pes
	my_pe
	_my_pe
	shmem_my_pe
	num_pes
	_num_pes
	shmem_n_pes
Symmetric heap	shmalloc
	shmemalign
	shfree
	shrealloc
Contiguous Put Operations	shmem_short_p
	shmem_int_p
	shmem_long_p
	shmem_float_p
	shmem_double_p
	shmem_longlong_p
	shmem_longdouble_p
	shmem_char_put
	shmem_short_put
	shmem_double_put
	shmem_float_put
	shmem_int_put
	shmem_long_put
	shmem_longdouble_put
	shmem_longlong_put
	shmem_put
	shmem_put32
	shmem_put64
	shmem_put128
	shmem_putmem

Table 6-3. SHMEM Application Programming Interface Calls (Continued)

Operation	Calls
Non-blocking Put Operations	`shmem_double_put_nb`
	`shmem_float_put_nb`
	`shmem_int_put_nb`
	`shmem_long_put_nb`
	`shmem_longdouble_put_nb`
	`shmem_longlong_put_nb`
	`shmem_put_nb`
	`shmem_put32_nb`
	`shmem_put64_nb`
	`shmem_put128_nb`
	`shmem_putmem_nb`
	`shmem_short_put_nb`
Strided Put Operations	`shmem_double_iput`
	`shmem_float_iput`
	`shmem_int_iput`
	`shmem_iput`
	`shmem_iput32`
	`shmem_iput64`
	`shmem_iput128`
	`shmem_long_iput`
	`shmem_longdouble_iput`
	`shmem_longlong_iput`
	`shmem_short_iput`
Indexed Put Operations	`shmem_ixput`
	`shmem_ixput32`
	`shmem_ixput64`
Put and Non-blocking Ordering, Flushing and Completion	`shmem_fence`
	`shmem_quiet`
	`shmem_wait_nb`
	`shmem_test_nb`
	`shmem_poll_nb` Same as shmem_test_nb, provided for compatibility
Contiguous Get Operations	`shmem_short_g`
	`shmem_int_g`
	`shmem_long_g`
	`shmem_float_g`
	`shmem_double_g`
	`shmem_longlong_g`
	`shmem_longdouble_g`
	`shmem_char_get`
	`shmem_short_get`
	`shmem_double_get`
	`shmem_float_get`
	`shmem_int_get`
	`shmem_long_get`
	`shmem_longdouble_get`
	`shmem_longlong_get`
	`shmem_get`
	`shmem_get32`
	`shmem_get64`
	`shmem_get128`
	`shmem_getmem`

Table 6-3. SHMEM Application Programming Interface Calls (Continued)

Operation	Calls
Non-blocking Get Operations	shmem_double_get_nb
	shmem_float_get_nb
	shmem_int_get_nb
	shmem_long_get_nb
	shmem_longdouble_get_nb
	shmem_longlong_get_nb
	shmem_short_get_nb
	shmem_get_nb
	shmem_get32_nb
	shmem_get64_nb
	shmem_get128_nb
	shmem_getmem_nb
Strided Get Operations	shmem_double_iget
	shmem_float_iget
	shmem_int_iget
	shmem_iget
	shmem_iget32
	shmem_iget64
	shmem_iget128
	shmem_long_iget
	shmem_longdouble_iget
	shmem_longlong_iget
	shmem_short_iget
Indexed Get Operations	shmem_ixget
	shmem_ixget32
	shmem_ixget64
Barriers	barrier
	shmem_barrier_all
	shmem_barrier
Broadcasts	shmem_broadcast
	shmem_broadcast32
	shmem_broadcast64
Concatenation	shmem_collect
	shmem_collect32
	shmem_collect64
	shmem_fcollect
	shmem_fcollect32
	shmem_fcollect64
Synchronization operations	shmem_int_wait
	shmem_long_wait
	shmem_longlong_wait
	shmem_short_wait
	shmem_wait
	shmem_int_wait_until
	shmem_long_wait_until
	shmem_longlong_wait_until
	shmem_short_wait_until
	shmem_wait_until

Table 6-3. SHMEM Application Programming Interface Calls (Continued)

Operation	Calls
Atomic operations	`shmem_double_swap`
	`shmem_float_swap`
	`shmem_short_swap`
	`shmem_int_swap`
	`shmem_long_swap`
	`shmem_longlong_swap`
	`shmem_swap`
	`shmem_short_cswap`
	`shmem_int_cswap`
	`shmem_long_cswap`
	`shmem_longlong_cswap`
	`shmem_short_mswap`
	`shmem_int_mswap`
	`shmem_long_mswap`
	`shmem_longlong_mswap`
	`shmem_short_inc`
	`shmem_int_inc`
	`shmem_long_inc`
	`shmem_longlong_inc`
	`shmem_short_add`
	`shmem_int_add`
	`shmem_long_add`
	`shmem_longlong_add`
	`shmem_short_finc`
	`shmem_int_finc`
	`shmem_long_finc`
	`shmem_longlong_finc`
	`shmem_short_fadd`
	`shmem_int_fadd`
	`shmem_long_fadd`
	`shmem_longlong_fadd`

Table 6-3. SHMEM Application Programming Interface Calls (Continued)

Operation	Calls
Reductions	`shmem_int_and_to_all`
	`shmem_long_and_to_all`
	`shmem_longlong_and_to_all`
	`shmem_short_and_to_all`
	`shmem_int_or_to_all`
	`shmem_long_or_to_all`
	`shmem_longlong_or_to_all`
	`shmem_short_or_to_all`
	`shmem_int_xor_to_all`
	`shmem_long_xor_to_all`
	`shmem_longlong_xor_to_all`
	`shmem_short_xor_to_all`
	`shmem_double_min_to_all`
	`shmem_float_min_to_all`
	`shmem_int_min_to_all`
	`shmem_long_min_to_all`
	`shmem_longdouble_min_to_all`
	`shmem_longlong_min_to_all`
	`shmem_short_min_to_all`
	`shmem_double_max_to_all`
	`shmem_float_max_to_all`
	`shmem_int_max_to_all`
	`shmem_long_max_to_all`
	`shmem_longdouble_max_to_all`
	`shmem_longlong_max_to_all`
	`shmem_short_max_to_all`
	`shmem_complexd_sum_to_all` Complex collectives are not implemented
	`shmem_complexf_sum_to_all` Complex collectives are not implemented
	`shmem_double_sum_to_all`
	`shmem_float_sum_to_all`
	`shmem_int_sum_to_all`
	`shmem_long_sum_to_all`
	`shmem_longdouble_sum_to_all`
	`shmem_longlong_sum_to_all`
	`shmem_short_sum_to_all`
	`shmem_complexd_prod_to_all` Complex collectives are not implemented
	`shmem_complexf_prod_to_all` Complex collectives are not implemented
	`shmem_double_prod_to_all`
	`shmem_float_prod_to_all`
	`shmem_int_prod_to_all`
	`shmem_long_prod_to_all`
	`shmem_longdouble_prod_to_all`
	`shmem_longlong_prod_to_all`
	`shmem_short_prod_to_all`
All-to-all (an extension beyond classic SHMEM)	`shmem_alltoall`
	`shmem_alltoall32`
	`shmem_alltoall64`

Table 6-3. SHMEM Application Programming Interface Calls (Continued)

Operation	Calls
Locks	`shmem_set_lock`
	`shmem_clear_lock`
	`shmem_test_lock`
Events	`clear_event`
	`set_event`
	`wait_event`
	`test_event`
General Operations (for compatibility)	`globalexit` Allows any process to abort the job
	`shmem_finalize` Call to terminate the SHMEM library
	`shmem_pe_accessible` Tests PE for accessibility
	`shmem_addr_accessible` Test address on PE for accessibility
Cache Operations (for compatibility)	`shmem_clear_cache_inv` Implemented as a no-op
	`shmem_clear_cache_line_inv` Implemented as a no-op
	`shmem_set_cache_inv` Implemented as a no-op
	`shmem_set_cache_line_inv` Implemented as a no-op
	`shmem_udcflush` Implemented as a no-op
	`shmem_udcflush_line` Implemented as a no-op
Stack/Pointer Operations (for compatibility)	`shmem_stack` Implemented as a no-op
	`shmem_ptr` Returns the address if the PE is my PE, otherwise NULL

6.12 SHMEM Benchmark Programs

The following SHMEM micro-benchmark programs are included:

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

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

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

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

The micro-benchmarks have the command line options shown in Table 6-4

Table 6-4. Intel SHMEM micro-benchmarks options

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

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

6.12.0.0.1 Intel SHMEM random access benchmark

`shmem-rand`: randomized `put`/`get` benchmark

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

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

Message sizes are specified in bytes (default = 8)

Options: See Table 6-5

Table 6-5. Intel 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 is scaled)
`-h`	displays the help page
`-l`	enable communication to local ranks
`-m INTEGER[K]`	memory size in MB (default = 8MB): or in KB with a K suffix
`-n`	use non-pipelined mode for NB ops (default pipelined)
`-o OP`	choose `OP` from `get`, `getnb`, `put`, `putnb`
`-p`	for blocking puts, no quiet every window (this is the default)
`-q`	for blocking puts, use quiet every window
`-r`	use ring pattern (default is random)

Table 6-5. Intel SHMEM random access benchmark options (Continued)

Option	Description
`-s`	enable communication to self
`-t FLOAT`	if the loop count is not given, run the test for this many seconds (default is 10s)
`-u`	run in uni-directional mode
`-v`	verbose mode (repeat for more verbose)
`-w INTEGER`	set the window size (default = 32)
`-x INTEGER`	window size limit (default = 16384)

6.12.0.0.2 Intel SHMEM all-to-all benchmark

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

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

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

Message sizes are specified in bytes (default 8)

Options: See Table 6-6

Table 6-6. Intel 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 is scaled)
`-h`	displays the help page
`-l`	enable communication to local ranks (including self)
`-m INTEGER[K]`	memory size in MB (default = 8MB): or in KB with a K suffix
`-n`	use non-pipelined mode for NB ops (default pipelined)
`-o OP`	choose `OP` from put, or `putnb`
`-p INTEGER`	offset for all-to-all schedule (default 1, usually set to ppn)
`-r`	randomize all-to-all schedule
`-s`	enable communication to self
`-t FLOAT`	if the loop count is not given, run the test for this many seconds (default is 10s)
`-v`	verbose mode (repeat for more verbose)
`-w INTEGER`	set the window size (default = 32)
`-x INTEGER`	window size limit (default = 16384)

6.12.0.0.3 Intel SHMEM barrier benchmark

`shmem-barrier`: barrier benchmark

Usage: `shmem-barrier [options]`

Options: See Table 6-7

Table 6-7. Intel SHMEM barrier benchmark options

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

6.12.0.0.4 Intel SHMEM reduce benchmark

`shmem-reduce`: reduce benchmark
Usage: shmem-reduce [options]
Options: See Table 6-8

Table 6-8. Intel SHMEM reduce benchmark options

Option	Description
`-b INTEGER`	number of barriers between reduces (default 0)
`-h`	displays the help page
`-i INTEGER[K]`	outer iterations (default 1)
`-r INTEGER`	inner iterations (default 10000)

§ §

7.0 Virtual Fabric support in PSM

7.1 Introduction

Performance Scaled Messaging (PSM) provides support for full Virtual Fabric (vFabric) integration, allowing users to specify IB Service Level (SL) and Partition Key (PKey), or to provide a configured Service ID (SID) to target a vFabric. Support for using IB path record queries to the Intel® True Scale Suite Fabric Manager (FM) during connection setup is also available, enabling alternative switch topologies such as Mesh/Torus. Note that this relies on the Distributed SA cache from Intel® True Scale Fabric Suite FastFabric (FF).

All PSM enabled MPIs can leverage these capabilities transparently, but only one MPI (Open MPI) is configured to support it natively. Native support here means that MPI specific mpirun switches are available to activate/deactivate these features. Other MPIs will require use of environment variables to leverage these capabilities. With MPI applications, the environment variables need to be propagated across all nodes/processes and not just the node from where the job is submitted/run. The mechanisms to do this are MPI specific, but for two common MPIs the following may be helpful:

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

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

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

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

Some of the features available require appropriate versions of associated software and firmware for correct operation. These requirements are listed in the relevant sections.

7.2 Virtual Fabric Support

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

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

7.3 Using SL and PKeys

SL and Pkeys can be specified natively for Open MPI. For other MPIs use the following list of environment variables to specify the SL and Pkey. The environment variables need to be propagated across all processes for correct operation.

Note: This is available with Open MPI v1.3.4rc4 and above only!

- **Open MPI**: Use mca parameters (`mtl_psm_ib_service_level` and `mtl_psm_ib_pkey`) to specify the pkey on the mpirun command line. Example:

```
mpirun -np 2 -machinefile machinefile -mca
mtl_psm_ib_service_level SL -mca mtl_psm_ib_pkey Pkey prog
prog_args
```

- Other MPIs can use the following environment variables that are propagated across all processes. This process is MPI library specific but samples on how to do this for Open MPI and MVAPICH2 are listed in the "Introduction" on page 109.
  - `IPATH_SL=`*SL* `# Service Level to Use` *0-15*
  - `PSM_PKEY=`*Pkey* `# Pkey to use`

7.4 Using Service ID

Full vFabric integration with PSM is available, allowing the user to specify a SID. For correct operation, PSM requires the following components to be available and configured correctly.

- Intel host FM Configuration – PSM MPI vFabrics need to be configured and enabled correctly in the `intel_fm.xml` file. 16 unique SIDs have been allocated in the sample file.
- OFED+ library needs to be installed on all nodes. This is available as part of FastFabric Toolset.
- Intel Distributed SA needs to be installed, configured and activated on all the nodes. This is part of FastFabric Toolset. Please refer to *Intel® True Scale Fabric Suite FastFabric User Guide* on how to configure and activate the Distributed SA. The SIDs configured in the FM configuration file should also be provided to the Distributed SA for correct operation.

Service ID can be specified natively for Open MPI. For other MPIs use the following list of environment variables. The environment variables need to be propagated across all processes for correct operation.

- Open MPI: Use mca parameters (`mtl_psm_ib_service_id` and `mtl_psm_path_query`) to specify the service id on the mpirun command line. Example:

```
mpirun -np 2 -machinefile machinefile -mca mtl_psm_path_query opp
-mca mtl_psm_ib_service_id SID prog prog_args
```

- Other MPIs can use the following environment variables:
  - `PSM_PATH_REC=opp # Path record query mechanism to use. Always specify opp`
  - `PSM_IB_SERVICE_ID=SID # Service ID to use`

7.5 SL2VL mapping from the Fabric Manager

PSM is able to use the SL2VL table as programmed by the FM. Prior releases required manual specification of the SL2VL mapping via an environment variable.

7.6 Verifying SL2VL tables on Intel 7300 Series HCAs

`iba_saquery` can be used to get the SL2VL mapping for any given port however, Intel 7300 series HCAs exports the SL2VL mapping via sysfs files. These files are used by PSM to implement the SL2VL tables automatically. The SL2VL tables are per port and available under `/sys/class/infiniband/`*`hca name`*`/ports/`*`port #`*`/sl2vl`. The directory contains 16 files numbered 0-15 that specify the SL. Listing the SL files returns the VL as programmed by the SL.

§ §

8.0 PSM Multi-rail

Multi-rail means that a process can use multiple network interface cards to transfer messages. With modern computer servers supporting multiple HCAs, multi-rail improves network performance for applications.

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

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

Note: Intel does not support the use of dual ported cards where both ports on the card are connected for use with PSM_MULTIRAIL. PSM_MUTIRAIL is the PSM environment variable that is used to enable this feature in PSM. If PSM_MULTIRAIL is not enabled, it is supported.

8.1 User Base

The system administrator sets up a PSM multi-rail system using multiple True Scale HCAs per node. If multiple fabrics are desired, the system administrator connects the HCA(s) to multiple fabrics, and configures each fabric with different subnet IDs.

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

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

8.2 Environment Variables

The following are environment variables that can be set:

PSM_MULTIRAIL=*n* – ***n*** can be any value. If this environment variable is set, to a non-zero value, PSM tries to setup multiple rails. Otherwise, multi-rail is turned off. How multi-rails are setup and how many rails are setup depends on environment variable PSM_MULTIRAIL_MAP is set or not.

PSM_MULTIRAIL_MAP=*unit:port,unit:port,unit:port*,… – ***unit*** is from 0, ***port*** is from 1. This environment variable tells PSM which unit/port pair is used to setup a rail, multiple specifications are separated by a comma. If only one rail is specified, it is equivalent to a single-rail case, the Unit/Port is specified instead of using Unit/Port assigned by QIB driver. PSM scans the above pattern until a violation or error is encountered, and uses the information it has gathered.

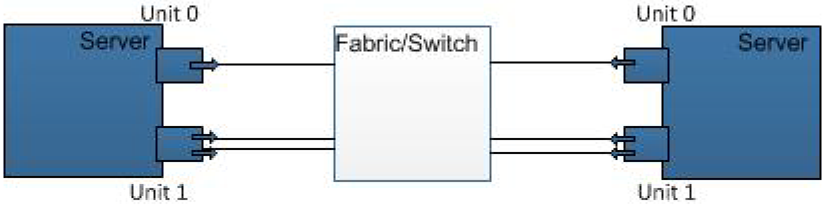
8.3 Examples of Single- and Multi-rail

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

Example 8-1. Single fabric, each node has two HCAs, Unit 0 has one port, Unit 1 has two ports

Figure 8-1 shows an example of a single fabric with each node having two cards. Unit 0 (qib0) has one port and Unit 1 (qib1) has two ports.

Figure 8-1. Single fabric, each node has two cards, Unit 0 has one port, Unit 1 has two ports



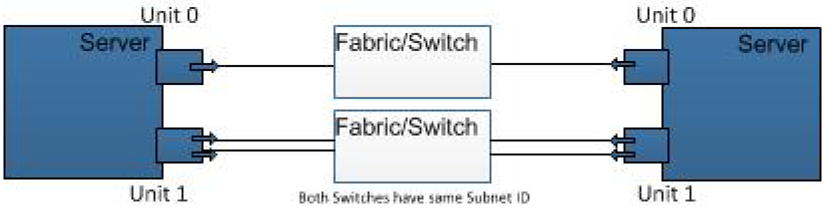
Environment Variables

- **PSM_MULTIRAIL** is not set. PSM is using single-rail, the Unit/Port/context selection is from the assignment of QIB driver. **IPATH_UNIT** and **IPATH_PORT** are set by the user to specify the Unit/Port to use.
- **PSM_MULTIRAIL** is set. PSM checks that there are two units in the system. The first available port is Port 1 for Unit 0. The next available port is Port 1 for Unit 1. PSM by default will use a PSM_MULTIRAIL_MAP of 0:1,1:1. Since this a single fabric, all of the ports have the same subnet ID. PSM sets up the first (master) connection over 0:1, and sets up the second slave connection over 1:1
- **PSM_MULTIRAIL=1** and **PSM_MULTIRAIL_MAP=1:2,0:1** The user specifies the map, how to use the Unit/Port, and PSM uses the given pairs. PSM sets up the master connection over Unit 1 Port 2 and sets up the slave connection over Unit 0 Port 1. Since Unit 1 Port 1 is available, Unit 1 Port 2 will never be selected if it is not specified in PSM_MULTIRAIL_MAP explicitly. The user can fine tune which port to use.

Example 8-2. Multi-fabrics, with same subnet ID

Figure 8-2 shows an example of multiple fabrics with the same subnet ID.

Figure 8-2. Multi-fabrics, with same subnet ID



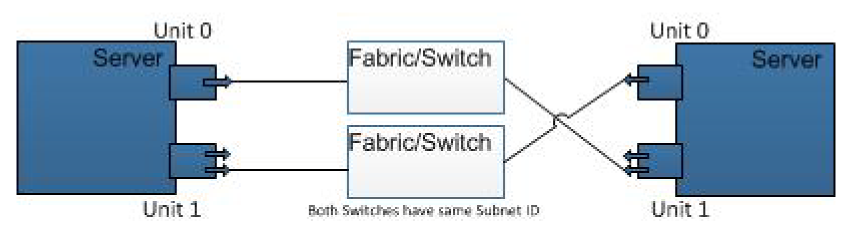
Environment Variables

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

Example 8-3. Multi-fabrics, single subnet ID, abnormal wiring.

Figure 8-3 shows an example of multiple fabrics with a single subnet ID, and abnormal wiring.

Figure 8-3. Multi-fabrics, with same subnet ID, and abnormal wiring



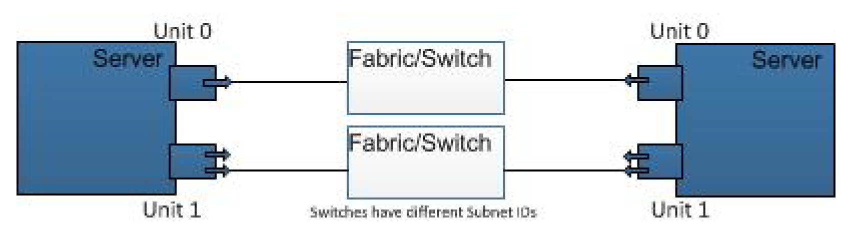
Environment Variables

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

Example 8-4. Multi-fabrics, different subnet IDs

Figure 8-4 shows an example of multiple fabrics with different subnet IDs.

Figure 8-4. Multi-fabrics, with different subnet IDs



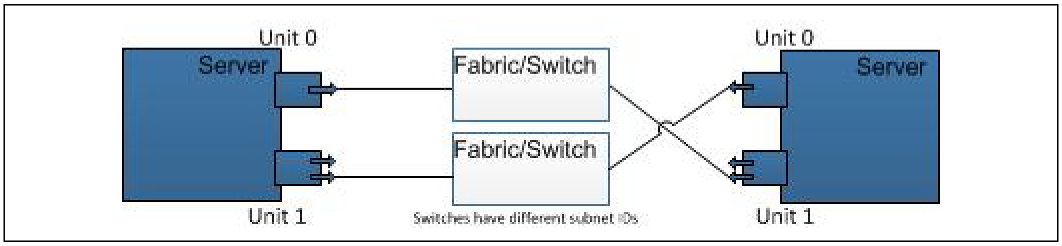
Environment Variables

- **PSM_MULTIRAIL** is not set. PSM does not work because there are multiple fabrics. Unit 0/Port 1 on first node has no connection to Unit 1/Port 1 on second node.
- **PSM_MULTIRAIL=1** automatic selection. Both nodes get Unit/Port pairs 0:1,1:2 first, after the PSM reordering based on subnet ID, the node on the left side will get 0:1,1:2 and the node on the right side gets 0:1,1:2. The PSM makes the master rail on 0:1 of left node with 0:1 on right node. The slave rail is setup on 1:2 of the left node with 1:2 of the right node. PSM works in this configuration/setting.
- **PSM_MULTIRAIL=1** and **PSM_MULTIRAIL_MAP=1:2,0:1**. The user specifies the Unit/Port pairs. PSM does not reorder them. Both nodes use 1:2 to make the connection on the second fabric as the master rail, and set up the second rail over 0:1 on both sides. PSM works fine in this configuration.

Example 8-5. Multi-fabrics, different subnet IDs, abnormal wiring.

Figure 8-5 shows an example of multiple fabrics with different subnet IDs and abnormal wiring.

Figure 8-5. Multi-fabrics, with different subnet IDs, and abnormal wiring



Environment Variables

- **PSM_MULTIRAIL** is not set. PSM does not work because there are multiple fabrics.
- **PSM_MULTIRAIL=1** automatic selection. Both nodes get Unit/Port pairs 0:1,1:2 first, after PSM reordering based on the subnet ID, the node on the left side will get 0:1,1:2 again and the node on the right side gets 1:2,0:1. The PSM makes the master rail on 0:1 of the left node with 1:2 on the right node. The slave rail is setup on 1:2 of left with 0:1 of right. PSM works in this configuration/setting.
- **PSM_MULTIRAIL=1** and **PSM_MULTIRAIL_MAP=1:2,0:1**. The user specifies the Unit/Port pairs. PSM does not reorder them. Both nodes use 1:2 to make a connection, it fails because there is no path between them. PSM does not work in this case.

§ §

9.0 Dispersive Routing

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

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

Dispersive routing, as implemented in the PSM, attempts to avoid congestion hotspots described above by “spraying” messages across these paths. A congested path will not bottleneck messages flowing down the alternate paths that are not congested. The current implementation of PSM supports fabrics with a maximum LMC of 3 (8 LIDs assigned per port). This can result in a maximum of 64 possible paths between a SLID, DLID pair ([SLID, DLID],[SLID, DLID+1], [SLID,DLID+2]…..[SLID,DLID+8],[SLID+1, DLID],[SLID+1, DLID+1]…..[SLID+7, DLID+8]). Keeping state associated with these many paths requires large amount of memory resources, with empirical data showing not much gain in performance beyond utilizing a small set of multiple paths. Therefore PSM reduces the number of paths actually used in the above case to 8 where the following paths are the only ones considered for transmission <V_Variable>— [SLID, DLID], [SLID + 1, DLID + 1], [SLID + 2, DLID + 2] ….. [SLID + N, DLID + N]. This makes the resource requirements manageable while providing most of the benefits of dispersive routing (congestion avoidance by utilizing multiple paths).

Internally, PSM utilizes dispersive routing differently for small and large messages. Large messages are any messages greater-than or equal-to 64K. For large messages, the message is split into message fragments of 128K by default (called a window). Each of these message windows is sprayed across a distinct path between ports. All packets belonging to a window utilize the same path however the windows themselves can take a different path through the fabric. PSM assembles the windows that make up an MPI message before delivering it to the application. This allows limited out of order semantics through the fabrics to be maintain with little overhead. Small messages on the other hand always utilize a single path when communicating to a remote node however different processes executing on a node can utilize different paths for their communication between the nodes. For example, two nodes A and B each with 8 processors per node. Assuming the fabric is configured for a LMC of 3, PSM constructs 8 paths through the fabric as described above and a 16 process MPI application that spans these nodes (8 process per node). Then:

- Each MPI process is automatically bound to a given CPU core numbered between 0-7. PSM does this at startup to get improved cache hit rates and other benefits.
- Small Messages sent from a process on core N will use path N.

Note: Only path N will be used by this process for all communications to any process on the remote node.

- For a large message, each process will utilize all of the 8 paths and spray the windowed messages across it.

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

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

Note: Multiple paths are still used in the fabric if multiple processes (each on a different CPU) are communicating.

- **Static_Dest**: The path selection is based on the CPU index of the destination process. Multiple paths can be used if data transfer is to different remote processes within a node. If multiple processes from Node A send a message to a single process on Node B only one path will be used across all processes.
- **Static_Base**: The only path that is used is the base path [SLID,DLID] between nodes regardless of the LMC of the fabric or the number of paths available. This is similar to how PSM operated till the IFS 5.1 release.

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

§ §

10.0 gPXE

gPXE is an open source (GPL) network bootloader. It provides a direct replacement for proprietary PXE ROMs. See http://etherboot.org/wiki/index.php for documentation and general information.

10.1 gPXE Setup

At least two machines and a switch are needed (or connect the two machines back-to-back and run FM on the server).

- A DHCP server
- A boot server or http server (can be the same as the DHCP server)
- A node to be booted
  Use a QLE7340 or QLE7342 adapter for the node.

The following software is included with the Intel OFED+ installation software package:

- gPXE boot image
- patch for DHCP server
- tool to install gPXE boot image in EPROM of card
- sample gPXE script

Everything that can be done with the proprietary PXE loader over Ethernet, can be done with the gPXE loader over IB. The gPXE boot code is only a mechanism to load an initial boot image onto the system. It is up to the downloaded boot image to do the rest.

For example, the boot image could be:

- A stand-alone memory test program
- A diskless kernel image that mounts its file systems via NFS
  Refer to http://www.faqs.org/docs/Linux-HOWTO/Diskless-HOWTO.html
- A Linux install image like kickstart, which then installs software to the local hard drive(s). Refer to http://www.faqs.org/docs/Linux-HOWTO/KickStart-HOWTO.html
- A second stage boot loader
- A live CD Linux image
- A gPXE script

10.1.1 Required Steps

1. Download a copy of the gPXE image.
   Located at:

- The executable to flash the EXPROM on the Intel HCAs is located at:
  `/usr/sbin/ipath_exprom`

- The gPXE driver for QLE7300 series HCAs (the EXPROM image) is located at: `/usr/share/infinipath/gPXE/iba7322.rom`

2. In order for dhcpd to correctly load, assign IP addresses to the HCA GUID. The dhcpd on the existing DHCP server may need to be patched. This patch will be provided via the gPXE rpm installation.
3. Write the ROM image to the HCA.

   This only needs to be done once per HCA.

```
ipath_exprom -e -w iba7xxx.rom
```

In some cases, executing the above command results in a hang. If you experience a hang, type **CTRL+C** to quit, then execute one flag at a time:

```
ipath_exprom -e iba7xxx.rom
```

```
ipath_exprom -w iba7xxx.rom
```

4. Enable booting from the HCA (gPXE device) in the BIOS

10.2 Preparing the DHCP Server in Linux

When the boot session starts, the gPXE firmware attempts to bring up an HCA network link. If it succeeds to bring up a connected link, the gPXE firmware communicates with the DHCP server. The DHCP server assigns an IP address to the gPXE client and provides it with the location of the boot program.

10.2.1 Installing DHCP

gPXE requires that the DHCP server runs on a machine that supports IP over IB.

Note: Prior to installing DHCP, make sure that Intel OFED+ is already installed on your DHCP server.

1. Download and install the latest DHCP server from www.isc.org.

   Standard DHCP fields holding MAC address are not large enough to contain an IPoIB hardware address. To overcome this problem, DHCP over IB messages convey a client identifier field used to identify the DHCP session. This client identifier field can be used to associate an IP address with a client identifier value, such that the DHCP server will grant the same IP address to any client that conveys this client identifier.
2. Unpack the latest downloaded DHCP server.

```
tar zxf dhcp-release.tar.gz
```

3. Uncomment the line `/* #define USE_SOCKETS */` in `dhcp-`*release*`/includes/site.h`
4. Change to the main directory.

```
cd dhcp-release
```

Note: If there is an older version of DHCP installed, save it before continuing with the following steps.

5. Configure the source.

```
./configure
```

6. When the configuration of DHCP is finished, build the DHCP server.

```
make
```

7. When the DHCP has successfully finished building, install DHCP.

```
make install
```

10.2.2 Configuring DHCP

1. From the client host, find the GUID of the HCA by using `p1info` or look at the GUID label on the HCA.
2. Turn the GUID into a MAC address and specify the port of the HCA that is going to be used at the end, using `b0` for port0 or `b1` for port1.

   For example for a GUID that reads `0x00117500005a6eec`, the MAC address would read: `00:11:75:00:00:5a:6e:ec:b0`
3. Add the MAC address to the DHCP server.

   The following is the sample `/etc/dhcpd.conf` file that specifies the HCA GUID for the hardware address:

```
#

# DHCP Server Configuration file.

#   see /usr/share/doc/dhcp*/dhcpd.conf.sample

#


ddns-update-style none;


subnet 10.252.252.0 netmask 255.255.255.0 {

option subnet-mask 255.255.255.0;

range dynamic-bootp 10.252.252.100 10.252.252.109;


host hl5-0 {

hardware unknown-32 00:11:75:00:00:7e:c1:b0;

option host-name "hl5";

}


host hl5-1 {

hardware unknown-32 00:11:75:00:00:7e:c1:b1;

option host-name "hl5";
}


filename "http://10.252.252.1/images/uniboot/uniboot.php";

}
```

In this example, host `hl5` has a dual port HCA. hl5-0 corresponds to port 0, and `hl5-1` corresponds to port 1 on the HCA.

4. Restart the DHCP server

10.3 Netbooting Over IB

The following procedures are an example of netbooting over IB, using an HTTP boot server.

10.3.1 Prerequisites

- Required steps from above have been executed.
- The BIOS has been configured to enable booting from the HCA. The gPXE IB device should be listed as the first boot device.
- Apache server has been configured with PHP on your network, and is configured to serve pages out of `/vault.`
- It is understood in this example that users would have their own tools and files for diskless booting with an http boot server.

Note: The dhcpd and apache configuration files referenced in this example are included as examples, and are not part of the Intel OFED+ installed software. Your site boot servers may be different, see their documentation for equivalent information.

Note: Instructions on installing and configuring a dhcp server or a boot server are beyond the scope of this document.

10.3.2 Boot Server Setup

Configure the boot server for your site.

Note: gPXE supports several file transfer methods such as TFTP, HTTP, iSCSI. This example uses HTTP since it generally scales better and is the preferred choice.

Note: This step involves setting up a http server and needs to be done by a user that understands server setup on the http server is being used

1. Install Apache.
2. Create an `images.conf` file and a `kernels.conf` file and place them in the `/etc/httpd/conf.d` directory. This sets up aliases for and tells apache where to find them:

```
/images <V_Variable>— http://<IP ADDRESS>/images/

/kernels <V_Variable>— http://<IP ADDRESS>/kernels/
```

The following is an example of the `images.conf` file

```
Alias /images /vault/images
```

```
<Directory "/vault/images">

    AllowOverride All

    Options Indexes FollowSymLinks

    Order allow,deny

    Allow from all

</Directory>
```

The following is an example of the `kernels.conf` file

```
Alias /kernels /boot



<Directory "/boot">

    AllowOverride None

    Order allow,deny

    Allow from all

</Directory>
```

3. Make a `uniboot` directory:

```
mkdir -p /vault/images/uniboot
```

4. Create a `initrd.img file`

   Prerequisites

- "gPXE Setup" on page 119 has been completed.
- "Preparing the DHCP Server in Linux" on page 120 has been completed

   To add an IB driver into the `initrd` file, The IB modules need to be copied to the diskless image. The host machine needs to be pre-installed with the Intel OFED+ Host Software that is appropriate for the kernel version the diskless image will run. The Intel OFED+ Host Software is available for download from http://downloadcenter.intel.com/

Note: The remainder of this section assumes that Intel OFED+ has been installed on the Host machine.

Warning: The following procedure modifies critical files used in the boot procedure. It must be executed by users with expertise in the boot process. Improper application of this procedure may prevent the diskless machine from booting.

   a. If `/vault/images/initrd.img` file is already present on the server machine, back it up. For example:

```
cp -a /vault/images/initrd.img /vault/images/ initrd.img.bak
```

b. The infinipath rpm will install the file `/usr/share/infinipath/gPXE/gpxe-qib-modify-initrd` with contents similar to the following example. You can either run the script to generate a new initrd image, or use it as an example, and customize as appropriate for your site.

```
# This assumes you will use the currently running version of
linux, and

# that you are starting from a fully configured machine of the same
type

# (hardware configuration), and BIOS settings.

#

# start with a known path, to get the system commands

PATH=/sbin:/usr/sbin:/bin:/usr/bin:$PATH



# start from a copy of the current initd image



mkdir -p /var/tmp/initrd-ib

cd /var/tmp/initrd-ib



kern=$(uname -r)

if [ -e /boot/initrd-${kern}.img ]; then

   initrd=/boot/initrd-${kern}.img

elif [ -e /boot/initrd ]; then

   initrd=/boot/initrd

else

   echo Unable to locate correct initrd, fix script and re-run

   exit 1

fi



cp ${initrd} initrd-ib-${kern}.img
```

```
# Get full original listing

gunzip -dc initrd-ib-${kern}.img | cpio -it --quiet | grep -v
'^\.$' | sort -o Orig-listing


# start building modified image

rm -rf new # for retries

mkdir new

cd new


# extract previous contents

gunzip -dc ../initrd-ib-${kern}.img | cpio --quiet -id


# add infiniband modules

mkdir -p lib/ib

find /lib/modules/${kern}/updates -type f | \

 egrep
'(iw_cm|ib_(mad|addr|core|sa|cm|uverbs|ucm|umad|ipoib|qib).ko|rdm
a_|ipoib_helper)' | \

 xargs -I '{}' cp -a  '{}' lib/ib


# Some distros have ipoib_helper, others don't require it

if [ -e lib/ib/ipoib_helper ]; then

   helper_cmd='/sbin/insmod /lib/ib/ipoib_helper.ko'

fi


# On some kernels, the qib driver will require the dca module

if modinfo -F depends ib_qib | grep -q dca; then

  cp $(find /lib/modules/$(uname -r) -name dca.ko) lib/ib

  dcacmd='/sbin/insmod /lib/ib/dca.ko'
```

```
else

  dcacmd=

fi



# IB requires loading an IPv6 module. If you do not have it in your
initrd, add it

if grep -q ipv6 ../Orig-listing; then

   # already added, and presumably insmod'ed, along with any
dependencies

   v6cmd=

else

   echo -e 'Adding IPv6 and related modules\n'

   cp /lib/modules/${kern}/kernel/net/ipv6/ipv6.ko lib

   IFS=' ' v6cmd='echo "Loading IPV6"

/sbin/insmod /lib/ipv6.ko'

   # Some versions of IPv6 have dependencies, add them.

   xfrm=$(modinfo -F depends ipv6)

   if [ ${xfrm} ]; then

       cp $(find /lib/modules/$(uname -r) -name ${xfrm}.ko) lib

       IFS=' ' v6cmd='/sbin/insmod /lib/'${xfrm}'.ko

'"$v6cmd"

       crypto=$(modinfo -F depends $xfrm)

       if [ ${crypto} ]; then

         cp $(find /lib/modules/$(uname -r) -name ${crypto}.ko) lib

       IFS=' ' v6cmd='/sbin/insmod /lib/'${crypto}'.ko

'"$v6cmd"

       fi

   fi

fi
```

```
# we need insmod to load the modules; if not present it, copy it

mkdir -p sbin

grep -q insmod ../Orig-listing || cp /sbin/insmod sbin



echo -e 'NOTE: you will need to config ib0 in the normal way in
your booted root

filesystem, in order to use it for NFS, etc.\n'



# Now build the commands to load the additional modules.  We add
them just after

# the last existing insmod command, so all other dependences will
be resolved

# You can change the location if desired or necessary.

# loading order is important.  You can verify the order works ahead
of time

# by running "/etc/init.d/openibd stop", and then running these
commands

# manually by cut and paste

# This will work on SLES, although different than the standard
mechanism



cat > ../init-cmds << EOF

# Start of IB module block

$v6cmd

echo "loading IB modules"

/sbin/insmod /lib/ib/ib_addr.ko

/sbin/insmod /lib/ib/ib_core.ko

/sbin/insmod /lib/ib/ib_mad.ko

/sbin/insmod /lib/ib/ib_sa.ko

/sbin/insmod /lib/ib/ib_cm.ko
```

```
/sbin/insmod /lib/ib/ib_uverbs.ko

/sbin/insmod /lib/ib/ib_ucm.ko

/sbin/insmod /lib/ib/ib_umad.ko

/sbin/insmod /lib/ib/iw_cm.ko

/sbin/insmod /lib/ib/rdma_cm.ko

/sbin/insmod /lib/ib/rdma_ucm.ko

$dcacmd

/sbin/insmod /lib/ib/ib_qib.ko

$helper_cmd

/sbin/insmod /lib/ib/ib_ipoib.ko

echo "finished loading IB modules"

# End of IB module block

EOF


# first get line number where we append (after last insmod if any,
otherwse

# at start

line=$(egrep -n insmod init | sed -n '$s/:.*//p')

if [ ! "${line}" ]; then line=1; fi

sed -e "${line}r ../init-cmds" init > init.new



# show the difference, then rename

echo -e 'Differences between original and new init command
script\n'

diff init init.new

mv init.new init

chmod 700 init
```

```
# now rebuilt the initrd image

find . | cpio --quiet -H newc -o | gzip > ../initrd-${kern}.img


cd ..


# get the file list in the new image

gunzip -dc initrd-${kern}.img | cpio --quiet -it | grep -v '^\.$'
| sort -o New-listing


# and show the differences.

echo -e '\nChanges in files in initrd image\n'

diff Orig-listing New-listing


# copy the new initrd to wherever you have configure the dhcp
server to look

# for it (here we assume it's /images)

mkdir -p /images

cp  initrd-${kern}.img /images


echo -e '\nCompleted initrd for IB'

ls -l /images/initrd-${kern}.img
```

c. Run the `usr/share/infinipath/gPXE/ gpxe-qib-modify-initrd` script to create the `initrd.img` file.

At this stage, the `initrd.img` file is ready and located at the location where the DHCP server was configured to look for it.

5. Create a `uniboot.php` file and save it to `/vault/images/uniboot.`

Note: The `uniboot.php` generates a gPXE script that will attempt to boot from the `/boot/vmlinuz-<VERSION>` kernel. If you want to boot from a different kernel, edit `uniboot.php` with the appropriate kernel string in the `$kver` variable.

The following is an example of a `uniboot.php` file:

```
<?

header ( 'Content-type: text/plain' );
```

```
function strleft ( $s1, $s2 ) {

  return substr ( $s1, 0, strpos ( $s1, $s2 ) );

}


function baseURL() {

  $s = empty ( $_SERVER["HTTPS"] ) ? '' :

    ( $_SERVER["HTTPS"] == "on" ) ? "s" : "";

  $protocol = strleft ( strtolower ( $_SERVER["SERVER_PROTOCOL"]
), "/" ).$s;

  $port = ( $_SERVER["SERVER_PORT"] == "80" ) ? "" :

    ( ":".$_SERVER["SERVER_PORT"] );

  return $protocol."://".$_SERVER['SERVER_NAME'].$port;

}


$baseurl = baseURL();

$selfurl = $baseurl.$_SERVER['REQUEST_URI'];

$dirurl = $baseurl.( dirname ( $_SERVER['SCRIPT_NAME'] ) );


$kver = "<VERSION>";


echo <<< EOF

#!gpxe

initrd /images/initrd.img

kernel /kernels/vmlinuz-${kver} bootfile=${selfurl}
ip=\${net0/ip}::\${net0/gateway}:\${net0/netmask}:\${net0/hostnam
e}:ib0:off vga=788 console=tty0 console=ttyS0,115200 debug
root=/dev/hdb1

boot

EOF;
```

```
?>
```

The generated gPXE script tells gPXE to load `/boot/vmlinuz-<VERSION>` and `/vault/images/initrd.img` files from the httpd server node and run them.

6. Copy `vmlinuz-<VERSION> to /boot` on the boot server.

   This is the kernel that will boot.

   This file can be copied from any machine that has RHEL5.3 installed.
7. Start httpd

10.3.3 Steps on the gPXE Client

1. Ensure that the HCA is listed as the first bootable device in the BIOS.
2. Reboot the test node(s) and enter the BIOS boot setup.

   This is highly dependent on the BIOS for the system but you should see a menu for boot options and a submenu for boot devices.

   Select **gPXE IB** as the first boot device.

   When you power on the system or press the reset button, the system will execute the boot code on the HCA that will query the DHCP server for the IP address and boot image to download.

   Once the boot image is downloaded, the BIOS/HCA is finished and the boot image is ready.
3. Verify system boots off of the kernel image on the boot server. The best way to do this is to boot into a different kernel from the one installed on the hard drive on the client, or to un-plug the hard drive on the client and verify that on boot up, a kernel and file system exist.

10.4 HTTP Boot Setup

gPXE supports booting diskless machines. To enable using an IB driver, the (remote) kernel or initrd image must include and be configured to load that driver. This can be achieved either by compiling the HCA driver into the kernel, or by adding the device driver module into the initrd image and loading it.

1. Make a new directory

```
mdir /vault/images/uniboot
```

2. Change directories

```
cd /vault/images/uniboot
```

3. Create a `initrd.img file` using the information and example in Step 4 of "Boot Server Setup" on page 122.
4. Create a `uniboot.php` file using the example in Step 4 of "Boot Server Setup" on page 122.
5. Create an `images.conf` file and a `kernels.conf` file using the examples in Step 2 of "Boot Server Setup" on page 122 and place them in the `/etc/httpd/conf.d` directory.
6. Edit `/etc/dhcpd.conf` file to boot the clients using HTTP

```
filename "http://172.26.32.9/images/uniboot/uniboot.php";
```

7. Restart the DHCP server

8. Start HTTP if it is not already running:

```
/etc/init.d/httpd start
```

§ §

Appendix A Benchmark Programs

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

These programs allow you to measure MPI latency, bandwidth, and message rate between two or more nodes in your cluster. The executables are installed by default under `/usr/mpi/`*`compiler`*`/`*`mpi`*`/tests/osu_benchmarks-3.1.1`. The remainder of this chapter will assume that the gcc-compiled version of Open MPI was installed in the default location of `/usr/mpi/gcc/openmpi-1.8.1-qlc` and that mpi-selector is used to choose this Open MPI version as the MPI to be used.

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 True Scale performance characteristics.

For additional MPI sample applications refer to Section 5 of the *Intel® True Scale Fabric Suite FastFabric Command Line Interface Reference Guide*.

A.1 Benchmark 1: Measuring MPI Latency Between Two Nodes

In the MPI community, *latency for a message of given size* is the time difference between a node program's 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 Ohio State University, measures the latency for a range of messages sizes from 0bytes 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-1.8.1-qlc/tests/osu_benchmarks-3.1.1/osu_lat
```

```
ency
```

`-H` (or `--hosts`) allows the specification of the host list on the command line instead of using a host file (with the `-m` or `-machinefile` option). Since only two hosts are listed, this implies that two host programs will be started (as if `-np 2` were specified). The output of the program looks like:

```
# OSU MPI Latency Test v3.1.1)

# Size          Latency (us)

0               1.67

1               1.68

2               1.69

4               1.68

8               1.68

16              1.93

32              1.92

64              1.92

128             1.99

256             2.12

512             2.38

1024            2.74

2048            3.52

4096            4.59

8192            6.52

16384           9.98

32768           17.65

65536           52.11

131072          84.07

262144          114.90

524288          241.97

1048576         422.41
```

```
2097152          783.21

4194304          1596.37
```

The first column displays the message size in bytes. The second column displays the average (one-way) latency in microseconds. This example shows the syntax of the command and the format of the output, and is not meant to represent actual values that might be obtained on any particular True Scale installation.

A.2 Benchmark 2: Measuring MPI Bandwidth Between Two Nodes

The `osu_bw` benchmark measures the maximum rate that you can pump data between two nodes. This benchmark also uses a ping-pong mechanism, similar to the `osu_latency` code, except in this case, the originator of the messages pumps a number of them (64 in the installed version) in succession using the non-blocking MPI_I send 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 acknowledgement when all of the sent data has been received.

You can run this program by typing:

```
$ mpirun -H host1,host2 \

 /usr/mpi/gcc/openmpi-1.8.1-qlc/tests/osu_benchmarks-3.1.1/osu_bw
```

Typical output might look like:

```
# OSU MPI Bandwidth Test v3.1.1

# Size        Bandwidth (MB/s)

1                        2.35

2                        4.69

4                        9.38

8                       18.80

16                      34.55

32                      68.89

64                     137.87

128                    265.80

256                    480.19

512                    843.70

1024                  1353.48

2048                  1984.11
```

```
4096                    2152.61

8192                    2249.00

16384                   2680.75

32768                   2905.83

65536                   3170.05

131072                  3224.15

262144                  3241.35

524288                  3270.21

1048576                 3286.05

2097152                 3292.64

4194304                 3283.87
```

The increase in measured bandwidth with the messages' size is because the contribution of each packet's overhead to the measured time becomes relatively smaller.

A.3 Benchmark 3: Messaging Rate Microbenchmarks

A.3.1 OSU Multiple Bandwidth / Message Rate test (`osu_mbw_mr`)

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

```
$ mpirun -H host1,host2 -npernode 12 \

/usr/mpi/gcc/openmpi-1.8.1-qlc/tests/osu_benchmarks-3.1.1/osu_mbw
_mr
```

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

```
# OSU MPI Multiple Bandwidth / Message Rate Test v3.1.1

# [ pairs: 12 ] [ window size: 64 ]

# Size                         MB/s           Messages/s
```

```
1                       22.77        22768062.43
2                       44.90        22449128.66
4                       91.75        22938300.02
8                      179.23        22403849.44
16                     279.91        17494300.07
32                     554.16        17317485.47
64                    1119.88        17498101.32
128                   1740.54        13597979.96
256                   2110.22         8243066.36
512                   2353.17         4596038.46
1024                  2495.88         2437386.38
2048                  2573.99         1256833.08
4096                  2567.88          626923.21
8192                  2757.54          336613.42
16384                 3283.94          200435.90
32768                 3291.54          100449.84
65536                 3298.20           50326.50
131072                3305.77           25221.05
262144                3310.39           12628.14
524288                3310.83            6314.90
1048576               3311.11            3157.72
2097152               3323.50            1584.77
4194304               3302.35             787.34
```

A.3.2 An Enhanced Multiple Bandwidth / Message Rate test (`mpi_multibw`)

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

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

The benchmark has been updated with code to dynamically determine what processes are on which host. Thefollowing is an example output when running `mpi_multibw`:

```
$ mpirun -H host1,host2 -npernode 12  \

  /usr/mpi/gcc/openmpi-1.8.1-qlc/tests/intel/mpi_multibw

# PathScale Modified OSU MPI Bandwidth Test

(OSU Version 2.2, PathScale $Revision: 1.1.2.1 $)

# Running on 12 procs per node (uni-directional traffic for each
process pair)


# Size          Aggregate Bandwidth (MB/s)       Messages/s

1               24.992623                        24992622.996615

2               50.015847                        25007923.312888

4               100.075479                       25018869.818990

8               200.115037                       25014379.610716

16              284.475601                       17779725.040265

32              568.950239                       17779694.953511

64              1137.899392                      17779677.998115

128             1758.183987                      13735812.394705

256             2116.159352                      8266247.468294

512             2355.027827                      4599663.724469

1024            2496.960650                      2438438.134886

2048            2574.260975                      1256963.366877

4096            2567.861960                      626919.423819

8192            2746.514440                      335267.875961

16384           3284.264487                      200455.596122

32768           3292.007839                      100464.106405
```

```
65536          3299.800622                    50350.961641

131072         3306.998105                    25230.393259

262144         3309.840069                    12626.037860

524288         3323.339300                    6338.766671

1048576        3323.068802                    3169.125368

2097152        3307.077899                    1576.937627

4194304        3300.327382                    786.859365

Searching for N/2 bandwidth.  Maximum Bandwidth of 3323.339300
MB/s...

Found N/2 bandwidth of 1662.009095 MB/s at size 121 bytes
```

Note the improved message rate at small message sizes of ~25 million compared to the rate of 22.8 million measured with `osu_mbw_mr`. Also note that it only takes a message of size 121 bytes to generate half of the peak uni-directional bandwidth.

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

```
$ mpirun -H host1,host2 -np 24  \

  /usr/mpi/gcc/openmpi-1.8.1-qlc/tests/intel/mpi_multibw -b

# PathScale Modified OSU MPI Bandwidth Test

(OSU Version 2.2, PathScale $Revision: 1.1.2.1 $)

# Running on 12 procs per node (bi-directional traffic for each
process pair)

# Size         Aggregate Bandwidth (MB/s)     Messages/s

1              34.572819                      34572819.324348

2              68.984920                      34492459.942272

4              137.870850                     34467712.532016

8              274.914966                     34364370.730843

16             438.182185                     27386386.585309

32             871.077525                     27221172.671073

64             1743.576039                    27243375.616870

128            3046.774606                    23802926.607917

256            3968.178042                    15500695.477711
```

```
512             4558.456908                     8903236.148204

1024            4876.777738                     4762478.259397

2048            5050.255245                     2465944.943769

4096            5063.142612                     1236118.801851

8192            5234.475557                     638974.066993

16384           6255.483598                     381804.418801

32768           6236.354159                     190318.425252

65536           6288.370045                     95952.912066

131072          6330.494823                     48297.842586

262144          6351.690777                     24229.777437

524288          6353.021307                     12117.426504

1048576         6353.890433                     6059.542115

2097152         6353.951840                     3029.800339

4194304         6354.671923                     1515.071851

Searching for N/2 bandwidth.  Maximum Bandwidth of 6354.671923
MB/s...

Found N/2 bandwidth of 3184.322181 MB/s at size 170 bytes
```

The higher peak bi-directional messaging rate of 34.6 million messages per second at the 1 byte size, compared to 25 million messages/sec. when run unidirectionally.

§ §

Appendix B Integration with a Batch Queuing System

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

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

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

In this Appendix there are two sections which deal with process and file clean-up after batch MPI/PSM jobs have completed: "Clean Termination of MPI Processes" and "Clean-up PSM Shared Memory Files".

B.1 Clean Termination of MPI Processes

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

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

```
# /sbin/fuser -v /dev/ipath
```

```
/dev/ipath:    22648m 22651m
```

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

```
# lsof /dev/ipath
```

This command displays a list of processes using InfiniPath. Additionally, to get all processes, including stats programs, ipath_sma, diags, and others, run the program in the following manner:

```
# /sbin/fuser -v /dev/ipath*
```

`lsof` can also take the same form:

```
# lsof /dev/ipath*
```

The following command terminates all processes using the Intel interconnect:

```
# /sbin/fuser -k /dev/ipath
```

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

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

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

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

B.2 Clean-up PSM Shared Memory Files

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

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

```
#!/bin/sh

files=`/bin/ls /dev/shm/psm_shm.* 2> /dev/null`;

for file in $files;

do

/sbin/fuser $file > /dev/null 2>&1;

if [ $? -ne 0 ];

then

/bin/rm $file > /dev/null 2>&1;

fi;

done;
```

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

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

§ §

Appendix C Troubleshooting

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

- "Using LEDs to Check the State of the HCA"
- "BIOS Settings"
- "Kernel and Initialization Issues"
- "OpenFabrics and InfiniPath Issues"
- "System Administration Troubleshooting"
- "Performance Issues"
- "Open MPI Troubleshooting"
- "HPL Residual Error Failure"

Troubleshooting information for hardware installation is found in the *Intel® True Scale Fabric Adapter Hardware Installation Guide* and software installation is found in the *Intel® True Scale Fabric Software Installation Guide*.

C.1 Using LEDs to Check the State of the HCA

The LEDs function as link and data indicators once the InfiniPath software has been installed, the driver has been loaded, and the fabric is being actively managed by a subnet manager.

Table 10-1 describes the LED states. The green LED indicates the physical link signal; the amber LED indicates the link. The green LED normally illuminates first. The normal state is *Green On*, *Amber On*. The QLE7240 and QLE7280 have an additional state, as shown in Table 10-1.

Table 10-1. LED Link and Data Indicators

LED States	Indication
Green OFF Amber OFF	The switch is not powered up. The software is neither installed nor started. Loss of signal. Verify that the software is installed and configured with `ipath_control -i`. If correct, check both cable connectors.
Green ON Amber OFF	Signal detected and the physical link is up. Ready to talk to SM to bring the link fully up. If this state persists, the SM may be missing or the link may not be configured. Use `ipath_control -i` to verify the software state. If all HCAs are in this state, then the SM is not running. Check the SM configuration, or install and run `opensmd`.

Table 10-1. LED Link and Data Indicators (Continued)

LED States	Indication
Green ON Amber ON	The link is configured, properly connected, and ready. Signal detected. Ready to talk to an SM to bring the link fully up. The link is configured. Properly connected and ready to receive data and link packets.
Green BLINKING (quickly) Amber ON	Indicates traffic
Green BLINKING† Amber BLINKING	Locates the HCA This feature is controlled by `ipath_control -b [On | Off]`

†. This feature is available only on the QLE7340, QLE7342, QLE7240 and QLE7280 adapters

C.2 BIOS Settings

This section covers issues related to BIOS settings.The most important setting is Advanced Configuration and Power Interface (ACPI). This setting must be enabled. If ACPI has been disabled, it may result in initialization problems, as described in "InfiniPath Interrupts Not Working" on page 144.

You can check and adjust the BIOS settings using the BIOS Setup utility. Check the hardware documentation that came with your system for more information.

C.3 Kernel and Initialization Issues

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

C.3.1 Driver Load Fails Due to Unsupported Kernel

If you try to load the InfiniPath driver on a kernel that InfiniPath software does not support, the load fails. Error messages similar to this display:

```
modprobe: error inserting
'/lib/modules/2.6.3-1.1659-smp/updates/kernel/drivers/infiniband/
hw/qib/ib_qib.ko': -1 Invalid module format
```

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

C.3.2 Rebuild or Reinstall Drivers if Different Kernel Installed

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

C.3.3 InfiniPath Interrupts Not Working

The InfiniPath driver cannot configure the InfiniPath link to a usable state unless interrupts are working. Check for this problem with the command:

```
$ grep ib_qib /proc/interrupts
```

Normal output is similar to this:

```
           CPU0       CPU1

185:     364263          0    IO-APIC-level   ib_qib
```

Note: The output you see may vary depending on board type, distribution, or update level.

If there is no output at all, the driver initialization failed. For more information on driver problems, see "Driver Load Fails Due to Unsupported Kernel" on page 144 or "InfiniPath ib_qib Initialization Failure" on page 147.

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

```
66: 0 0  PCI-MSI  ib_qib

185:0 0  IO-APIC-level  ib_qib
```

The following message appears when driver has initialized successfully, but no interrupts are seen within 5 seconds.

```
ib_qib 0000:82:00.0: No interrupts detected.
```

A zero count in all CPU columns means that no InfiniPath 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 `infinipath` initialization failures

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

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

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

To track down other initialization failures, see "InfiniPath ib_qib Initialization Failure" on page 147.

The program `ipath_checkout` can also help flag these kinds of problems. See "ipath_checkout" on page 184 for more information.

C.3.4 OpenFabrics Load Errors if ib_qib Driver Load Fails

When the ib_qib driver fails to load, the other OpenFabrics drivers/modules will load and be shown by `lsmod`, but commands like `ibstatus`, `ibv_devinfo`, and `ipath_control -i` will fail as follows:

```
# ibstatus

Fatal error:  device '*': sys files not found
(/sys/class/infiniband/*/ports)
```

```
# ibv_devinfo

libibverbs: Fatal: couldn’t read uverbs ABI version.

No IB devices found

# ipath_control -i

InfiniPath driver not loaded ?

No InfiniPath info available
```

C.3.5 InfiniPath ib_qib Initialization Failure

There may be cases where ib_qib 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>:ipath_userinit: assign_port command failed: Network is
down

<nodename>:can't open /dev/ipath, network down
```

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

```
MPIRUN<node_where_started>: 1 rank has not yet exited 60 seconds
after rank 0 (node <nodename>) exited without reaching
MPI_Finalize().

MPIRUN<node_where_started>:Waiting at most another 60 seconds for
the remaining ranks to do a clean shutdown before terminating 1
node processes.
```

If this error appears, check to see if the InfiniPath driver is loaded by typing:

```
$ lsmod | grep ib_qib
```

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

```
# modprobe -v ib_qib

# lsmod | grep ib_qib

# dmesg | grep -i ib_qib | tail -25
```

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

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 ib_qib
```

This command may generate more than one screen of output.

Also, check the link status with the commands:

```
$ cat /sys/class/infiniband/qib0/device/status_str
```

These commands are normally executed by the `ipathbug-helper` script, but running them separately may help locate the problem.

See also "status_str" on page 191 and "ipath_checkout" on page 184.

C.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.

C.4 OpenFabrics and InfiniPath Issues

The following sections cover issues related to OpenFabrics (including Subnet Managers) and InfiniPath.

C.4.1 Stop Infinipath Services Before Stopping/Restarting InfiniPath

The following Infinipath services must be stopped before stopping/starting/restarting InfiniPath:

- FM
- OpenSM
- SRP

Here is a sample command and the corresponding error messages:

```
# /etc/init.d/openibd stop

Unloading infiniband modules: sdp cm umad uverbs ipoib sa ipath
mad coreFATAL:Module ib_umad is in use.

Unloading infinipath modules FATAL: Module ib_qib is in use.

[FAILED]
```

C.4.2 Manual Shutdown or Restart May Hang if NFS in Use

If you are using NFS over IPoIB and use the manual `/etc/init.d/openibd stop` (or `restart`) command, the shutdown process may silently hang on the `fuser` command contained within the script. This is because fuser cannot traverse down the tree from the mount point once the mount point has disappeared. To remedy this problem, the fuser process itself needs to be killed. Run the following command either as a root user or as the user who is running the `fuser` process:

```
# kill -9 fuser
```

The shutdown will continue.

This problem is not seen if the system is rebooted or if the filesystem has already been unmounted before stopping infinipath.

C.4.3 Load and Configure IPoIB Before Loading SDP

SDP generates Connection Refused errors if it is loaded before IPoIB has been loaded and configured. To solve the problem, load and configure IPoIB first.

C.4.4 Set `$IBPATH` for OpenFabrics Scripts

The environment variable `$IBPATH` must be set to `/usr/bin`. If this has not been set, or if you have it set to a location other than the installed location, you may see error messages similar to the following when running some OpenFabrics scripts:

```
/usr/bin/ibhosts: line 30: /usr/local/bin/ibnetdiscover: No such
file or directory
```

For the OpenFabrics commands supplied with this InfiniPath release, set the variable (if it has not been set already) to `/usr/bin`, as follows:

```
$ export IBPATH=/usr/bin
```

C.4.5 SDP Module Not Loading

If the settings for debug level and the zero copy threshold from InfiniPath release 2.0 are present in the release 2.2 `/etc/modprobe.conf file` (RHEL) or `/etc/modprobe.conf.local` (SLES) file, the SDP module may not load. To solve the problem, remove the following line.

```
options ib_sdp sdp_debug_level=4
sdp_zcopy_thrsh_src_default=10000000
```

C.4.6 `ibsrpdm` Command Hangs when Two HCAs are Installed but Only Unit 1 is Connected to the Switch

If multiple HCAs (unit 0 and unit 1) are installed and only unit 1 is connected to the switch, the `ibsrpdm` command (to set up an SRP target) can hang. If unit 0 is connected and unit 1 is disconnected, the problem does not occur.

When only unit 1 is connected to the switch, use the `-d` option with `ibsrpdm`. Then, using the output from the `ibsrpdm` command, echo the new target information into `/sys/class/infiniband_srp/srp-ipath1-1/add_target`.

For example:

```
# ibsrpdm -d /dev/infiniband/umad1 -c

# echo \
id_ext=21000001ff040bf6,ioc_guid=21000001ff040bf6,dgid=fe80000000
00000021000001ff040bf6,pkey=ffff,service_id=f60b04ff01000021 >
/sys/class/infiniband_srp/srp-ipath0-1/add_target
```

C.4.7 Outdated ipath_ether Configuration Setup Generates Error

Ethernet emulation (`ipath_ether`) has been removed in this release, and, as a result, an error may be seen if the user still has an alias set previously by `modprobe.conf` (for example, `alias eth2 ipath_ether`).

When `ifconfig` or `ifup` are run, the error will look similar to the following (assuming `ipath_ether` was used for `eth2`):

```
eth2: error fetching interface information: Device not found
```

To prevent the error message, remove the following files (assuming `ipath_ether` was used for `eth2`):

`/etc/sysconfig/network-scripts/ifcfg-eth2` (for RHEL)

`/etc/sysconfig/network/ifcfg-eth2` (for SLES)

Intel recommends using the IP over IB protocol (IPoIB-CM), included in the standard OpenFabrics software releases, as a replacement for `ipath_ether`.

C.5 System Administration Troubleshooting

The following sections provide details on locating problems related to system administration.

C.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. If FastFabric is not installed in the fabric, there are two diagnostic tools, `ibhosts` and `ibtracert`, that may also be helpful. The tool `ibhosts` lists all the IB nodes that the subnet manager recognizes. To check the IB path between two nodes, use the `ibtracert` command.

C.6 Performance Issues

The following sections discuss known performance issues.

C.6.1 Large Message Receive Side Bandwidth Varies with Socket Affinity on Opteron Systems

On Opteron systems, when using the QLE7240 or QLE7280 in DDR mode, there is a receive side bandwidth bottleneck for CPUs that are not adjacent to the PCI Express root complex. This may cause performance to vary. The bottleneck is most obvious when using SendDMA with large messages on the farthest sockets. The best case for SendDMA is when both sender and receiver are on the closest sockets. Overall performance for PIO (and smaller messages) is better than with SendDMA.

C.6.2 Erratic Performance

Sometimes erratic performance is seen on applications that use interrupts. An example is inconsistent SDP latency when running a program such as `netperf`. This may be seen on AMD-based systems using the QLE7240 or QLE7280 adapters. If this happens, check to see if the program `irqbalance` is running. This program is a Linux daemon that distributes interrupts across processors. However, it may interfere with prior interrupt request (IRQ) affinity settings, introducing timing anomalies. After stopping this process (as a root user), bind IRQ to a CPU for more consistent performance. First, stop `irqbalance`:

```
# /sbin/chkconfig irqbalance off

# /etc/init.d/irqbalance stop
```

Next, find the IRQ number and bind it to a CPU. The IRQ number can be found in one of two ways, depending on the system used. Both methods are described in the following paragraphs.

Note: Take care when cutting and pasting commands from PDF documents, as quotes are special characters and may not be translated correctly.

C.6.2.1 Method 1

Check to see if the IRQ number is found in `/proc/irq/`*`xxx`*, where *`xxx`* is the IRQ number in `/sys/class/infiniband/ipath*/device/irq`. Do this as a root user. For example:

```
# my_irq=`cat /sys/class/infiniband/qib0/device/irq`

# ls /proc/irq
```

If `$my_irq` can be found under `/proc/irq/`, then type:

```
# echo 01 > /proc/irq/$my_irq/smp_affinityMethod 2
```

If command from Method 1, `ls /proc/irq`, cannot find `$my_irq`, then use the following commands instead:

```
# my_irq=`cat /proc/interrupts|grep ib_qib|awk \

’{print $1}’|sed -e ’s/://’`

# echo 01 > /proc/irq/$my_irq/smp_affinity
```

This method is not the first choice because, on some systems, there may be two rows of `ib_qib` output, and you will not know which one of the two numbers to choose. However, if you cannot find `$my_irq` listed under `/proc/irq` (Method 1), this type of system most likely has only one line for `ib_qib` listed in `/proc/interrupts`, so you can use Method 2.

Here is an example:

```
# cat /sys/class/infiniband/ipath*/device/irq

98

# ls /proc/irq
```

```
0   10    11   13   15   233   4   50   7   8   90

1   106   12   14   2    3     5   58   66   74  9
```

(Note that you cannot find 98.)

```
# cat /proc/interrupts|grep ib_qib|awk \

'{print $1}'|sed -e 's/://'

106

# echo 01 > /proc/irq/106/smp_affinity
```

Using the `echo` command immediately changes the processor affinity of an IRQ.

Note: The contents of the `smp_affinity` file may not reflect the expected values, even though the affinity change has taken place.

Note: If the driver is reloaded, the affinity assignment will revert to the default, so you will need to reset it to the desired value.

You can look at the stats in `/proc/interrupts` while the adapter is active to observe which CPU is fielding `ib_qib` interrupts.

C.6.2.2 Immediately change the processor affinity of an IRQ

To immediately change the processor affinity of an IRQ, execute a command similar to the following, as a root user:

```
echo 01 > /proc/irq/$my_irq/smp_affinity
```

The contents of the `smp_affinity` file may not reflect the expected values, even though the affinity change has taken place. If the driver is reloaded, the affinity assignment will revert to the default, so you will need to reset it to the desired value. Look at the stats in `/proc/interrupts` while the adapter is active to see which CPU is fielding `ib_qib` interrupts.

C.6.3 Performance Warning if `ib_qib` Shares Interrupts with `eth0`

When `ib_qib` shares interrupts with `eth0`, performance may be affected the OFED ULPs, such as IPoIB. A warning message appears in `syslog`, and also on the console or `tty` session where `/etc/init.d/openibd start` is run (if messages are set up to be displayed). Messages are in this form:

```
Nov  5 14:25:43 <nodename> infinipath: Shared interrupt will
affect performance: vector 169: devices eth0, ib_qib
```

Check `/proc/interrupts`: "169" is in the first column, and "devices" are shown in the last column.

You can also contact your system vendor to see if the BIOS settings can be changed to avoid the problem.

C.7 Open MPI Troubleshooting

Problems specific to compiling and running Open MPI programs are described in the following sections.

C.7.1 Invalid Configuration Warning

Open MPI warns about a invalid configuration every time it is run with the following warning:

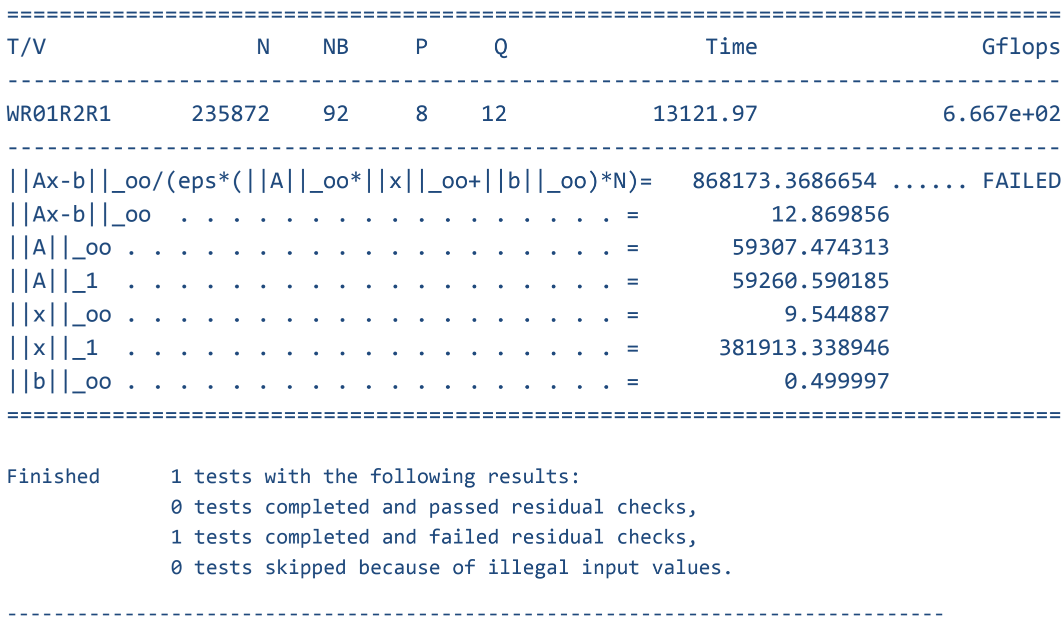
```
WARNING: There are more than one active ports on host 'st2107',
but the default subnet GID prefix was detected on more than one of
these ports.  If these ports are connected to different physical
IB networks, this configuration will fail in Open MPI.  This
version of Open MPI requires that every physically separate IB
subnet that is used between connected MPI processes must have
different subnet ID values.
```

When connecting 2 ports of an HCA to different fabrics, it is a mandatory requirement that the SubnetPrefix for those two fabrics be different and non-default (for example, not FE80000000000000) based on the FM configuration file. The `config_generate` tool for the FM will help generate such files. Refer to the *Intel® True Scale Fabric Suite Fabric Manager User Guide* for more information about the `config_generate` tool.

C.8 HPL Residual Error Failure

High Performance Linpack (HPL) running on clusters with some architectures may sometimes result in a residual error, as shown in Figure 10-1.

Figure 10-1. Screenshot of Linpack test results showing residual failure



```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R1      235872    92     8    12           13121.97              6.667e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   868173.3686654 ...... FAILED
||Ax-b||_oo  . . . . . . . . . . . . . . . . . =          12.869856
||A||_oo . . . . . . . . . . . . . . . . . . . =       59307.474313
||A||_1  . . . . . . . . . . . . . . . . . . . =       59260.590185
||x||_oo . . . . . . . . . . . . . . . . . . . =           9.544887
||x||_1  . . . . . . . . . . . . . . . . . . . =      381913.338946
||b||_oo . . . . . . . . . . . . . . . . . . . =           0.499997
================================================================================

Finished      1 tests with the following results:
              0 tests completed and passed residual checks,
              1 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.

--------------------------------------------------------------------------------
```

To avoid the residual error, edit the Kcopy configuration as noted below and copy it to each compute node.

Note: The kcopy configuration file has various paths and names in the different Linux distributions as shown in the following list:

- For SLES or RHEL use file `/etc/modprobe.d/kcopy.conf`

Change the kcopy configuration file as follows:

1. Open the configuration file noted above in edit mode.
2. Add the following option:

```
options kcopy cache_coherent=1
```

3. Save the configuration file.
4. Copy the kcopy configuration file to every compute node.
5. Reboot or reload the kcopy module on every compute node.

§ §

Appendix D Write Combining

D.1 Introduction

Write Combining improves write bandwidth to the Intel True Scale driver by writing multiple words in a single bus transaction (typically 64 bytes). Write combining applies only to x86_64 systems.

The x86 Page Attribute Table (PAT) mechanism allocates Write Combining (WC) mappings for the PIO buffers, and is the default mechanism for WC.

If PAT is unavailable or PAT initialization fails, the software will generate a message in the log and fall back to the Memory Type Range Registers (MTRR) mechanism. If write combining is not working properly, lower than expected bandwidth may occur.

The following sections provide instructions for enabling and disabling WC using PAT and MTRR, and for verifying that write combining is working.

D.2 PAT and Write Combining

The `wc_pat` parameter is set in `/etc/modprobe.conf` (on Red Hat systems) or `/etc/modprobe.conf.local` (on SLES systems) to:

- 0) Disable PAT and use MTRR.
- 1) Configure WC by programming the PAT at the memory page level instead of the physical memory ranges.
- 2) Configure WC by programming the PAT at the memory page level and overwriting the operating system PAT configuration to enable WC uniformly across CPUs that have it disabled. This is the default setting.

The default `wc_pat` parameter is:

```
option ib_qib wc_pat=2
```

If PAT is unavailable or PAT initialization fails, the code generates a message in the log and falls back to the MTRR mechanism. To use MTRR, disable PAT by setting the `wc_pat` parameter to 0 (as a root user):

```
option ib_qib wc_pat=0
```

Revert to using MTRR-only behavior by following one of the two suggestions in "MTRR Mapping and Write Combining".

The driver must be restarted after the changes have been made.

Note: There will not be a WC entry in `/proc/mtrr` when using PAT.

D.3 MTRR Mapping and Write Combining

Two suggestions for properly enabling MTRR mapping for write combining are described in the following sections.

See "Performance Issues" on page 150 for more details on a related performance issue.

D.3.1 Edit BIOS Settings to Fix MTRR Issues

You can edit the BIOS setting for MTRR mapping. The BIOS setting looks similar to:

```
MTRR Mapping                [Discrete]
```

For systems with very large amounts of memory (32GB or more), it may also be necessary to adjust the BIOS setting for the *PCI hole granularity* to 2GB. This setting allows the memory to be mapped with fewer MTRRs, so that there will be one or more unused MTRRs for the InfiniPath driver.

Some BIOS' do not have the MTRR mapping option. It may have a different name, depending on the chipset, vendor, BIOS, or other factors. For example, it is sometimes referred to as *32 bit memory hole*. This setting must be enabled.

If there is no setting for MTRR mapping or 32 bit memory hole, and you have problems with degraded performance, contact your system or motherboard vendor and ask how to enable write combining.

D.3.2 Use the `ipath_mtrr` Script to Fix MTRR Issues

Intel also provides a script, `ipath_mtrr`, which sets the MTRR registers, enabling maximum performance from the InfiniPath driver. This Python script is available as a part of the InfiniPath software download, and is contained in the `infinipath*` RPM. It is installed in `/bin`.

To diagnose the machine, run it with no arguments (as a root user):

```
# ipath_mtrr
```

The test results will list any problems, if they exist, and provide suggestions on what to do.

To fix the MTRR registers, use:

```
# ipath_mtrr -w
```

Restart the driver after fixing the registers.

This script needs to be run after each system reboot. It can be set to run automatically upon restart by adding this line in `/etc/sysconfig/infinipath`:

```
IPATH_MTRR_ACTIVE=1
```

See the `ipath_mtrr(8)` man page for more information on other options.

D.4 Verify Write Combining is Working

To see if write combining is working correctly and to check the bandwidth, run the following command:

```
$ ipath_pkt_test -B
```

With write combining enabled, the QLE7140 and QLE7240 report in the range of 1150–1500 MBps. The QLE7280 reports in the range of 1950–3000 MBps.

You can also use `ipath_checkout` (use option 5) to check bandwidth.

Increased latency and low bandwidth may indicate a problem. The interconnect could be operating in a degraded performance mode with latency increasing to several microseconds, and bandwidth decreasing to as little as 200 MBps.

Upon driver startup, you may see these errors:

```
ib_qib 0000:04:01.0: infinqib0: Performance problem: bandwidth to
PIO buffers is only 273 MiB/sec
.
.
.
```

If you do not see any of these messages on your console, but suspect this problem, check the `/var/log/messages` file. Some systems suppress driver load messages but still output them to the log file.

§ §

Appendix E Commands and Files

The most useful commands and files for debugging, and common tasks, are presented in the following sections. Many of these commands and files have been discussed elsewhere in the documentation. This information is summarized and repeated here for your convenience.

E.1 Check Cluster Homogeneity with `ipath_checkout`

Many problems can be attributed to the lack of homogeneity in the cluster environment. Use the following items as a checklist for verifying homogeneity. A difference in any one of these items in your cluster may cause problems:

- Kernels
- Distributions
- Versions of the Intel boards
- Runtime and build environments
- `.o` files from different compilers
- Libraries
- Processor/link speeds
- PIO bandwidth
- MTUs

With the exception of finding any differences between the runtime and build environments, `ipath_checkout` will pick up information on all the above items. Other programs useful for verifying homogeneity are listed in Table 10-2, "Useful Programs" on page 160. More details on `ipath_checkout` are in "`ipath_checkout`" on page 184.

E.2 Restarting InfiniPath

When the driver status appears abnormal on any node, you can try restarting (as a root user). Type:

```
# /etc/init.d/openibd restart
```

These two commands perform the same function as `restart`:

```
# /etc/init.d/openibd stop
```

```
# /etc/init.d/openibd start
```

Also check the `/var/log/messages` file for any abnormal activity.

E.3 Summary and Descriptions of Commands

Commands are summarized in Table 10-2, "Useful Programs" on page 160. Names in blue text are linked to a corresponding section that provides further details. Check the man pages for more information on the programs.

Table 10-2. Useful Programs

Program Name	Function
chkconfig	Checks the configuration state and enables/disables services, including drivers. Can be useful for checking homogeneity.
`dmesg`	Prints out bootup messages. Useful for checking for initialization problems.
`iba_opp_query`	Retrieves path records from the Distributed SA and is somewhat similar to `iba_saquery`. It is intended for testing the Distributed SA (`dist_sa`) and for verifying connectivity between nodes in the fabric.
`iba_hca_rev`	Scans the system and reports hardware and firmware information about all the HCAs in the system.
`iba_manage_switch`	Allows management of externally managed switches (including 12200, 12200-18, and HP BLc Intel 4X QDR) without the IFS software.
`iba_packet_capture`	Enables packet capture and subsequent dump to file
`ibhosts`[†]	Checks that all hosts in the fabric are up and visible to the subnet manager and to each other
`ibstatus`[†]	Checks the status of IB devices when OpenFabrics is installed
`ibtracert`[†]	Determines the path that IB packets travel between two nodes
`ibv_devinfo`[†]	Lists information about IB devices in use. Use when OpenFabrics is enabled.
`ident`[††]	Identifies RCS keyword strings in files. Can check for dates, release versions, and other identifying information.
`ipath_checkout`[†††]	A bash shell script that performs sanity testing on a cluster using Intel hardware and InfiniPath software. When the program runs without errors, the node is properly configured.
`ipath_control`[†††]	A shell script that manipulates various parameters for the InfiniPath driver. This script gathers the same information contained in `boardversion`, `status_str`, and `version`.
`ipath_mtrr`[†††]	A Python script that sets the MTRR registers.
`ipath_pkt_test`[†††]	Tests the IB link and bandwidth between two Intel HCAs, or, using an IB loopback connector, tests within a single Intel HCA
`ipathstats`[†††]	Displays driver statistics and hardware counters, including performance and "error" (including status) counters
`lsmod`	Shows status of modules in the Linux kernel. Use to check whether drivers are loaded.
`modprobe`	Adds or removes modules from the Linux kernel.
`mpirun`[†††]	A front end program that starts an MPI job on an InfiniPath cluster. Use to check the origin of the drivers.
`mpi_stress`	An MPI stress test program designed to load up an MPI interconnect with point-to-point messages while optionally checking for data integrity.
ps	Displays information on current active processes. Use to check whether all necessary processes have been started.
`rpm`	Package manager to install, query, verify, update, or erase software packages. Use to check the contents of a package.
`strings`[††††]	Prints the strings of printable characters in a file. Useful for determining contents of non-text files such as date and version.

[†] These programs are contained in the OpenFabrics `openib-diags` RPM.

[††] These programs are contained within the `rcs` RPM for your distribution.

[†††] These programs are contained in the Open `mpi-frontend` RPM.

††††. These programs are contained within the `binutils` RPM for your distribution.

E.3.1 dmesg

`dmesg` prints out bootup messages. It is useful for checking for initialization problems. You can check to see if problems were detected during the driver and Intel hardware initialization with the command:

```
$ dmesg|egrep -i infinipath|qib
```

This command may generate more than one screen of output.

E.3.2 iba_opp_query

This command retrieves path records from the Distributed SA and is somewhat similar to `iba_saquery`. It is intended for testing the Distributed SA (`intel_sa`) and for verifying connectivity between nodes in the fabric. For information on configuring and using the Distributed SA, refer to "Intel Distributed Subnet Administration" on page 27.

`iba_opp_query` does not access the SM when doing queries, it only accesses the local Distributed SA database. For that reason, the kinds of queries that can be done are much more limited than with `iba_saquery`. In particular, it can only find paths that start on the machine where the command is run. (In other words, the source LID or source GID must be on the local node.) In addition, queries must supply either a source and destination LID, or a source and destination GID. They cannot be mixed. In addition, you will usually need to provide either a SID that was specified in Distributed SA configuration file, or a pkey that matches such a SID.

E.3.2.1 Usage

```
iba_opp_query [-v level] [-hca hca] [-p port] [-s LID] [-d LID]
[-S GID] [-D GID] [-k pkey] [-i sid] [-H]
```

E.3.2.2 Options

`-v/--verbose` *level* – Debug level. Should be a number between 1 and 7. Default is 5.

`-s/--slid` *LID* – Source LID. Can be in decimal, hex (0x##) or octal (0##)

`-d/--dlid` *LID* – Destination LID. Can be in decimal, hex (0x##) or octal (0##)

`-S/--sgid` *GID* – Source GID. (Can be in GID ("0x########:0x########") or inet6 format ("##:##:##:##:##:##:##:##"))

`-D/--dgid` *GID* – Destination GID. (Can be in GID ("0x########:0x########") or inet6 format ("##:##:##:##:##:##:##:##"))

`-k/--pkey` *pkey* – Partition Key

`-i/--sid` *sid* – Service ID

`-h/--hca` *hca* – The HCA to use. (Defaults to the first HCA.) The HCA can be identified by name ("mthca0", "qib1", et cetera) or by number (1, 2, 3, et cetera).

`-p/--port` *port* – The port to use. (Defaults to the first port)

`-H/--help` – Provides this help text.

All arguments are optional, but ill-formed queries can be expected to fail. You must provide at least a pair of LIDs, or a pair of GIDs.

E.3.2.3 Sample output:

```
# iba_opp_query --slid 0x31 --dlid 0x75 --sid 0x107

Query Parameters:

    resv1                   0x0000000000000107

    dgid                    ::

    sgid                    ::

    dlid                    0x75

    slid                    0x31

    hop                     0x0

    flow                    0x0

    tclass                  0x0

    num_path                0x0

    pkey                    0x0

    qos_class               0x0

    sl                      0x0

    mtu                     0x0

    rate                    0x0

    pkt_life                0x0

    preference              0x0

    resv2                   0x0

    resv3                   0x0

Using HCA qib0

Result:

    resv1                   0x0000000000000107

    dgid                    fe80::11:7500:79:e54a

    sgid                    fe80::11:7500:79:e416

    dlid                    0x75

    slid                    0x31
    hop                     0x0
    flow                    0x0
    tclass                  0x0
    num_path                0x0
    pkey                    0xffff
    qos_class               0x0
    sl                      0x1
    mtu                     0x4
    rate                    0x6
    pkt_life                0x10
    preference              0x0
    resv2                   0x0
    resv3                   0x0
```

E.3.2.4 Explanation of Sample Output:

This is a simple query, specifying the source and destination LIDs and the desired SID. The first half of the output shows the full "query" that will be sent to the Distributed SA. Unused fields are set to zero or are blank.

In the center, the line "`Using HCA qib0`" tells us that, because we did not specify which HCA to query against, the tool chose one for us. (Normally, the user will never have to specify which HCA to use. This is only relevant in the case where a single node is connected to multiple physical IB fabrics.)

Finally, the bottom half of the output shows the result of the query. Note that, if the query had failed (because the destination does not exist or because the SID is not found in the Distributed SA) you will receive and error instead:

```
# iba_opp_query --slid 0x31 --dlid 0x75 --sid 0x108

Query Parameters:

    resv1                   0x0000000000000108
    dgid                    ::
    sgid                    ::
    dlid                    0x75
    slid                    0x31
    hop                     0x0
    flow                        0x0
    tclass                      0x0
    num_path                    0x0
    pkey                        0x0
    qos_class                   0x0
    sl                          0x0
    mtu                         0x0
    rate                        0x0
    pkt_life                    0x0
    preference                  0x0
    resv2                       0x0
    resv3                       0x0
Using HCA qib0
******
Error: Get Path returned 22 for query: Invalid argument
******
```

E.3.2.5 Examples:

Query by LID and SID:

```
iba_opp_query -s 0x31 -d 0x75 -i 0x107
iba_opp_query --slid 0x31 --dlid 0x75 --sid 0x107
```

Queries using octal or decimal numbers:

```
iba_opp_query --slid 061 --dlid 0165 --sid 0407 (using octal numbers)
iba_opp_query –slid 49 –dlid 113 –sid 263 (using decimal numbers)
```

Note that these queries are the same as the first two, only the base of the numbers has changed.

Query by LID and PKEY:

```
iba_opp_query --slid 0x31 --dlid 0x75 –pkey 0x8002
```

Query by GID:

```
iba_opp_query -S fe80::11:7500:79:e416 -D fe80::11:7500:79:e54a
```

```
--sid 0x107

iba_opp_query -S 0xfe80000000000000:0x001175000079e416 -D
0xfe80000000000000:0x001175000079e394 --sid 0x107
```

As before, these queries are identical to the first two queries – they are just using the GIDs instead of the LIDs to specify the ports involved.

E.3.3 iba_hca_rev

This command scans the system and reports hardware and firmware information about all the HCAs in the system. Running `iba_hca_rev -v`(as a root user) produces output similar to the following when run from a node on the IB fabric:

```
# iba_hca_rev -v

######################

st2092  - HCA 0a:00.0

ID: FALCON QDR

PN: MHQH29B-XTR

EC: A2

SN: MT1029X00540

V0: PCIe Gen2 x8

V1: N/A

YA: N/A

FW: 2.9.1000

Image type:       ConnectX

FW Version:       2.9.1000

Device ID:        26428

Description:     Node             Port1            Port2
Sys image

GUIDs:         0002c903000ba8e0 0002c903000ba8e1 0002c903000ba8e2
0002c903000ba8e3

MACs:                                  0002c90ba8e0     0002c90ba8e1

Board ID:          (MT_0D80120009)

VSD:

PSID:             MT_0D80120009
```

```
Firmware Configuration:

;; Generated automatically by iniprep tool on Sun Jun 05 11:50:37
IDT 2011 from ./b0_falcon.prs

;;

;; PRS  FILE FOR Eagle

;; $Id: b0_falcon.prs,v 1.18 2011-02-14 11:47:28 achiad Exp $




[PS_INFO]

Name = MHQH29B-XTR_A2

Description = ConnectX-2 VPI adapter card; dual-port; 40Gb/s QSFP;
PCIe2.0 x8 5.0GT/s; tall bracket; RoHS R6


[ADAPTER]

PSID = MT_0D80120009

pcie_gen2_speed_supported = true

adapter_dev_id = 0x673c

silicon_rev = 0xb0


gpio_mode1 = 0x0

gpio_mode0 = 0x050e070f

gpio_default_val = 0x0502010f


[HCA]

hca_header_device_id = 0x673c

hca_header_subsystem_id = 0x0017

dpdp_en = true

eth_xfi_en = true
```

```
mdio_en_port1 = 0

[IB]

phy_type_port1 = XFI

phy_type_port2 = XFI


read_cable_params_port1_en = true

read_cable_params_port2_en = true


;;Polarity

eth_tx_lane_polarity_port1=0x0

eth_tx_lane_polarity_port2=0x0

eth_rx_lane_polarity_port1=0x1

eth_rx_lane_polarity_port2=0xD


;;Lane reversal

eth_tx_lane_reversal_port1=on

eth_tx_lane_reversal_port2=on

eth_rx_lane_reversal_port1=on

eth_rx_lane_reversal_port2=on


port1_sd0_ob_preemp_pre_qdr = 0x0

port2_sd0_ob_preemp_pre_qdr = 0x0

port1_sd1_ob_preemp_pre_qdr = 0x0

port2_sd1_ob_preemp_pre_qdr = 0x0

port1_sd2_ob_preemp_pre_qdr = 0x0

port2_sd2_ob_preemp_pre_qdr = 0x0

port1_sd3_ob_preemp_pre_qdr = 0x0

port2_sd3_ob_preemp_pre_qdr = 0x0
```

```
port1_sd0_ob_preemp_post_qdr = 0x6

port2_sd0_ob_preemp_post_qdr = 0x6

port1_sd1_ob_preemp_post_qdr = 0x6

port2_sd1_ob_preemp_post_qdr = 0x6

port1_sd2_ob_preemp_post_qdr = 0x6

port2_sd2_ob_preemp_post_qdr = 0x6

port1_sd3_ob_preemp_post_qdr = 0x6

port2_sd3_ob_preemp_post_qdr = 0x6


port1_sd0_ob_preemp_main_qdr = 0x0

port2_sd0_ob_preemp_main_qdr = 0x0

port1_sd1_ob_preemp_main_qdr = 0x0

port2_sd1_ob_preemp_main_qdr = 0x0

port1_sd2_ob_preemp_main_qdr = 0x0

port2_sd2_ob_preemp_main_qdr = 0x0

port1_sd3_ob_preemp_main_qdr = 0x0

port2_sd3_ob_preemp_main_qdr = 0x0


port1_sd0_ob_preemp_msb_qdr = 0x0

port2_sd0_ob_preemp_msb_qdr = 0x0

port1_sd1_ob_preemp_msb_qdr = 0x0

port2_sd1_ob_preemp_msb_qdr = 0x0

port1_sd2_ob_preemp_msb_qdr = 0x0

port2_sd2_ob_preemp_msb_qdr = 0x0

port1_sd3_ob_preemp_msb_qdr = 0x0

port2_sd3_ob_preemp_msb_qdr = 0x0
```

```
port1_sd0_muxmain_qdr = 0x1f

port2_sd0_muxmain_qdr = 0x1f

port1_sd1_muxmain_qdr = 0x1f

port2_sd1_muxmain_qdr = 0x1f

port1_sd2_muxmain_qdr = 0x1f

port2_sd2_muxmain_qdr = 0x1f

port1_sd3_muxmain_qdr = 0x1f

port2_sd3_muxmain_qdr = 0x1f


mellanox_qdr_ib_support = true

mellanox_ddr_ib_support = true


spec1_2_ib_support = true

spec1_2_ddr_ib_support = true

spec1_2_qdr_ib_support = true


auto_qdr_tx_options = 8

auto_qdr_rx_options = 7


auto_ddr_option_0.tx_preemp_pre = 0x2

auto_ddr_option_0.tx_preemp_msb = 0x1

auto_ddr_option_0.tx_preemp_post = 0x0

auto_ddr_option_0.tx_preemp_main = 0x1b


auto_ddr_option_1.tx_preemp_pre = 0x8

auto_ddr_option_1.tx_preemp_msb = 0x0

auto_ddr_option_1.tx_preemp_post = 0x2

auto_ddr_option_1.tx_preemp_main = 0x10
```

```
auto_ddr_option_1.tx_preemp = 0x0


auto_ddr_option_2.tx_preemp_pre = 0xa

auto_ddr_option_2.tx_preemp_msb = 0x0

auto_ddr_option_2.tx_preemp_post = 0x2

auto_ddr_option_2.tx_preemp_main = 0x12

auto_ddr_option_2.tx_preemp = 0x0


auto_ddr_option_3.tx_preemp_pre = 0xf

auto_ddr_option_3.tx_preemp_msb = 0x1

auto_ddr_option_3.tx_preemp_post = 0x3

auto_ddr_option_3.tx_preemp_main = 0x1f

auto_ddr_option_3.tx_preemp = 0x2


auto_ddr_option_4.tx_preemp_pre = 0x4

auto_ddr_option_4.tx_preemp_msb = 0x1

auto_ddr_option_4.tx_preemp_post = 0x5

auto_ddr_option_4.tx_preemp_main = 0x12

auto_ddr_option_4.tx_preemp = 0x0


auto_ddr_option_5.tx_preemp_pre = 0x5

auto_ddr_option_5.tx_preemp_msb = 0x1

auto_ddr_option_5.tx_preemp_post = 0x3

auto_ddr_option_5.tx_preemp_main = 0x13

auto_ddr_option_5.tx_preemp = 0x0


auto_ddr_option_6.tx_preemp_pre = 0x3

auto_ddr_option_6.tx_preemp_msb = 0x1
```

```
auto_ddr_option_6.tx_preemp_post = 0x4

auto_ddr_option_6.tx_preemp_main = 0x1f

auto_ddr_option_6.tx_preemp = 0x0


auto_ddr_option_7.tx_preemp_pre = 0x8

auto_ddr_option_7.tx_preemp_msb = 0x1

auto_ddr_option_7.tx_preemp_post = 0x3

auto_ddr_option_7.tx_preemp_main = 0x17

auto_ddr_option_7.tx_preemp = 0x0


auto_ddr_option_8.tx_preemp_pre = 0xf

auto_ddr_option_8.tx_preemp_msb = 0x1

auto_ddr_option_8.tx_preemp_post = 0x3

auto_ddr_option_8.tx_preemp_main = 0x14

auto_ddr_option_8.tx_preemp = 0x2


auto_ddr_option_9.tx_preemp_pre = 0x8

auto_ddr_option_9.tx_preemp_msb = 0x0

auto_ddr_option_9.tx_preemp_post = 0x3

auto_ddr_option_9.tx_preemp_main = 0x17

auto_ddr_option_9.tx_preemp = 0x0


auto_ddr_option_10.tx_preemp_pre = 0x8

auto_ddr_option_10.tx_preemp_msb = 0x0

auto_ddr_option_10.tx_preemp_post = 0x3

auto_ddr_option_10.tx_preemp_main = 0x17

auto_ddr_option_10.tx_preemp = 0x0
```

```
auto_ddr_option_11.tx_preemp_pre = 0xf
auto_ddr_option_11.tx_preemp_msb = 0x0
auto_ddr_option_11.tx_preemp_post = 0x3
auto_ddr_option_11.tx_preemp_main = 0x19
auto_ddr_option_11.tx_preemp = 0x0

auto_ddr_option_12.tx_preemp_pre = 0xf
auto_ddr_option_12.tx_preemp_msb = 0x0
auto_ddr_option_12.tx_preemp_post = 0x3
auto_ddr_option_12.tx_preemp_main = 0x19
auto_ddr_option_12.tx_preemp = 0x0

auto_ddr_option_13.tx_preemp_pre = 0x0
auto_ddr_option_13.tx_preemp_msb = 0x0
auto_ddr_option_13.tx_preemp_post = 0x0
auto_ddr_option_13.tx_preemp_main = 0x5
auto_ddr_option_13.tx_preemp = 0x0

auto_ddr_option_14.tx_preemp_pre = 0x0
auto_ddr_option_14.tx_preemp_msb = 0x0
auto_ddr_option_14.tx_preemp_post = 0x0
auto_ddr_option_14.tx_preemp_main = 0x5
auto_ddr_option_14.tx_preemp = 0x0

auto_ddr_option_15.tx_preemp_pre = 0x0
auto_ddr_option_15.tx_preemp_msb = 0x0
auto_ddr_option_15.tx_preemp_post = 0x0
auto_ddr_option_15.tx_preemp_main = 0x5
```

```
auto_ddr_option_15.tx_preemp = 0x0


;;;;; Integer parameter. Values range : 0x0 - 0xf.




auto_ddr_option_0.rx_offs_lowpass_en = 0x0

auto_ddr_option_1.rx_offs_lowpass_en = 0x0

auto_ddr_option_2.rx_offs_lowpass_en = 0x0

auto_ddr_option_3.rx_offs_lowpass_en = 0x0

auto_ddr_option_4.rx_offs_lowpass_en = 0x0

auto_ddr_option_5.rx_offs_lowpass_en = 0x0

auto_ddr_option_6.rx_offs_lowpass_en = 0x0

auto_ddr_option_7.rx_offs_lowpass_en = 0x0

auto_ddr_option_0.rx_offs = 0x0

auto_ddr_option_1.rx_offs = 0x0

auto_ddr_option_2.rx_offs = 0x0

auto_ddr_option_3.rx_offs = 0x0

auto_ddr_option_4.rx_offs = 0x0

auto_ddr_option_5.rx_offs = 0x0

auto_ddr_option_6.rx_offs = 0x0

auto_ddr_option_7.rx_offs = 0x0

auto_ddr_option_0.rx_equal_offs = 0x0

auto_ddr_option_1.rx_equal_offs = 0x0

auto_ddr_option_2.rx_equal_offs = 0x0

auto_ddr_option_3.rx_equal_offs = 0x0

auto_ddr_option_4.rx_equal_offs = 0x0

auto_ddr_option_5.rx_equal_offs = 0x0
```

```
auto_ddr_option_6.rx_equal_offs = 0x0

auto_ddr_option_7.rx_equal_offs = 0x0

auto_ddr_option_0.rx_muxeq = 0x0

auto_ddr_option_1.rx_muxeq = 0x0

auto_ddr_option_2.rx_muxeq = 0x0

auto_ddr_option_3.rx_muxeq = 0x0

auto_ddr_option_4.rx_muxeq = 0x0

auto_ddr_option_5.rx_muxeq = 0x0

auto_ddr_option_6.rx_muxeq = 0x0

auto_ddr_option_7.rx_muxeq = 0x0

auto_ddr_option_0.rx_muxmain = 0x1f

auto_ddr_option_1.rx_muxmain = 0x1f

auto_ddr_option_2.rx_muxmain = 0x1f

auto_ddr_option_3.rx_muxmain = 0x1f

auto_ddr_option_4.rx_muxmain = 0x1f

auto_ddr_option_5.rx_muxmain = 0x1f

auto_ddr_option_6.rx_muxmain = 0x1f

auto_ddr_option_7.rx_muxmain = 0x1f

auto_ddr_option_0.rx_main = 0x1

auto_ddr_option_1.rx_main = 0xf

auto_ddr_option_2.rx_main = 0xf

auto_ddr_option_3.rx_main = 0xf

auto_ddr_option_4.rx_main = 0xe

auto_ddr_option_5.rx_main = 0xe

auto_ddr_option_6.rx_main = 0xf

auto_ddr_option_7.rx_main = 0xf

auto_ddr_option_0.rx_extra_hs_gain = 0x0

auto_ddr_option_1.rx_extra_hs_gain = 0x3
```

```
auto_ddr_option_2.rx_extra_hs_gain = 0x2

auto_ddr_option_3.rx_extra_hs_gain = 0x4

auto_ddr_option_4.rx_extra_hs_gain = 0x1

auto_ddr_option_5.rx_extra_hs_gain = 0x2

auto_ddr_option_6.rx_extra_hs_gain = 0x7

auto_ddr_option_7.rx_extra_hs_gain = 0x0

auto_ddr_option_0.rx_sigdet_th = 0x1

auto_ddr_option_1.rx_sigdet_th = 0x1

auto_ddr_option_2.rx_sigdet_th = 0x1

auto_ddr_option_3.rx_sigdet_th = 0x1

auto_ddr_option_4.rx_sigdet_th = 0x1

auto_ddr_option_5.rx_sigdet_th = 0x1

auto_ddr_option_6.rx_sigdet_th = 0x1

auto_ddr_option_7.rx_sigdet_th = 0x1

auto_ddr_option_0.rx_equalization = 0x4

auto_ddr_option_1.rx_equalization = 0x0

auto_ddr_option_2.rx_equalization = 0x0

auto_ddr_option_3.rx_equalization = 0x0

auto_ddr_option_4.rx_equalization = 0x0

auto_ddr_option_5.rx_equalization = 0x0

auto_ddr_option_6.rx_equalization = 0x0

auto_ddr_option_7.rx_equalization = 0x0


auto_ddr_option_9.rx_muxeq = 0x0

auto_ddr_option_9.rx_muxmain = 0x1f

auto_ddr_option_9.rx_main = 0xf

auto_ddr_option_9.rx_extra_hs_gain = 0x0

auto_ddr_option_9.rx_equalization = 0x0
```

```
auto_ddr_option_10.rx_muxeq = 0x0

auto_ddr_option_10.rx_muxmain = 0x1f

auto_ddr_option_10.rx_main = 0xf

auto_ddr_option_10.rx_extra_hs_gain = 0x0

auto_ddr_option_10.rx_equalization = 0x0


auto_ddr_option_11.rx_muxeq = 0x04

auto_ddr_option_11.rx_muxmain = 0x1f

auto_ddr_option_11.rx_main = 0xf

auto_ddr_option_11.rx_extra_hs_gain = 0x4

auto_ddr_option_11.rx_equalization = 0x7f


auto_ddr_option_12.rx_muxeq = 0x6

auto_ddr_option_12.rx_muxmain = 0x1f

auto_ddr_option_12.rx_main = 0xf

auto_ddr_option_12.rx_extra_hs_gain = 0x4

auto_ddr_option_12.rx_equalization = 0x7f


auto_ddr_option_13.rx_muxeq = 0x0

auto_ddr_option_13.rx_muxmain = 0x1f

auto_ddr_option_13.rx_main = 0xf

auto_ddr_option_13.rx_extra_hs_gain = 0x3

auto_ddr_option_13.rx_equalization = 0x0


auto_ddr_option_14.rx_muxeq = 0x0

auto_ddr_option_14.rx_muxmain = 0x1f

auto_ddr_option_14.rx_main = 0xf
```

```
auto_ddr_option_14.rx_extra_hs_gain = 0x3
auto_ddr_option_14.rx_equalization = 0x0

auto_ddr_option_15.rx_muxeq = 0x0
auto_ddr_option_15.rx_muxmain = 0x1f
auto_ddr_option_15.rx_main = 0xf
auto_ddr_option_15.rx_extra_hs_gain = 0x3
auto_ddr_option_15.rx_equalization = 0x0

center_mix90phase = true
auto_kr_option_6.rx_extra_hs_gain = 0x3

ext_phy_board_port1 = FALCON
ext_phy_board_port2 = FALCON

[PLL]
lbist_en  = 0
lbist_shift_freq  = 3
pll_stabilize = 0x13
flash_div = 0x3
lbist_array_bypass = 1
lbist_pat_cnt_lsb = 0x2
core_f = 44
core_r = 27
ddr_6_db_preemp_pre = 0x3
ddr_6_db_preemp_main = 0xe

[FW]
```

```
Firmware Verification:


     FS2 failsafe image. Start address: 0x0. Chunk size 0x80000:


     NOTE: The addresses below are contiguous logical addresses.
Physical addresses on

          flash may be different, based on the image start address
and chunk size


     /0x00000038-0x00001233 (0x0011fc)/ (BOOT2) - OK

     /0x00001234-0x0000280f (0x0015dc)/ (BOOT2) - OK

     /0x00002810-0x000034ef (0x000ce0)/ (Configuration) - OK

     /0x000034f0-0x00003533 (0x000044)/ (GUID) - OK

     /0x00003534-0x0000366b (0x000138)/ (Image Info) - OK

     /0x0000366c-0x0000946f (0x005e04)/ (DDR) - OK

     /0x00009470-0x0000ab53 (0x0016e4)/ (DDR) - OK

     /0x0000ab54-0x00016b43 (0x00bff0)/ (DDR) - OK

     /0x00016b44-0x0001fb57 (0x009014)/ (DDR) - OK

     /0x0001fb58-0x000720ab (0x052554)/ (DDR) - OK

     /0x000720ac-0x0007308f (0x000fe4)/ (DDR) - OK

     /0x00073090-0x00099787 (0x0266f8)/ (DDR) - OK

     /0x00099788-0x0009d11f (0x003998)/ (DDR) - OK

     /0x0009d120-0x000a0b8b (0x003a6c)/ (DDR) - OK

     /0x000a0b8c-0x000a1037 (0x0004ac)/ (Configuration) - OK

     /0x000a1038-0x000a1093 (0x00005c)/ (Jump addresses) - OK

     /0x000a1094-0x000a1707 (0x000674)/ (FW Configuration) - OK

     /0x00000000-0x000a1707 (0x0a1708)/ (Full Image) - OK
```

```
FW image verification succeeded. Image is bootable.


######################
```

E.3.4 `iba_manage_switch`

(**Switch**) Allows management of externally managed switches (including 12200, 12200-18, and HP BLc Intel 4X QDR) without using the IFS software. It is designed to operate on one switch at a time, taking a mandatory target GUID parameter.

E.3.4.1 Usage

```
iba_manage_switch -t target-guid [-H] [-v] [-h hca] [-p port] [
-x] [-S] [ -f fileName] [-r] [-C configOption] [-i integer-value]
[-s string-value] [-c captureFile] operation
```

E.3.4.2 Options

`-H` – help (this message)

`-v` – verbose - additional output

`-t` *`target-guid`* – guid of target switch in hex format, for example `0x00066a00e3001234`

`-h` *`hca`* – HCA number, default is first HCA

`-p` *`port`* – port number, default is first active port

`-x` – clobber previous results file

`-S` – enforce password, will be prompted for each subcommand

`-f` *`fileName`* – fileName of the `emfw` file to be used in `fwUpdate` operation - must be a valid `emfw` file with `.emfw` suffix

`-r` – reset switch after `fwUpdate` (only valid with `fwUpdate` operation)

`-C` *`configOption`* – configuration option for `setConfigValue` operation
  `mtucap` (mtu capability) – use `-i` for integer value (4-2048, 5-4096)
  `vlcap` (vl capability) – use `-i` for integer value (1=1VL, 2=2VLs, 3=4VLs, 4=8VLs, 5=15VLs)
  `linkwidth` (link width supported) – use `-i` for integer value (1=1X, 2=4X, 3=1X/4X, 4=8X, 5=1X/8X, 6=4X/8X, 7=1X/4X/8X)
  `vlcreditdist` (VL credit distribution) – use `-i` for integer value (0, 1, 2, 3, or 4)
  `linkspeed` (link speed supported) – use `-i` for integer value (1=SDR, 2=DDR, 3=SDR/DDR, 4=QDR, 7=SDR/DDR/QDR)

`-i` *`integer-value`* – integer value

`-s` *`string-value`* – string value

`-c` *`captureFile`* – filename of capture output file *`operation`* – operation to perform:

- `fwUpdate` – perform firmware update using `fileName` parameter, must be an `emfw` file
- `fwVerify` – perform firmware validation, validate firmware in primary/secondary EEPROMs, report which was booted
- `ping` – test for switch presence
- `reboot` – reboot switch
- `setConfigValue` – update configuration value, use `–C` for configuration option and `–i` for integer value
- `setIBNodeDesc` – set the IB node description, use `–s` for string value of node desc
- `setPassword` – set the vendor key (prompts for password to be used for subsequent switch access)
- `showConfig` – report user-configurable settings
- `showFwVersion` – report firmware version running on switch
- `showPowerCooling` – report status of power supplies and fans
- `capture` – perform capture of switch
- `showVPD` – report VPD information of switch

E.3.4.3 Example

```
iba_manage_switch -t 0x00066a00e3001234 -f
  Intel_12000_V1_firmware.7.0.0.0.27.emfw fwUpdate

iba_manage_switch -t 0x00066a00e3001234 reboot

iba_manage_switch -t 0x00066a00e3001234 showFwVersion

iba_manage_switch -t 0x00066a00e3001234 -s i12k1234
  setIBNodeDesc

iba_manage_switch -t 0x00066a00e3001234 -C mtucap -i 4
  setConfigValue

iba_manage_switch -H
```

The results are recorded in `iba_manage_switch.res` file in the current directory. Use the `–x` option to clobber and create a new file.

E.3.5 iba_packet_capture

This tool operates in cooperation with IB snoop device in the QIB driver. It enables packet capture and subsequent dump to file.

The `snoop_enable` variable must be set to 1 (enabled) in the `modprobe.conf /ib_qib.conf` file to create snoop devices and capture devices. If `snoop_enable` is set to 0 (disable) then no snoop and capture devices are created.

This tool captures packets in memory in a large ring buffer and dumps the packet information to a file when it is instructed by a set option. The tool supports filtering of several IB fields. The tool is primarily intended for internal Intel use.

Edit the file `/etc/modprobe.d/ib_qib.conf` to add `snoop_enable=1` to the options line. If the file does not exist, it can be created with the following text:

options `ib_qib snoop_enable=1`

E.3.5.1 Usage

```
iba_packet_capture [-o outfile] [-d devfile] [-f filterfile] [-a
alarm] [-s maxblocks] [-v]
```

E.3.5.2 Options

`-o` *`outfile`* – output file for captured packets – default is packetDump.pcap

`-d` *`devfile`* – snoop device file for capturing packets – default is /dev/ipath_capture_00_01

`-f` *`filterfile`* – filter file used for filtering – if absent, no filtering

`-a` *`alarm`* – number of seconds for alarm trigger to dump capture and exit

`-s` *`maxblocks`* – max 64 byte blocks of data to capture in units of Mi (1024*1024)

`-v` – verbose output

To stop capture and trigger dump, kill with SIGINT (Ctrl-C) or SIGUSR1 (with the kill command). The program will dump packets to file and exit

A sample filter file is located at `/opt/iba/samples/filterFile.txt`. This file should be copied to the user's home directory for editing and used with the packet capture utility.

E.3.6 `ibhosts`

This tool determines if all the hosts in your IB fabric are up and visible to the subnet manager and to each other. It is installed from the `openib-diag` RPM. Running `ibhosts` (as a root user) produces output similar to the following when run from a node on the IB fabric:

```
# ibhosts

Ca : 0x0008f10001280000 ports 2 "Voltaire InfiniBand
Fiber-Channel Router"

Ca : 0x0011750000ff9869 ports 1 "idev-11"

Ca : 0x0011750000ff9878 ports 1 "idev-05"

Ca : 0x0011750000ff985c ports 1 "idev-06"

Ca : 0x0011750000ff9873 ports 1 "idev-04"
```

E.3.7 `ibstatus`

This program displays basic information on the status of IB devices that are currently in use when OpenFabrics RPMs are installed. It is installed from the `openib-diag` RPM.

Following is a sample output for the DDR HCAs:

```
# ibstatus

Infiniband device 'qib0' port 1 status:
        default gid:      fe80:0000:0000:0000:0011:7500:0078:a5d2

        base lid:         0x1

        sm lid:           0x4

        state:            4: ACTIVE

        phys state:       5: LinkUp

        rate:             40 Gb/sec (4X QDR)

        link_layer:       InfiniBand
```

E.3.8 ibtracert

The tool `ibtracert` determines the path that IB packets travel between two nodes. It is installed from the `openib-diag` RPM. The IB LIDs of the two nodes in this example are determined by using the `ipath_control -i` command on each node. The `ibtracert` tool produces output similar to the following when run (as a root user) from a node on the IB fabric:

```
# ibtracert 0xb9 0x9a

   From ca {0x0011750000ff9886} portnum 1 lid 0xb9-0xb9 "iqa-37"

   [1] -> switch port {0x0002c9010a19bea0}[1] lid 0x14-0x14
"MT47396 Infiniscale-III"

   [24] -> switch port {0x00066a0007000333}[8] lid 0xc-0xc
"SilverStorm 9120 GUID=0x00066a000200016c Leaf 6, Chip A"

   [6] -> switch port {0x0002c90000000000}[15] lid 0x9-0x9
"MT47396 Infiniscale-III"

   [7] -> ca port {0x0011750000ff9878}[1] lid 0x9a-0x9a "idev-05"

   To ca {0x0011750000ff9878} portnum 1 lid 0x9a-0x9a "idev-05"
```

E.3.9 ibv_devinfo

This program displays information about IB devices, including various kinds of identification and status data. It is installed from the `openib-diag` RPM. Use this program when OpenFabrics is enabled. `ibv_devinfo` queries RDMA devices. Use the `-v` option to see more information. For example:

```
# ibv_devinfo

hca_id: qib0

        transport:                      InfiniBand (0)

        fw_ver:                         0.0.0

        node_guid:                      0011:7500:0078:a5d2
        sys_image_guid:                 0011:7500:0078:a5d2
        vendor_id:                      0x1175
        vendor_part_id:                 29474
        hw_ver:                         0x2
        board_id:                       InfiniPath_QLE7340
        phys_port_cnt:                  1
                port:   1
                        state:                  PORT_ACTIVE (4)
                        max_mtu:                4096 (5)
                        active_mtu:             4096 (5)
                        sm_lid:                 4
                        port_lid:               1
                        port_lmc:               0x00
                        link_layer:             IB
```

E.3.10 `ident`

The `ident` strings are available in `ib_qib.ko`. Running `ident` provides driver information similar to the following. For Intel RPMs on a SLES distribution, it will look like the following example:

ident/lib/modules/*OS_version*/updates/kernel/drivers/infiniband/hw/qib/ib_qib.ko

```
/lib/modules/OS_version/updates/kernel/drivers/infiniband/hw/qib/
ib_qib.ko:

$Id: Intel OFED Release x.x.x $
$Date: yyyy-mm-dd-hh:mm $
```

Note: For Intel RPMs on a RHEL distribution, the drivers folder is in the updates folder instead of the kernels folder as follows:

```
/lib/modules/OS_version/updates/drivers/infiniband/hw/qib/ib_qib.
ko
```

If the `/lib/modules/`*`OS_version`*`/updates` directory is not present, then the driver in use is the one that comes with the core kernel. In this case, either the `kernel-ib` RPM is not installed or it is not configured for the current running kernel.

If the `updates` directory is present, but empty except for the subdirectory `kernel`, then an OFED install is probably being used, and the `ident` string will be empty. For example:

```
$ cd /lib/modules/OS_version/updates

$ ls

kernel

$ cd kernel/drivers/infiniband/hw/qib/
lib/modules/2.6.18-8.el5/updates/kernel/drivers/infiniband/hw/qib

$ ident ib_qib.ko

        ib_qib.ko:

        ident warning: no id keywords in ib_qib.ko
```

Note: `ident` is in the optional `rcs` RPM, and is not always installed.

E.3.11 ipath_checkout

The `ipath_checkout` tool is a `bash` script that verifies that the installation is correct and that all the nodes of the network are functioning and mutually connected by the InfiniPath fabric. It is installed from the `infinipath` RPM. It must be run on a front end node, and requires specification of a nodefile. For example:

```
$ ipath_checkout [options] nodefile
```

The `nodefile` lists the hostnames of the nodes of the cluster, one hostname per line. The format of `nodefile` is as follows:

```
hostname1

hostname2

...
```

Note: The hostnames in the `nodefile` are Ethernet hostnames, not IPv4 addresses.

Note: To create a `nodefile`, use the `ibhosts` program. It will generate a list of available nodes that are already connected to the switch.

`ipath_checkout` performs the following seven tests on the cluster:

1. Executes the `ping` command to all nodes to verify that they all are reachable from the front end.
2. Executes the `ssh` command to each node to verify correct configuration of `ssh`.
3. Gathers and analyzes system configuration from the nodes.
4. Gathers and analyzes RPMs installed on the nodes.
5. Verifies InfiniPath hardware and software status and configuration, including tests for link speed, PIO bandwidth (incorrect MTRR settings), and MTU size.
6. Verifies the ability to `mpirun` jobs on the nodes.
7. Runs a bandwidth and latency test on every pair of nodes and analyzes the results.

The options available with `ipath_checkout` are shown in Table E.3.11.1.

E.3.11.1 Options

`-h, --help` – These options display help messages describing how a command is used.

`-v, --verbose`
`-vv, --vverbose`
`-vvv, --vvverbose` – These options specify three successively higher levels of detail in reporting test results. There are four levels of detail in all, including the case where none of these options are given.

`-c, --continue` – When this option is not specified, the test terminates when any test fails. When specified, the tests continue after a failure, with failing nodes excluded from subsequent tests.

`-k, --keep` – This option keeps intermediate files that were created while performing tests and compiling reports. Results are saved in a directory created by mktemp and named infinipath_XXXXXX or in the directory name given to --workdir.

`--workdir=DIR` – Use DIR to hold intermediate files created while running tests. DIR must not already exist.

`--run=LIST` – This option runs only the tests in LIST. See the seven tests listed previously. For example, `--run=123` will run only tests 1, 2, and 3.

`--skip=LIST` – This option skips the tests in LIST. See the seven tests listed previously. For example, `--skip=2457` will skip tests 2, 4, 5, and 7.

`-d, --debug` – This option turns on the `-x` and `-v` flags in bash(1).

In most cases of failure, the script suggests recommended actions. Also refer to the ipath_checkout man page.

E.3.12 `ipath_control`

The `ipath_control` tool is a shell script that manipulates various parameters for the InfiniPath driver. It is installed from the `infinipath` RPM. Many of the parameters are used only when diagnosing problems, and may require special system configurations. Using these options may require restarting the driver or utility programs to recover from incorrect parameters.

Most of the functionality is accessed via the `/sys` filesystem. This shell script gathers the same information contained in these files:

```
/sys/class/infiniband/qib0/device/boardversion

/sys/class/infiniband/qib0/ports/1/linkcontrol/status_str

/sys/class/infiniband/qib0/device/driver/version
```

These files are also documented in Table 10-4, “Useful Files” on page 191 and Table 10-5, “`status_str` File Contents” on page 192.

Other than the `-i` option, this script must be run with root permissions. See the man pages for `ipath_control` for more details.

Here is sample usage and output:

```
% ipath_control -i

$Id: Intel OFED Release x.x.x $ $Date: yyyy-mm-dd-hh:mm $

0: Version: ChipABI 2.0, InfiniPath_QLE7342, InfiniPath1 6.1, SW
Compat 2

0: Serial: RIB0941C00005 LocalBus: PCIe,5000MHz,x8

0,1: Status: 0xe1 Initted Present IB_link_up IB_configured

0,1: LID=0x1 GUID=0011:7500:0079:e574

0,1: HRTBT:Auto LINK:40 Gb/sec (4X QDR)

0,2: Status: 0x21 Initted Present [IB link not Active]

0,2: LID=0xffff GUID=0011:7500:0079:e575
```

The `-i` option combined with the `-v` option is very useful for looking at the IB width/rate and PCIe lanes/rate. For example:

```
% ipath_control -iv

$Id: Intel OFED Release x.x.x $ $Date: yyyy-mm-dd-hh:mm $

0: Version: ChipABI 2.0, InfiniPath_QLE7342, InfiniPath1 6.1, SW
Compat 2

0: Serial: RIB0941C00005 LocalBus: PCIe,5000MHz,x8

0,1: Status: 0xe1 Initted Present IB_link_up IB_configured

0,1: LID=0x1 GUID=0011:7500:0079:e574

0,1: HRTBT:Auto LINK:40 Gb/sec (4X QDR)

0,2: Status: 0x21 Initted Present [IB link not Active]

0,2: LID=0xffff GUID=0011:7500:0079:e575

0,2: HRTBT:Auto LINK:10 Gb/sec (4X)
```

Note: On the first line, `Release` *version* refers to the current software release. The second line contains chip architecture version information.

E.3.13 `ipath_mtrr`

Note: Use `ipath_mtrr` if you are not using the default PAT mechanism to enable write combining.

MTRR is used by the InfiniPath driver to enable write combining to the Intel on-chip transmit buffers. This option improves write bandwidth to the Intel chip by writing multiple words in a single bus transaction (typically 64 bytes). This option applies only to x86_64 systems. It can often be set in the BIOS.

However, some BIOS' do not have the MTRR mapping option. It may have a different name, depending on the chipset, vendor, BIOS, or other factors. For example, it is sometimes referred to as *32 bit memory hole.* This setting must be enabled.

If there is no setting for MTRR mapping or 32 bit memory hole, contact your system or motherboard vendor and ask how to enable write combining.

You can check and adjust these BIOS settings using the BIOS Setup utility. For specific instructions, follow the hardware documentation that came with your system.

Intel also provides a script, `ipath_mtrr`, which sets the MTRR registers, enabling maximum performance from the InfiniPath driver. This Python script is available as a part of the InfiniPath software download, and is contained in the `infinipath*` RPM. It is installed in `/bin`.

To diagnose the machine, run it with no arguments (as a root user):

```
# ipath_mtrr
```

The test results will list any problems, if they exist, and provide suggestions on what to do.

To fix the MTRR registers, use:

```
# ipath_mtrr -w
```

Restart the driver after fixing the registers.

This script needs to be run after each system reboot. It can be set to run automatically upon restart by adding this line in `/etc/sysconfig/infinipath`:

```
IPATH_MTRR_ACTIVE=1
```

See the `ipath_mtrr(8)` man page for more information on other options.

E.3.14 ipath_pkt_test

This program is installed from the `infinipath` RPM. Use `ipath_pkt_test` to do one of the following:

- Test the IB link and bandwidth between two InfiniPath HCAs.
- Using an IB loopback connector, test the link and bandwidth within a single InfiniPath HCA.

The `ipath_pkt_test` program runs in either ping-pong mode (send a packet, wait for a reply, repeat) or in stream mode (send packets as quickly as possible, receive responses as they come back).

Upon completion, the sending side prints statistics on the packet bandwidth, showing both the payload bandwidth and the total bandwidth (including IB and InfiniPath headers). See the man page for more information.

E.3.15 ipathstats

The `ipathstats` program is useful for diagnosing InfiniPath problems, particularly those that are performance related. It is installed from the `infinipath` RPM. It displays both driver statistics and hardware counters, including both performance and "error" (including status) counters.

Running `ipathstats -c 10`, for example, displays the number of packets and 32-bit words of data being transferred on a node in each 10-second interval. This output may show differences in traffic patterns on different nodes, or at different stages of execution. See the man page for more information.

E.3.16 lsmod

When you need to find which InfiniPath and OpenFabrics modules are running, type the following command:

```
# lsmod | egrep ’ib_|rdma_|findex’
```

E.3.17 modprobe

Use this program to load/unload the drivers. You can check to see if the driver has loaded by using this command:

```
# modprobe -v ib_qib
```

The `-v` option typically only prints messages if there are problems.

The configuration file that `modprobe` uses is `/etc/modprobe.conf` (`/etc/modprobe.conf.local` on SLES). In this file, various options and naming aliases can be set.

E.3.18 mpirun

`mpirun` determines whether the program is being run against a Intel or non-Intel driver. It is installed from the `mpi-frontend` RPM. Sample commands and results are shown in the following paragraphs.

Intel-built:

```
$ mpirun -np 2 -m /tmp/id1 -d0x101 mpi_latency 1 0
asus-01:0.ipath_setaffinity: Set CPU affinity to 1, port 0:2:0
(1 active chips)
asus-01:0.ipath_userinit: Driver is Intel-built
```

Non-Intel built:

```
$ mpirun -np 2 -m /tmp/id1 -d0x101 mpi_latency 1 0
asus-01:0.ipath_setaffinity: Set CPU affinity to 1, port 0:2:0
(1 active chips)
asus-01:0.ipath_userinit: Driver is not Intel-built
```

E.3.19 mpi_stress

This is an MPI stress test program designed to load up an MPI interconnect with point-to-point messages while optionally checking for data integrity. By default, it runs with all-to-all traffic patterns, optionally including oneself and one’s local shared memory (shm) peers. It can also be set up with multi-dimensional grid traffic patterns; this can be parameterized to run rings, open 2D grids, closed 2D grids, cubic lattices, hypercubes, and so on.

Optionally, the message data can be randomized and checked using CRC checksums (strong but slow) or XOR checksums (weak but fast). The communication kernel is built out of non-blocking point-to-point calls to load up the interconnect. The program is not designed to exhaustively test out different MPI primitives. Performance metrics are displayed, but should be carefully interpreted in terms of the features enabled.

This is an MPI application and should be run under `mpirun` or its equivalent.

The following example runs 16 processes and a specified hosts file using the default options (all-to-all connectivity, 64 to 4MB messages in powers of two, one iteration, no data integrity checking):

```
$ mpirun -np 16 -m hosts mpi_stress
```

There are a number of options for `mpi_stress`; this one may be particularly useful:

`-P`

This option *poisons* receive buffers at initialization and after each receive; pre-initialize with random data so that any parts that are not being correctly updated with received data can be observed later.

See the `mpi_stress(1)` man page for more information.

E.3.20 rpm

To check the contents of an installed RPM, use these commands:

```
$ rpm -qa infinipath\* mpi-\*
```

```
$ rpm -q --info infinipath # (etc)
```

The option`-q` queries. The option `--qa` queries all. To query a package that has not yet been installed, use the `-qpl` option.

E.3.21 strings

Use the `strings` command to determine the content of and extract text from a binary file.

The command `strings` can also be used. For example, the command:

```
$ strings -a /usr/lib/libinfinipath.so.4.0 | grep Date:
```

produces this output:

```
$Date: 2009-02-26 12:05 Release2.3 InfiniPath $
```

Note: The `strings` command is part of `binutils` (a development RPM), and may not be available on all machines.

E.4 Common Tasks and Commands

Table 10-3 lists some common commands that help with administration and troubleshooting. Note that `mpirun` in `nonmpi` mode can perform a number of checks.

Table 10-3. Common Tasks and Commands Summary

Function	Command

Check the system state	`ipath_checkout [options] hostsfile`
	`ipathbug-helper -m hostsfile \`[†]
	`> ipath-info-allhosts`
	`mpirun -m hostsfile -ppn 1 \`[†]
	`-np numhosts -nonmpi ipath_control -i`
	Also see the file:
	`/sys/class/infiniband/ipath*/device/status_str`
	where * is the unit number. This file provides information about the link state, possible cable/switch problems, and hardware errors.

Verify hosts via an Ethernet ping	`ipath_checkout --run=1 hostsfile`
Verify `ssh`	`ipath_checkout --run=2 hostsfile`
Show `uname -a` for all hosts	`mpirun -m hostsfile -ppn 1 \`[†] `-np numhosts -nonmpi  uname -a`
Reboot hosts	As a root user: `mpirun -m hostsfile -ppn 1 \`[†] `-np numhosts -nonmpi  reboot`

Run a command on all hosts	`mpirun -m hostsfile -ppn 1 \`[†]
	`-np numhosts -nonmpi` **`command`**
	`Examples:`
	`mpirun -m hostsfile -ppn 1 \`[†]
	`-np numhosts -nonmpi hostname`
	`mpirun -m hostsfile -ppn 1 \`[†]
	`-np numhosts -nonmpi date`

Copy a file to all hosts	Using bash:
	`$ for i in $( cat hostsfile )`
	`do`
	`scp source $i:destination`
	`done`

Summarize the fabric components	`ipathbug-helper -m hostsfile \`[†] `> ipath-info-allhosts`

Table 10-3. Common Tasks and Commands Summary (Continued)

Function	Command

Show the status of host IB ports	`ipathbug-helper -m hostsfile \`[†]
	`> ipath-info-allhosts`
	`mpirun -m hostsfile -ppn 1 \`[†]
	`-np numhosts -nonmpi ipath_control -i`

Verify that the hosts see each other	`ipath_checkout --run=5 hostsfile`
Check MPI performance	`ipath_checkout --run=7 hostsfile`
Generate all hosts problem report information	`ipathbug-helper -m hostsfile \`[†] `> ipath-info-allhosts`

[†] The `\` indicates commands that are broken across multiple lines.

E.5 Summary and Descriptions of Useful Files

Useful files are summarized in Table 10-4. Names in blue text are linked to a corresponding section that provides further details.

Table 10-4. Useful Files

File Name	Function
`boardversion`	File that shows the version of the chip architecture.
`status_str`	File that verifies that the InfiniPath software is loaded and functioning
/var/log/messages	Logfile where various programs write messages. Tracks activity on your system
`version`	File that provides version information of installed software/drivers

E.5.1 boardversion

It is useful to keep track of the current version of the chip architecture. You can check the version by looking in this file:

```
/sys/class/infiniband/qib0/device/boardversion
```

Example contents are:

```
ChipABI 2.0,InfiniPath_QLE7280,InfiniPath1 5.2,PCI 2,SW Compat 2
```

This information is useful for reporting problems to Technical Support.

Note: This file returns information of where the form factor HCA is installed. The PCIe half-height, short form factor is referred to as the QLE7340 or QLE7342.

E.5.2 status_str

Check the file `status_str` to verify that the InfiniPath software is loaded and functioning. The file is located here:

```
/sys/class/infiniband/qib/device/status_str
```

Table 10-5 shows the possible contents of the file, with brief explanations of the entries.

Table 10-5. `status_str` File Contents

File Contents	Description
Initted	The driver has loaded and successfully initialized the IBA6110 or IBA7220 ASIC.
Present	The IBA6110 or IBA7220 ASIC has been detected (but not initialized unless Initted is also present).
IB_link_up	The IB link has been configured and is in the active state; packets can be sent and received.
IB_configured	The IB link has been configured. It may or may not be up and usable.
NOIBcable	Unable to detect link present. This problem can be caused by one of the following problems with the HCAs: • No cable is plugged into the HCA. • The HCA is connected to something other than another IB device, or the connector is not fully seated. • The switch where the HCA is connected is down.
Fatal_Hardware_Error	Check the system log (default is `/var/log/messages)` for more information, then call Technical Support.

This same directory contains other files with information related to status. These files are summarized in Table 10-6.

Table 10-6. Status—Other Files

File Name	Contents
lid	IB LID. The address on the IB fabric, similar conceptually to an IP address for TCP/IP. *Local* refers to it being unique only within a single IB fabric.
mlid	The Multicast Local ID (MLID), for IB multicast. Used for InfiniPath ether broadcasts, since IB has no concept of broadcast.
guid	The GUID for the InfiniPath chip, it is equivalent to a MAC address.
nguid	The number of GUIDs that are used. If `nguids=2` and two chips are discovered, the first chip is assigned the requested GUID (from `eeprom`, or `ipath_sma`), and the second chip is assigned GUID+1.
serial	The serial number of the Intel HCA.
unit	A unique number for each card or chip in a system.
status	The numeric version of the `status_str` file, described in Table 10-5.

E.5.3 `version`

You can check the version of the installed InfiniPath software by looking in:

```
/sys/class/infiniband/qib0/device/driver/module/version
```

Intel-built drivers have contents similar to:

```
Version_Major.Version_Minor $Id: Intel True Scale Fabric OFED
Release x.x.x$ $Date: DDD MMM dd hh:mm:ss timezone yyyy $
```

Non-Intel-built drivers (in this case `kernel.org`) have contents similar to:

```
Version_Major.Version_Minor $Id: qib kernel.org driver $
```

E.6 Summary of Configuration Files

Table 10-7 contains descriptions of the configuration and configuration template files used by the InfiniPath and OpenFabrics software.

Table 10-7. Configuration Files

Configuration File Name	Description
/etc/modprobe.conf	Specifies options for modules when added or removed by the `modprobe` command. Also used for creating aliases. The PAT write-combing option is set here. For Red Hat 5.*X* systems.
/etc/modprobe.d/ib_qib.conf	Specifies options for modules when added or removed by the `modprobe` command. Also used for creating aliases. The PAT write-combing option is set here. For Red Hat 6.*X* systems.
/etc/modprobe.conf.local	Specifies options for modules when added or removed by the `modprobe` command. Also used for creating aliases. The PAT write-combing option is set here. For SLES systems.
/etc/infiniband/openib.conf	The primary configuration file for InfiniPath, OFED modules, and other modules and associated daemons. Automatically loads additional modules or changes IPoIB transport type.
/etc/sysconfig/infinipath	Contains settings, including the one that sets the `ipath_mtrr` script to run on reboot.
/etc/sysconfig/network/ifcfg-*NAME*	Network configuration file for network interfaces For SLES systems.
/etc/sysconfig/network-scripts/ifcfg-*NAME*	Network configuration file for network interfaces For Red Hat systems.
Sample and Template Files	**Description**
/usr/share/doc/initscripts-*/sysconfig.txt	File that explains many of the entries in the configuration files For Red Hat systems.

§ §

Appendix F Recommended Reading

Reference material for further reading is provided in this appendix.

F.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

F.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

F.3 Reference and Source for SLURM

The open-source resource manager designed for Linux clusters is located at:
http://www.llnl.gov/linux/slurm/

F.4 InfiniBand*

The InfiniBand* specification can be found at the InfiniBand* Trade Association (IBTA) website:
http://www.infinibandta.org/

F.5 OpenFabrics

Information about the OpenFabrics Alliance (OFA) is located at:
http://www.openfabrics.org

F.6 Clusters

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

F.7 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

F.8 Rocks

Extensive documentation on installing Rocks and custom Rolls can be found at:

http://www.stackiq.com

F.9 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/