Intel® Ethernet Fabric Suite Fabric Host Software

User Guide

Rev. 1.1

July 2021

Doc. No.: 632489, Rev.: 1.1

Revision History

Date	Revision	Description
July 2021	1.1	• GPU support added and discussed in the following sections and environment variables: Installed Layout, Intel® Ethernet Fabric Suite PSM3 Support for GPUDirect*, RDMA Modes and Rendezvous Module, PSM3 Rendezvous Kernel Module, PSM3 and NVIDIA* CUDA* Support, PSM3_CUDA, PSM3_CUDA_THRESH_RNDV, PSM3_GPUDIRECT, PSM3_IDENTIFY, PSM3_MQ_RNDV_NIC_THRESH, PSM3_MQ_RNDV_NIC_WINDOW, PSM3_MR_CACHE_MODE, PSM3_MULTI_EP, PSM3_PRINT_STATSMASK, PSM3_QP_PER_NIC, PSM3_RDMA, PSM3_RV_GPU_CACHE_SIZE, PSM3_RV_MR_CACHE_SIZE, CUDA* Application Failures • Descriptions changed for the following environment variables: PSM3_TRACEMASK (0x100 selection no longer includes environment variable information). • Description of send completion semantics improved in PSM3 Two-Sided Messaging. • Minor improvements in Managing MPI Versions with the MPI Selector Utility, PSM3 Rendezvous Kernel Module, PSM3_ERRCHK_TIMEOUT, PSM3_RV_HEARTBEAT_INTERVAL, PSM3_RV_Q_DEPTH, and PSM3_RV_RECONNECT_TIMEOUT.
February 2021	1.0	Initial release. • Use of the Intel® MPI Library as packaged within the Intel® OneAPI packaging is now mentioned in Setting Up the Intel® MPI Library. • Runtime options for using the Intel® MPI Library have been updated in Running MPI Applications with Intel® MPI Library. • The flag used to control logging of RV connection statistics has changed in PSM3_PRINT_STATSMASK. • Improved description of multi-rail configuration in PSM3 Multi-Rail Support, PSM3_MULTIRAIL and PSM3_MULTIRAIL_MAP. • The Rendevous module now supports RC QP connection recovery via PSM3_RV_RECONNECT_TIMEOUT and PSM3_RV_HEARTBEAT_INTERVAL. • Descriptions improved for the following environment variables: PSM3_CONNECT_TIMEOUT, FI_PSM3_LAZY_CONN, PSM3_MR_CACHE_MODE, PSM3_MR_CACHE_SIZE, PSM3_MULTI_EP, PSM3_NIC, PSM3_PRINT_STATS, PSM3_QP_PER_NIC, PSM3_RCVTHREAD, PSM3_RCVTHREAD_FREQ, PSM3_RV_MR_CACHE_SIZE, PSM3_RV_Q_DEPTH, PSM3_TRACEMASK, FI_PSM3_UUID • Descriptions added for the following environment variables.: PSM3_ERRCHK_TIMEOUT, PSM3_FLOW_CREDITS, and PSM3_NIC_SELECTION_ALG. • More details about UUID (job key) are provided in Job Identifiers. • The ordering of sections has been refined to provide a better sequencing of the document.
December 2020	0.7	Beta release. The *Intel® Ethernet Performance Scaled Messaging 3 (PSM3) User Guide* has been combined with the *Intel® Ethernet Fabric Suite Host Software User Guide* The following changes have been incorporated into the sections which were previously part of the *Intel® Ethernet Performance Scaled Messaging 3 (PSM3) User Guide*:
		continued...

Date	Revision	Description
		• Descriptions added for the following new environment variables: PSM3_IB_SERVICE_ID, PSM3_MTU, PSM3_NUM_RECV_CQES, PSM3_RV_MR_CACHE_SIZE, PSM3_RV_Q_DEPTH, PSM3_RV_QP_PER_CONN • Rendezvous module now supports multiple QPs per connection via PSM3_RV_QP_PER_CONN • Rendezvous module statistics are now available via new selections in PSM3_PRINT_STATSMASK • The filename used for statistics enabled via PSM3_PRINT_STATS has been changed to include the hostname so that shared fileystems can be accommodated • Descriptions improved for the following environment variables: PSM3_ALLOW_ROUTERS, PSM3_MR_CACHE_MODE

Contents

Figures

Tables

Preface

This manual is part of the documentation set for the Intel® Ethernet Fabric Suite Fabric (Intel® EFS Fabric), which is an end-to-end solution consisting of Network Interface Cards (NICs), fabric management and diagnostic tools.

The Intel® EFS Fabric delivers the next generation, High-Performance Computing (HPC) network solution that is designed to cost-effectively meet the growth, density, and reliability requirements of HPC and AI training clusters.

Intended Audience

The intended audience for the Intel® Ethernet Fabric Suite (Intel® EFS) document set is network administrators and other qualified personnel.

Intel® Ethernet Fabric Suite Documentation Library

Intel® Ethernet Fabric Suite publications are available at the following URL:

https://www.intel.com/content/www/us/en/design/products-and-solutions/networking-and-io/high-performance-networking/technical-library.html

Use the tasks listed in this table to find the corresponding Intel® Ethernet Fabric Suite document.

Task	Document Title	Description
Installing host software Installing NIC firmware	*Intel® Ethernet Fabric Suite Software Installation Guide*	Describes using a Text-based User Interface (TUI) to guide you through the installation process. You have the option of using command line interface (CLI) commands to perform the installation or install using the Linux* distribution software.
Managing a fabric using FastFabric	*Intel® Ethernet Fabric Suite FastFabric User Guide*	Provides instructions for using the set of fabric management tools designed to simplify and optimize common fabric management tasks. The management tools consist of Text-based User Interface (TUI) menus and command line interface (CLI) commands.
Running MPI applications on Intel® EFS Running middleware that uses Intel® EFS	*Intel® Ethernet Fabric Suite Host Software User Guide*	Describes how to set up and administer the Network Interface Card (NIC) after the software has been installed and provides a reference for users working with Intel® PSM3. Performance Scaled Messaging 3 (PSM3) is an Open Fabrics Interface (OFI, aka libfabric) provider which implements an optimized user-level communications protocol. The audience for this document includes cluster administrators and those running or implementing Message-Passing Interface (MPI) programs.

continued...

Task	Document Title	Description
Optimizing system performance	*Intel® Ethernet Fabric Performance Tuning Guide*	Describes BIOS settings and parameters that have been shown to ensure best performance, or make performance more consistent, on Intel® Ethernet Fabric Suite Software. If you are interested in benchmarking the performance of your system, these tips may help you obtain better performance.
Learning about new release features, open issues, and resolved issues for a particular release	*Intel® Ethernet Fabric Suite Software Release Notes*	

How to Search the Intel® Ethernet Fabric Suite Documentation Set

Many PDF readers, such as Adobe* Reader and Foxit* Reader, allow you to search across multiple PDFs in a folder.

Follow these steps:

1. Download and unzip all the Intel® Ethernet Fabric Suite PDFs into a single folder.
2. Open your PDF reader and use **CTRL-SHIFT-F** to open the Advanced Search window.
3. Select **All PDF documents in...**
4. Select **Browse for Location** in the dropdown menu and navigate to the folder containing the PDFs.
5. Enter the string you are looking for and click **Search**.

Use advanced features to further refine your search criteria. Refer to your PDF reader Help for details.

Documentation Conventions

The following conventions are standard for Intel® Ethernet Fabric Suite documentation:

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

  See License Agreements on page 13 for more information.

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

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

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

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

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

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

  Refer to *Intel® Ethernet Fabric Suite Software Installation Guide* for details.

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

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

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

Best Practices

- Intel recommends that users update to the latest versions of Intel® Ethernet Fabric Suite software to obtain the most recent functional and security updates.
- To improve security, the administrator should log out users and disable multi-user logins prior to performing provisioning and similar tasks.

License Agreements

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

Technical Support

Technical support for Intel® Ethernet Fabric Suite products is available 24 hours a day, 365 days a year. Please contact Intel Customer Support or visit https://www.intel.com/content/www/us/en/support/contact-support.html for additional detail.

1.0 Introduction

This guide provides detailed information and procedures to set up and administer the Intel® Ethernet Fabric Suite host software after software installation. The Intel® Ethernet Fabric Suite host software takes advantage of the given host's Network Interface Card (NIC) to access and use the network. The audience for this guide includes both cluster administrators and those running or implementing Message-Passing Interface (MPI) programs, who have different but overlapping interests in the details of the technology.

For details about the other documents for the Intel® Ethernet Fabric Suite product line, refer to Intel® Ethernet Fabric Suite Documentation Library on page 11 in this document.

For installation details, see the following document:

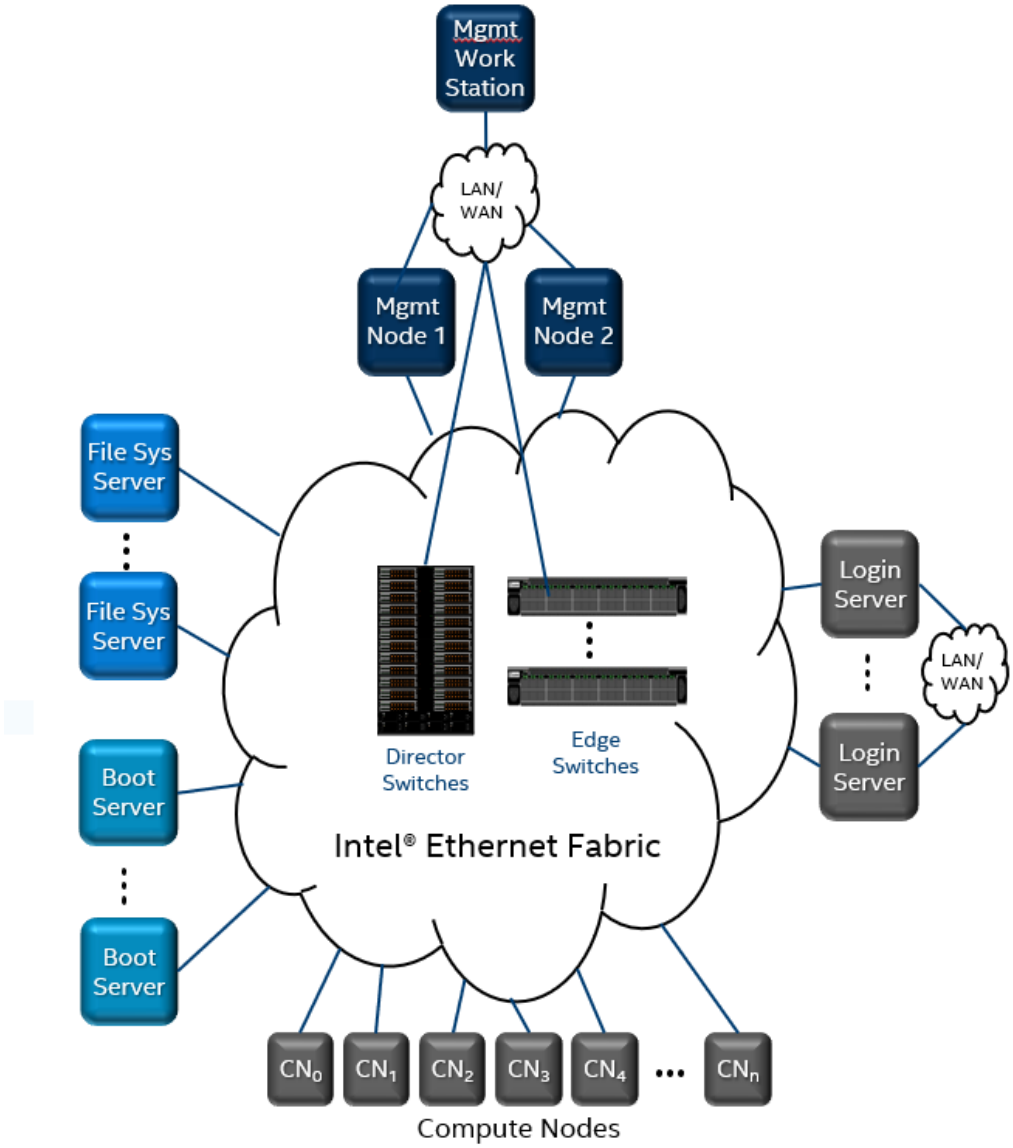
- *Intel® Ethernet Fabric Suite Software Installation Guide*

1.1 Intel® Ethernet Fabric Suite Overview

The Intel® Ethernet Fabric Suite (Intel® EFS) interconnect fabric design enables a broad class of multiple node computational applications requiring scalable, tightly-coupled processing, memory, and storage resources. With open standard APIs developed by the OpenFabrics Alliance* (OFA) Open Fabrics Interface (OFI) workgroup, NICs and switches in the Intel® EFS family are optimized to provide the low latency, high bandwidth, and high message rate needed by High Performance Computing (HPC) and AI training applications.

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

Figure 1. Intel® EFS Fabric



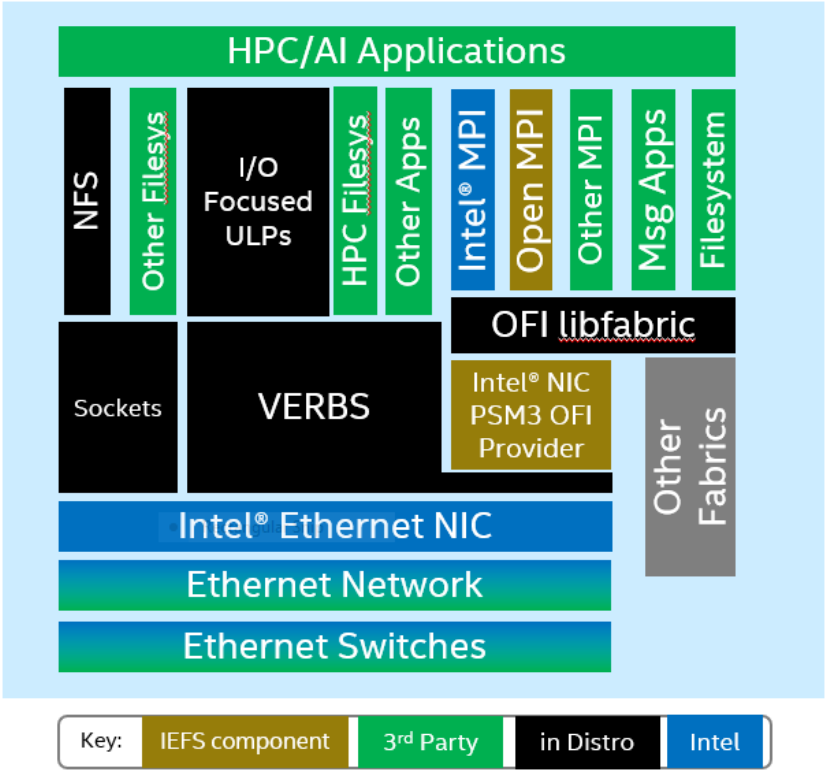
The software ecosystem is built around OFA software and includes three key APIs.

1. The OFA OFI represents a long-term direction for high-performance user level and kernel level network APIs.
2. OFA Verbs provides support for existing remote direct memory access (RDMA) applications.
3. Sockets is supported and permits many existing applications to immediately run on Intel® Ethernet Fabric Suite as well as provide TCP/IP features such as IP routing and network bonding.

Higher-level communication libraries, such as the Message Passing Interface (MPI), are layered on top of these low level OFA APIs. This permits existing HPC applications to immediately take advantage of advanced Intel® Ethernet Fabric Suite features.

Intel® Ethernet Fabric Suite combines the Network Interface Card (NIC), Intel® Ethernet Fabric Suite switches, and fabric management tools into an end-to-end solution. The host fabric software stack is shown in the following figure.

Figure 2. Intel® EFS Host Fabric Software Stack



1.1.1 Network Interface Card

Each host is connected to the fabric through a Network Interface Card (NIC). The NIC translates instructions between the host processor and the fabric. It includes the logic necessary to implement the physical and link layers of the fabric architecture, so that a node can attach to a fabric and send and receive packets to other servers or devices. NICs also include specialized logic for executing and accelerating upper layer protocols, such as RDMA transport layers.

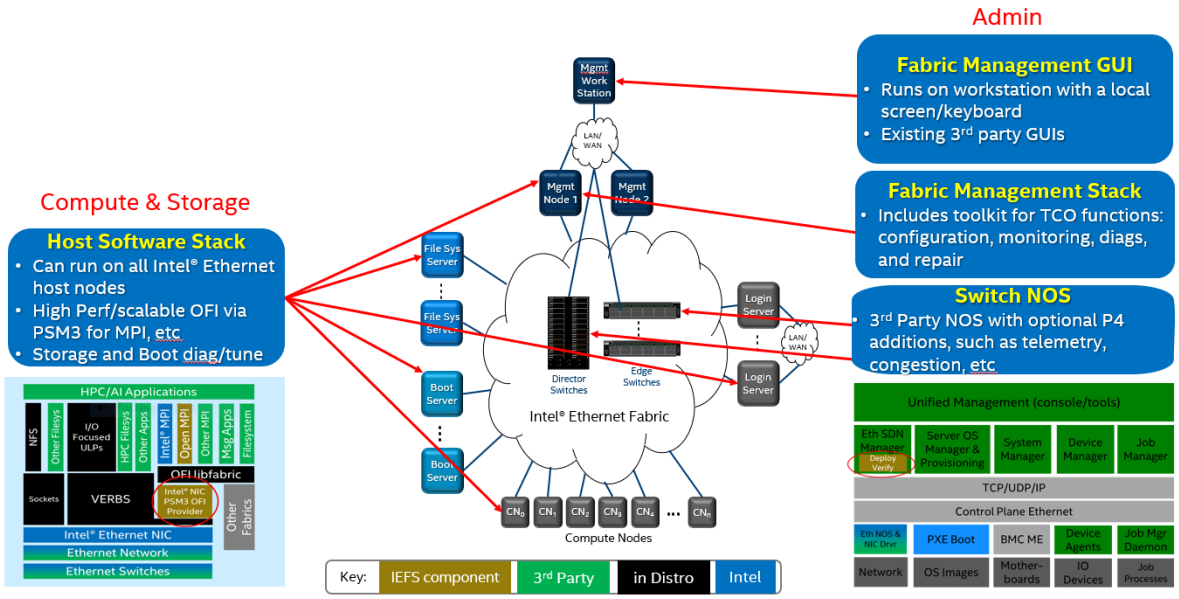
1.2 Intel® Ethernet Fabric Suite Software Overview

For software applications, Intel® EFS maintains consistency and compatibility with existing standard APIs through the open source OpenFabrics Alliance* (OFA) software stack on Linux* distribution releases.

Software Components

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

Figure 3. Intel® EFS Fabric and Software Components



Software Component Descriptions
Switch Network Operating System (NOS) Intel® EFS supports a variety of third party NOS solutions on standard Ethernet switches. Each of these switches may include features such as: • An embedded processor that runs switch management and control functions. • System management capabilities, including signal integrity, thermal monitoring, and voltage monitoring, among others. • Ethernet* port access using command line interface (CLI) or graphical user interface (GUI).
Host Software Stack • Runs on all Intel® EFS -connected host nodes and supports compute, management, and I/O nodes. • Provides a rich set of APIs including OFI, sockets, and OFA verbs. • Provides high performance, highly scalable MPI implementation via the Intel® PSM3 OFI (also known as libfabric) provider, and multiple MPI middlewares. • Includes Boot over Fabric mechanism for configuring a server to boot over the Intel® Ethernet Fabric using the NIC Unified Extensible Firmware Interface (UEFI) firmware. User documents: • *Intel® Ethernet Fabric Suite Host Software User Guide* • *Intel® Ethernet Fabric Performance Tuning Guide*
Fabric Management Stack Intel® EFS supports a variety of third party Ethernet management solutions including popular Software Defined Networking (SDN) stacks. As part of the management solution the Intel® EFS FastFabric tools are provided to aid deployment verification, fabric tuning, and diagnosis. • Runs on Intel® EFS -connected management nodes. • Includes a toolkit for configuration, monitoring, diagnostics, and repair. User documents: • *Intel® Ethernet Fabric Suite FastFabric User Guide*

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

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

2.1 Cluster Setup

Prerequisites

Make sure that hardware and low level driver installation has been completed according to the instructions supplied with the hardware.

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

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

Cluster Setup Tasks

Perform the following tasks when setting up the cluster:

1. Check that the BIOS is set properly according to the information provided in the *Intel® Ethernet Fabric Performance Tuning Guide*.
2. Check other performance tuning settings. See the *Intel® Ethernet Fabric Performance Tuning Guide* .
3. For RDMA use cases, configure Priority Flow Control (PFC) on the network adapters and Ethernet switches. For more detail see the *Intel® Ethernet Fabric Performance Tuning Guide* and the Intel® Ethernet 800 Series Linux Flow Control Configuration Guide for RDMA Use Cases.
4. Set up the host environment to use `ssh` using one of the following methods:
   - Use the `ethsetupssh` CLI command. See the man pages or the *Intel® Ethernet Fabric Suite FastFabric User Guide* for details.
   - Use the FastFabric textual user interface (TUI) to set up `ssh`. See the *Intel® Ethernet Fabric Suite FastFabric User Guide* for details.
5. Verify the cluster setup using the the FastFabric textual user interface (TUI). See the *Intel® Ethernet Fabric Suite FastFabric User Guide* for details.

2.2 Using MPI

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

Prerequisites

Before you continue, the following tasks must be completed:

1. Verify that the Intel hardware and software have been installed on all the nodes.
2. Verify that the host environment is set up to use `ssh` on your cluster as described in Cluster Setup.

Set Up and Run MPI

The following steps are the high level procedures with links to the detailed procedures for setting up and running MPI:

1. Set up Intel® MPI Library. See Intel® MPI Library Installation and Setup.
2. Compile MPI applications. See Compiling MPI Applications with Intel® MPI Library.
3. Run MPI applications. See Running MPI Applications with Intel® MPI Library.

Additional Considerations

- To test using other MPIs that run over OFI (also known as libfabric), such as Open MPI, see Using Other MPIs.
- Use the MPI Selector Utility to switch between multiple versions of MPI. See Managing MPI Versions with the MPI Selector Utility.
- Refer to Intel® Ethernet Fabric Suite Cluster Setup and Administration, and the *Intel® Ethernet Fabric Performance Tuning Guide* for information regarding fabric performance tuning.
- Review your process placement controls. See https://software.intel.com/en-us/articles/controlling-process-placement-with-the-intel-mpi-library for detailed information.
- Refer to Using Other MPIs to learn about using other MPI implementations.

3.0 Intel® Ethernet Fabric Suite Cluster Setup and Administration

This section describes what the cluster administrator needs to know about the Intel® Ethernet Fabric Suite software and system administration.

3.1 Installation Packages

The following software installation packages are available for an Intel® Ethernet Fabric.

3.2 Installed Layout

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

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

Table 1. Installed Files and Locations

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

Documentation	`/usr/share/man`
	`/usr/share/doc/libpsm3-fi/`
	Intel® Ethernet Fabric Suite user documentation can be found on the Intel web site. See Intel® Ethernet Fabric Suite Documentation Library for URL.

Configuration	`/etc/eth-tools`
	`/etc/sysconfig/eth-tools`
	`/usr/share/eth-tools/samples/`

Initialization	`/usr/lib/systemd/system/iefs.service`
OpenMPI source	`/usr/src/eth/MPI`
MPI benchmark source	`/usr/src/eth/mpi_apps` See *Intel® Ethernet Fabric Suite FastFabric User Guide* for more information on how to build and run these benchmarks
Intel® PSM3 OFI Provider	`/usr/lib64/libfabric/libpsm3-fi.so*`
Intel® Ethernet Fabric Kernel Modules	RHEL*: `/lib/modules/<kernel version>/extra/iefs-kernel-updates/*.ko` SLES*: `/lib/modules/<kernel version>/updates/iefs-kernel-updates/*.ko`

3.3 Intel® Ethernet Fabric and OFA Driver Overview

The Network Interface Card (NIC) software will include the appropriate kernel module and user space libraries to enable use of TCP/IP via sockets and RoCE via OFA verbs APIs.

In addition the Intel® Ethernet Fabric Suite includes the Intel® Performance Scaled Messaging 3 (Intel® PSM3 OFI provider and the rendevous kernel module (`rv`)). See PSM3 OFI Provider.

3.4 Managing the Intel® Ethernet Fabric Rendezvous Kernel Module

The startup script for the rendezvous module (`rv`) is installed automatically as part of the software installation, and typically does not need to be changed. It runs as a system service.

The rendezvous module has several configuration variables that set MR cache sizes, control connection handling, define events to create trace records, and set the debug level.

3.4.1 More Information on Configuring and Loading Drivers

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

Also refer to the `/usr/share/doc/initscripts-*/sysconfig.txt` file for general information on configuration files.

4.0 Running MPI on Network Interface Cards

This section provides information on using the Message Passing Interface (MPI) on Network Interface Cards (NICs). 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® Ethernet Host Software

The high-performance open-source MPI packaged with Intel® Ethernet Fabric Suite Basic installation package is Open MPI . This MPI has support for the OpenFabrics Alliance* (OFA Open Fabrics Interface (OFI) (also known as libfabric). OFI allows MPI to make use of the Intel® PSM3 provider for optimized message passing both locally and across the network.

There are other MPIs that are not packaged with Intel® Ethernet Fabric Suite Basic installation package that use the OFI API, including the Intel® MPI Library.

For more information on other MPIs including Open MPI, see Using Other MPIs.

4.2 Intel® MPI Library

The Intel® MPI Library is a high-performance interconnect-independent multi-fabric library implementation of the industry-standard Message Passing Interface, v3.1 (MPI-3.1) specification. Intel® Ethernet Fabric Suite supports the 64-bit version of Intel® MPI Library. The Intel® MPI Library is not included in the Intel® Ethernet Fabric Suite software, but is available separately. Go to http://software.intel.com/en-us/intel-mpi-library for more information.

NOTE

The Intel® MPI Library is feature rich and highly optimized. For many applications it will offer superior performance and capabilities as compared to other MPI libraries. It's use with Intel® Ethernet Fabric Suite is highly recommended.

4.2.1 Intel® MPI Library Installation and Setup

Download the Intel® MPI Library from http://software.intel.com/en-us/intel-mpi-library and follow the installation instructions. The following subsections provide setup instructions for the Intel® MPI Library.

4.2.1.1 Setting Up the Intel® MPI Library

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

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

```
$ source $prefix/intel64/bin/mpivars.sh
```

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

```
$ source $prefix/intel64/bin/mpivars.csh
```

NOTE

In the examples above `$prefix` is the path to the Intel® MPI installation. For example, if Intel® Parallel Studio is installed `$prefix` may be `/opt/intel/compilers_and_libraries_<version>/linux/mpi/`

NOTE

When using Intel® MPI packaged within the Intel® oneAPI packaging, instead of using `mpivars.sh` as shown above, `source $prefix/env/vars.sh` prior to launching MPI jobs. For example. if Intel® is installed `$prefix` may be `/opt/intel/oneapi/<version>/mpi/`

4.2.1.2 Compiling MPI Applications with Intel® MPI Library

Generally, recompilation is not required for MPICH-based applications. For applications compiled for other MPIs, Intel recommends that you use the included wrapper scripts that invoke the underlying compiler. The default underlying compiler is GCC*, including gfortran.

NOTE

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

Table 2. Intel® MPI Library Wrapper Scripts

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

continued...

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

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

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

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

4.2.2 Running MPI Applications with Intel® MPI Library

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

```
$ export I_MPI_FABRICS=shm:ofi
$ export I_MPI_OFI_PROVIDER=psm3
$ mpirun -np 4 -ppn 1 -hostfile mpi_hosts mpi_app_name
```

NOTE

The -ppn 1 option shown above places one process on each host listed in *mpi_hosts*. Hosts may be listed more than once to place more than 1 process on a given host.

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

On systems where you are using Intel® Ethernet Fabric Suite Software, you can ensure that the Intel® Ethernet Fabric is used by Intel® MPI Library by exporting I_MPI_FABRICS=shm:ofi and I_MPI_OFI_PROVIDER=psm3 prior to the mpirun command. More details are available in section "Intel® MPI Library Settings" of the *Intel® Ethernet Fabric Performance Tuning Guide*.

4.3 Allocating Processes

MPI ranks are processes that communicate through the Intel® PSM3 OFI provider for best performance. These MPI ranks are called Intel® PSM3 processes.

Typically, MPI jobs are run with each rank mapped to a CPU and associated with a NIC.

Optimal performance may be achieved by ensuring that the Intel® PSM3 process affinity is assigned to the CPU of the Non-Uniform Memory Access (NUMA) node local to the NIC that it is operating.

See this article https://software.intel.com/en-us/articles/controlling-process-placement-with-the-intel-mpi-library for information on controlling process placement with Intel® MPI Library.

4.4 Environment Variables

The PSM3 OFI Provider section provides more information about Intel® PSM3 and the environment variables that may be used to control various PSM3 options and features when using MPIs such as the Intel® MPI Library. Refer to that section for complete details.

Intel® MPI Library provides its own environment variables that may control MPI features and may also effect various PSM3 features. It also provides mechanisms (`-genv` option) to set variables in all processes of a job such that variables are only active after the `mpirun` command has been issued and while the MPI processes are active. See the Intel® MPI Library documentation for information, specifically: https://software.intel.com/en-us/mpi-developer-reference-linux

4.5 Intel MPI Library and Hybrid MPI/OpenMP Applications

Intel MPI Library supports hybrid MPI/OpenMP applications. Instead of `MPI_Init/MPI_INIT` (for C/C++ and Fortran respectively), the program must call `MPI_Init_thread/MPI_INIT_THREAD` for initialization.

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

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

NOTE

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

The number of OpenMP threads is on a per-node basis and is controlled by the `OMP_NUM_THREADS` environment variable in the user's environment. `OMP_NUM_THREADS` is used by other compilers' OpenMP products, but is not an Intel MPI Library environment variable. Use this variable to adjust the split between MPI processes and OpenMP threads. Usually, the number of MPI processes (per node) times the number of OpenMP threads is set to match the number of CPUs per node.

An example case is a node with four CPUs, running one MPI process and four OpenMP threads. In this case, `OMP_NUM_THREADS` is set to `4`.

4.6 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.

4.6.1 MPI Errors

Almost all MPI routines (except `MPI_Wtime` and `MPI_Wtick`) return an error code. It is returned either as the function return value in C functions or as the last argument in a Fortran subroutine call. Before the value is returned, the current MPI error handler is called. By default, this error handler terminates the MPI job. Therefore, you can get information about MPI exceptions in your code by providing your own handler for `MPI_ERRORS_RETURN`. For details, see the "MPI_Comm_set_errhandler" man page: https://www.mpich.org/static/docs/v3.2/www3/MPI_Comm_set_errhandler.html

For details on MPI error codes, see the "Error codes and classes" man page: https://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/node149.htm

4.6.2 Using Debuggers

See Debugging in the Intel®Developer Zone for details on debugging with Intel® MPI Library.

5.0 Using Other MPIs

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

5.1 Introduction

Intel® EFS Software supports multiple high-performance MPI implementations. Most implementations run over OFA's Open Fabrics Interface (OFI, aka libfabric) and hence can use the Intel® PSM3 OFI provider. Some supported MPI implementation are shown in the following table. Use the `mpi-selector-menu` command to choose which MPI to use, as described in Managing MPI Versions with the MPI Selector Utility.

Table 3. Supported MPI Implementations

MPI Implementation	Runs Over	Compiled With	Comments
Intel® MPI Library	PSM3 (via OFI)	GCC Intel (ICC)	Provides MPI-1 and MPI-2 functionality. Available from Intel®.
Open MPI	PSM3 (via OFI)	GCC	Provides some MPI-2 functionality (one-sided operations and dynamic processes). A build with support for CUDA* enabled GPU applications is provided. Available as part of the Intel® EFS Software download. Can be managed by `mpi-selector`.

NOTE

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

5.2 Installed Layout

By default, Open MPI is installed in the following directory tree:

`/usr/mpi/gcc/$mpi-mpi_version`

NOTE

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

The Intel® EFS Software-supplied MPI is pre-compiled with GCC*. It also has `-ofi` appended after the MPI version number, for example:

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

CUDA* enabled versions of Open MPI included with Intel® EFS Software will have `-cuda-ofi` appended after the MPI version number, for example:

```
/usr/mpi/gcc/openmpi-VERSION-cuda-ofi
```

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

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

```
/usr/mpi/gcc/$mpi/bin/mpirun
```

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

5.3 Open MPI

Open MPI is an open source MPI implementation from the Open MPI Project. The precompiled version of Open MPI that runs over OFI (and can use the Intel® PSM3 OFI provider) and is built with the GCC* is available with the Intel download.

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

5.3.1 Installing Open MPI

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

5.3.2 Setting up Open MPI

Intel recommends that you use the `mpi-selector` tool, because it performs the necessary `$PATH` and `$LD_LIBRARY_PATH` setup to include the Open MPI installation path.

If the `mpi-selector` tool is not used, you must setup the paths explicitly or `source` a script provided with MPI such as *`$mpi_home`*`/bin/mpivars.sh`. Where *`$mpi_home`* is the directory path where Open MPI is installed, such as `/usr/mpi/gcc/openmpi-`*`VERSION`*`-ofi`.

5.3.3 Setting up Open MPI with SLURM

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

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

This key is used by the Intel® PSM3 OFI provider to uniquely identify each different job end point used on the fabric. If two MPI jobs are running on the same node sharing the same NIC and using Intel® PSM3, each one must have a different key.

5.3.4 Compiling Open MPI Applications

Intel recommends that you use the included wrapper scripts that invoke the underlying compiler instead of attempting to link to the Open MPI libraries manually. This allows the specific implementation of Open MPI to change without forcing changes to linker directives in users' Makefiles.

The following table lists the included wrapper scripts.

Table 4. Open MPI Wrapper Scripts

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

To compile your program in C, enter the following:

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

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

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

Table 5. Command Line Options for Scripts

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

5.3.5 Running Open MPI Applications

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

- `openmpi_gcc_ofi-X.X.X`

For example, if you chose `openmpi_gcc_ofi-X.X.X`, the following mpirun command would run using the Intel® PSM3 OFI provider:

```
$ mpirun -np 4 -machinefile mpi_hosts -mca mtl ofi -x FI_PROVIDER=psm3
mpi_app_name
```

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

NOTE

The `-mca mtl ofi` parameter is required to select use of OFI and `-x FI_PROVIDER=psm3` is required to ensure the Intel® PSM3 OFI provider is used. If an OFI provider name is not specified, Open MPI selects the first one listed by the `fi_info` utility.

5.3.6 Configuring MPI Programs for Open MPI

When configuring an MPI program, for example, generating header files and/or Makefiles for Open MPI, you usually need to specify `mpicc`, `mpicxx`, and so on as the compiler, rather than the GNU* Compiler Collection (i.e., GCC*; including gcc, g++, gfortran, etc.).

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

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

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

```
$ setenv CC mpicc
```

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

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

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

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

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

```
LD=mpif90
```

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

5.3.7 Using Another Compiler

Open MPI and all other Message Passing Interfaces (MPIs) that run on Intel® Ethernet Fabric Suite support multiple compilers, including:

- The GNU* Compiler Collection (i.e., GCC*; including gcc, g++, gfortran, etc.)
- Intel compiler

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

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

Table 6. Intel Compilers

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

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

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

5.3.7.1 Compiler and Linker Variables

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

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

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

5.3.8 Running in Shared Memory Mode

Open MPI supports running exclusively in shared memory mode. No Network Interface Card 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.

For shared memory execution, the Open MPI component that performs best is the `vader BTL`. To run using this component, it is necessary to request it with the following command line options `–mca pml ob1 –mca btl vader,self`. It is also recommended to explicitly restrict the node where the MPI processes will run by editing the `hostfile`. For example, if the file is named `onehost` and it is in the working directory, enter the following:

```
$  echo "idev-64 slots=8" > onehost
```

Where `idev-64` is the name of the host and `slots=8` is the maximum number of MPI processes to allowed to run in the node. Typically, this is equal to the number of cores on the node.

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

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

```
$ mpirun -np 2 -hostfile onehost -mca pml ob1 -mca btl vader,self  osu_latency
```

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

```
$ mpirun -np 2 -hostfile onehost -mca pml ob1 -mca btl vader,self  osu_bw
```

NOTE

For some applications, use of the Intel® PSM3 OFI provider and its shm protocol may provider better performance.

5.3.9 Using the mpi_hosts File

A `hostfile` (also called *machines file*, *nodefile*, or *hostsfile*) must be created to list the hosts that will be used for a given parallel job.

The two supported formats for the `hostfile` are:

```
hostname1
hostname2
...
```

or

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

In the first format, if the -np count (number of processes to spawn in the mpirun command) is greater than the number of lines in the machine file, the hostnames are repeated (in order) as many times as necessary for the requested number of processes. Also, if the -np count is less than the number of lines in the machine file, mpirun still processes the entire file and tries to pack processes to use as few hosts as possible in the hostfile.

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

Intel recommends that you use the second format and various command line options to schedule the placement of processes to hosts and cores. For example, use the mpirun option -npernode to specify how many processes should be scheduled on each host on each pass through the hostfile. (The -npernode option is similar to the Intel® MPI Library option -ppn.) In the case of nodes with 8 cores each, if the hostfile line is specified as hostname1 slots=8 max-slots=8, then Open MPI assigns a maximum of 8 processes to the node and there can be no over-subscription of the 8 cores.

There are several ways of specifying the hostfile:

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

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

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

  Also, -machinefile is a synonym for -hostfile.

- Use the -H, -hosts, or --host command line option, followed by a host list. The host list can follow one of the following examples:

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

- Use the file ./mpi_hosts, if it exists.

  If you are working in the context of a batch queuing system, it may provide a job submission script that generates an appropriate mpi_hosts file. For more details, see the website:

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

5.3.10 Using the Open MPI mpirun script

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

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

The general syntax is:

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

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

```
./program-name
```

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

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

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

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

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

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

This option overrides the `slots=process_count` specifications, if any, in the lines of the `mpi_hosts` file. As a general rule, `mpirun` distributes the *n* node processes among the nodes without exceeding, on any node, the maximum number of instances specified by the `slots=process_count` option. The value of the `slots=process_count` option is specified by either the `-npernode` command line option or in the `mpi_hosts` file.

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

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

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

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

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

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

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

Killing the `mpirun` program kills all the processes in the job. Use **CTRL+C** to kill `mpirun`.

5.3.11 Using Console I/O in Open MPI Programs

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

NOTE

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

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

It is possible to redirect standard I/O for Open MPI applications by using the typical shell redirection procedure on `mpirun`, as shown in the following example:

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

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

5.3.12 Process Environment for mpirun

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

- Remote Execution:

  https://www.open-mpi.org/doc/v2.0/man1/mpirun.1.php#sect18

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

5.3.13 Further Information on Open MPI

For more information about Open MPI, see:

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

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

5.4 Managing MPI Versions with the MPI Selector Utility

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

The MPI Selector is installed as a part of the Linux* distribution and it includes the following basic functions:

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

Here is an example for listing the available MPIs:

```
$ mpi-selector --list
openmpi_gcc_ofi-X.X.X
```

Changes to the default take effect in the next shell that is started. See the `mpi-selector` man page for more information.

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

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

Additional details can be found at:

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

6.0 Intel® Ethernet Fabric Suite PSM3 Support for GPUDirect*

GPUDirect* is an NVIDIA* technology that allows for third party network adapters to directly read and write to CUDA* host and device memory to enhance performance for latency and bandwidth. PSM3 has support for CUDA*, GPUDirect* Copy, GPUDirect* Send DMA and GPUDirect* RDMA. The Intel® Ethernet Fabric Suite also includes a pre-built CUDA* enabled version of Open MPI for OpenFabrics Alliance* (OFA Open Fabrics Interface (OFI) (also known as libfabric). Additionally the MPI benchmark source included with FastFabric includes some CUDA* enabled benchmarks which FastFabric can assist to build and distribute within a cluster.

PSM3's GPUDirect features can be used to accelerate CUDA* based workloads and benchmarks for servers with NVIDIA* Kepler-architecture based GPUs or newer.

NOTE

After the Intel® Ethernet Fabric Suite Software is properly installed with CUDA*, the CUDA* feature in the PSM3 provider must be enabled in combination with a CUDA*-enabled MPI in order to potentially accelerate CUDA* applications. Refer to PSM3 OFI Provider and the *Intel® Ethernet Fabric Performance Tuning Guide* for more information.

Refer to the following Intel® Ethernet Fabric Suite publications within the Intel® Ethernet Fabric Suite Documentation Library:

- *Intel® Ethernet Fabric Suite Software Installation Guide*
- *Intel® Ethernet Fabric Performance Tuning Guide*
- *Intel® Ethernet Fabric Suite Host Software User Guide*
- *Intel® Ethernet Fabric Suite FastFabric User Guide*

Refer to the following for more information about GPUDirect:

- https://developer.nvidia.com/gpudirect
- http://docs.nvidia.com/cuda/gpudirect-rdma/index.html

6.1 Using PSM3 Features for GPUDirect*

The PSM3 features required for GPUDirect* and CUDA* are disabled (0) by default. To use GPUDirect* and CUDA*:

- Use a CUDA*-enabled application.
- Ensure you are using a CUDA*-enabled MPI or middleware, such as `openmpi-x.x.x-cuda-ofi`.
- Enable PSM3 features:
  - `PSM3_CUDA=1`

— `PSM3_GPUDIRECT=1`

NOTES

- Not enabling these features for CUDA*-enabled workloads may result in application segfaults or other crashes.
- When `PSM3_GPUDIRECT=1`, this implicitly also sets `PSM3_CUDA=1`
- When `PSM3_GPUDIRECT=1`, the rendezvous module (rv) will be required to assist in GPUDirect* features. In which case the CUDA* enabled rendezvous module must be loaded.
- CUDA* applications may be run without use of GPUDirect* by specifying only `PSM3_CUDA=1`, in which case the rendezvous module may not be required. See RDMA Modes and Rendezvous Module for more information.
- For special cases, there may be reasons to disable GPUDirect* support; however, you should leave CUDA* support enabled. Refer to PSM3 OFI Provider or the *Intel® Ethernet Fabric Performance Tuning Guide* for more information.
- Enabling CUDA* and/or GPUDirect* for an application which does not use CUDA*, may reduce the performance of the job.

7.0 PSM3 OFI Provider

7.1 Introduction

The Intel® Performance Scaled Messaging 3 (Intel® PSM3) provider implements a high-performance protocol that runs above the communications interfaces provided by the Intel® Ethernet Fabric Suite family of products. PSM3 enables mechanisms necessary to implement higher level communications interfaces in parallel environments such as MPI and AI training frameworks.

PSM3 targets clusters of multicore processors and transparently implements two levels of communication: inter-node communication and intra-node shared memory communication.

7.2 Differences between PSM3 and PSM2

Intel® PSM3 interface differs from Intel® Omni-Path PSM2 in the following ways:

- PSM3 includes new features and optimizations for Intel® Ethernet Fabric hardware and processors.
- The PSM3 protocol only supports the Open Fabrics Interface (OFI, aka libfabric). As such the PSM API is no longer exported.
- PSM3 includes additional performance improvements and new features.
- PSM3 supports standard Ethernet networks and leverages standard RoCEv2 protocols as implemented by Intel® Ethernet Fabric Suite NICs.

For details on supported versions of MPI Libraries, refer to the *Intel® Ethernet Fabric Suite Software Release Notes*.

7.3 Compatibility

PSM3 can coexist with other software distributions such as OpenFabrics that allow applications to simultaneously target PSM3-based and non-PSM3 based applications on a single node without changing any system-level configuration.

However, unless otherwise noted, PSM3 does not support running PSM3-based and non-PSM3 based communication within the same user process.

PSM3 is currently a single-threaded library. This means that you cannot make any concurrent PSM3 library calls (with the exception of PSM3 Multi-Endpoint Functionality). While threads may be a valid execution model for the wider set of potential PSM3 clients, applications should currently expect better effective use of Intel® Ethernet Fabric Suite resources (and hence better performance) by dedicating a single PSM3 communication endpoint to every CPU core.

Except where noted, PSM3 does not assume a single program, multiple data (SPMD) parallel model, and extends to multiple program, multiple data (MPMD) environments in specific areas. However, PSM3 assumes the runtime environment to be homogeneous on all nodes in bit width (64-bit only) and endianness (little or big), and fails at startup if any of these assumptions do not hold.

7.4 Job Identifiers

Every PSM3 job is assigned a Universally Unique Job Identifier (UUID), sometimes referred to as a job key. Typically this identifier is automatically generated by the application launch mechanism or the job scheduler. All processes in a given job must have the same UUID and the same Linux user ID (see Linux `getuid(3)` man page).

The UUID is used by PSM3 for the following:

- To filter out stale packets or unexpected communications from other jobs.
- To aid in hashing for selection among multiple NICs when PSM3_NIC_SELECTION_ALG is applicable.
- To separate intra-node communications and coordination resources used by PSM3 such as Linux shared memory and semaphores.
- To separate resources and parameters for different jobs within the PSM3 Rendezvous Kernel Module

The UUID may be supplied to PSM3 in a variety of ways (in order of priority, earlier entry in list wins if UUID is supplied in multiple ways to a single process):

- The user, middleware or job launch script may explicitly export FI_PSM3_UUID
- The value may be supplied to PSM3 via the OFI API's auth_key field by the middleware or a application coded directly to OFI. This is the preferred mechanism as it more appropriately uses the OFI API in a manner which is not provider specific.
- PSM3 may generate it's own value based on the Linux user id.

NOTE

The UUID is 128 bits, but is not cryptographic in nature. Customers desiring heightened security for jobs, should use additional Ethernet security mechanisms such as network isolation/firewalling, VLANs or lower level packet encryption techniques. Depending on overall network design and hardware, such mechanisms may impact performance.

NOTE

Various middlewares and job schedulers may provider additional mechanisms for a user to specify the UUID to be passed to PSM3 or control it's generation. For example the Intel® MPI Library has controls such as `I_MPI_SPAWN=1` which permit it to generate the UUID based on the JOBID provided by various job schedulers such as Slurm, PBSpro and LSF, such mechanisms can enable communications via PSM3 for multi-part jobs.

7.5 Endpoint Communication Model

PSM3 follows an endpoint communication model where an endpoint is defined as an object (or handle) instantiated to support sending and receiving messages to other endpoints. In order to prevent PSM3 from being tied to a particular parallel model (such as SPMD), OFI (libfabric) retains control over the parallel layout of endpoints. Opening endpoints and connecting endpoints to enable communication are two decoupled mechanisms. If the OFI application (also called middleware, such as MPI) does not dynamically change the number of endpoints beyond parallel startup, it can combine both mechanisms at startup. OFI applications can manipulate the location and amount of endpoints at runtime by explicitly connecting sets or subsets of endpoints.

As a side effect, this greater flexibility allows the OFI application to manage a two-stage initialization process. In the first stage of opening an endpoint, the OFI application obtains an opaque handle to the endpoint and a globally distributable endpoint identifier. Prior to the second stage of connecting endpoints, the OFI application must distribute all relevant endpoint identifiers through an out-of-band mechanism. Once the endpoint identifiers are successfully distributed to all processes that need to communicate, the OFI application may connect all endpoint identifiers to the locally opened endpoint. In connecting the endpoints, the OFI application obtains an opaque endpoint address, which is used for all PSM3 communication operations.

Internal to the PSM3 provider, there is an optional lazy connection model that permits actual connect establishment to be delayed until the first communications with a given remote endpoint. See FI_PSM3_LAZY_CONN for more details.

7.6 PSM3 Multi-Endpoint Functionality

PSM3 Multi-Endpoint (Multi-EP) functionality is enabled by default.

PSM3 has added minimal thread safety for using with Multi-EP in a performant manner. Along with each endpoint (EP) created, an associated matched queue (MQ) is created that tracks message completion and ordering.

Related Information

- **Intel® MPI Library Multi-Thread (MT)**

  Intel® MPI MT design is motivated by the need to improve communication throughput and concurrency in hybrid MPI applications on Intel hardware, particularly when using Intel® Ethernet Fabric Suite (Intel® EFS). However, the design is universal, so it can be used on any other hardware that is supported with specific abstractions (Scalable Endpoints). The design is entirely based on the Open Fabric Interface (OFI) libfabric concept of Scalable Endpoints (SEP).

  For details, go to: https://software.intel.com/en-us/intel-mpi-library/documentation

- **OpenFabrics Alliance* (OFA) Open Fabric Interfaces libfabric**

  The psm3 provider supports scalable endpoints.

  For details, go to: https://ofiwg.github.io/libfabric/

7.7 PSM3 Multi-Rail Support

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

7.7.1 Multi-Rail Overview

A multi-rail configuration provides load balancing capabilities, potentially adding a higher degree of fabric performance .

The multi-rail feature can be applied to a single plane or multiple planes. By enabling multi-rail, a process can use multiple network interface cards (NICs) to transfer messages. When a single PSM3 process is using multiple NICs, one will be selected as the primary rail. PSM3 will use the primary rail for connection establishment and certain control messages and to ensure message ordering remains compliant with the MPI API specification.

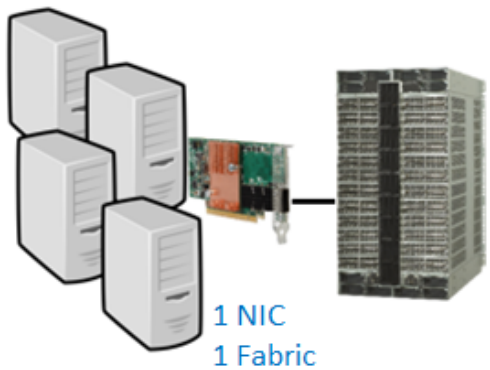
Even when each PSM3 process is using only a single NIC, the NIC used by various processes may differ, in which case multiple NICs may effectively be used and load balanced during a given job. The degree of load balancing will depend on the application's traffic patterns.

TERMINOLOGY:

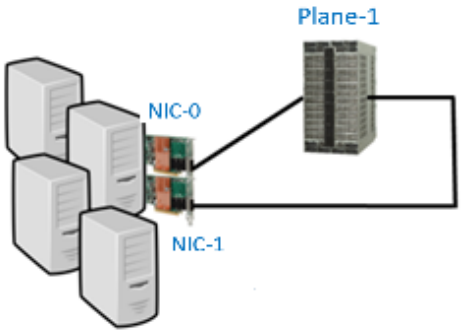
- Planes can sometimes be referred to as *fabrics*.
- Hosts can also be referred to as *nodes*.
- NICs can also be referred to as *rails*.
- Processes can also be referred to as *ranks*.

Three basic scenarios include:

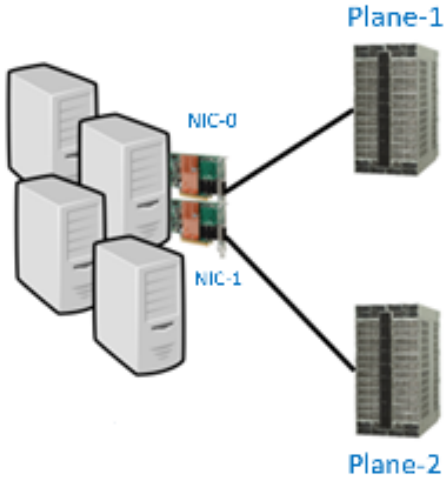
- Single-rail in a single plane: This scenario, shown in the following figure, consists of one NIC in a server connected to one plane. This is the default configuration during installation. This configuration provides the performance required by most applications in use today.



- Dual-rail in a single plane: This scenario, shown in the following figure, consists of two NICs in the same server connected to the same plane. This configuration may provide improved MPI message rate, latency, and bandwidth to the node.



- Dual-rail in dual planes: This scenario, shown in the following figure, consists of two NICs in the same server connected to separate planes. Depending on the platform, this configuration may provide improved MPI message rate, latency, and bandwidth to the node. Typically each plane is configured with a distinct set of switches, so that the planes are isolated from each other for performance reasons. NICs in different planes will not attempt to communicate with each other, so the physical networks for the planes do not require any interconnection. NICs in the first plane should not be on the same Ethernet IP subnet as any NICs in the second plane.



NOTE

Other multi-rail scenarios can be configured with more than two NICs per server and/or more than two planes.

Within each plane, there can be 1 or more Ethernet IP subnets. When there is more than 1 subnet per plane, Ethernet IP routing must be configured such that all NICs within the given plane can communicate with each other. By default Intel® PSM3 assumes NICs in different IP subnets cannot communicate with each other.

7.7.2 Multi-Rail Usage

The system administrator sets up a multi-rail system using multiple Intel® Ethernet Fabric NICs per node. If desired, the system administrator connects the NICs to multiple planes, and configures each plane with a different Ethernet IP subnet.

MPI application programmers can run their application over a multi-rail system to improve performance or increase the number of hardware resources available for jobs. By default, Intel® PSM3 selects one NIC per process in a round-robin fashion, in an attempt to evenly distribute the use of hardware resources and provide performance benefits.

In some scenarios, it may be beneficial to enable the PSM3_MULTIRAIL environment variable to make each PSM3 process load balance and stripe messages across more than the single selected NUMA-local or nearest NIC.

On a multi-plane system, if you do not want to use multi-rail, Intel recommends that you set the PSM3_NIC environment variable (starting from 0 or specifying a specific NIC by name) to notify the PSM3 job which NIC to use or set PSM3_NIC_SELECTION_ALG to control the algorithm used to select a NIC for each process. In this case, the NICs selected for a given job should be on the same plane, otherwise, the job might try to use NICs from different planes and cause the job to fail or hang because there is no path between planes. In this case, some jobs may successfully run across different planes, but this behavior cannot be guaranteed.

If multi-rail is turned on, PSM3 can reorder and automatically match the NICs by using the Ethernet IP subnet. That is why unique Ethernet IP subnets are required for each plane. For more information, refer to the *Intel® Ethernet Fabric Performance Tuning Guide*, MPI Performance.

7.7.3 Environment Variables

NOTE

Specification of multi-rail environment variables is typically not required for PSM3 to use multiple NICs that are on the same plane with a single subnet. However, it may provide performance benefits in certain scenarios and is required for multi-subnet or multi-plane configurations.

The following environment variables can be set:

- `PSM3_MULTIRAIL` = `n` where `n` can be a value between 0 and 2.

  `PSM3_MULTIRAIL` controls how PSM3 load balances and stripes messages across NICs on the system.

  If set to a non-zero value, PSM3 sets up multiple rails per process, up to a maximum of thirty-two rails. How multi-rails are set up and how many rails are used depends on how the environment variables `PSM3_MULTIRAIL_MAP` and `PSM3_ALLOW_ROUTERS` are set.

  When set to zero, each PSM3 process will use only its NUMA-local or nearest selected NIC as selected via `PSM3_NIC` or `PSM3_NIC_SELECTION_ALG`.

  The `PSM3_MULTIRAIL` options are:

  — 0 : Single NIC per process configuration (default)

  - 1 : Enable Multi-rail capability and each process will use all available NIC(s) in the system
  - 2 : Enable Multi-rail capability and limits NIC(s) used by a given process to NIC(s) within a single NUMA socket. PSM3 will look for an available NIC (and select at least one, even if it is not NUMA-local to the process)
- `PSM3_MULTIRAIL_MAP = unit,unit,unit,...`

  Specifies the NIC to use for each rail. Multiple specifications are separated by a comma. `unit` starts from 0 or may specify an RDMA device name. The term `unit` refers to a NIC.

  If only one rail is specified, it is equivalent to a single-rail case. The `unit` is specified instead of using the `unit` values assigned by PSM3.
- `PSM3_ALLOW_ROUTERS= n` where `n` can be 0 or 1. Normally PSM3 assumes any local or remote NICs with distinct Ethernet IP subnets are on distinct planes. When this value is 1, PSM3 assumes routers are present such that all Ethernet IP subnets can communicate with each other. When `PSM3_ALLOW_ROUTERS= 1` in multi-plane configurations, `PSM3_MULTIRAIL_MAP` must be used to explicitly specify the mapping.

NOTE

In a typical use case, if `PSM3_MULTIRAIL_MAP` is specified, the same value will be specified and used for all processes in the whole job.

If `PSM3_MULTIRAIL` is set to 1 or 2, the following occurs:

- For individual messages less than `PSM3_MQ_RNDV_NIC_WINDOW` (default of 128 KB), the available NICs are used in a round-robin fashion to send messages.
- For individual messages greater than `PSM3_MQ_RNDV_NIC_WINDOW`, messages are striped such that a single message can take advantage of more than one NIC.
- The `PSM3_MQ_RNDV_NIC_WINDOW` value may be decreased to enforce striping of messages smaller than the default.

  Messages will not be striped at a size lower than `PSM3_MQ_RNDV_NIC_THRESH` because an eager, instead of rendezvous, receive protocol is used. Multi-rail striping only occurs for rendezvous. When decreasing this value beware that the additional CPU and network overhead for setting up the rendezvous transfers may defeat any bandwidth gained by striping smaller messages.

If `PSM3_MULTIRAIL` is not set or set to 0, the following occurs:

- Each rank sends all its messages using a single NUMA-local or nearest NIC.

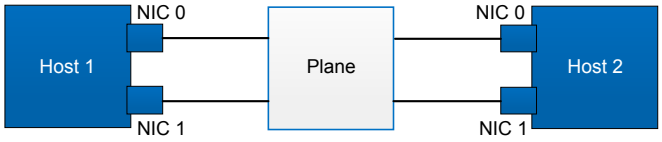
For more information see PSM3 OFI Provider.

7.7.4 Multi-Rail Configuration Examples

This section contains examples of multi-rail used in a single plane and in multiple planes.

Single plane, multi-rail

The following figure shows an example of a single plane with each host having two NICs. In this example, all NICs have the same Ethernet IP subnet.



Example Environment Variables

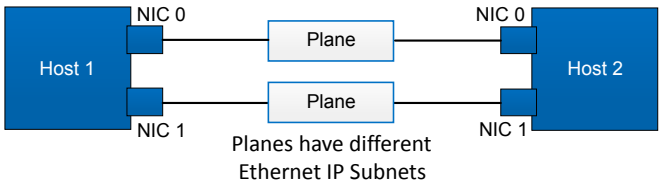
- `PSM3_MULTIRAIL` is not set. Each PSM3 rank uses a single NIC . If you do not want PSM3 to potentially use both NICs, you must specify the single desired NIC using `PSM3_NIC`.
- `PSM3_MULTIRAIL=1`. PSM3 discovers that there are two NICs in the system. The first available NIC is 0. The next available NIC is 1. PSM3, by default, uses a `PSM3_MULTIRAIL_MAP` of 0,1. PSM3 sets up the first (primary) connection over NIC 0, and sets up the second (secondary) connection over NIC 1.
- `PSM3_MULTIRAIL=1` and `PSM3_MULTIRAIL_MAP=1,0`. PSM3 uses the given units as specified. PSM3 sets up the primary connection over NIC 1 and sets up the secondary connection over NIC 0.
- `PSM3_MULTIRAIL=2` is set. Each rank will use only NICs on the same NUMA node as the rank unless there are none, then it will use NICs on other NUMA nodes. Refer to PSM3 OFI Provider for more informaiton.

NOTE

If some or all of the NICs have different Ethernet IP subnets, `PSM3_ALLOW_ROUTERS=1` must be specified, in which case PSM3 will ignore any differences in Ethernet IP subnets and assume each NIC can communicate with all the other NICs.

Multi-plane

The following figure shows an example of multiple planes with different Ethernet IP subnets. In this example, all NICs within a given plane are on the same Ethernet IP subnet.



NOTE

The Ethernet IP subnets for any dual plane configuration must be unique per plane.

Example Environment Variables

- `PSM3_MULTIRAIL` is not set. PSM3 may not work because there are multiple planes. NIC 0 on first host has no connection to NIC 1 on second host.
- `PSM3_MULTIRAIL=1`. PSM3 discovers that there are two NICs in the system. PSM3, by default, uses a `PSM3_MULTIRAIL_MAP` of 0,1. PSM3 connects the primary rail on NIC 0 of the left host with NIC 0 on the right host. The secondary rail is set up on NIC 1 of the left host with NIC 1 of the right host. PSM3 works in this configuration/setting.
- `PSM3_MULTIRAIL=1` and `PSM3_MULTIRAIL_MAP=1,0`. PSM3 uses the given units as specified. PSM3 does not reorder them. Both hosts use NIC 1 to make the connection for the primary subnet via the primary rail, and set up the secondary rail over NIC 0 on both sides. PSM3 works fine in this configuration.
- `PSM3_MULTIRAIL=2`. PSM3 may not work because there are multiple planes. NIC 0 on first host has no connection to NIC 1 on second host.

NOTE

If some or all of the NICs within a given plane have different Ethernet IP subnets, `PSM3_ALLOW_ROUTERS=1` must be specified, in which case PSM3 will ignore any differences in Ethernet IP subnets and assume each NIC can communicate with all the other NICs. In this situation, PSM3 cannot distinguish the planes based on their Ethernet IP subnets, so `PSM3_MULTIRAIL=1` and `PSM3_MULTIRAIL_MAP` must be specified so each PSM3 process properly uses the two planes in a primary/secondary configuration.

7.8 PSM3 Two-Sided Messaging

Intel® PSM3 implements a queue-based communication model with tag matching in which message consumers use metadata to match incoming messages against a list of preposted receive buffers. This Matched Queue (MQ) mechanism has semantics that are consistent with those presented by MPI 1.2, and all the features and side-effects of message passing find their way into PSM3 queues.

A successful tag match requires that the tag provided by the receiver matches the tag provided by the sender for every message sent . Since MQ two-sided message passing is a receiver-directed communication model, the tag matching done at the receiver involves matching a sent message's send tag with the tag and tag selector attached to every preposted receive buffer. The incoming send tag is compared to the posted receive tag but only for the portion specified in the tag selector. The tag selector can be used to mask off parts (or even all) of the bitwise comparison between sender and receiver tags. Receivers may also specify where the message must come from. A successful match causes the message to be received into the buffer where the tag is matched. If the incoming message is too large, it is truncated to the size of the posted receive buffer.

MQ messages are either received as expected or unexpected:

- The received message is *expected* if the incoming message tag matches the combination of tag and tag selector of at least one of the user-provided preposted receive buffers .

- The received message is *unexpected* if the incoming message tag does not match any combination of tag and tag selector from all the user-provided preposted receive buffers.

Unexpected messages are messages buffered by the PSM3 provider until a receive buffer that can match the unexpected message is provided. With PSM3 MQ two-sided messaging and MPI alike, unexpected messages can occur as a side-effect of the programming model, whereby the arrival of messages can be slightly out of step with receive buffer ordering. Unexpected messages can also be triggered by the difference between the rate at which a sender produces messages and the rate at which a paired receiver can post buffers and hence consume the messages.

In all cases, too many unexpected messages can negatively affect performance. Use some of the following mechanisms to reduce the effect of added memory allocations and copies that result from unexpected messages:

- If and when possible, receive buffers should be posted as early as possible.
- Use rendezvous messaging that can be controlled with PSM3_MQ_RNDV_NIC_THRESH and PSM3_MQ_RNDV_SHM_THRESH options. These options default to values determined to make effective use of bandwidth, and are not advisable for all communication message sizes. However, rendezvous messaging inherently prevents unexpected messages by synchronizing the sender with the receiver.
- PSM3 MQ statistics, such as the amount of received unexpected messages and the aggregate amount of unexpected bytes, are available via PSM3_PRINT_STATS and PSM3_PRINT_STATSMASK.

Whenever a match occurs, whether the message is expected or unexpected, the message may be truncated. Message truncation occurs when the size of the preposted buffer is less than the size of the incoming matched message. MQ correctly handles message truncation by always copying the appropriate amount of bytes to ensure it does not overwrite any receiver data. It is valid to send less data than the amount of data that has been preposted.

Message completion in Matched Queues follows local completion semantics. When sending a message, it is deemed complete when PSM3 guarantees that the source data has been queued to be sent and that the entire input source data memory location can be safely overwritten. As with standard MPI two-sided message passing, MQ does not make any remote completion guarantees for sends. MQ does however, allow a sender to send a synchronous message and uses the MPI two-sided definition of synchronous. For synchronous messages a send completion is reported only after a matching receive buffer has been posted by the receiver. Synchronous send completions also wait until the source data memory location can be safely overwritten, but can occur prior to the data being fully delivered to the receiver's buffer.

A receive is deemed complete after it has matched its associated receive buffer with an incoming send and that the data from the send has been completely delivered to the receive buffer.

Progress is typically ensured via periodic calls into PSM3 by the OFI application or middleware (often as a consequence of user application calls into the middleware). However, PSM3 also has a progress thread (controlled by PSM3_RCVTHREAD and PSM3_RCVTHREAD_FREQ) that periodically checks for incoming application messages or PSM3 control messages (such as acks) and makes forward progress.

7.9 RDMA Modes and Rendezvous Module

PSM3 supports multiple RDMA Modes. The various modes allow trade-offs between performance and memory footprint. All PSM3 modes make use of kernel bypass for low latency. Within the PSM3 protocol there are three basic types of communications:

- Control Messages: Messages that facilitate PSM internal operations such as connection establishment, credit exchange, and acknowledgments. They carry no application data.
- Eager Messages: Smaller application messages are classified as eager.
- Rendezvous Messages: Larger application messages and some application messages that request a synchronous end-to-end acknowledgment.

The transition point between Eager and Rendezvous is controlled by the PSM3_MQ_RNDV_NIC_THRESH (and PSM3_CUDA_THRESH_RNDV for CUDA* jobs) environment variable.

Eager messages allow the lowest overhead and latency. The Eager protocol makes use of end-to-end credit exchange and bounce buffers so that PSM3 can immediately initiate and take ownership of the transfer. Eager messages thus allow PSM3 to immediately report completion to the OFI application since the application buffer may now be reused.

Rendezvous messages perform an end-to-end Request-To-Send (RTS) Clear-To-Send (CTS) protocol prior to data transfer. The CTS is not issued by the receiver until tag matching (see PSM3 Two-Sided Messaging) has completed and successfully identified an OFI application buffer to receive the message. Upon receipt of the CTS, the sending PSM3 provider begins transfer of the message. For rendezvous messages, the OFI application completion is not reported until the transfer is done and the OFI application buffer is no longer being used. While rendezvous messages have more initial overhead, they permit the use of RDMA to perform zero-copy data transfers, reduce CPU overhead and achieve higher network performance. In addition, the rendezvous protocol provides greater receiver pacing such that transmission of large amounts of data will not overflow the receiver's buffers and cause inefficiencies such as packet retransmission or fabric flow control events.

For messages larger than PSM3_MQ_RNDV_NIC_WINDOW, the message will be split into multiple transmissions. A Single RTS is used, however a CTS is used for each transmission. These separate transmissions may by initiated in parallel and may be load balanced across multiple QPs (PSM3_QP_PER_NIC and PSM3_RV_QP_PER_CONN) or endpoints (PSM3_MULTIRAIL) for reduced latency and increased bandwidth.

When using RDMA for rendezvous messages, the application buffer must be pinned in physical memory and an RDMA Memory Region (MR) must be registered. As part of registering the MR, the MMU in the NIC is programmed with network virtual to physical address mappings and a `rkey` is created to remotely identify the MR. PSM3 exchanges the `rkey` in the CTS. This registration and exchange represents additional mandatory overheads in use of RDMA.

Use of RDMA also requires additional resources, namely:

- One or more RC QPs must be created for each connection.
- Each RC QP must have Work Requests (WQEs) to permit sending and receiving data.

- When using RC QPs for eager messages, each RC QP must also have a pool of receive buffers prepared to accept incoming eager data.

The PSM3_RDMA environment variable selects the RDMA mode to be used for a given job.

For the more advanced modes (PSM3_RDMA of 1, 2 or 3), a kernel rendezvous module (named `rv`) is used to assist PSM3 in performing these operations. Some of the important features of the rendezvous module include:

- Caching of RDMA Memory regions (MRs) to reduce overhead when application buffers are frequently used.
- Scalable implementation of shared RC QPs to reduce total QPs needed per endpoint and hence memory footprint.
- Multiple QP load balancing to increase bandwidth and reduce head of line blocking.
- RC QP connection recovery.
- Interacting with NVIDIA* GPU drivers to manage pinned GPU memory for use in GPUDirect* Copy, GPUDirect* Send DMA and GPUDirect* RDMA.

Comparing the PSM3_RDMA options:

- `0` – Only use UD QPs. This offers low latency and the best scalability. However single process bandwidth for large messages may be lower than other modes.
- `1` – Use rendezvous module for node to node level RC QPs for rendezvous. Large messages take advantage of RDMA via the rendezvous module. While the kernel calls add some modest latency, the benefits of RDMA and higher bandwidth may provide better application performance. This mode also has the lowest memory requirements as compared to mode `2` and `3`. Eager messages are handled the same as mode `0`.
- `2` – Use user space RC QPs for rendezvous. Large messages take advantage of RDMA via user space RC QPs in each PSM process. This avoids the kernel call overhead for message transfer. The rendezvous module may be used for MR caching (See PSM3_MR_CACHE_MODE). However, this mode requires significantly more QPs per endpoint since each PSM process must establish RC QP connections to every other remote process in the job. For high process per node (ranks per node) jobs, the number of QPs can grow to be 1000s. For example, a 100-node job with 50 processes per node needs 99*(50*50)=247,500 RC QPs per node (plus the UD QPs for eager messages). Eager messages are handled the same as mode `0`.
- `3` – Use user space RC QPs for eager and rendezvous This augments mode `2` by also using the same RC QPs for eager messages. The RC QPs can offer slightly lower latency than the UD QP, however each RC QP must have a pool of receive buffers. So in addition to the high QP count in mode `2`, a significant amount of additional memory is required for per QP eager receive buffers.

For PSM3_RDMA mode 1, the rendezvous module creates RC QPs in the kernel at the endpoint-to-endpoint level and shares these QPs across all PSM3 processes in a given job using a given NIC. This significantly reduces the number of QPs and communications memory required.

In modes 1, 2, and 3, the rendezvous module may also provide MR caching. For mode 2 and 3, use of the rendezvous module cache is controlled by PSM3_MR_CACHE_MODE. For mode 1, the rendezvous module MR cache is always

used. The rendezvous module MR cache retains MRs after they are done being used, so that future RDMA use of the same buffer can avoid memory registration overheads. The cache registers itself with the kernel MMU notifier mechanism so that any buffers which the application frees will be automatically purged from the cache. The size of the rendezvous module MR cache per process is controlled via the `mr_cache_size` kernel module parameter for the `rv` module. Values for `mr_cache_size` are specified in units of megabytes.

In addition, for modes 1, 2, and 3, a user space MR table is retained. The table reference counts all currently in use MRs and permits concurrent transfers (such as messages that have been split into multiple transmissions) using the same buffer to share the same MR for improved efficiency. This table is sized via PSM3_MR_CACHE_SIZE.

When using mode 0 or 1, PSM3 is extremely resilient to packet loss and fabric disruptions. In mode 1, PSM3 coordinates with the rendezvous module to retransmit lost RDMA messages and the rendezvous module will recover affected RC QPs. This allows PSM3 to ride through disruptions which would exceed the traditional limitations of RDMA RC QP timeout and retry mechanisms (as controlled via PSM3_QP_TIMEOUT and PSM3_QP_RETRY). See PSM3_RV_RECONNECT_TIMEOUT for more information.

NOTE

For non-CUDA* applications, RDMA mode 0 does not use the rendezvous module. However, when GPUDirect* is enabled, all RDMA modes require the CUDA enabled rendezvous module. See Intel® Ethernet Fabric Suite PSM3 Support for GPUDirect*

NOTE

GPUDirect* is not allowed with RDMA mode 2 or 3. When this combination is specified, a warning is reported and mode 1 is used.

7.10 PSM3 Rendezvous Kernel Module

For PSM3's more advanced modes (PSM3_RDMA of 1, 2 or 3 or PSM3_GPUDIRECT), a kernel rendezvous module (named `rv`) is used to assist PSM3 in performing RDMA and/or GPU operations. Some of the important features of the rendezvous module include:

- Caching of RDMA Memory regions (MRs) to reduce overhead when application buffers are frequently used.
- Scalable implementation of shared RC QPs to reduce total QPs needed per endpoint and hence memory footprint.
- Multiple QP load balancing to increase bandwidth and reduce head of line blocking.
- RC QP connection recovery.
- Interacting with NVIDIA* GPU drivers to manage pinned GPU memory for use in GPUDirect* Copy, GPUDirect* Send DMA and GPUDirect* RDMA.

Comparing the PSM3_RDMA options:

- `0` – Only use UD QPs. The rendezvous module is not used unless PSM3_GPUDIRECT is enabled.

- `1` – Use rendezvous module for node to node level RC QPs for messages using the rendezvous protocol.
- `2` – Use user space RC QPs for rendezvous. The rendezvous module may be used for MR caching (See PSM3_MR_CACHE_MODE).
- `3` – Use user space RC QPs for eager and rendezvous. The rendezvous module may be used for MR caching (See PSM3_MR_CACHE_MODE).

For PSM3_RDMA mode 1, the rendezvous module creates RC QPs in the kernel at the endpoint-to-endpoint level and shares these QPs across all PSM3 processes in a given job using a given NIC. This significantly reduces the number of QPs and communications memory required.

In modes 1, 2, and 3, the rendezvous module may also provide MR caching. For mode 2 and 3, use of the rendezvous module cache is controlled by PSM3_MR_CACHE_MODE. For mode 1, the rendezvous module MR cache is always used. The rendezvous module MR cache retains MRs after they are done being used, so that future RDMA use of the same buffer can avoid memory registration overheads. The cache registers itself with the kernel MMU notifier mechanism so that any buffers which the application frees will be automatically purged from the cache.

The rendezvous module has the following kernel parameters:

- `mr_cache_size` - The size of the rendezvous module CPU MR cache per process in units of megabytes. May be controlled per job via PSM3_RV_MR_CACHE_SIZE. Default is 256.
- `mr_cache_size_gpu` - The size of the rendezvous module CPU MR cache per process in units of megabytes for jobs where PSM3_CUDA is enabled. May be controlled per job via PSM3_RV_MR_CACHE_SIZE. Default is 1024.
- `gpu_cache_size` - The size of the rendezvous module GPU registration cache per process in units of megabytes for jobs not using GPUDirect* RDMA. May be controlled per job via PSM3_RV_GPU_CACHE_SIZE. Default is 256.
- `gpu_rdma_cache_size` - The size of the rendezvous module GPU registration cache per process in units of megabytes for jobs using GPUDirect* RDMA. May be controlled per job via PSM3_RV_GPU_CACHE_SIZE. Default is 1024.
- `num_conn` - The number of RC QPs per rendezvous module node to node connection for PSM3_RDMA mode 1. May be controlled per job via PSM3_RV_QP_PER_CONN. Default is 4.
- `q_depth` - Sets the maximum concurrent queued IOs per node to node connection in the rendezvous module. May be controlled per job via PSM3_RV_Q_DEPTH. A node to node connection consists of `num_conn` RC QPs, a send CQ and a recv CQ. Each will be sized such that up to `q_depth` total IOs can be queued for a given node to node connection. Default is 4000.
- `service_id` - The service ID to be used by the rendezvous module when establishing RC QP connections via the Connection Manager (CM) in the kernel for the job. May be controlled per job via PSM3_IB_SERVICE_ID. Default is 0x1000125500000001ULL.
- `enable_user_mr` - Enable user mode MR caching. When 1, the rendezvous module can cache user space MRs for CPU Send DMA, GPUDirect* Send DMA or PSM3_RDMA mode 2 and 3 when PSM3_MR_CACHE_MODE is 1. Default is 0.

User mode MR caching can be used for:

- PSM3_RDMA mode 2 and 3 when PSM3_MR_CACHE_MODE is 1. When `enable_user_mr` is not enabled, such caching is not allowed and PSM3_RDMA mode 2 and 3 may only be used with PSM3_MR_CACHE_MODE set to 0.
- Any PSM3_RDMA mode when CPU Send DMA or GPUDirect* Send DMA is enabled. However if `enable_user_mr` is not enabled, Send DMA will disable itself with a warning.

7.10.1 More Information on Configuring and Loading Drivers

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

Also refer to the `/usr/share/doc/initscripts-*/sysconfig.txt` file for general information on configuration files.

7.11 PSM3 and NVIDIA* CUDA* Support

PSM3 supports GPU buffer transfers through NVIDIA CUDA, GPUDirect* Copy, GPUDirect* Send DMA and GPUDirect* RDMA. This support is integrated in conjunction with a CUDA-enabled rendezvous kernel module (rv). To use this feature, both PSM3 and the rendezvous kernel module must be CUDA-enabled and present in the system. When enabled, PSM3 helps accelerate transfers of GPU memory buffers with Intel®'s RoCE NICs. You must enable this feature at runtime.

PSM3's CUDA* support includes the following capabilities:

- GPUDirect* Copy - For smaller messages, data copies may be more efficient than DMA. This feature allows for optimized copies to and from the GPU.
- GPUDirect* Send DMA - For medium sized messages, DMA by the sender may be more efficient than full RDMA. This feature allows for the sender to DMA directly from the GPU.
- GPUDirect* RDMA - For larger messages, RDMA provides benefits as outlined in RDMA Modes and Rendezvous Module. This feature allows both the sender and receiver to DMA directly from and to the GPU.
- CUDA* Copies - In some situations, or when GPUDirect* is disabled within PSM3, PSM3 may use CUDA* calls to copy data out of and/or into the GPU. The algorithms in PSM3 are optimized to take advantage of async CUDA* copy mechanisms to optimize performance for larger messages. Where appropriate, PSM3 may also pipeline such copies to increase performance.

NOTE

Since there are additional checks in software critical paths (and such checks may have a minor performance impact), it is only recommended that you only enable this feature if you need CUDA-based support.

NOTE

There are no performance penalties for using a CUDA* enabled rendezvous module (rv) during a PSM3 job which has not enabled PSM3 CUDA* support.

By default, CUDA support is disabled. To enable it at runtime, refer to PSM3_CUDA and PSM3_GPUDIRECT. These environment variables must be set before the application is launched. Additionally, if an MPI or middleware application is used, then both the MPI and middleware typically need to be CUDA-enabled.

When enabled, PSM3 will check the locality of all buffers passed into send and receive operations. When appropriate, PSM3 in conjunction with the rendezvous driver will enable the Intel® Ethernet Fabric Suite NIC to directly read from and write into the GPU buffer. This enhanced behavior eliminates the need for an application or middleware to move a GPU-based buffer to host memory before using it in a PSM3 operation, providing a performance advantage.

CUDA support is limited to using a single GPU per process. The application should set up the CUDA runtime and pre-select a GPU card (through the use of `cudaSetDevice()` or a similar CUDA API) prior to calling OFI or `MPI_Init()`, if using MPI. While systems with a single GPU may not have this requirement, systems with multiple GPUs may see reduced performance without proper initialization. Therefore, it is recommended that you initialize the CUDA runtime before the OFI or `MPI_Init()` call.

Sizing the GPU Registration Cache

When using any of the PSM3 GPUDirect* features, the PSM3 Rendezvous Kernel Module will be used for managing pinned GPU memory and registering GPU MRs. In this role, the PSM3 Rendezvous Kernel Module maintains a GPU registration cache per process to optimize performance for GPU buffers which are frequently used for PSM3 IOs involving any of the GPUDirect* features. The size of this per process cache is controllable via PSM3 Rendezvous Kernel Module module parameters (`gpu_cache_size` and `gpu_rdma_cache_size`) and can be overridden by the PSM3 env variable PSM3_RV_GPU_CACHE_SIZE.

The GPU registration cache must be sized less than the practical limits of the given GPU model's ability to pin memory. The upper bound of memory pinning for the GPU can be found by looking at the `BAR1 Memory Usage Total` reported by `nvidia-smi -q`. However, the practical limit is often smaller as CUDA* may reserve some BAR space for itself and other elements of CUDA* may also consume BAR space.

To determine the GPU registration cache's behaviors during a given run, use PSM3_PRINT_STATS and review it's output files. If a non-zero value is observed for the `rv_gpu_failed_pin` counter, this indicates the nVidia driver was unable to fulfill some memory pinning requests. Typically this is due to reaching the practical limit of pinned memory and BAR space for the given GPU model. When this is observed, the `rv_gpu_max_size` value will indicate the maximum memory (in Megabytes) which was successfully pinned during the job and can be a reasonable upper limit for the GPU registration cache size used in future runs on this GPU device.

NOTE

When decreasing the GPU registration cache size, other PSM3 parameters may also need to be adjusted. See PSM3_RV_GPU_CACHE_SIZE for more details about the relationships between GPU registration cache size and other PSM3 parameters.

NOTE

Typically, only 1 process (or rank) is assigned to a given GPU. If more than 1 process (or rank) will be run per GPU, the per process GPU registration cache size must also be correspondingly reduced (e.g. if 4 processes will share a GPU, the per process GPU registration cache must be cut to 1/4 of the practical limit of GPU pinned memory). In this case, any analysis of PSM3_PRINT_STATS output must consider all per-process output files from the job and use the sum of `rv_gpu_failed_pin` across all processes to determine if the practical limit has been reached and the sum of `rv_gpu_size` at each point in the job will indicate maximum total GPU memory successfully pinned at that point in the job.

Notes for Middleware Developers

PSM3 indicates it's runtime support for CUDA* via the `FI_HMEM` OFI (libfabric) capability flag. This flag is only reported when a CUDA* enabled version of PSM3 is used in conjunction with runtime enablement of CUDA* via `PSM3_CUDA` and/or `PSM3_GPUDIRECT`. When `FI_HMEM` is reported, PSM3 checks the location of all IO buffers, in which case, the middleware should not need to pre-check the buffer locations or move buffers to host memory before passing them into OFI (libfabric) APIs. Doing so may cause performance degradation. If developers are adding CUDA support to existing middlewares, Intel recommends minimal or no processing of the buffer before passing it into OFI APIs.

OFI APIs accept void* data types for buffer pointers, thus making it generic for both host and GPU based buffers.

It is worth mentioning that some MPI and middleware implementations may require special handling for collective operations, especially for Reduction operations performed within the middleware against GPU buffers. In which case some high-level middleware GPU support may be necessary if implementing support for collectives.

7.12 Environment Variables

This section describes how to control PSM3 behavior using environment variables.

7.12.1 FI_PSM3_INJECT_SIZE

Controls the limit for use of the OFI inject strategy for message sending and completion handling. Only message sizes below this value will use that strategy. The inject strategy can offer improved message rate for smaller messages.

Default: `64`

NOTE

A value of `512` may offer better performance for many applications.

7.12.2 FI_PSM3_LAZY_CONN

Controls when connections are established. The lazy connection model connects pairs of endpoints on first use of the given communication path. For jobs that exhibit sparse communication patterns, such as ring-based communications or nearest neighbor

communications, this can speed job startup and reduce the amount of memory needed for communications resources. However, for jobs that use most of the communications paths, such as those that make use of AlltoAll collectives, performing connections at job start can be more efficient.

Options:

- `0` – Disabled. Establishes all connections when requested via OFI application, typically during job startup (default).
- `1` – Enabled. Establishes each connection only when first used for communications with the given remote endpoint.

Default: `FI_PSM3_LAZY_CONN=0` (disabled)

NOTE

The lazy connection model requires that both sides of the connection participate in its establishment and that both sides indicate the need for such a connection (such as posting an application receive request on one side and posting an application send request on the other). If the application is not well synchronized for these operations on both ends of the connection, one side may end up waiting a significant amount of time. Such wait time is considered part of the connection process and counted against the connect timeout. To help avoid lazy connection failures, during lazy connection establishment a connect timeout of 30 seconds is used by default. In some cases it may be necessary to increase this timeout by specifying PSM3_CONNECT_TIMEOUT.

NOTE

Due to PSM3's scalability and low memory footprint characteristics, use of `FI_PSM3_LAZY_CONN` is typically unnecessary and may have other side effects, its use is hence discouraged.

7.12.3 FI_PSM3_UUID

Sets the Universally Unique Identifier (UUID) per job. All processes in a given job must use the same value. It is preferable to use a unique value per concurrent job, especially when more than one job is running on a given endpoint at a time. PSM3 uses this value to detect potentially stale connection or disconnection attempts and the rendezvous module uses it to identify each job so it can separate QP resources so that they are not shared across jobs. The value is also used to seed hash functions and linux intra-node Interprocess communications (IPC) used by PSM3_NIC_SELECTION_ALG to distribute processes among multiple NICs. The value must be specified as a string of 16 hex digits separated by dashes at the exact points shown in the default.

Default: `00FF00FF-0000-0000-0000-00FF00FF00FF`

7.12.4 PSM3_ALLOW_ROUTERS

Indicates whether endpoints with different IP subnets should be considered accessible.

- `0` – Consider end points with different IPv4 subnets unaccessible.
- `1` – Consider all endpoints accessible, even if they have different IPv4 subnets

Default: `0`

The IPv4 subnet is defined by taking the IPv4 address for a NIC and masking it with the IPv4 subnet mask. When set to `1`, PSM3 will assume IP routers are configured in the network such that all endpoints can communicate with each other regardless of IPv4 subnet. When set to `0` PSM3 will assume a multi-subnet configuration where individual subnets are separated and unable to communicate. See *Intel® Ethernet Fabric Suite Host Software User Guide*, Multi-Rail Support in PSM3 for more details about multi-subnet configurations.

If set incorrectly, it may prevent jobs from executing which require NICs in different subnets to communicate. This variable can influence the association of NICs to processes for PSM3_MULTIRAIL.

7.12.5 PSM3_CONNECT_TIMEOUT

Overrides the End-point connection timeout to allow for handling systems that may have a slow startup time. This value will override the timeout passed in `FI_PSM3_CONN_TIMEOUT`. Values are presented in seconds. Values used outside the valid range will be adjusted to fit within the valid range.

Options:

- `0` – Disabled
- `1` – Sets the timeout value to 2 seconds.
- Enter a timeout value from 2 (minimum) to 9,223,372,036 (maximum) in seconds.

Default: The value passed in by `FI_PSM3_CONN_TIMEOUT` (10 seconds).

7.12.6 PSM3_CUDA

Enables CUDA* support in PSM3 when set. Requires the PSM3 provider to be compiled with CUDA* support.

For additional details, see the *Intel® Ethernet Fabric Performance Tuning Guide*.

NOTE

If GPU buffers are used in the workloads and `PSM3_CUDA` is not set to 1, undefined behavior will result.

Default: `PSM3_CUDA=0`

See also: PSM3_GPUDIRECT

7.12.7 PSM3_CUDA_THRESH_RNDV

Sets the eager-to-rendezvous switchover threshold in bytes for messages sent from a GPU buffer. Rendezvous is used for larger messages and for PSM3_RDMA>= 1, uses RDMA for both transmit and receive. Smaller values lead to increased bandwidth; larger values lead to decreased latency. Tuning this value is complex and dependent on PSM3_MQ_RNDV_NIC_WINDOW.

Options:

- Any value between 1 and 4 GB. Larger values may disable the threshold entirely.

Default : `PSM3_CUDA_THRESH_RNDV=8000`

See RDMA Modes and Rendezvous Module for more details.

Also see PSM3_MQ_RNDV_NIC_THRESH.

7.12.8 PSM3_DEBUG_FILENAME

Controls where additional debug output which has been selected by PSM3_TRACEMASK will be placed. The filename specified may use markers of `%h` and `%p` which will be replaced with the hostname and process id respectively. This may be useful when output is going to a shared filesystem. When not specified, output from multiple processes will be intermingled within the same file. In either case each line is output with the process label (typically hostname.rank#).

For example, `PSM3_DEBUG_FILENAME=debug.%h.%p` may generate files such as `debug.host01.678` and `debug.host01.679` for job processes 678 and 679 on host01.

Default: unspecified, any enabled debug output goes to `stdout`.

NOTE

Informative messages and error messages are only output to stdout.

7.12.9 PSM3_DEVICES

Enables one or more of the following devices for communication:

- `self` - allows a process to send messages to itself
- `shm` - allows a process to send messages to other processes on the same host via linux shared memory
- `nic` - allows a process to send messages to processes on other hosts

For PSM3 jobs that do not require shared-memory communications, PSM3_DEVICES can be specified as `self,nic`. Similarly, for shared-memory only jobs, PSM3_DEVICES can be specified as `shm,self`. You must ensure that the endpoint included in a job does not require a device that has been explicitly disabled (eg. omitted). In some instances, enabling only the devices that are required may improve performance.

Default: `PSM3_DEVICES="self,shm,nic"`

For shared-memory only jobs use: `PSM3_DEVICES="self,shm"`

7.12.10 PSM3_DISABLE_MMAP_MALLOC

Disables `mmap` for `malloc()`.

Uses `glibc mallopt()` to disable all uses of `mmap` by setting `M_MMAP_MAX` to 0 and `M_TRIM_THRESHOLD` to -1. Refer to the Linux* man page for `mallopt()` for details.

Default: `PSM3_DISABLE_MMAP_MALLOC=NO`

NOTE

Choosing `YES` may reduce the memory footprint required by your program, at the potential expense of increasing CPU overhead associated with memory allocation and memory freeing. The default `NO` option is better for performance.

7.12.11 PSM3_ERRCHK_TIMEOUT

Controls the timeouts used for error recovery from lost user space packets. The syntax is:

```
PSM3_ERRCHK_TIMEOUT=min[:max[:factor]]
```

Default value: `PSM3_ERRCHK_TIMEOUT=160:640:2`. If a field is omitted, its default value will be used.

Fields:

`min` `max`	The values of `min` and `max` set the range of timeouts to use when waiting for acknowledgments. The values are in units of milliseconds. For example, values of 160:640 mean that timeouts start at 160 milliseconds but can go as large as 640 milliseconds. If the `max` value specified is less than the `min`, the value of `min` is used as both the `min` and `max`.
`factor`	`factor` controls how aggressively the timeout is adjusted within the specified range. `factor` specifies the value used to multiply the currently selected timeout value. Adjustment means that subsequent packet lost packet recovery waits longer for the acknowledgment. For example when 160:640:2 is used, the first error recovery for a given packet will allow up to 160 milliseconds before attempting recovery. The second recovery will wait 320 milliseconds and the third will wait 640 milliseconds. Any subsequent error recovery attempts will each wait 640 milliseconds.

NOTE

Timeout selection is a trade-off between how quickly lost packets are recovered from vs the additional fabric load due to packets used for recovery. For example, if packets are delayed longer than the selected timeout due to congestion, but ultimately arrive, PSM3 may unnecessarily issue additional error recovery packets which can make the congestion worse. Use of larger `max` and `factor` values can help mitigate the error recovery load induced by PSM3 when the fabric is highly congested or recovering from temporary outages (such as rerouting around lost links or switches).

7.12.12 PSM3_FLOW_CREDITS

The number of concurrent unacked packets (credits) permitted per flow. A flow is a stream of packets between a specific pair of endpoints.

Decreasing this value may help to reduce network pressure, at the possible expense of overall message rate or bandwidth.

Default: 64

7.12.13 PSM3_GPUDIRECT

GPUDirect* is a technology that enables a direct path for data exchange between a graphics processing unit (GPU) and a third-party peer device using standard features of PCI Express. For more information, see the NVIDIA* CUDA* documentation: https://developer.nvidia.com/gpudirect and http://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

When set, this enables GPUDirect* Copy, GPUDirect* Send DMA and GPUDirect* RDMA support. This allows increased performance through direct data exchange between GPU and NIC. When enabled, the rendezvous driver is required with CUDA* support. For details, see the *Intel® Ethernet Fabric Suite Software Installation Guide*.

Default: `PSM3_GPUDIRECT=0`

NOTE

When `PSM3_GPUDIRECT=1`, this implicitly also sets `PSM3_CUDA=1`

Also see: PSM3_CUDA

7.12.14 PSM3_IB_SERVICE_ID

Sets the service ID to be used by the rendezvous module when establishing RC QP connections via the Connection Manager (CM) in the kernel for the job. If a value of 0 is specified, then the rendezvous module's `service_id` module parameter controls the selection.

Default: `PSM3_IB_SERVICE_ID=0x1000125500000001`

The same service id can safely be shared by multiple jobs. However, all jobs which are using the same PSM3_FI_UUID (as defaulted, set explicitly or set via the MPI middleware) must use the same service id.

7.12.15 PSM3_IDENTIFY

Enable verbose output of process PSM3 software version identification including library location, build date, CUDA* support, rendezvous module API version (if rendezvous module is being used, see RDMA Modes and Rendezvous Module), process rank IDs, total ranks per node, total ranks in the job, CPU core, and NIC(s) selected.

Options:

- `0` – disable - no output.
- `1` – enabled on all processes.
- `1:` – enabled only on rank 0 (abbreviation for `PSM3_IDENTIFY=1:*:rank0`).
- `1:pattern` – enabled only on processes whose label matches the extended glob pattern.

Default: `0`

NOTE

For more information on extended glob patterns see the linux man pages for glob(7) and fnmatch(3).

The label for a process is typically of the form `hostname:rank#` such as `myhost047:rank3` where rank is the relative process number in the job. If MPI, the MPI runtime and the job scheduler have not indicated the rank to PSM3, the label will be of the form `hostname:pid#` where pid is the linux process id. The form of labels for a given cluster can be observed at the beginning of various PSM3 output messages, such as those from `PSM3_IDENTIFY`.

Some Example uses of patterns:

- `PSM3_IDENTIFY=1*:rank0` – only identify rank 0. When the user is confident that the same PSM3 software and configuration is installed on all nodes used in the job, this can provide a more concise output.
- `PSM3_IDENTIFY=1:myhost047:*` - only identify processes on myhost047. All processes on that host will provide output. This can be helpful if only a single host's configuration is suspect.
- `PSM3_IDENTIFY=1:+(*:rank0|*:rank1)` – only identify rank 0 and 1. This is an example of an extended glob pattern.

NOTE

Depending on how jobs are launched, patterns may need to be enclosed in single quotes to prevent expansion of wildcards against local filenames during the launch script.

NOTE

Output occurs for every selected process in the job. As such this can generate quite a lot of output especially when all processes are selected in a high process count job.

In the PSM3_IDENTIFY output, CUDA* enablement of the PSM3 OFI Provider is indicated via `-cuda` after the PSM protocol version. Such as: `PSM3_IDENTIFY PSM3 v3.0-cuda built for IEFS version`. Non-CUDA enabled PSM3 OFI Providers appear such as: `PSM3_IDENTIFY PSM3 v3.0 built for IEFS version`.

If PSM3 was built with rendezvous module (rv) support, the PSM3_IDENTIFY output will indicate it, such as: `PSM3_IDENTIFY built against rv interface v1.1`. If PSM3 was built with rendezvous module and CUDA* support, the rendezvous GPU API version is shown, such as `PSM3_IDENTIFY built against rv interface v1.1 gpu v1.0 cuda`.

When PSM3 is using the rendezvous module (rv) for the given job, the PSM3_IDENTIFY output will indicate the version of rendezvous module API loaded, such as: `PSM3_IDENTIFY run-time rv interface v1.1 mr`. When "mr" is present it indicates `enable_user_mr` is enabled within the rendezvous module. If both PSM3 and rendezvous module support CUDA*, the rendezvous GPU API version is also shown such as: `PSM3_IDENTIFY run-time rv interface v1.1 mr gpu`

`v1.0 cuda`. Finally, if the rendezvous module supports CUDA*, but PSM3 does not, this will be indicated such as `PSM3_IDENTIFY run-time rv interface v1.1 mr cuda`. Also see PSM3 Rendezvous Kernel Module.

7.12.16 PSM3_MEMORY

Sets the memory usage mode. Controls the amount of memory used for MQ entries by setting the number of entries. Setting this value also sets PSM3_MQ_RECVREQS_MAX and PSM3_MQ_RNDV_NIC_THRESH to preset internal values, see Options for details.

Options:

NOTE

You must enter the desired option as text, not a numerical value.

- `min` – Reserves memory to hold 65536 pending requests.
- `normal` – Reserves memory to hold 1048576 pending requests.
- `large` – Reserves memory to hold 16777216 pending requests.

Default: `PSM3_MEMORY=normal`

7.12.17 PSM3_MQ_RECVREQS_MAX

Sets the maximum number of `irecv` requests pending completion.

- PSM3_MQ_RECVREQS_MAX must be a power-of-two (2^n) value.
- When PSM3_MEMORY is `min` or `normal`, PSM3_MQ_RECVREQS_MAX must be at least `1024`.
- When PSM3_MEMORY is `large`, PSM3_MQ_RECVREQS_MAX must be at least `8192`.

Default: `PSM3_MQ_RECVREQS_MAX=1048576`

7.12.18 PSM3_MQ_RNDV_NIC_THRESH

Sets the eager-to-rendezvous switchover threshold in bytes for messages sent from a CPU buffer. Rendezvous is used for larger messages and for PSM3_RDMA=1, 2, or 3, uses RDMA for both transmit and receive. Smaller values lead to increased bandwidth; larger values lead to decreased latency. Tuning this value is complex and dependent on PSM3_MQ_RNDV_NIC_WINDOW.

Options:

- Any value between 1 and 4 GB. Larger values may disable the threshold entirely.

Default : `PSM3_MQ_RNDV_NIC_THRESH=64000`

See RDMA Modes and Rendezvous Module for more details.

Also see PSM3_CUDA_THRESH_RNDV.

7.12.19 PSM3_MQ_RNDV_NIC_WINDOW

Sets the windowing size in bytes for how large messages are split for transmission.

Larger values may reduce CPU loading, smaller values may provide better distribution of bandwidth in workloads with many simultaneous destinations like an MPI collective operation, but will increase CPU loading. Additionally when PSM3_MULTIRAIL is active or PSM3_QP_PER_NICis > 1 or PSM3_RV_QP_PER_CONNis > 1, this value controls the granularity at which messages are striped across multiple NICs and/or QPs respectively.

Options:

- Any value between 1 and 4 MB; page-aligned values work best.

Defaults: `PSM3_MQ_RNDV_NIC_WINDOW=131072` when PSM3_CUDA disabled. Default is 2097152 when PSM3_CUDA enabled.

See RDMA Modes and Rendezvous Module for more details.

7.12.20 PSM3_MQ_RNDV_SHM_THRESH

Sets the threshold (in bytes) for shared memory eager-to-rendezvous switchover.

Default: `PSM3_MQ_RNDV_SHM_THRESH=16000`

7.12.21 PSM3_MQ_SENDREQS_MAX

Sets the maximum number of `isend` requests pending completion.

- PSM3_MQ_SENDREQS_MAX must be a power-of-two (2^n) value.
- When PSM3_MEMORY is `min` or `normal`, PSM3_MQ_SENDREQS_MAX must be at least `1024`.
- When PSM3_MEMORY is `large`, PSM3_MQ_SENDREQS_MAX must be at least `8192`.

Default: `PSM3_MQ_SENDREQS_MAX=1048576`

7.12.22 PSM3_MR_CACHE_MODE

Controls the user space MR cache as follows:

- `0` – No MR caching.
- `1` – Use rendezvous module for MR caching.

Default: `1`

When PSM3_RDMA selects mode 1, 2 or 3, a user space MR table is retained per local endpoint. This table permits MRs to be reference counted and reused if an OFI application simultaneously has more than one send or receive operation in flight using the same buffer. This may occur when an application is sending the same data to multiple remote processes (such as during MPI_Broadcast) or when PSM3 has divided a large message into multiple smaller rendezvous transmissions (see PSM3_MQ_RNDV_NIC_WINDOW).

Use of the rendezvous module for true MR caching can greatly reduce MR registration overhead when a set of buffers are repeatedly used for communications. Unlike the user space MR table, the rendezvous module MR cache will retain some MRs after they are done their current transfer, so they may be reused by future transfers. Such buffer reuse is common in many implementations of middleware collective algorithms such as MPI_AllReduce.

When PSM3_RDMA selects mode 1 or PSM3_GPUDIRECT is enabled, this setting is ignored and the rendezvous module MR cache is always used.

See RDMA Modes and Rendezvous Module, PSM3_MR_CACHE_SIZE and PSM3_RV_MR_CACHE_SIZE.

7.12.23 PSM3_MR_CACHE_SIZE

Sets the maximum number of MRs to retain per endpoint in the user space MR table.

Default: `8 * (PSM3_NUM_SEND_RDMA + 32)`.

If a value less than the minimum (PSM3_NUM_SEND_RDMA + 32) is specified, it will be automatically increased to the minimum.

See RDMA Modes and Rendezvous Module and PSM3_MR_CACHE_MODE.

7.12.24 PSM3_MTU

Sets upper bound on maximum packet size. The default MTU is controlled by the NIC driver. This may be used to decrease the value selected by the driver.

Valid values are:

- `<= 0` – use MTU selected by NIC driver (typically 4096).
- `1` – 256
- `2` – 512
- `3` – 1024
- `4` – 2048
- `5` – 4096
- `256`
- `512`
- `1024`
- `2048`
- `4096`

Default: -1

NOTE

Other values will be rounded up to the next larger valid value. Using a bad value or one larger than the MTU selected by the driver will silently use the driver selected value.

7.12.25 PSM3_MULTI_EP

Enables more than one PSM3 endpoint to be opened in a process.

Options:

- `0` Disabled.
- `1` Enabled.

Default: `1`

NOTE

For each endpoint opened, a full complement of rails and QPs will be used. See PSM3 Multi-Rail Support and PSM3_QP_PER_NIC.

NOTE

When using PSM3_GPUDIRECT an independent set of GPU registration caches are allocated per opened endpoint. This can negatively impact GPU BAR space consumption which can negatively impact performance.

7.12.26 PSM3_MULTIRAIL

Enables multi-rail capability so each process can use multiple network interface cards to transfer messages. The PSM3 multi-rail feature can be applied to a single plane with multiple rails (multiple NICs), or multiple planes.

Options:

- `0` – Single NIC per process configuration.
- `1` – Enable Multi-rail capability and each process will use all available NIC(s) in the system.
- `2` – Enable Multi-rail capability and limit NIC(s) used by a given process to NIC(s) within a single NUMA socket.

  PSM3 looks for at least one available NIC in the same NUMA socket on which you pin the task. If no such NICs are found, PSM3 falls back to `PSM3_MULTIRAIL=1` behavior and uses any other available NIC(s) for the given process. You are responsible for physical placement of NIC(s). Job launchers, middleware, and end users are responsible for correctly affinitizing MPI ranks and processes for best performance.

Default: `0`

When PSM3_QP_PER_NIC is specified `>1`, the specified number of QPs will be created per rail. The round robin distribution of traffic will vary the rails then the QPs in the sequence: first rail first QP, second rail first QP, first rail second QP, second rail second QP, etc.

For more detail on this feature please see PSM3 Multi-Rail Support. Also see PSM3_MULTIRAIL_MAP and PSM3_ALLOW_ROUTERS.

7.12.27 PSM3_MULTIRAIL_MAP

Tells PSM3 which NIC is to use for each rail. If only one rail is specified, it is equivalent to a single-rail case.

NOTE

`PSM3_MULTIRAIL_MAP` overrides any auto-selection and affinity logic in PSM3, regardless of whether PSM3_MULTIRAIL is set to 1 or 2. `PSM3_MULTIRAIL_MAP` is ignored when PSM3_MULTIRAIL is 0. For details, see PSM3 Multi-Rail Support.

Options: `unit,unit,unit,...`

The `unit` may be specified in the same manner as PSM3_NIC, namely as a Device Unit number or RDMA device name (as shown in `ibv_devices`). When a unit number is specified, it is relative to the alphabetic sort of the RDMA device names. Unit 0 is the 1st name.

Multiple rail specifications are separated by commas. A trailing comma will be ignored. In some cases extraneous whitespace may cause parse errors, so white space should be avoided.

It is valid to specify a given unit more than once, in which case an additional complement of PSM3_QP_PER_NIC QPs will be created and included in the round robin scheduling. For example, if `PSM3_QP_PER_NIC=2` and `PSM3_MULTIRAIL_MAP=0,0,1` a total of 4 QPs will be created on NIC 0 and 2 QPs on NIC 1. They will be scheduled as NIC 0 first QP, NIC 0 3rd QP, NIC 1 first QP, NIC 0 second QP, NIC 0 forth QP, NIC 1 second QP.

`PSM3_MULTIRAIL_MAP` may be used without PSM3_QP_PER_NIC to create the same effect.

- `PSM3_MULTIRAIL_MAP=0,0` and `PSM3_QP_PER_NIC=1` is equivalent to `PSM3_MULTIRAIL_MAP=0` and `PSM3_QP_PER_NIC=2`.
- `PSM3_MULTIRAIL_MAP=0,1,0,1` and `PSM3_QP_PER_NIC=1` is equivalent to `PSM3_MULTIRAIL_MAP=0,1` and `PSM3_QP_PER_NIC=2`.

NOTE

The first `unit` specified will be used as the primary rail for the process. The primary rail is used for connection establishment and some other non-load balanced control messages.

NOTE

The Intel® PSM3 implementation has a limit of thirty-two (32) NICs per node and 32 QPs per endpoint.

7.12.28 PSM3_NIC

Specifies the Device Unit number or RDMA device name (as shown in `ibv_devices`). When a unit number is specified, it is relative to the alphabetic sort of the RDMA device names. Unit 0 is the 1st name.

Default: `PSM3_NIC=any`. The NIC for each process is selected based on PSM3_NIC_SELECTION_ALG.

NOTE

PSM3 detects all RDMA devices, so if multiple types of RDMA devices are present, a device other than an Intel® Ethernet Fabric NIC may be selected. At this time, PSM3 is only supported for use with Intel® Ethernet Fabric NICs. See *Intel® Ethernet Fabric Suite Software Release Notes* for more details on devices supported.

NOTE

The Intel® PSM3 implementation has a limit of thirty-two (32) NICs per node.

NOTE

This parameter is ignored when PSM3_MULTIRAIL is 1 or 2.

7.12.29 PSM3_NIC_SELECTION_ALG

Specifies the algorithm to use for selecting the NIC per process when PSM3_NIC is not specified (or specifed as `any`) and PSM3_MULTIRAIL is 0.

For each of the algorithms below, only NICs whose port is active (`ibv_devinfo` shows `state PORT_ACTIVE`) will be considered. Additionally any RoCE NICs (`ibv_devinfo` shows `link_layer Ethernet` and `transport InfiniBand`) will only be considered if they have a RoCE IPv4 port GID (`ibv_devinfo -v` includes a port GID of the form: `0:ffff:xxxx:xxxx` where `xxxx:xxxx` is the IPv4 addr).

Options:

- `RoundRobin` or `rr` – selects a NIC on the same NUMA socket as the process. If more than 1 NIC is on the process' NUMA socket, those NICs will be distributed evenly across all processes on the given NUMA socket. Processes which have no NIC on their NUMA socket will simply be distributed across all available NICs.
- `Packed` or `p` – all processes will use the 1st available NIC
- `RoundRobinAll` or `rra` – The processes will be distributed across all available NICs without considering NUMA locality of the NIC vs process.

Default:

- If all NICs are on the same subnet - `RoundRobin`
- If all NICs are RoCE and PSM3_ALLOW_ROUTERS is enabled - `RoundRobin`
- Otherwise - `Packed`

NOTE

The above selection algorithms are only applied with regard to processes within the same job. If multiple jobs are run on the same node at the same time, the distribution of processes to NICs may be uneven or the jobs may each even use different values for `PSM3_NIC_SELECTION_ALG`.

NOTE

PSM3 detects all RDMA devices, so if multiple types of RDMA devices are present, a device other than an Intel® Ethernet Fabric NIC may be selected. At this time, PSM3 is only supported for use with Intel® Ethernet Fabric NICs. See *Intel® Ethernet Fabric Suite Software Release Notes* for more details on devices supported.

NOTE

The Intel® PSM3 implementation has a limit of thirty-two (32) NICs per node.

NOTE

PSM3_IDENTIFY may be used to show the rank identification and NIC selected for each process in a job.

7.12.30 PSM3_NUM_RECV_CQES

Sets the number of receive queue entries (CQEs) to allocate.

Default: `0`

When 0, in PSM3_RDMA modes 0, 1 and 2, the user space receive CQ is sized at PSM3_NUM_RECV_WQES + 1032 CQEs. In PSM3_RDMA mode 3, the user space receiver CQE is sized at PSM3_NUM_RECV_WQES + 5032 CQEs.

In all modes, a single completion queue (CQ) is used to handle incoming packet completions. Larger values may improve performance for some applications.

See RDMA Modes and Rendezvous Module for more details.

7.12.31 PSM3_NUM_RECV_WQES

Sets the number of receive WQEs to allocate.

Default: `4095`

In all PSM3_RDMA modes, the UD QP is sized at PSM3_NUM_RECV_WQES + 1032 WQEs. In PSM3_RDMA mode 3, the user space RC QP size is set to PSM3_NUM_RECV_WQES ÷ 4 WQEs.

In all modes, this also sets the number of UD receive eager bounce buffers (each of size PSM3_MTU) which are allocated per local endpoint. In PSM3_RDMA mode 3, an additional PSM3_NUM_RECV_WQES ÷ 4 buffers are also allocated per user space RC QP.

See RDMA Modes and Rendezvous Module for more details.

7.12.32 PSM3_NUM_SEND_RDMA

Sets the maximum number of concurrent outgoing RDMA writes to allow per local NIC.

Tuning this value can improve performance, but also can increase the amount of memory needed for send work request elements (WQEs) on queue pairs (QPs).

Default: `128`

See RDMA Modes and Rendezvous Module for more details.

7.12.33 PSM3_NUM_SEND_WQES

Sets the number of send WQEs to allocate.

Default: `4080`

The value affects QP allocation for each PSM3_RDMA mode as follows:

- `0` – Is the exact UD QP send Q size.
- `1` – Sets only UD QP send Q size.
- `2` – Sets UD QP send Q size. The user space RC QP size is controlled by PSM3_NUM_SEND_RDMA.
- `3` – Sets UD QP send Q size. The user space RC QP size is set to PSM3_NUM_SEND_WQES + PSM3_NUM_SEND_RDMA.

In all modes, this also sets the number of send bounce buffers (each of size PSM3_MTU) that are allocated per local endpoint.

See RDMA Modes and Rendezvous Module for more details.

7.12.34 PSM3_PRINT_STATS

Sets the frequency of PSM3 statistics output in units of seconds.

Options:

- `0` – Disables statistics output.
- `-1` – Performs statistics output once at the successful completion of a job.
- `>0` – Specifies the frequency of statistics output in seconds.

Default: `0`

When enabled, statistics are output for each local endpoint in each process on each node in a job to a local file whose name is of the form: `./psm3-perf-stat-[hostname]-[pid]`. The selection of statistics to output is controlled by PSM3_PRINT_STATSMASK. For each statistic, both the cumulative value as well as the delta (in parenthesis) since the previous output is shown.

7.12.35 PSM3_PRINT_STATSMASK

Selects which statistics will be included in PSM3_PRINT_STATS output.

Options:

The following selections may be summed to select more than one type of statistic:

- `1` – MQ message passing statistics
- `2` – CUDA* call statistics
- `0x000100` – Receive progress thread statistics

- 0x000200 – IPS protocol statistics
- 0x000400 – Rendezvous statistics
- 0x000800 – MR cache statistics
- 0x002000 – rendezvous module completion event statistics
- 0x004000 – rendezvous module connection statistics
- 0x100000 – Show zero values (Without this selection, only non-zero statistics are shown.)

Default: 0x00fffff

7.12.36 PSM3_QP_PER_NIC

Sets the number of UD QPs per local endpoint. In addition for PSM3_RDMA modes 2 and 3 this also sets the number of user space RC QPs to establish per remote endpoint .

Default: 1

For some applications, use of multiple QPs may increase performance, especially for large messages. However, this represents a multiplier to the amount of communications memory needed as each QP requires its own set of WQEs and receive buffers.

This feature interacts with PSM3_MULTIRAIL and PSM3_MULTIRAIL_MAP. When PSM3_QP_PER_NIC is specified >1, the specified number of QPs will be created per rail. The round robin distribution of traffic will vary the rails then the QPs. For example given 2 rails and 2 QPs per rail, the round robin sequence will be: first rail first QP, second rail first QP, first rail second QP, second rail second QP.

NOTE

The Intel® PSM3 implementation has a limit of thirty-two (32) NICs per node and 32 QPs per endpoint.

NOTE

When using PSM3_GPUDIRECT use of more than 1 QP per NIC is discouraged. Each such QP will have an independent GPU registration cache and this can negatively impact GPU BAR space consumption which can negatively impact performance.

See for more details RDMA Modes and Rendezvous Module and PSM3 Multi-Rail Support.

7.12.37 PSM3_QP_RETRY

Sets the retry limit for user space and rendezvous module RC QPs.

Default: 7

This value only affects RC QPs for PSM3_RDMA modes 2 and 3. Values of `0` to `7` are permitted. For mode 1, the rendezvous module always uses a retry limit of `7`. The retry limit along with PSM3_QP_TIMEOUT determine the maximum time an RC QP may attempt retransmission of a packet before giving up, resulting in job failure.

In general, a value of `7` is recommended. This provides the maximum resiliency to network glitches and disruptions and can permit smaller values of PSM3_QP_TIMEOUT to be used.

7.12.38 PSM3_QP_TIMEOUT

Sets the timeout value for user space and rendezvous module RC QPs, in units of microseconds.

Default: `536870`

This value only affects RC QPs for PSM3_RDMA modes 1, 2 and 3. The actual timeout configured in the hardware is rounded up to the next hardware-supported value. The values supported by hardware are constrained to 4.096 × 2^n for n=0 to 31. This allows timeouts from 4.096 microseconds to 2.4 hours. The timeout along with PSM3_QP_RETRY determines the maximum time an RC QP may attempt retransmission of a packet before giving up, resulting in job failure.

Tuning of this value represents a trade-off between resiliency to network glitches and disruptions versus latency impact of packet loss. Higher values increase resiliency, however they also increase the latency for recovery.

7.12.39 PSM3_RCVTHREAD

Enables the receiver thread. PSM3 uses an extra background thread per rank to make MPI communication progress more efficient. This thread does not aggressively compete with resources against the main computation thread, but can be disabled by setting `PSM3_RCVTHREAD=0`. The frequency at which it runs is controlled via PSM3_RCVTHREAD_FREQ.

Default: `PSM3_RCVTHREAD=1`

NOTE

It is recommended to enable the receiver thread. In addition to communications progress, this is the only place where NIC fatal errors are detected and reported.

7.12.40 PSM3_RCVTHREAD_FREQ

Controls the frequency of polling by the receiver thread. The syntax is:

```
PSM3_RCVTHREAD_FREQ=min_freq[:max_freq[:shift_freq]]
```

Default value: `PSM3_RCVTHREAD_FREQ=10:100:1`. If any field is outside the allowed range, these default values will be used for all fields.

Allowed values:

`min_freq: [0 - 1000]` `max_freq: [min - 1000]`	The values of `min_freq` and `max_freq` are frequency in Hz (times per second) and specify the duration of sleeps between thread wakeups. For example, values of 10:100 mean that sleeps start at 100 milliseconds but can go as small as 10 milliseconds. Providing an empty value, or `min_freq` equal to 0 or `max_freq` equal to 0 will result in no receiver thread periodic polling. In which case the receiver thread will only wake when urgent packets (such as error recovery packets) are received.
`shift_freq: [0 - 10]`	`shift_freq` controls how aggressively the sleep duration is adjusted within the specified range. `shift_freq` specifies a power of 2 (2^{shift_freq}) that is used to multiply the sleep duration within the specified range. Adjustment means that sleep duration is reduced when work is found continually pending or queued and increased when work is found not to be pending.

See PSM3_RCVTHREAD.

NOTE

Higher frequencies for receiver thread polling can improve performance for applications which infrequently call into MPI. However, if run too frequently the receiver thread can introduce host CPU jitter and overheads, especially for applications which call into MPI frequently enough. Regardless of the `PSM3_RCVTHREAD_FREQ` specified, when PSM3_RCVTHREAD is enabled (1), the receiver thread will always be immediately woken to process urgent messages used for packet loss recovery, this insures timely recovery.

7.12.41 PSM3_RDMA

Controls the use of RC QPs and RDMA.

Options:

- `0` – Use only UD QPs.
- `1` – Use rendezvous module for node to node level RC QPs for rendezvous.
- `2` – Use user space RC QPs for rendezvous.
- `3` – Use user space RC QPs for eager and rendezvous.

Default: `0` when PSM3_GPUDIRECT disabled. Default is 1 when PSM3_GPUDIRECT enabled.

In all modes, a UD QP is created per endpoint to handle control messages (connection establishment, credit exchange, acknowledgments). In modes 1, 2, and 3, RDMA is used with RC QPs to optimize message transfer performance and reduce CPU overhead.

As the job size and number of processes (for example, ranks) per node increases, the memory required for these resources can be significant, especially for mode 3 where even a modest sized job may require gigabytes of communications buffers.

An important innovation introduced with PSM3 is the rendezvous kernel module. For more details on RDMA modes and the rendezvous module see RDMA Modes and Rendezvous Module.

NOTE

GPUDirect* is not allowed with RDMA mode 2 or 3. When this combination is specified, a warning is reported and mode 1 is used.

7.12.42 PSM3_RDMA_SENDSESSIONS_MAX

Controls the maximum number of RDMA send descriptor objects that PSM3 will create. The actual number of RDMA send descriptors created by PSM3 is determined by number of simultaneous sends and may be less than this value. See also PSM3_NUM_SEND_RDMA.

If a sender needs a RDMA send descriptor but none are available, PSM3 will issue a warning:

```
Non-fatal temporary exhaustion of send rdma descriptors
```

The send will retry when a descriptor becomes available.

Increasing the value of PSM3_RDMA_SENDSESSIONS_MAX can improve performance and reduce the frequency of, or eliminate, these warnings entirely.

Setting this value overrides the default maximum number of RDMA send descriptors determined by PSM3_MEMORY.

The maximum value for PSM3_RDMA_SENDSESSIONS_MAX is `1073741824` (2^{30}). Value must be a power of two (2^n).

Defaults:

- `1` – when `PSM3_MEMORY=min`
- `8192` – when `PSM3_MEMORY=normal`
- `16384` – when `PSM3_MEMORY=large`

7.12.43 PSM3_RTS_CTS_INTERLEAVE

Interleaves the handling of Ready-to-Send (RTS) packets with Clear-to-Send (CTS) packets in the PSM3 rendezvous protocol. This may improve link bandwidth by reducing link idle time for many-senders to one-receiver communication patterns.

Options:

- `1` – Enabled
- `0` – Disabled (default)

Default: `PSM3_RTS_CTS_INTERLEAVE=0` (disabled)

7.12.44 PSM3_RV_GPU_CACHE_SIZE

Sets the maximum amount of GPU memory to be pinned per process by the rendezvous module's GPU registration cache. In units of megabytes.

- `0` – cache size will be selected by rendezvous module's `gpu_cache_size` or `gpu_rdma_cache_size` module parameter (default of 256 and 1024 respectively in units of megabytes).
- `>0` – maximum pinned memory per process, in units of megabytes.

Default: `0`

The rendezvous modules's GPU registration cache is used in the following modes:

- When GPUDirect* RDMA is enabled, (e.g. PSM3_GPUDIRECT is enabled along with PSM3_RDMA mode >=1), the cache is used for GPUDirect* Copy, GPUDirect* send DMA and GPUDirect* RDMA. In this case the default size is selected by the `gpu_rdma_cache_size` module parameter (default of 1024 in units of megabytes).
- When GPUDirect* is enabled without GPUDirect* RDMA, (e.g. PSM3_GPUDIRECT is enabled along with PSM3_RDMA mode 0), the cache is used for GPUDirect* Copy and GPUDirect* send DMA . In this case the default size is selected by the `gpu_cache_size` module parameter (default of 256 in units of megabytes).

Use of the rendezvous module for GPU caching can greatly reduce GPU memory pinning overhead and GPU MR registration overhead when a set of GPU buffers are repeatedly used for communications. When PSM3_GPUDIRECT is enabled, the rendezvous module is required and the rendezvous module GPU registration cache will be used. The rendezvous module GPU registration cache will retain some GPU pinned memory and GPU MRS after they are done their current transfer, so they may be reused by future transfers. Such buffer reuse is common in many implementations of middleware collective algorithms such as MPI_AllReduce. For some applications, performance may be improved by growing this value, however values which result in `number_of_processes*PSM3_RV_GPU_CACHE_SIZE` near or exceeding the total GPU memory (or the limit which a given GPU model permits to be pinned and mapped into PCIe) can negatively affect PSM3 performance and application performance. Overly large GPU registration cache sizes may even may cause application or OS failures due to pinning too much GPU memory.

The minimum GPU registration cache size for PSM3_RDMA mode >=1 is (PSM3_NUM_SEND_RDMA + 32) * PSM3_MQ_RNDV_NIC_WINDOW + the largest GPUDirect* Copy size (default of 64,000 bytes).

The minimum GPU registration cache size for PSM3_RDMA mode 0 is the largest GPUDirect* Copy size (default of 64,000 bytes).

See RDMA Modes and Rendezvous Module, PSM3 Rendezvous Kernel Module, PSM3 and NVIDIA* CUDA* Support and PSM3_RV_MR_CACHE_SIZE.

7.12.45 PSM3_RV_HEARTBEAT_INTERVAL

Sets the interval between connection heartbeats for PSM3_RDMA mode 1. The value is in units of milliseconds. If a value of 0 is specified, then the rendezvous modules's heartbeat mechanism is disabled for the job.

Default: `1000`

The heartbeat mechanism helps detect rare situations where only one side of the RC QP connection observes an issue. If RC QP recovery is enabled and the observing side happens to also be the passive side of the connection, it could end up waiting for a re-connection which will never be issued.

The heartbeats use 0 length RC QP payload packets and are only issued on RC QPs which have seen no traffic in either direction for the duration of the interval. As such the mechanism incurs minimal overhead and is recommended for use even when RC QP recovery is disabled.

All jobs which are using the same PSM3_FI_UUID (as defaulted, set explicitly or set via the MPI middleware) must use the same heartbeat interval.

When both are enabled, the heartbeat interval selected must be set smaller than PSM3_RV_RECONNECT_TIMEOUT.

See for more details RDMA Modes and Rendezvous Module.

7.12.46 PSM3_RV_MR_CACHE_SIZE

Sets the maximum amount of CPU memory to be pinned per process by the rendezvous module's CPU MR cache. In units of megabytes.

- `0` – cache size will be selected by rendezvous module's mr_cache_size or mr_cache_size_gpu module parameter (default of 256 and 1024 respectively in units of megabytes).
- `>0` – maximum pinned CPU memory per process, in units of megabytes.

Default: `0`

The rendezvous modules's CPU MR cache is used in the following modes:

- When PSM3_RDMA selects mode 1
- When PSM3_RDMA selects mode 2 or 3 and PSM3_MR_CACHE_MODE is 1

Use of the rendezvous module for CPU MR caching can greatly reduce MR registration overhead when a set of buffers are repeatedly used for communications. Unlike the user space MR table, the rendezvous module CPU MR cache will retain some MRs after they are done their current transfer, so they may be reused by future transfers. Such buffer reuse is common in many implementations of middleware collective algorithms such as MPI_AllReduce. For some applications, performance may be improved by growing this value, however values which result in `number_of_processes*PSM3_RV_MR_CACHE_SIZE` near or exceeding the total server memory can negatively effect application performance due to swapping or even may cause application or OS failures due to pinning too much memory.

The minimum cache size is (PSM3_NUM_SEND_RDMA + 32) * PSM3_MQ_RNDV_NIC_WINDOW

See RDMA Modes and Rendezvous Module and PSM3_RV_GPU_CACHE_SIZE.

7.12.47 PSM3_RV_Q_DEPTH

Sets the maximum concurrent queued IOs per node to node connection in the rendezvous module. If a value of 0 is specified, then the rendezvous modules's `q_depth` parameter controls the selection.

Default: 0

The default in the rendezvous module is sized with some headroom for jobs with up to 100 processes per node. For jobs with higher process per node counts, growing this value may help application performance, especially for large messages.

All jobs which are using the same PSM3_FI_UUID (as defaulted, set explicitly or set via the MPI middleware) must use the same queue depth.

See for more details RDMA Modes and Rendezvous Module and PSM3 Rendezvous Kernel Module.

7.12.48 PSM3_RV_QP_PER_CONN

Sets the number of RC QPs per rendezvous module connection for PSM3_RDMA mode 1. If a value of 0 is specified, then the rendezvous modules's num_conn parameter controls the selection.

Default: 0

For some applications, use of multiple QPs per connection may increase performance for large messages.

All jobs which are using the same PSM3_FI_UUID (as defaulted, set explicitly or set via the MPI middleware) must use the same number of QPs per rv connection.

See for more details RDMA Modes and Rendezvous Module.

7.12.49 PSM3_RV_RECONNECT_TIMEOUT

Sets the RC QP minimum re-connection timeout in seconds for PSM3_RDMA mode 1. If a value of 0 is specified, then the rendezvous modules's RC QP connection recovery feature is disabled for the job.

Default: 30

This only affects the time permitted for connection recovery. The time limit for initial connection establishment is controlled by PSM3_CONNECT_TIMEOUT and FI_PSM3_CONN_TIMEOUT and is typically set to a lower value than PSM3_RV_RECONNECT_TIMEOUT, see PSM3_CONNECT_TIMEOUT for details. There may be some delays in detecting the connection loss or starting the re-connection mechanism, hence the actual time limit may be slightly larger than specified.

All jobs which are using the same PSM3_FI_UUID (as defaulted, set explicitly or set via the MPI middleware) must use the same timeout.

When enabled, the re-connect timeout selected must be set larger than PSM3_RV_HEARTBEAT_INTERVAL.

See for more details RDMA Modes and Rendezvous Module.

7.12.50 PSM3_SEND_REAP_THRESH

Sets the number of outstanding send WQEs before reaping CQEs.

Default: 256

7.12.51 PSM3_TRACEMASK

Sets the information to output from various parts of PSM3 for debugging.

Options:

The following selections may be summed to select more than one type of output:

- `0x1` (default) – Informative messages (mostly warnings) are printed. This value should be considered a minimum.
- `0x2` – Higher level debug information, mostly during initialization and shutdown.
- `0x10` – Higher level debug verbs information, mostly during initialization and shutdown.
- `0x20` – PSM3 connection establishment and disconnection.
- `0x40` – Detailed debug output regarding packet movement, quite a significant amount of output since multiple messages are output per packet.
- `0x80` – Detailed debug output regarding packet contents, quite a significant amount of output since multiple messages are output per packet.
- `0x100` – PSM3 process startup and shutdown.
- `0x200` – Detailed debug output regarding memory registration, quite a significant amount of output since multiple messages are output per message using rendevous RDMA.
- `0x400` – PSM3 environment variables. Also see PSM3_VERBOSE_ENV.

Default: `PSM3_TRACEMASK=0x1`

PSM3_DEBUG_FILENAME may be used to control where debug output is placed.

The following are some useful sample settings:

- `0x101` – Startup and finalization messages are added to the output.
- `0x133` – Details of startup and shutdown.
- `0x3f3` – Every communication event is logged, this should be used for extreme debugging only.
- `0xffff` – All debug messages will be output. This should be used only in extreme debugging situations as it will generate lots of debug output.

Also see PSM3_IDENTIFY, PSM3_PRINT_STATS, and PSM3_VERBOSE_ENV.

NOTE

In addition to debugging information from PSM3, additional higher level information may be independently enabled for output via OFI environment variables such as `FI_LOG_LEVEL, FI_LOG_PROV, FI_LOG_SUBSYS and FI_PERF_CNTR`. See the OFI (libfabric) man pages and documentation for more information.

NOTE

The exact meaning of each value as well as the messages selected and their format may change in future releases.

7.12.52 PSM3_VERBOSE_ENV

Enable verbose output of PSM3 environment variables. Help text for environment variables are output by each process along with their defaults and any non-default values selected.

Options:

- `0` – disable - no output.
- `1` – enabled on all processes. Only variables found in the environment are output, no help text.
- `1:` – enabled only on rank 0 (abbreviation for `PSM3_VERBOSE_ENV=1:*:rank0`). Only variables found in the environment are output, no help text.
- `1:pattern` – enabled only on processes whose label matches the extended glob pattern. Only variables found in the environment are output, no help text.
- `2` – enabled on all processes. All variables are output with help text.
- `2:` – enabled only on rank 0 (abbreviation for `PSM3_VERBOSE_ENV=2:*:rank0`). All variables are output with help text.
- `2:pattern` – enabled only on processes whose label matches the extended glob pattern. All variables are output with help text.

Default: `0`

NOTE

For more information on extended glob patterns see the linux man pages for glob(7) and fnmatch(3).

The label for a process is typically of the form `hostname:rank#` such as `myhost047:rank3` where rank is the relative process number in the job. If MPI, the MPI runtime and the job scheduler have not indicated the rank to PSM3, the label will be of the form `hostname:pid#` where pid is the linux process id. The form of labels for a given cluster can be observed at the beginning of various PSM3 output messages, such as those from `PSM3_IDENTIFY`.

Some Example uses of patterns:

- `PSM3_VERBOSE_ENV=2:*:rank0` – only output for rank 0. When the user only needs the help text or is confident that the same PSM3 environment variables are being supplied to all processes in the job, this can provide a more concise output.
- `PSM3_VERBOSE_ENV=1:myhost047:*` - only output for processes on myhost047. All processes on that host will provide output. This can be helpful if only a single host's environment is suspect.
- `PSM3_VERBOSE_ENV=1:+(*:rank0|*:rank1)` – only output for rank 0 and 1. This is an example of an extended glob pattern.

NOTE

Depending on how jobs are launched, the value for `PSM3_VERBOSE_ENV` may need to be enclosed in single quotes to prevent expansion of wildcards against local filenames during the launch script.

NOTE

Output occurs for every selected process in the job. As such this can generate quite a lot of output especially when all processes are selected in a high process count job.

NOTE

Variables with names of the form FI_PSM3_* are not included in this output.

NOTE

Variables which are not applicable due to selections in other variables may not be output. For example, when PSM3_NIC is specified, PSM3_NIC_SELECTION_ALG is not output. Similarly when PSM3_MULTIRAIL is not enabled, PSM3_MULTIRAIL_MAP is not output.

8.0 Integration with a Batch Queuing System

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

For Open MPI, there are resources at openmpi.org that document how to use the MPI with different batch queuing systems, located at the following links:

- Torque / PBS Pro: http://www.open-mpi.org/faq/?category=tm
- SLURM: http://www.open-mpi.org/faq/?category=slurm
- Bproc: http://www.open-mpi.org/faq/?category=bproc
- LSF: https://www.open-mpi.org/faq/?category=building#build-rte-lsf

8.1 Clean Termination of MPI Processes

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

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

```
/sbin/fuser -v /dev/infiniband/* /dev/rv
/dev/infiniband/uverbs0:    22648m 22651m
/dev/rv:                    22648m 22651m
```

In this example, processes 22648 and 22651 are using the Intel interconnect.

Another example using the `lsof` command:

```
lsof /dev/infiniband/* /dev/rv
```

This command displays a list of processes using Intel® Ethernet Fabric Suite.

Run the following command to terminate all processes using the Intel interconnect:

```
/sbin/fuser -k /dev/infiniband/* /dev/rv
```

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

NOTE

`/dev/rv` will only be used by MPI PSM3 processes which are using the rendezvous module. As such the list of processes associated with it may not be the complete list of MPI processes using PSM3.

NOTE

`/dev/infiniband/*` may be used by processes other than MPI processes and may be used by libraries other than PSM3. As such care must be used when using `fuser -k` or similar mechanisms to select processes to kill.

NOTE

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

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

If this error occurs, refer to Clean Up PSM3 Shared Memory Files.

8.2 Clean Up PSM3 Shared Memory Files

If a PSM3 job terminates abnormally, such as with a segmentation fault, there could be POSIX shared memory files left over in the `/dev/shm` directory. The files are owned by the user and can be deleted either by the user or by root.

To clean up the system, create, save, and run the following PSM3 SHM cleanup script as root on each node. Either log on to the node, or run remotely using `pdsh` or `ssh`.

```
#!/bin/sh
files=`/bin/ls /dev/shm/psm3_shm.* 2> /dev/null`;
for file in $files;
do
/sbin/fuser $file > /dev/null 2>&1;
if [ $? -ne 0 ];
then
/bin/rm $file > /dev/null 2>&1;
fi;
done;
```

When the system is idle, you can remove all of the shared memory files, including stale files, with the following command:

```
# rm -rf /dev/shm/psm3_shm.*
```

9.0 Troubleshooting

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

- BIOS Settings
- Kernel and Initialization Issues
- System Administration Troubleshooting
- CUDA* Application Failures
- Performance Issues

Additional troubleshooting information can be found in:

- The documentation which came with the Ethernet NICs and Switches being used.
- *Intel® Ethernet Fabric Suite Software Installation Guide*

Intel® Ethernet Fabric Suite user documentation can be found on the Intel web site. See Intel® Ethernet Fabric Suite Documentation Library on page 11 for URL.

9.1 BIOS Settings

Refer to the *Intel® Ethernet Fabric Performance Tuning Guide* for information relating to checking, setting, and changing BIOS settings.

9.2 Kernel and Initialization Issues

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

- Rendezvous Module Load Fails Due to Unsupported Kernel
- Rebuild or Reinstall Rendezvous Module if Different Kernel Installed
- Intel® Ethernet Fabric Suite Rendezvous Module Initialization Failure
- MPI Job Failures Due to Initialization Problems

9.2.1 Rendezvous Module Load Fails Due to Unsupported Kernel

If you try to load the Intel® Ethernet Fabric Suite rendezvous module on a kernel that the Intel® Ethernet Fabric Suite software does not support, the load fails with error messages that point to `rv.ko`.

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

9.2.2 Rebuild or Reinstall Rendezvous Module if Different Kernel Installed

If you are not using the DKMS-enabled version of the kernel module, you must reboot and then rebuild or reinstall the Intel® Ethernet Fabric Suite kernel modules (rendezvous module). Intel recommends that you use the instalation Textual User Interface (TUI) (`INSTALL`) to perform this rebuild or reinstall. Refer to the *Intel® Ethernet Fabric Suite Software Installation Guide* for more information.

9.2.3 Intel® Ethernet Fabric Suite Rendezvous Module Initialization Failure

There may be cases where the rendezvous module was not properly loaded or 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 -hostfile myhosts -genv  I_MPI_FABRICS=shm:ofi -genv
I_MPI_OFI_PROVIDER=psm3 -genv PSM3_RDMA=1 osu_latency
...
<nodename.pid>: Unable to open rendezvous module for port 1 of irdma-ens785f0.
<nodename.pid>: Unable to initialize verbs
<nodename.pid>: PSM3 can't open nic unit: 0 (err=23)
```

If this error appears, check to see if the Intel® Ethernet Fabric Suite rendezvous module is loaded with the command:

```
$ lsmod | grep rv
```

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

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

The output indicates whether the driver has loaded or not. Printing out messages using `dmesg` may help to locate any problems with the rendezvous module.

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

```
$ dmesg | grep -i rv
```

This command may generate more than one screen of output.

If the module successfully loaded, check the access permissions on the module with the command:

```
$ ls -l /dev/rv
```

9.2.4 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:

```
OFI addrinfo() failed (ofi_init.c:986:MPIDI_OFI_mpi_init_hook:No data available)
```

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

9.3 System Administration Troubleshooting

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

9.3.1 Flapping/Unstable NIC Links

Although PSM is designed to withstand temporary link outages, there may be NIC link instabilities which cannot be survived. One of the symptoms of a "flapping" or unstable NIC link is repeated down/up messages in dmesg, for example:

```
:> dmesg -T | grep NIC
[03:59:59 2020] ice 0000:83:00.1 cvl1: NIC Link is Down
[04:00:00 2020] ice 0000:83:00.1 cvl1: NIC Link is up ...
[04:13:15 2020] ice 0000:83:00.1 cvl1: NIC Link is Down
[04:13:16 2020] ice 0000:83:00.1 cvl1: NIC Link is up ...
[04:13:16 2020] ice 0000:83:00.1 cvl1: NIC Link is Down
[04:13:17 2020] ice 0000:83:00.1 cvl1: NIC Link is up ...
[04:45:55 2020] ice 0000:83:00.1 cvl1: NIC Link is Down
[04:45:56 2020] ice 0000:83:00.1 cvl1: NIC Link is up ...
[04:45:56 2020] ice 0000:83:00.1 cvl1: NIC Link is Down
[04:45:57 2020] ice 0000:83:00.1 cvl1: NIC Link is up ...
```

As seen in the above example, the NIC link has gone down and up five times over the span of 45 minutes on an idle system. This is the sign of an unhealthy link. The threshold for a "flapping" NIC needs to be defined by the system admin according to the cluster usage. For example, if no re-cabling is being performed through the duration of a node's uptime, then a low number (or no) link flapping should be seen. If nodes are being re-cabled or other system-admin related tasks requiring a link bounce are being performed, then a certain number of link flapping occurrences may be expected.

Sometimes re-seating the cable on the NIC and switch side may help, but other times this may indicate an issue with a cable or switch configuration. These issues should be addressed on every node before the system is put into production or unexpected behavior may result. Some switch operating systems (such as the Arista EOS) contain features to automatically disable ports when configurable thresholds are exceeded (see the Arista EOS user manual, section titled "Link Flap Monitoring", for details).

9.3.2 Broken Intermediate Link

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

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

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

Intel recommends using FastFabric to help diagnose this problem. For details, see the *Intel® Ethernet Fabric Suite FastFabric User Guide*.

9.4 CUDA* Application Failures

If a CUDA application fails to launch or segfaults early in it's execution, check the following:

- The MPI used and/or all middleware is CUDA* enabled.
- The PSM3 OFI provider being used is CUDA* enabled. (See PSM3_IDENTIFY)
- The PSM3_CUDA and/or PSM3_GPUDIRECT options are enabled.
- If PSM3_GPUDIRECT is enabled, verify the CUDA* enabled rendezvous module is loaded. (See PSM3_IDENTIFY)

See Intel® Ethernet Fabric Suite PSM3 Support for GPUDirect* and PSM3 and NVIDIA* CUDA* Support

9.5 Performance Issues

See the *Intel® Ethernet Fabric Performance Tuning Guide* for details about Intel® Ethernet Fabric optimizing performance and handling performance issues.

10.0 Recommended Reading

This section contains lists of reference material for further reading.

10.1 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

10.2 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

10.3 Reference and Source for SLURM

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

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

10.4 OpenFabrics Alliance*

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

http://www.openfabrics.org

10.5 Clusters

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

10.6 Networking

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

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

10.7 Other Software Packages

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

http://modules.sourceforge.net/

11.0 Descriptions of Command Line Tools

This section provides a complete description of each Intel® Ethernet Host Software command line tool and its parameters.

Additional Intel® Ethernet Fabric Suite FastFabric command line tools are described in the *Intel® Ethernet Fabric Suite FastFabric User Guide*.

11.1 Basic Single Host Operations

The tools described in this section are available on each host where the Intel® Ethernet Host Software stack tools have been installed. The tools can enable FastFabric toolset operations against cluster nodes when used on a Management Node withIntel® Ethernet Fabric Suite installed, however, they can also be directly used on an individual host.

11.1.1 ethautostartconfig

Provides a command line interface to configure autostart options for various Ethernet utilities.

Syntax

```
ethautostartconfig --[Action] [Utility]
```

Options

`--help` Produces full help text.

`--status` Shows status of setting.

`--enable` Enables the setting.

`--disable` Disables the setting.

`--list` Lists all available utilities.

`Utility` Identifies the utility to be acted upon.

11.1.2 ethsystemconfig

Provides a command line interface to configure system options for various Ethernet utilities.

Syntax

```
ethsystemconfig --[Action] [Utility]
```

Options

`--help`	Produces full help text.
`--status`	Shows status of setting.
`--enable`	Enables the setting.
`--disable`	Disables the setting.
`--list`	Lists all available utilities.
`Utility`	Identifies the utility to be acted upon.

11.1.3 iefsconfig

(Switch and Host) Configures the Intel® Ethernet Fabric Suite Software through command line interface or TUI menus.

Syntax

```
iefsconfig [-G] [-v|-vv] [-u|-s|-e comp] [-E comp] [-D comp]
 [--answer keyword=value]
```

or

```
iefsconfig -C
```

or

```
iefsconfig -V
```

Options

No option	Starts the Intel® Ethernet Fabric Suite Software TUI.
`--help`	Produces full help text.
`-G`	Installs GPU Direct components (must have NVidia drivers installed).
`-v`	Specifies verbose logging.
`-vv`	Specifies very verbose debug logging.
`-u`	Uninstalls all ULPs and drivers with default options.

`-s`	Enables autostart for all installed drivers.
`-e` *comp*	Uninstalls the given component with default options. This option can appear more than once on the command line.
`-E` *comp*	Enables autostart of a given component. This option can appear with `-D` or more than once on the command line.
`-D` *comp*	Disables autostart of given component. This option can appear with `-E` or more than once on the command line.
`-C`	Outputs list of supported components. **NOTE:** Supported components may vary according to OS. Refer to *Intel® Ethernet Fabric Suite Software Release Notes*, OS RPMs Installation Prerequisites for the list of components by supported OS. Supported components include: `eth_tools psm3 eth_module fastfabric eth_rdma openmpi_gcc_ofi mpisrc delta_debug` Supported components when using command on a Management Node with Intel® Ethernet Fabric Suite installed, include: `fastfabric` Supported component name aliases include: `eth mpi psm_mpi` Additional component names allowed for `-E` and `-D` options: `snmp`
`-V`	Outputs version.
`--answer` *keyword=value*	Provides an answer to a question that may occur during the operation. Answers to questions not asked are ignored. Invalid answers result in prompting for interactive installs, or using default options for non-interactive installs. **Possible Questions** (*keyword=value*): `ARPTABLE_TUNING` Adjust kernel ARP table size for large fabrics? `ROCE_ON` RoCE RDMA transport? `LIMITS_SEL` Resource Limits Selector

Example

```
# iefsconfig
Intel Ethernet x.x.x.x.x Software
```

```
   1) Show Installed Software
   2) Reconfigure Eth RDMA
   3) Reconfigure Driver Autostart
   4) Generate Supporting Information for Problem Report
   5) FastFabric (Host/Admin)
   6) Uninstall Software

   X) Exit
```

11.1.4 ethcapture

Captures critical system information into a zipped tar file. The resulting tar file should be sent to Customer Support along with any Intel® Ethernet Fabric problem report regarding this system.

NOTE

The resulting host capture file can require significant amounts of space on the host. The actual size varies, but sizes can be multiple megabytes. Intel recommends ensuring that adequate disk space is available on the host system.

Syntax

```
ethcapture [-d detail] output_tgz_file
```

Options

`--help` Produces full help text.

`-d detail` Captures level of detail:

- `1` (Local) Obtains local information from host. This is the default if no options are entered.
- `2` (Fabric) In addition to *Local*, also obtains basic fabric information by queries to the SM and fabric error analysis using `ethreport`.
- `3` (Analysis) In addition to *Fabric*, also obtains `ethallanalysis` results. If `ethallanalysis` has not yet been run, it is run as part of the capture.

NOTE

Detail levels 2 – 3 can be used when fabric operational problems occur. If the problem is node-specific, detail level 1 should be sufficient. Detail levels 2 – 3 require an operational Fabric. Typically your support representative requests a given detail level. If a given detail level takes excessively long or fails to be gathered, try a lower detail level.

For detail levels 2 – 3, the additional information is only available on a node with Intel® Ethernet Fabric Suite FastFabric Toolset installed.

output_tgz_file Specifies the name of a file to be created by `ethcapture`. The file name specified is overwritten if it already exists. Intel recommends using the `.tgz` suffix in the file name supplied. If the filename given does not have a `.tgz` suffix, the `.tgz` suffix is added.

Examples

```
ethcapture mycapture.tgz
ethcapture -d 3 030127capture.tgz
```