Intel® Ethernet Fabric Suite Fabric Host Software

User Guide

Rev. 1.11

June 2025

Doc. No.: 632489, Rev.: 1.11

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All product plans and roadmaps are subject to change without notice.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Copyright © 2020–2024, Intel Corporation. All rights reserved.

Revision History

Date	Revision	Description
June 2025	1.11	Product 12.1.0.0 release - Changes to this document include: • PSM3_RDMA mode 2 and 3 (user space RC QPs) now support QP connection recovery, see PSM3 Verbs RDMA Modes and Rendezvous Module. As part of this, PSM3_RV_RECONNECT_TIMEOUT is deprecated and replaced with PSM3_RECONNECT_TIMEOUT • Added PSM3_RV_FR_PAGE_LIST_LEN
February 2025	1.10	Product 12.0.0.0 release - Changes to this document include: • Improved PSM3_FLOW_CREDITS to have a better flow control for RC verbs • Improved PSM3_MTU to allow a value larger than NIC MTU for RC verbs • Removed selection `0x10` from PSM3_TRACEMASK
July 2024	1.9	Product 11.7.0.0 release - Changes to this document include: • Default for PSM3_RDMA is now 1 when rendezvous module is available or `PSM3_GPUDIRECT` is enabled. • PSM3_FLOW_CREDITS now allows dynamic credit adjustment based on network behavior. • Default for PSM3_MQ_RNDV_SHM_GPU_THRESH is now 63 for NVIDIA GPUs.
March 2024	1.8	Product 11.6.0.0 release - Changes to this document include: • Added new GPU aware NIC selection algorithms. See Multi-Rail Environment Variables, Multi-Rail Configuration Examples, PSM3 Multi-IP Environment Variables, PSM3 Multi-IP Configuration Examples, Building the PSM3 RPM, PSM3_MULTIRAIL, PSM3_MULTIRAIL_MAP, and PSM3_NIC_SELECTION_ALG. • PSM3_MULTIRAIL_MAP now allows for per process selection of NICs. Also updated Multi-Rail Environment Variables, Multi-Rail Configuration Examples, and PSM3 Multi-IP Environment Variables. • Documented Intel MPI support of NVIDIA GPUs, see Using PSM3 Features for NVIDIA GPUDirect. • Documented Intel MPI setting of PSM3_GPUDIRECT in Environment Variables for Intel® MPI Library Jobs, Using PSM3 Features for Direct Access to Intel GPUs, Using PSM3 Features for NVIDIA GPUDirect, and PSM3_GPUDIRECT. • Changes to default for PSM3_GPUDIRECT_RDMA_RECV_LIMIT and PSM3_GPUDIRECT_RDMA_SEND_LIMIT for Intel GPUs. • PSM3_PRINT_STATS now allows an optional pattern to enable statistics output for a subset of the processes in a job. • Added PSM3_PRINT_STATS_PREFIX and updated PSM3 Performance Statistics, PSM3_PRINT_STATS and PSM3_PRINT_STATS_HELP. • The use of colon to separate DSA work queue lists in PSM3_DSA_WQS has been deprecated, a semicolon is now used to separate DSA work queue lists. • Added PSM3_RNDV_NIC_WINDOW and PSM3_GPU_RNDV_NIC_WINDOW and deprecated PSM3_MQ_RNDV_NIC_WINDOW. Updated Multi-Rail Environment Variables, PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3_GPU_THRESH_RNDV, PSM3_MR_CACHE_MODE, PSM3_MR_CACHE_SIZE_MB, PSM3_MTU, PSM3_RV_GPU_CACHE_SIZE and PSM3_RV_MR_CACHE_SIZE.

continued...

Date	Revision	Description
		• Added PSM3_MQ_RNDV_SHM_GPU_THRESH and updated PSM3_MQ_RNDV_SHM_THRESH. • Default for PSM3_GPU_THRESH_RNDV when using sockets HAL is now 4GB. • PSM3_GPUDIRECT_RDMA_RECV_LIMIT, PSM3_GPUDIRECT_RDMA_SEND_LIMIT, and PSM3_GPU_THRESH_RNDV allow input of `max`. • PSM3_UDP_GSO now allows direct control over the maximum segmentation offload size. • Correctly document default for PSM3_GPUDIRECT_RDMA_SEND_LIMIT for NVIDIA GPUs. • Improved descriptions of environment variable reporting in PSM3_PRINT_STATSMASK and PSM3_VERBOSE_ENV. • Improved description of PSM3_DEVICES and PSM3 Architecture and Hardware Abstraction Layer. • Added Confirming the PSM3 Provider is Selected and referenced in Running MPI Applications with Intel® MPI Library, Running Open MPI Applications, Running Applications that Use oneCCL, Running with oneCCL, and Running with NVIDIA NCCL. • Improved description of ethbw. • Improved description in PSM3 Rendezvous Kernel Module. • Added discussion of `--enable-psm3-dsa` and `--enable-psm3-umr-cache` build options as well as use of Intel icx compiler to Building the PSM3 RPM
September 2023	1.7	Product 11.5.1.0 release - Changes to this document include: • Documented PSM3_GPUDIRECT_RDMA_RECV_LIMIT. • Changes to default for PSM3_GPUDIRECT_RDMA_SEND_LIMIT and PSM3_MQ_RNDV_NIC_WINDOW for Intel GPU. • Added PSM3_IDENTIFY output of GPU library versions. • NVIDIA GPU cross version interoperability improved. See Building the PSM3 RPM. • Added new compilers in Compiling MPI Applications with Intel® MPI Library. • Updated example in Running with Multiple PSM3 Variations. • Improved description of user space MR cache performance analysis. See PSM3 Performance Statistics • Improved description of PSM3_TRACEMASK and PSM3_DEBUG_FILENAME. • Assorted grammatical, formatting and style improvements through the document.
May 2023	1.6	Product 11.5.0.0 release - Changes to this document include: • Documented the user space MR cache. See PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Rendezvous Kernel Module, PSM3_MR_CACHE_MODE, PSM3_MR_CACHE_SIZE and PSM3_MR_CACHE_SIZE_MB. • Intel® Xeon® Processor Data Streaming Accelerator (DSA) shared work queue support added and discussed in the following sections: PSM3 Data Streaming Accelerator Support, PSM3_DSA_MULTI, PSM3_DSA_WQS, PSM3_IDENTIFY and dsa_setup. • Documented additional settings for running Intel GPU applications. See PSM3 and Intel GPU Support and Intel GPU Application Failures. • Documented new PSM3_PRINT_STATSMASK bit of 0x40 to include process launch information. Also see PSM3 Performance Statistics. • Documented additional PSM3 defaults which may be overridden by the Intel® MPI Library. See Environment Variables for Intel® MPI Library Jobs and PSM3_HAL. • Added additional suggested debug procedures in MPI Job Failures Due to Initialization Problems.

continued...

Date	Revision	Description
		• Updated `PSM3_TCP_BIND_SRC` default value to 1. See PSM3_TCP_BIND_SRC. • Updated Building the PSM3 RPM to include information about compiler selection, including how to build with the Intel® C++ Compiler (Classic) (`icc`). Added note that psm3 must be built against the CUDA major version that it will be run against. • Added Section about Running with Multiple PSM3 Variations. • Refined Introduction section. • Rendezvous kernel module `gpu_cache_keep` option removed. See PSM3 Rendezvous Kernel Module. • Removed troubleshooting for "MPI Job Failures in Virtual Machines". PSM3 now functions without error even if the `/sys/class/*/*/device/local_cpuset` file for the selected NIC is missing.
March 2023	1.5	Product 11.4.1.0 release - Changes to this document include: • Added a per server PSM3 configuration file (`/etc/psm3.conf`) as discussed in PSM3 Config File. Also see PSM3_DISABLE_MMAP_MALLOC, PSM3_TRACEMASK and PSM3_VERBOSE_ENV. • Intel® Xeon® Processor Data Streaming Accelerator (DSA) support added and discussed in the following sections: PSM3 Data Streaming Accelerator Support, PSM3_DSA_WQS, PSM3_IDENTIFY and dsa_setup. • Added PSM3_FORCE_SPEED and PSM3_TCP_BIND_SRC. • Deprecated PSM3_CUDA_THRESH_RNDV and replaced with PSM3_GPU_THRESH_RNDV. • Added more information about PSM3 performance statistics. See PSM3 Performance Statistics. Also added statistics help via PSM3_PRINT_STATS_HELP and improved description of PSM3_PRINT_STATS and PSM3_PRINT_STATSMASK. • Added troubleshooting for "MPI Job Failures in Virtual Machines". • Added `gpu_cache_keep` option to rendezvous kernel module: PSM3 Rendezvous Kernel Module for use with Intel GPUs. • Improved description of PSM3 and Intel GPU Support and PSM3 and NVIDIA CUDA Support. • Documented PSM3 defaults which may be overridden by the Intel® MPI Library. See Environment Variables for Intel® MPI Library Jobs, PSM3 Config File, FI_PSM3_INJECT_SIZE, FI_PSM3_LAZY_CONN, FI_PSM3_UUID and PSM3_MULTI_EP.
November 2022	1.4	Product 11.4.0.0 release - Changes to this document include: • Intel GPU support added and discussed in the following sections:PSM3 Support for GPUs, PSM3 Support for Intel GPUs,Running oneCCL on Network Interface Cards, PSM3 Architecture and Hardware Abstraction Layer, PSM3 and GPU Support, PSM3 and Intel GPU Support, HAL and Protocol-Specific Configuration Controls, PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Sockets Modes, Building the PSM3 RPM, PSM3_CUDA_THRESH_RNDV, PSM3_DEVICES, PSM3_GPUDIRECT, PSM3_GPUDIRECT_RDMA_SEND_LIMIT, PSM3_IDENTIFY, PSM3_MQ_RNDV_NIC_THRESH, PSM3_MQ_RNDV_NIC_WINDOW, PSM3_ONEAPI_ZE, PSM3_PRINT_STATSMASK, PSM3_RV_GPU_CACHE_SIZE, and Intel GPU Application Failures. • PSM3_MULTIRAIL_MAP now permits selection of addresses when combined with PSM3 Multi-IP Support and PSM3_ADDR_PER_NIC • Added PSM3_MULTIRAIL=-1 mode. Also see PSM3 Multi-Rail Support, Running oneCCL on Network Interface Cards, and PSM3 Support for NVIDIA NCCL. • Added ethshmcleanup and discussed in Clean Up PSM3 Shared Memory Files. • Added PSM3_GPUDIRECT_RDMA_SEND_LIMIT. • Updated Installing the NVIDIA NCCL OFI Plugin

continued...

Date	Revision	Description
June 2022	1.3	Product 11.3.0.0 release - Changes to this document include: • PSM3 now permits use of multiple IP addresses per NIC. See PSM3 Multi-IP Support and PSM3_ADDR_PER_NIC • Address and NIC filtering now also incorporates multiple IP address configuration as discussed in NIC and Address Filtering, PSM3 Multi-IP Support, and PSM3_ADDR_PER_NIC • Clarified PSM3 Multi-Rail Support enablement for middleware load balancing. • Added ability to control polling when using TCP/IP in the sockets HAL, see PSM3_TCP_SKIPPOLL_COUNT. • Added control over send rate when using TCP/IP in the sockets HAL, see PSM3_TCP_SNDPACING_THRESH. • Clarified PSM3_ALLOW_ROUTERS in relation to multi-plane configurations in PSM3 Multi-Rail Support.
March 2022	1.2	Product 11.2.0.0 release - Changes to this document include: • Use of PSM3 with oneCCL is now described: Running oneCCL on Network Interface Cards • Use of PSM3 with Intel® MPI has been updated: Intel® MPI Library • Use of PSM3 with NVIDIA NCCL is now described: PSM3 Support for NVIDIA NCCL • Address and NIC filtering added including use of wildcards in PSM3_NIC and its applicability to all PSM3 modes of operation. In addition IPv6 and InfiniBand handling was added and is also discussed in the following sections and environment variables: NIC and Address Filtering, PSM3 Multi-Rail Support, PSM3_ADDR_FMT, PSM3_ALLOW_ROUTERS, PSM3_IDENTIFY, PSM3_MULTIRAIL_MAP, PSM3_NIC, PSM3_NIC_SELECTION_ALG, PSM3_NIC_SPEED, PSM3_SUBNETS • Support for TCP/IP sockets was added along with runtime selection between a verbs Hardware Abstraction Layer (HAL) and a sockets HAL. This is discussed in the following sections: PSM3 Architecture and Hardware Abstraction Layer, PSM3 Sockets Modes, HAL and Protocol-Specific Configuration Controls, NIC and Address Filtering and PSM3_HAL. New parameters specific to the sockets HAL are discussed in: PSM3_SOCKETS, PSM3_TCP_PORT_RANGE, PSM3_TCP_RCVBUF, PSM3_TCP_SNDBUF, PSM3_UDP_GSO, PSM3_UDP_RCVBUF, PSM3_UDP_SNDBUF. Related updates were also made in the following sections: Using PSM3 Features for NVIDIA GPUDirect, PSM3_CUDA_THRESH_RNDV, PSM3_DEVICES, PSM3_FLOW_CREDITS. PSM3_GPUDIRECT, PSM3_IB_SERVICE_ID, PSM3_MQ_RNDV_NIC_WINDOW, PSM3_MR_CACHE_MODE, PSM3_MR_CACHE_SIZE, PSM3_MTU, PSM3_MULTIRAIL, PSM3_MULTIRAIL_MAP, PSM3_NIC, PSM3_NIC_SPEED, PSM3_NUM_RECV_CQES, PSM3_NUM_RECV_WQES, PSM3_NUM_SEND_RDMA, PSM3_NUM_SEND_WQES, PSM3_QP_PER_NIC, PSM3_RDMA, PSM3_QP_RETRY, PSM3_QP_TIMEOUT, PSM3_RDMA_SENDSESSIONS_MAX, PSM3_RV_GPU_CACHE_SIZE, PSM3_RV_HEARTBEAT_INTERVAL, PSM3_RV_MR_CACHE_SIZE, PSM3_RV_Q_DEPTH, PSM3_RV_QP_PER_CONN, PSM3_RV_RECONNECT_TIMEOUT, PSM3_SEND_REAP_THRESH, PSM3_TRACEMASK. • A new section was added describing PSM3 `rpmbuild` options: Building the PSM3 RPM • The rendezvous kernel module now has a Fast-Registration Pool. see PSM3 Rendezvous Kernel Module • The behavior of multi-rail message striping has been clarified in: Multi-Rail Environment Variables • The list of /dev/shm objects to remove after failed jobs or between jobs has been refined, see: Clean Up PSM3 Shared Memory Files • PSM3_IDENTIFY output for rendezvous module run-time interface now uses `user_mr` to indicate if the rendezvous module was loaded with `enable_user_mr=1`.

continued...

Date	Revision	Description
		• Clarified PSM3_TRACEMASK can disable output from PSM3_VERBOSE_ENV. • Improved description of PSM3_DEVICES • Updated web links in the following sections: Intel® MPI Library, Intel® MPI Library Installation and Setup, PSM3 Multi-Endpoint Functionality, Using Debuggers, Using the mpi_hosts File, Using the Open MPI mpirun script, Process Environment for mpirun, Further Information on Open MPI, Integration with a Batch Queuing System, Reference and Source for SLURM • PSM3_MULTI_EP has been deprecated. It is recommended this always be enabled. The default is enabled.
July 2021	1.1	Product 11.1.0.0 release - Changes to this document include: • NVIDIA GPU support added and discussed in the following sections and environment variables: Installed Layout, PSM3 Support for NVIDIA GPUDirect, PSM3 Verbs 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	Product 11.0.0.0 initial release.

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/support/articles/000088090/ethernet-products/intel-ethernet-software.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, also called libfabric) provider which implements an optimized user-level communications protocol. The audience for

continued...

Task	Document Title	Description
		this document includes cluster administrators and those running or implementing Message-Passing Interface (MPI) programs.
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 to a figure, table, or section in this guide. Links to websites are also shown in blue. For example:

  See License Agreements 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. 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

Creating a technical support ticket for Intel® Ethernet Fabric Suite products is available 24 hours a day, 365 days a year. Contact Intel® Customer Support or visit https://www.intel.com/content/www/us/en/support.html for additional details.

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 15 in this document.

For installation details, see the following document:

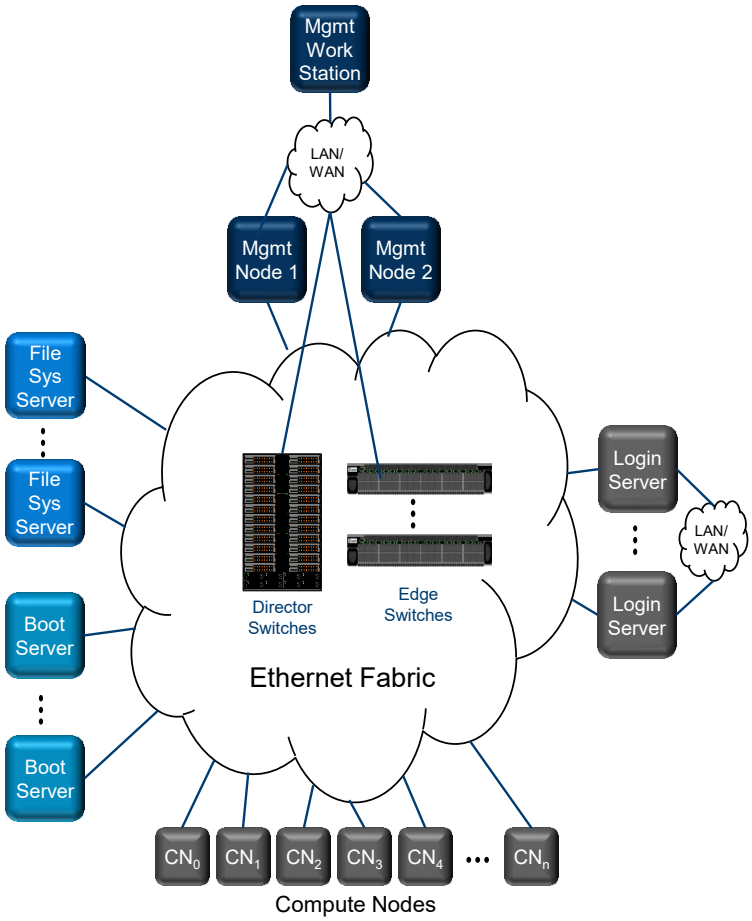
- *Intel® Ethernet Fabric Suite Software Installation Guide*

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 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



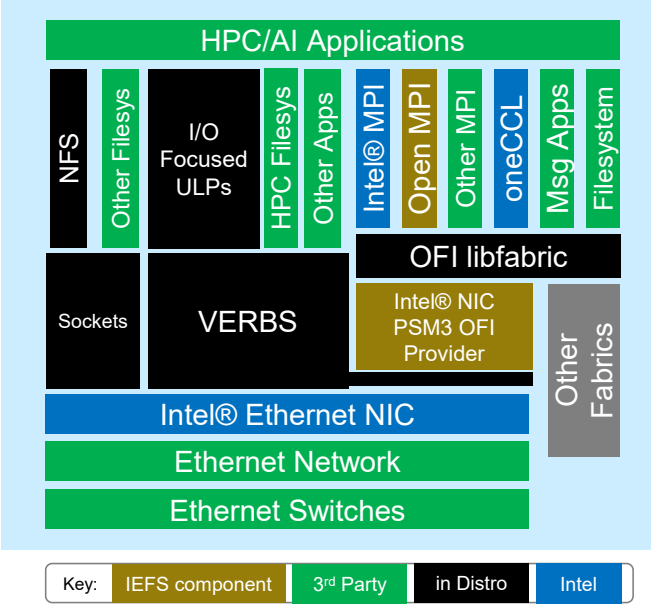
The software ecosystem is built around OFA software and includes three key APIs.

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 are 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), standard third-party Ethernet 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.

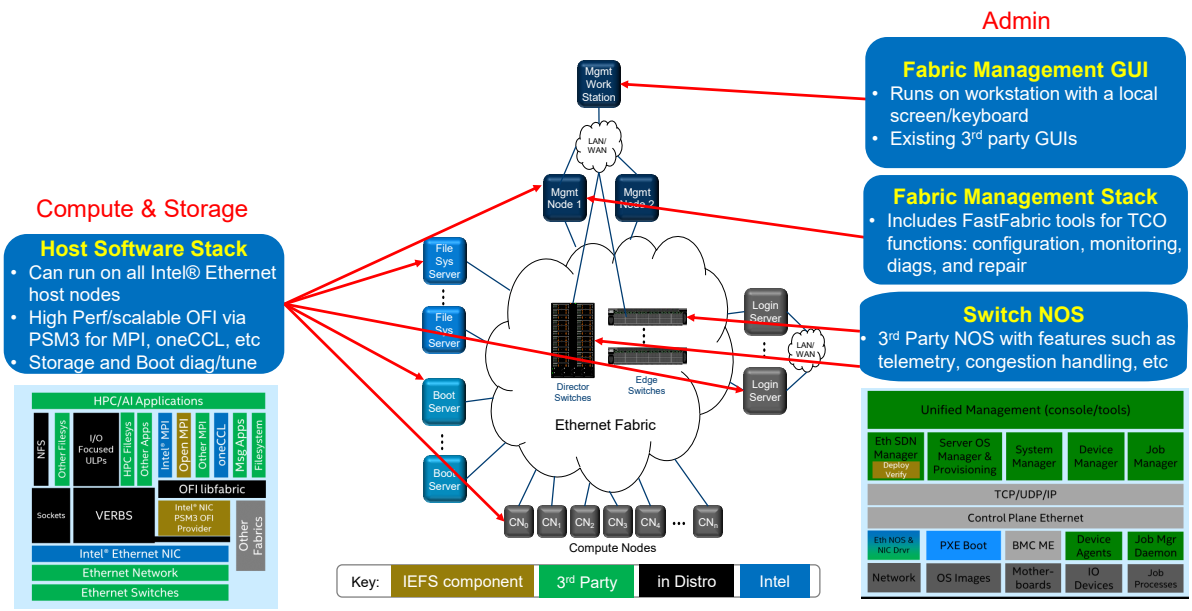
1.2 Intel® Ethernet Fabric Suite Software Overview

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 table.

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 through 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 hardware installation has been completed according to the instructions in the following documents:
-
- Make sure that software installation and driver configuration have 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`
Open MPI 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	RHEL and SLES: `/usr/lib64/libfabric/libpsm3-fi.so*` Ubuntu: `/usr/lib/x86_64-linux-gnu/libpsm3-fi.so*`
Intel® Ethernet Fabric Kernel Modules	RHEL and Ubuntu: `/lib/modules/<kernel version>/extra/iefs-kernel-updates/*.ko`

continued...

File Type	Location
	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 includes the appropriate kernel module and user space libraries to enable use of TCP/IP via sockets and RoCE (RDMA over Converged Ethernet) 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). 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 on the Intel® MPI Library and the Intel® oneAPI package that includes it.

NOTE

The Intel® MPI Library is feature-rich and highly optimized. For many applications it offers superior performance and capabilities as compared to other MPI libraries. Its 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 as part of the Intel® oneAPI package 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`.

Prior to launching MPI jobs, run the following command:

```
$ source $prefix/setvars.sh
```

NOTE

In the example above, `$prefix` is the path to the Intel® oneAPI installation. For example, if Intel® oneAPI is installed to its default location, `$prefix` may be `/opt/intel/oneapi/`

Alternatively, the following command may be used:

```
$ source $mpi_home/env/vars.sh
```

NOTE

In the example above, `$mpi_home` is the path to the Intel® MPI Library installation. For example, if Intel® oneAPI is installed to its default location, `$mpi_home` may be `/opt/intel/oneapi/mpi/<version>/`

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
`mpif77`	Fortran 77
`mpif90`	Fortran 90
`mpiifort`	Fortran 77/90 (uses Intel Fortran compiler)
`mpifc`	Fortran
`mpiifx`	Fortran (ifx)
`mpicc`	C

continued...

Wrapper Script Name	Language
mpicxx	Classic C++ and Data Parallel C++ (DPC++)
mpiicc	C (uses Intel C compiler)
mpiicpc	Classic C++ (uses Intel C++ compiler)
mpiicpx	Data Parallel C++
mpiicx	Data Parallel C
mpigcc	C (uses gcc)
mpigxx	C++ (uses g++)

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.

NOTE

When using mpicxx, the I_MPI_CXX environment variable may be used to select the compiler. The default is g++. When the Intel compilers are installed, you may also select icx (DPC++) or icc (classic C and C++). See https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-windows/top/environment-variable-reference/compilation-environment-variables.html for more information.

NOTE

See https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/command-reference/compiler-commands.html for more information on compiler commands and compiler selection.

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 one 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*. Also see Confirming the PSM3 Provider is Selected.

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 Controlling Process Placement with the Intel® MPI Library for information on controlling process placement with Intel® MPI Library.

4.4 Environment Variables for Intel® MPI Library Jobs

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.

The Intel® MPI Library provides its own environment variables that may control MPI features and may also effect which PSM3 features are used. 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

The following parameters may be exported by the Intel® MPI Library prior to loading PSM3. As such, the PSM3 defaults documented in PSM3 Environment Variables (as well as any settings in `/etc/psm3.conf`, see PSM3 Config File) may be overridden by the Intel® MPI Library. To see the actual values being used, set `PSM3_VERBOSE_ENV=1`: (see PSM3_VERBOSE_ENV).

- FI_PSM3_INJECT_SIZE - May be adjusted by the Intel® MPI Library.
- FI_PSM3_LAZY_CONN - Set to 1 for scale larger than `I_MPI_LARGE_SCALE_THRESHOLD` when `I_MPI_DYNAMIC_CONNECTION` is enabled (enabled by default).
- FI_PSM3_UUID - Set to a unique UUID generated by the Intel® MPI Library Hydra process manager as part of job launch (may be overwritten by `I_MPI_HYDRA_UUID`).
- PSM3_GPUDIRECT - Set to 1 when `I_MPI_OFFLOAD > 0` and `I_MPI_OFFLOAD_RDMA=1`. See discussion on *GPU Support* and *GPU Buffers Support* in the *Intel® MPI Library Developer Reference for Linux* OS*.
- PSM3_HAL - May be adjusted to select `verbs`.

- PSM3_MULTI_EP - Set to 1 for Multi-Endpoint (Multi-EP) and hand-off features (see Environment Variables for Multi-EP in the *Intel® MPI Library Developer Reference for Linux* OS*).

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. The error code 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 https://www.intel.com/content/www/us/en/developer/tools/documentation.html 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 MPI implementations run over OFA's Open Fabrics Interface (OFI, aka libfabric) and hence can use the Intel PSM3 OFI provider. Some supported MPI implementations 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/`*`COMPILER/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` may be replaced by *`$prefix`*.

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

```
/usr/mpi/COMPILER/MPI-VERSION/bin/mpirun
```

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` set up 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 endpoint 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 your 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 outputs the available MPI choices, such as:

- `openmpi_gcc_ofi-X.X.X`
- `openmpi_gcc_cuda_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 the 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.

Also see Confirming the PSM3 Provider is Selected.

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 (that is, 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 and Intel recommends that you specify the linker as `mpicc`, `mpif90`, etc. This specification automatically includes the correct flags and libraries, rather than manually trying to configure to pass the flags and libraries explicitly. For example:

```
LD=mpif90
```

These scripts pass the 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 (that is, 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, 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`. Intel also recommends that you 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 eight cores each, if the `hostfile` line is specified as `hostname1 slots=8 max-slots=8`, then Open MPI assigns a maximum of eight processes to the node and there can be no over-subscription of the eight 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 comma-separated host list such as:

  ```
  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

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 at https://www.open-mpi.org/doc; in particular, the following sections of the mpirun man page:

- Remote Execution

- Exported Environment Variables
- Setting MCA Parameters

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 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)
- Modify the 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 MPI take effect in the next shell that is started. See the `mpi-selector` man page for more information.

Each MPI registers 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 the 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`), which will be invoked on the remote nodes.

Additional details can be found at:

- Process Environment for mpirun
- Environment Variables for Intel® MPI Library Jobs
- Compiler and Linker Variables

6.0 Running oneCCL on Network Interface Cards

This section provides information on using the Intel® oneAPI Collectives Communications Library (oneCCL) on Network Interface Cards (NICs). Examples are provided for setting up the user environment, and references are provided for compiling and running oneCCL programs.

6.1 Introduction

oneAPI is a set of APIs and tools that provide a multi-vendor, cohesive environment for developing and executing high-performance applications on CPUs as well as various accelerated processing elements such as GPUs. Within oneAPI, oneCCL is a scalable and high-performance communication library for Deep Learning (DL) and Machine Learning (ML) workloads. It develops the ideas originated in the Intel® Machine Learning Scaling Library and expands the design and API to encompass new features and use cases.

6.2 oneCCL

oneCCL provides an efficient implementation of communication patterns used in deep learning. oneCCL uses MPI and libfabric to provide lower-level communications. oneCCL provides a rich set of features to enable application development and optimized execution including:

- Common API suitable for popular deep learning distributed frameworks (such as PyTorch and Horovod)
- Optimized for high performance on Intel CPUs and GPUs.
- Optimized to drive scalability of communication patterns by allowing to easily trade-off compute for communication performance
- Works across various interconnects, including Intel® Ethernet Fabric Suite

oneCCL is not included in the Intel® Ethernet Fabric Suite software, but is available separately. Refer to the following for more information about oneAPI and oneCCL:

- https://www.oneapi.io/
- https://www.oneapi.io/spec/
- https://spec.oneapi.io/versions/latest/elements/oneCCL/source/index.html
- https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html
- https://www.intel.com/content/www/us/en/developer/tools/oneapi/oneccl.html

NOTE

oneCCL is feature rich and highly optimized. For many applications it offers superior performance and capabilities as compared to other libraries. Its use with Intel® Ethernet Fabric Suite for relevant applications is highly recommended.

6.2.1 oneCCL Installation and Setup

Download oneCCL from https://www.intel.com/content/www/us/en/developer/tools/oneapi/oneccl.html as part of the Intel® oneAPI package and follow the installation instructions. The following subsections provide setup instructions for oneCCL.

6.2.1.1 Setting Up oneCCL

Before using oneCCL, make sure the `PATH` and `LD_LIBRARY_PATH` are set up.

When using oneCCL from the Intel® oneAPI Base Toolkit, run the following command:

```
$ source $prefix/setvars.sh
```

NOTE

In the example above, `$prefix` is the path to the Intel® oneAPI installation. For example, the default installation location is `/opt/intel/oneapi/`.

6.2.1.2 Compiling Applications Using oneCCL

Intel® oneAPI supports a variety of programming languages that can use oneCCL, including:

- Data Parallel C++ (DPC++)
- Classic C++

For more information, see https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html

6.2.2 Running Applications that Use oneCCL

When running applications that use oneCCL, Intel® MPI is typically used to launch jobs. See Intel® MPI Library for more details.

Also see Confirming the PSM3 Provider is Selected.

6.3 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.

The Intel® MPI Library and oneCCL each have their own environment variables that may control MPI and oneCCL features and may also effect which PSM3 features are used. The Intel® MPI process launcher (`mpirun`) 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 processes are active. See the Intel® MPI Library documentation for information, specifically https://software.intel.com/en-us/mpi-developer-reference-linux, and the oneCCL documentation, specifically https://www.intel.com/content/www/us/en/develop/documentation/oneccl-developer-guide-and-reference/top.html

oneCCL provides mechanisms for explicit control over multi-rail configurations. For more information, see https://www.intel.com/content/www/us/en/develop/documentation/oneccl-developer-guide-and-reference/top/env-variables.html and the section on Multi-NIC. In some configurations, when oneCCL is explicitly doing multi-rail load balancing, it may be advantageous to set PSM3_MULTIRAIL=`-1` to avoid oneCCL mistakenly using the `autoselect_one` fabric interface.

6.4 Debugging oneAPI and oneCCL Applications

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. To help in this area, Intel® oneAPI toolkit includes the Intel® Distribution for GDB. The Intel® Distribution for GDB delivers a unified debugging experience that allows for efficient and simultaneous debug of cross-platform parallel and threaded applications developed in Data Parallel C++ (DPC++), C, C++, OpenMP, SYCL, or Fortran. See https://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-for-gdb.html for more information.

7.0 PSM3 Support for GPUs

PSM3 has been designed such that it can perform well with both CPU-based and GPU-based servers and applications. When using GPUs, PSM3 includes Direct GPU access optimizations for data movement directly to and from GPUs. This includes both Direct Intel GPU access and NVIDIA GPUDirect

For more information on use and tuning of PSM3 wih GPUs, 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*

NOTE

Throughout this document, the term *Direct GPU access* is used to refer to data movement directly to and from both Intel and NVIDIA GPUs, including:

- Direct GPU Copy (also referred to as GPUDirect Copy for NVIDIA GPUs)
- Direct GPU Send DMA (also referred to as GPUDirect Send DMA for NVIDIA GPUs)
- Direct GPU RDMA (also referred to as GPUDirect RDMA for NVIDIA GPUs)

7.1 PSM3 Support for Intel GPUs

When using the oneAPI environment on Intel® Data Center GPUs, PSM3 can take advantage of various GPU specific optimizations. Including:

- oneAPI and Level Zero (ZE) APIs and libraries
- Direct GPU access optimizations for data movement directly to and from Intel GPUs
- The oneAPI Collectives Communications Library (oneCCL)

Refer to the following for more information about oneAPI:

- https://www.oneapi.io/
- https://www.oneapi.io/spec/
- https://spec.oneapi.io/level-zero/latest/index.html
- https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html
- https://www.intel.com/content/www/us/en/developer/tools/oneapi/oneccl.html

Multiple Processes per Intel GPU

For the majority of use cases, MPI applications using GPUs assign one to four MPI ranks per GPU tile. These MPI ranks can run concurrently, including all the required communications. In the case where it is desired to run more than four ranks per GPU tile, it may be necessary to use the Computer Aggregation Layer (CAL). See https://github.com/intel/compute-aggregation-layer for more details.

7.1.1 PSM3 Support for Direct Intel GPU Access

oneAPI via its Level Zero library includes technology that allows for network adapters to directly read and write to oneAPI host and GPU memory to enhance performance for latency and bandwidth. PSM3 has support for oneAPI Level Zero (ZE), Direct GPU Copy, Direct GPU Send DMA, and Direct GPU RDMA. Additionally, the MPI benchmark source included with FastFabric includes some Intel GPU-enabled benchmarks that FastFabric can assist to build and distribute within a cluster.

PSM3's Direct GPU features can be used to accelerate oneAPI-based workloads and benchmarks for servers with Intel® Data Center GPUs.

NOTE

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

7.1.1.1 Using PSM3 Features for Direct Access to Intel GPUs

The PSM3 features required for Direct GPU access and Level Zero are disabled (0) by default. To use these features:

- Use a oneAPI GPU-enabled application.
- Ensure you are using a oneAPI GPU-enabled MPI or middleware, such as `/opt/intel/oneapi/mpi/<version>`.
- Enable PSM3 features:
  - `PSM3_ONEAPI_ZE=1`
  - `PSM3_GPUDIRECT=1`

NOTES

- Not enabling these features for GPU-enabled workloads may result in application segfaults or other crashes.
- When `PSM3_GPUDIRECT=1`, this implicitly also sets `PSM3_ONEAPI_ZE=1`.
- When `PSM3_GPUDIRECT=1`, the rendezvous module (rv) will be required to assist in Direct GPU access features. The Intel GPU-enabled rendezvous module (oneapize) must be loaded.
- GPU applications may be run without use of Direct GPU access by specifying only `PSM3_ONEAPI_ZE=1`. The rendezvous module may not be required. See PSM3 Verbs RDMA Modes and Rendezvous Module for more information.
- When using the sockets Hardware Abstraction Layer (HAL), only Direct GPU Copy is available. See PSM3 Architecture and Hardware Abstraction Layer.
- For special cases, there may be reasons to disable Direct GPU access support; however, you should leave oneAPI Level Zero support enabled. Refer to PSM3 OFI Provider or the *Intel® Ethernet Fabric Performance Tuning Guide* for more information.
- Enabling PSM3 oneAPI Level Zero and/or Direct GPU access for an application that does not use the GPU, or a middleware that handles all the GPU aspects itself, may reduce the performance of the job.
- `PSM3_GPUDIRECT=1` may be automatically set by Intel MPI, see Environment Variables for Intel® MPI Library Jobs.

7.1.2 PSM3 Support for oneCCL

oneCCL is a oneAPI library that implements various collective algorithms optimized for CPU and GPU environments, including both multi-node and multiple CPU or GPU per node environments. The oneCCL library may be used by various applications and middlewares, such as various deep learning frameworks. oneCCL is designed to directly use OpenFabrics Alliance (OFA Open Fabrics Interface (OFI) (also known as libfabric) for communications.

Refer to https://docs.oneapi.io/versions/latest/index.html and https://oneapi-src.github.io/oneCCL/ for more information about oneCCL.

7.1.2.1 Running with oneCCL

A simple way to confirm oneCCL is working properly is via the oneAPI oneCCL benchmarks that are included in oneAPI as example oneCCL programs.

NOTE

During the first job after installation of oneCCL, Intel recommends that you set the environment variable `CCL_LOG_LEVEL=info`. Then, review the output to ensure PSM3 is being used. Intel also recommends that you enable PSM3_IDENTIFY to confirm PSM3 and the proper NICs are being selected. Also see Confirming the PSM3 Provider is Selected.

NOTE

In some configurations, when oneCCL is explicitly doing multi-rail load balancing, it may be advantageous to set PSM3_MULTIRAIL=-1 to avoid oneCCL mistakenly using the `autoselect_one` fabric interface.

Also see PSM3 and Intel GPU SupportPSM3 and NVIDIA CUDA Support

7.2 PSM3 Support for NVIDIA GPUs

When using the NVIDIA CUDA environment on Kepler architecture-based GPUs or newer GPUs, PSM3 can take advantage of various GPU specific optimizations, including:

- NVIDIA CUDA APIs and libraries
- NVIDIA GPUDirect optimizations for data movement to and from NVIDIA GPUs
- The NVIDIA Collectives Communications Library (NCCL)

NVIDIA Multi-Process Service (MPS)

For the majority of use cases, MPI applications using GPUs assign one MPI rank per GPU. All of the communication for the GPU is handled through the single MPI rank. In the case where it is desired to use multiple MPI ranks per GPU, significantly lower performance may be seen. In this case, it may be beneficial to deploy NVIDIA Multi-Process Service(MPS). See https://docs.nvidia.com/deploy/mps/index.html for more details.

7.2.1 PSM3 Support for NVIDIA 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 that 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 for more information about GPUDirect:

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

NOTE

Throughout this document, the term *Direct GPU access* is used to refer to data movement directly to and from both Intel and NVIDIA GPUs. This includes:

- Direct GPU Copy (also referred to as GPUDirect Copy for NVIDIA GPUs)
- Direct GPU Send DMA (also referred to as GPUDirect Send DMA for NVIDIA GPUs)
- Direct GPU RDMA (also referred to as GPUDirect RDMA for NVIDIA GPUs)

7.2.1.1 Using PSM3 Features for NVIDIA 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 `/opt/intel/oneapi/mpi/<version>` or `openmpi-x.x.x-cuda-ofi`.
- Enable PSM3 features:
  - `PSM3_CUDA=1`
  - `PSM3_GPUDIRECT=1`

NOTES

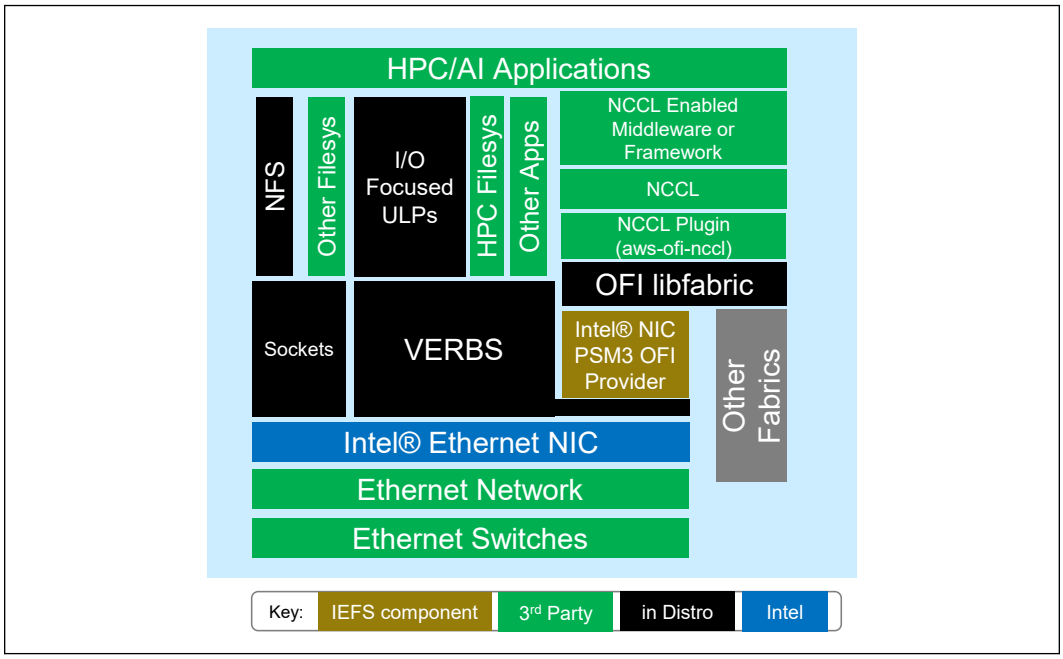
- Not enabling these features for CUDA-enabled workloads may result in application segfaults or other crashes.
- When `PSM3_GPUDIRECT=1`, this implicitly also sets `PSM3_CUDA=1`.
- When `PSM3_GPUDIRECT=1`, the rendezvous module (rv) will be required to assist in GPUDirect features. The CUDA-enabled rendezvous module must be loaded.
- CUDA applications may be run without use of GPUDirect by specifying only `PSM3_CUDA=1`. The rendezvous module may not be required. See PSM3 Verbs RDMA Modes and Rendezvous Module for more information.
- When using the sockets Hardware Abstraction Layer (HAL), only GPUDirect Copy is available. See PSM3 Architecture and Hardware Abstraction Layer.
- 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 PSM3 CUDA and/or GPUDirect for an application that does not use CUDA, or a middleware that handles all the GPU aspects itself, may reduce the performance of the job.
- `PSM3_GPUDIRECT=1` may be automatically set by Intel MPI, see Environment Variables for Intel® MPI Library Jobs.

7.2.2 PSM3 Support for NVIDIA NCCL

NCCL is an NVIDIA library that implements various collective algorithms optimized for multi-GPU environments, including both multi-node and multiple GPU per node environments. The NCCL library may be used by various applications and

middlewares, such as various deep learning frameworks. In order for NCCL to use a network not already built into the NCCL library, a NCCL plugin is required. In order to use OFI and PSM3 as the network for NCCL, the aws-ofi-nccl plugin is used.

Figure 4. Intel® EFS Host Fabric Software Stack When Using NCCL



Refer to the following for more information about NCCL:

- https://developer.nvidia.com/nccl
- https://github.com/NVIDIA/nccl

7.2.2.1 Installing the NVIDIA NCCL OFI Plugin

The aws-ofi-nccl plugin is distributed as source code that must be built by the end user. The source code for the aws-ofi-nccl plugin can be found at https://github.com/aws/aws-ofi-nccl/releases.

Prior to building the plugin, the following must have been installed on the system (including both runtime and development libraries and headers):

- NVIDIA CUDA
- NCCL
- OFI (libfabric)
- Open MPI enabled for OFI and CUDA (included in Intel® Ethernet Fabric Suite)

In order to build and install this plugin, follow the instructions at https://github.com/aws/aws-ofi-nccl.

After completion of a successful build, the plugin will have been installed to `/usr/local/lib`, unless it is installed to custom path through the use of `--prefix=PATH`.

7.2.2.2 Running with NVIDIA NCCL

When running applications or frameworks that use NVIDIA NCCL, the network plugin will be found via the `LD_LIBRARY_PATH`. The aws-ofi-nccl plugin will by default be installed to `/usr/local/lib`, which is part of the default `LD_LIBRARY_PATH`.

A simple way to confirm NCCL is working properly is via the NVIDIA NCCL test suite that can be found at https://github.com/nvidia/nccl-tests.

After having installed nccl_tests, Open MPI's mpirun command may be used to launch any of the nccl-test programs, such as this example of a four-process job:

```
mpirun -np 4 -machinefile mpi_hosts -mca mtl ofi -x FI_PROVIDER=psm3 -x
PSM3_CUDA=1 -x PSM3_GPUDIRECT=1 -x PSM3_RDMA=1 nccl-tests/all_reduce_perf -b 8 -e
128M -f 2
```

NOTE

See Setting up Open MPI for more information on various ways to setup and run Open MPI.

NOTE

During the first job after installation of NCCL and the plugin, Intel recommends that you set the environment variable `NCCL_DEBUG=info`. Then, look for `via NET/AWS Libfabric/1/GDRDMA` in the output to confirm the aws-nccl-plugin is being used and NCCL has recognized that the plugin and PSM3 are CUDA enabled. Intel also recommends that you enable PSM3_IDENTIFY to confirm PSM3 is being selected. Also see Confirming the PSM3 Provider is Selected.

NOTE

Depending on the exact NIC and GPU placement in the server, in order to enable GPU Direct, the environment variable `NCCL_NET_GDR_LEVEL` may need to be set to a non-default value. See https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html

NOTE

In some configurations, when NCCL is explicitly doing multi-rail load balancing, it may be advantageous to set PSM3_MULTIRAIL=`-1` to avoid NCCL mistakenly using the `autoselect_one` fabric interface.

See https://github.com/NVIDIA/nccl-tests for more details on the arguments to nccl-tests applications. See Open MPI for more details on running Open MPI. Also see PSM3 and NVIDIA CUDA Support

8.0 PSM3 OFI Provider

8.1 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 high-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.

8.2 Differences Between PSM3 and PSM2

The Intel PSM3 interface differs from PSM2 (used by Omni-Path) in the following ways:

- PSM3 includes new features and optimizations for Intel® Ethernet Fabric hardware and newer 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 (RDMA over Converged Ethernet, version 2) and sockets TCP/IP 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*.

8.3 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 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 are not met.

8.4 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 that is not provider specific.
- PSM3 may generate its own value based on the Linux user id.

NOTE

The UUID is 128 bits, but is not cryptographic in nature. If you want heightened security for jobs, you 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 you to specify the UUID to be passed to PSM3 or control its generation. For example, the Intel® MPI Library has controls such as `I_MPI_SPAWN=1` that 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.

8.5 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.

8.6 PSM3 Multi-Endpoint Functionality

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

PSM3 has added minimal thread safety for use 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 Library 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 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 and view the Intel® MPI and oneAPI documentation.

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

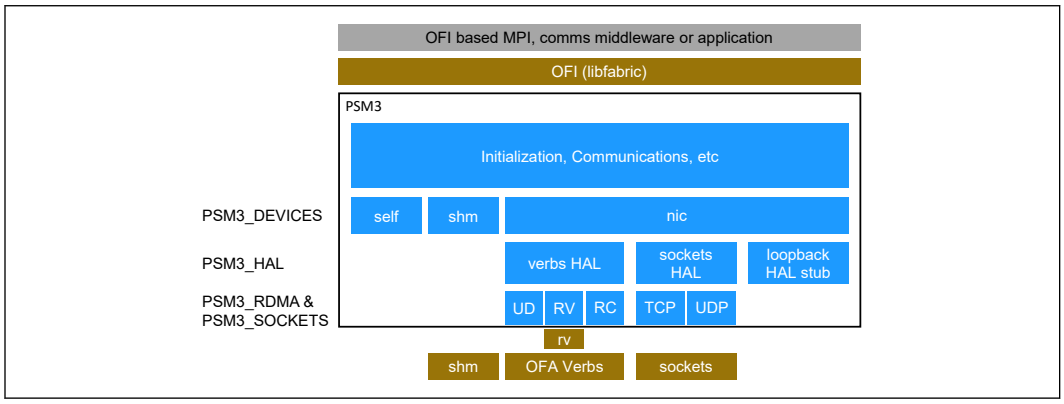
  The `psm3` provider supports scalable endpoints.

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

8.7 PSM3 Architecture and Hardware Abstraction Layer

The following figure is a simplified view of the key elements in the PSM3 architecture and shows the interactions between various concepts such as PSM3_DEVICES, PSM3_HAL, and data movement mechanisms such as PSM3 Verbs RDMA Modes and Rendezvous Module (PSM3_RDMA) and PSM3 Sockets Modes (PSM3_SOCKETS).

Figure 5. PSM3 Architecture



PSM3 is an OFI (libfabric) provider. As such, PSM3 implements the necessary OFI provider API and runs immediately below libfabric. Above libfabric, a variety of communications middleware or applications may be run. MPI is one such example of a communications middleware.

Internal to PSM3 are a set of sophisticated algorithms and mechanisms to efficiently implement communications in support of the libfabric APIs.

PSM3 has three major subsystems for communications, referred to as `self`, `shm`, and `nic` and controlled via PSM3_DEVICES.

- `self` - Allows a process to send messages to itself.
- `shm` - Allows a process to send messages to other processes on the same host via a variety of mechanisms including: Linux shared memory (shm), direct CPU process to CPU process copies, direct GPU to GPU transfers, and/or the Data Streaming Accelerator (DSA).
- `nic` - Allows a process to send messages to processes on other hosts.

A given job may use any combination of these three subsystems. By default, all three are enabled.

In most use cases, PSM3's `nic` subsystem will be enabled and used to communicate to processes on other hosts that are part of a multi-node job within a cluster. Within the `nic` subsystem, a single Hardware Abstraction Layer (HAL) is selected to provide a wire protocol and data movement strategy across the network. HAL selection is further explained in NIC and Address Filtering and may be directly controlled by PSM3_HAL. The following HAL implementations are available:

- `verbs` - Makes use of Open Fabrics Alliance verbs APIs to move data using RDMA-enabled NICs.
- `sockets` - Makes use of Linux sockets APIs to move data using the TCP/IP family of protocols.

- `loopback` - A stub internal to PSM3 that is only used when the `nic` device is not enabled. This facilitates some basic initialization steps, but does not perform any actual communications.

A given HAL may have options for data movement protocol and strategy. Such options may trade-off performance, scalability, memory footprint, etc.

The preferred HAL is `verbs`. The `verbs` HAL takes advantage of the Open Fabrics Alliance verbs API to make use of kernel bypass and RDMA to achieve optimized latency, and bandwidth with low CPU utilization. The `verbs` HAL implements multiple data movement protocols as outlined in PSM3 Verbs RDMA Modes and Rendezvous Module which may be directly controlled by PSM3_RDMA. Some of the modes will take advantage of the PSM3 Rendezvous Kernel Module to optimize scalability and efficiency.

An alternate HAL is `sockets`. The `sockets` HAL makes use of the Linux sockets API to implement a couple data movement protocols as outlined in PSM3 Sockets Modes which may be directly controlled by PSM3_SOCKETS. Most NICs will support sockets, but sockets lacks kernel bypass and often depends on interrupts and additional data copies to move data. However, in systems without an RDMA-capable NIC, this may be the best choice for running applications.

HAL selection occurs early during process launch and exactly one HAL will be selected for all PSM3 node-to-node communications by a given process. All processes in a job must use the same HAL. By default, PSM3 will select the HAL based on which acceptable NICs it finds. For more information, see NIC and Address Filtering.

NOTE

The PSM3 Rendezvous Kernel Module (rv) may also be used to enable Direct GPU communication optimizations when using some GPU devices. While not directly depicted above, the rv module can be used by the `verbs` or `sockets` HALs for this purpose. See PSM3 Support for Direct Intel GPU Access, PSM3 and Intel GPU Support, PSM3 Support for NVIDIA GPUDirect, PSM3 and NVIDIA CUDA Support, and PSM3_GPUDIRECT.

NOTE

PSM3 build options control which HALs are included in the PSM3 binary as well as which data movement protocols are available within each included HAL. See Building the PSM3 RPM.

8.8 NIC and Address Filtering

Modern servers often have more than one NIC, and each NIC may have varied performance, connectivity, and features. As such, PSM3 must select which NIC(s) it will use for a given job. To this end, PSM3 attempts to make reasonable NIC selections by default, but may need end user guidance to ensure the preferred NIC(s) are selected. To accomplish this, PSM3 has a number of filters which are applied based on NIC capabilities, names, addresses, address types, and current link speed to select which NIC(s) will be considered. In addition, each NIC may have more than one address. For example, a NIC might have both IPv4 and IPv6 addresses assigned, and it may have multiple IPv4 and/or multiple IPv6 addresses assigned.

The following filters are available and applied to decide which NIC(s) and addresses within each NIC will be considered for use in a given job:

- PSM3_HAL - This filter can limit NIC selection to NICs that support the API and capabilities required by the given Hardware Abstraction Layer (HAL), namely NICs that support the verbs API versus those which support the sockets API.
- NIC port status - NICs whose link is not *active* or do not have a valid, assigned address are always excluded from selection.
- PSM3_NIC - This filter can limit NIC selection to a specific NIC name, name pattern, or unit number, essentially excluding all other NICs from selection.
- PSM3_ADDR_FMT - This filter can limit which addresses within a NIC are considered. NICs that have no addresses of the given type are excluded.
- PSM3_SUBNETS - This filter can limit which addresses within a NIC are considered based on their subnet. NICs that have no addresses matching the list of acceptable subnets are excluded.
- PSM3_ADDR_PER_NIC - This filter can limit NIC and address type selection. NICs and address types within a NIC that have less than PSM3_ADDR_PER_NIC unfiltered addresses of the given type are excluded.
- PSM3_NIC_SPEED - This filter can limit which NICs are considered based on their link speed. NICs whose current link speed does not match are excluded.

After applying the above filters, the set of NICs that pass all of the filters will be considered for use in the job. Within each considered NIC, the first PSM3_ADDR_PER_NIC addresses that pass all of the filters will be used for the given NIC. The actual selection of NIC per process may then occur based on other selections such as those discussed in PSM3 Multi-Rail Support and PSM3 Multi-IP Support and controlled via PSM3_ALLOW_ROUTERS, PSM3_MULTIRAIL, PSM3_MULTIRAIL_MAP, and PSM3_NIC_SELECTION_ALG.

NOTE

Even if a NIC is filtered out, it is still assigned a unit number based on an alphabetic sort by name among the NICs supported by a given HAL. As such, unit numbers remain constant within a given HAL regardless of which NICs have been filtered out. Such unit numbers may be used in environment variables such as PSM3_NIC and PSM3_MULTIRAIL_MAP, however those variables must select a unit that has not been filtered out. Be aware that the unit number of a given hardware NIC is often different within each HAL.

NOTE

The HAL, NIC(s), and addresses selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the HAL, NIC, and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

NOTE

PSM3 detects all RDMA and sockets devices, so if multiple types of RDMA or sockets 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.

8.9 PSM3 Multi-Rail Support

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

8.9.1 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 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.

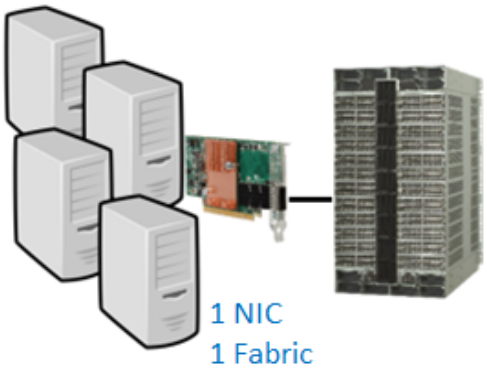
Even when each PSM3 process uses only a single NIC, the NIC used by various processes may differ. The multiple NICs may effectively be used and load balanced during a given job. The degree of load balancing will depend on the application's traffic patterns.

TERMINOLOGY:

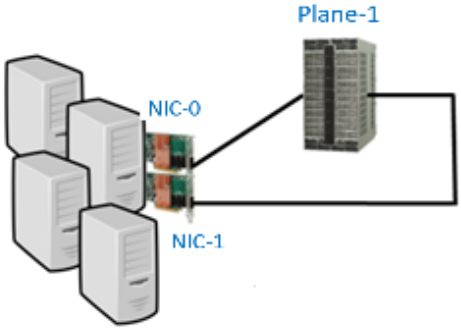
- Planes can sometimes be referred to as *fabrics*.
- Hosts can also be referred to as *nodes*.
- NICs can also be referred to as *rails*.
- Processes can also be referred to as *ranks*.

Three basic scenarios include:

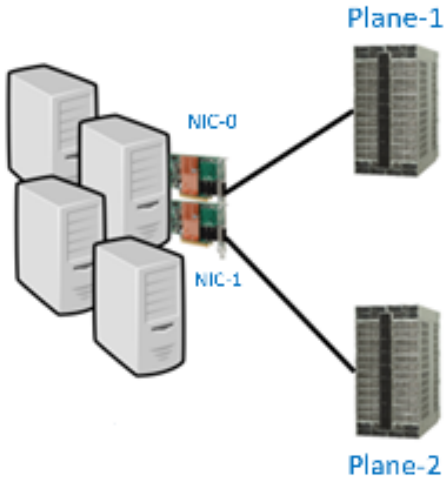
- Single-rail in a single plane: This scenario, shown in the following figure, consists of one NIC in a server connected to one plane. This is the default configuration during installation. This configuration provides the performance required by most applications in use today.



- 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 one or more Ethernet IP subnets. When there is more than one 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.

8.9.2 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 want to only use a single NIC per process, Intel recommends that you use one or more of the mechanisms in NIC and Address Filtering to limit the NICs used to a single plane. As needed, PSM3_NIC_SELECTION_ALG may then also be used 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 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.

NOTE

Some middleware implementations, such as MPI, OneCCL or NCCL, may provide mechanisms to directly control NIC usage and load balancing in the middleware. In this case, it may be beneficial to avoid and disable PSM3 multi-rail and automatic NIC selection mechanisms.

8.9.3 Multi-Rail 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.

NOTE

The mechanisms outlined in NIC and Address Filtering may be used to limit which NICs PSM3 will consider using.

The following environment variables can be set:

- `PSM3_MULTIRAIL=n`, where `n` can be a value between -1 and 4.

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

  If set to a 1, 2, 3 or 4, 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 NIC and Address Filtering or `PSM3_NIC_SELECTION_ALG`.

  When set to -1, all PSM3 NIC selection and load balancing mechanisms are disabled and the middleware above is responsible for selecting NICs and load balancing.

  The `PSM3_MULTIRAIL` options are:

  — `-1` - No NIC autoselection nor multi-rail within PSM3. The middleware above is responsible for NIC selection and load balancing.

  - `0` - Single NIC per process configuration (default fabric interface). However, the middleware above may also chose to explicitly open PSM3 fabric interfaces for individual NICs and do its own load balancing.
  - `1` - Enables multi-rail capability and each process will use all available NIC(s) in the system.
  - `2` - Enables 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).
  - `3` - Enables multi-rail capability and limits NIC(s) used by a given process to NIC(s) within a single NUMA socket and equally close to the process's GPU (when applicable). PSM3 will look for an available NIC (and select at least one, even if it is not NUMA-local to the process).
  - `4` - Enables multi-rail capability and limits NIC(s) used by a given process to those equally close to the process's GPU and if possible, also within the process's NUMA socket.
- `PSM3_MULTIRAIL_MAP=rail,rail,rail,...`

  Specifies the NIC and address(es) to use for each rail. Multiple `rail` are separated by a comma. Each `rail` may be specified as an RDMA device name or device unit number (starting at 0), and optionally a PSM3 Multi-IP Support address index.

  If only one rail is specified, it is equivalent to a single-rail case. The `rail` specified is the only rail used.

  `PSM3_MULTIRAIL_MAP` also allows specification of unique sets of rails per process on a given node, see PSM3_MULTIRAIL_MAP for more information.
- `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, if unfiltered NICs are not alphabetically in the same order on each node, `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 on all nodes in the whole job.

If `PSM3_MULTIRAIL` is set to 1, 2, 3, or 4, the following occurs:

- For individual messages less than the window size selected by `PSM3_RNDV_NIC_WINDOW` (`PSM3_GPU_RNDV_NIC_WINDOW` for GPU messages), the available NICs are used in a round-robin fashion to send messages.
- When using RDMA and rendezvous for larger messages, messages greater than the window size selected by `PSM3_RNDV_NIC_WINDOW` (`PSM3_GPU_RNDV_NIC_WINDOW` for GPU messages) are striped such that a single message can take advantage of more than one NIC. See PSM3 Verbs RDMA Modes and Rendezvous Module.

- The use of smaller window sizes in `PSM3_RNDV_NIC_WINDOW` and `PSM3_GPU_RNDV_NIC_WINDOW` may be used to force striping of smaller messages.
- Messages will not be striped at a size lower than `PSM3_MQ_RNDV_NIC_THRESH` (`PSM3_GPU_THRESH_RNDV` for GPU messages) because an eager, instead of rendezvous, receive protocol is used. Multi-rail striping only occurs for rendezvous when using RDMA. When decreasing this value, be aware 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 and the default PSM3 fabric interface is opened, the following occurs:

- Each rank sends all its messages using a single nearestNIC selected based on PSM3_NIC_SELECTION_ALG

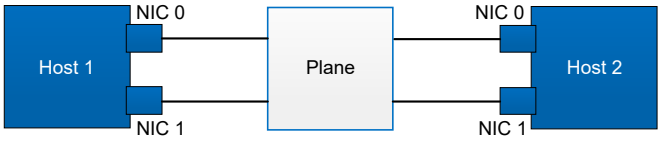
For more information see PSM3 OFI Provider.

8.9.4 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 or is 0. Each PSM3 rank uses a single NIC. The PSM3 provider will present a unique OFI fabric interface for each NIC. The first interface will be an `autoselect_one` interface permitting PSM3 to automatically select among the NICs, with different choices per process. Middleware above PSM3 may alternatively use any of the other presented fabric interfaces to open a specific NIC. Such middleware may also choose to open more than one specific NIC and perform load balancing in the middleware. If you do not want PSM3 to potentially use different NICs for each rank, you must specify the single desired NIC using PSM3_NIC or another filtering mechanism in NIC and Address Filtering.
- `PSM3_MULTIRAIL=-1`. PSM3 behaves similar to `PSM3_MULTIRAIL=0` except the `autoselect_one` interface is not presented. Middleware above PSM3 must explicitly select which of the presented fabric interfaces to open and may choose to perform load balancing in the middleware.

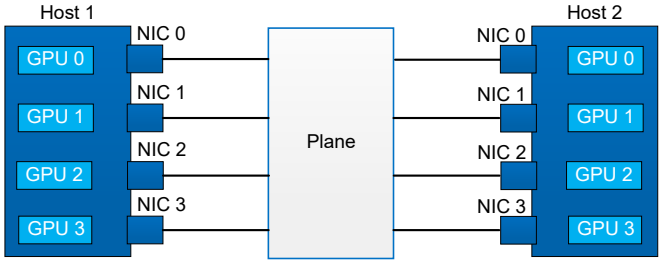
- `PSM3_MULTIRAIL=1`. PSM3 discovers that there are two NICs in the system. The first available NIC is 0. The next available NIC is 1. PSM3, by default, uses a `PSM3_MULTIRAIL_MAP` that includes all NICs (`NIC0,NIC1`). PSM3 sets up the first (primary) connection over NIC 0 and the second (secondary) connection over NIC 1.
- `PSM3_MULTIRAIL=1` and `PSM3_MULTIRAIL_MAP=NIC1,NIC0`. PSM3 uses the given units as specified. PSM3 sets up the primary connection over NIC 1 and the secondary connection over NIC 0.
- `PSM3_MULTIRAIL=2`, `3`, or `4` is set. Each rank will use only NICs close to the process (each of 2, 3, 4 have different criteria for closeness). Refer to PSM3 OFI Provider and PSM3_MULTIRAIL for more information.

NOTE

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

Single plane, multi-rail with GPUs

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



In general the behavior is the same as the previous dual rail example. However, if the platform design utilizes PCIe switches such that GPU 0 and NIC 0 are on the same PCIe switch, NIC 1 and GPU 1 are their own PCIe switch, etc., then Direct GPU Access may perform better by using features in the middleware or application to control which CPU and GPU each rank will use and then taking advantage of `PSM3_MULTIRAIL=4` or `PSM3_MULTIRAIL_MAP` to select the appropriate NIC for each process.

If the platform design does not include PCIe switches, then Direct GPU Access may perform better by using features in the middleware or application to control which CPU and GPU each rank will use and then taking advantage of `PSM3_MULTIRAIL=3` or `PSM3_MULTIRAIL_MAP` to select the appropriate NIC for each process.

Example Environment Variables

- `PSM3_MULTIRAIL=4`. PSM3 uses the location of the process's GPU with a secondary consideration of the process's CPU's NUMA socket to select NICs. In this example the process using GPU 0 will use NIC 0, the process using GPU 1 will use NIC 1, etc. This choice may provide better Direct GPU Access performance than other choices.
- `PSM3_MULTIRAIL=3`. PSM3 uses the location of the process's CPU with a secondary consideration of the process's GPU to select NICs. In a platform with PCIe switches, this will typically behave the same as `PSM3_MULTIRAIL=4`. However. in a platform without PCIe switches, NIC 0 and 1 may be equally close to GPU 0 and 1, in which case the process on GPU 0 may use NIC 0 and NIC 1, the process on GPU 1 may use NIC 0 and NIC 1, etc. On such platforms, this choice may provide better Direct GPU Access performance than other choices.
- `PSM3_MULTIRAIL=1` and `PSM3_MULTIRAIL_MAP=NIC0;NIC1;NIC2;NIC3`. PSM3 uses the local rank of each process to select the desired NICs. In this example local rank 0 will use NIC 0, local rank 1 will use NIC 1, etc. Assuming the middleware or application has similarly chosen for local rank 0 to use GPU 0, local rank 1 to use GPU 1, etc., this choice may provide good Direct GPU Access performance.

NOTE

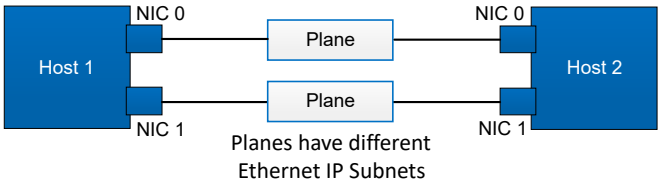
If some or all of the NICs have different Ethernet IP subnets, `PSM3_ALLOW_ROUTERS=1` must be specified. In this case, PSM3 will ignore any differences in Ethernet IP subnets and assume each NIC can communicate with all the other NICs.

NOTE

When using NVIDIA GPUs, if more than 1 GPU is visible to the current process, at the time of PSM3 endpoint initialization PSM3 will identify the GPU being used by the process via `cuCtxGetCurrent` and `cuCtxGetDevice`. To limit the GPUs visible to a given process, the NVIDIA environment variable `CUDA_VISIBLE_DEVICES` can be exported with a list of GPU device numbers.

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 or is 0. PSM3 may not work because there are multiple planes. NIC 0 on Host 1 has no connectivity to NIC 1 on Host 2.
- `PSM3_MULTIRAIL=-1`. The PSM3 provider will present a unique OFI fabric interface for each NIC. Middleware above PSM3 must explicitly select which of the presented fabric interfaces to open and may choose to perform load balancing in the middleware. Such middleware must be aware of the multi-plane nature of the configuration.
- `PSM3_MULTIRAIL=1`. PSM3 discovers that there are two NICs in the system. PSM3, by default, uses a `PSM3_MULTIRAIL_MAP` that includes all NICs (`NIC0,NIC1`). PSM3 connects the primary rail on NIC 0 of Host 1 with NIC 0 on Host 2. The secondary rail is set up on NIC 1 of Host 1 with NIC 1 of Host 2. PSM3 works in this configuration/setting.
- `PSM3_MULTIRAIL=1` and `PSM3_MULTIRAIL_MAP=NIC1,NIC0`. 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`, `3`, or `4`. PSM3 may not work because there are multiple planes. NIC 0 on Host 1 has no connectivity to NIC 1 on Host 2.

NOTE

If some or all of the NICs within a given plane have different Ethernet IP subnets, `PSM3_ALLOW_ROUTERS=1` must be specified, and 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.

8.10 PSM3 Multi-IP Support

In addition to being able to use multiple network interface cards (NICs) to transfer messages, PSM3 also supports using multiple IP addresses per NIC. This section defines terminology, explains user scenarios, and describes implementation details for MPI application programmers.

8.10.1 Multi-IP Overview

Modern NICs allow multiple IP addresses to be defined for a single, physical, connection to the network. This allows a single physical connection to receive messages that were routed through different paths in the fabric, potentially reducing congestion.

Multi-IP is an extension of multi-rail. It further extends the load balancing capabilities provided by multi-rail by allowing data traffic to be spread across multiple switch-to-switch links, even in a single plane. Depending on the application, this may further improve application performance by reducing network congestion.

The multi-IP feature can be applied to a single NIC, single plane or multiple plane configuration. By enabling multi-IP, a process can use multiple IP addresses to transfer messages (essentially treating IP addresses as if they were unique rails).

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

NOTE

This feature is only useful in fabrics that have multiple switch-to-switch links between sources and destinations, such as a fat-tree, and where the edge switches have been configured so that different IP addresses for a given NIC are routed over different switch-to-switch links. Fabrics that do not offer multiple switch-to-switch links between destinations, or are not correctly configured, will not see an improvement in performance when using multiple IP addresses per NIC.

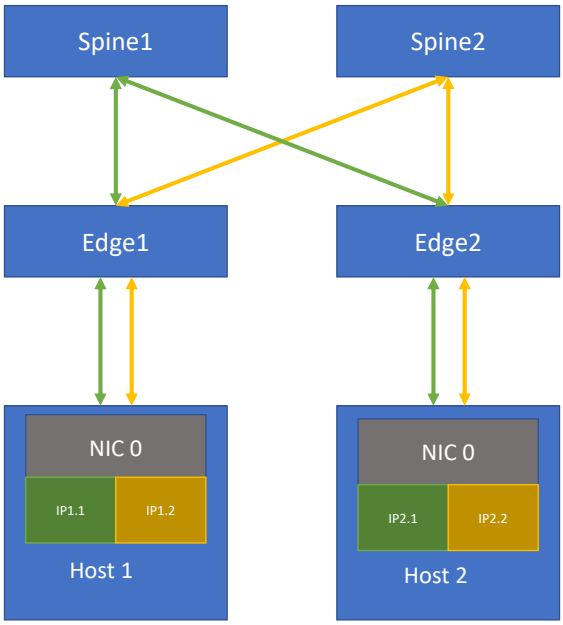
TERMINOLOGY:

- 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*.

The basic scenarios include:

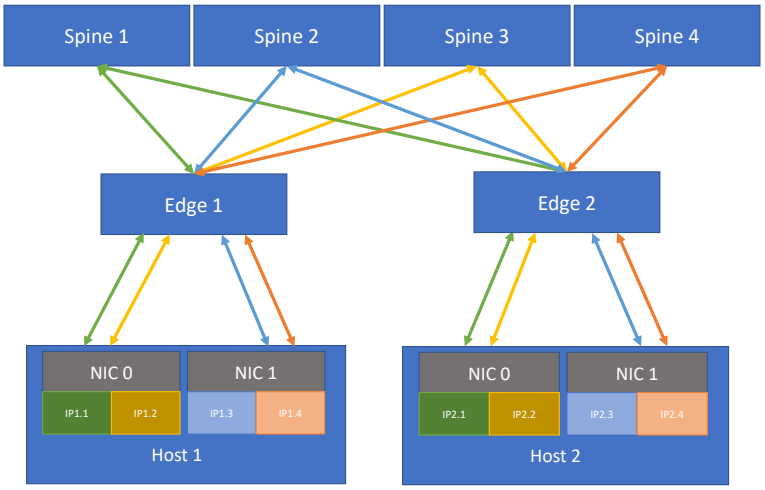
- **One IP address per NIC, per plane:** This is the default configuration. Each NIC is assigned a single IP address and no attempt is made to load balance across IP addresses.
- **Multiple IP addresses per NIC in a single plane:** In this scenario, a fabric with redundant switch-to-switch links is configured such that different IP addresses take different paths to the same NIC. Depending on the application, this may provide improved MPI message rates, latency, and bandwidth to the node.

  For example, in the following figure, each host has a single NIC configured with two IP addresses and the edge switches (Edge1 and Edge2) are configured such that traffic for the IP addresses shown in green (IP1.1 and IP2.1) are routed through the switch labeled Spine1 while traffic for the IP addresses in yellow (IP1.2 and IP2.2) are routed through the switch labeled Spine2.



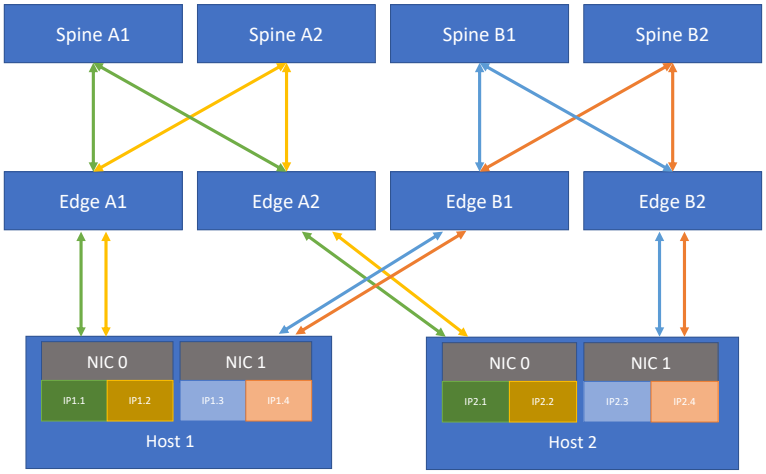
- **Multiple IP addresses, multiple NICs in a single plane:** This scenario, shown in the following figure, consists of hosts with two NICs each, with two IP addresses per NIC. As in the previous example, the edge switches are configured so that traffic will be routed to a specific switch (Spine 1 or Spine 3 for NIC 0 or Spine 2 or Spine 4 for NIC 1) in order to distribute traffic more evenly across the switch-to-switch links.

  Depending on the fabric and the application, this configuration may provide improved MPI message rate, latency, and bandwidth to the node. Depending on the PSM3 configuration, NIC 0 may or may not attempt to communicate with NIC 1 in each node.



- **Multiple IP addresses per NIC in multiple planes:** This scenario, shown in the following figure, consists of hosts with two NICs each, with two IP addresses per NIC. As in the previous example, the edge switches are configured so that traffic will be routed to a specific switch (Spine A1 or Spine A2 for NIC 0 or Spine B1 or Spine B2 for NIC 1) in order to distribute traffic more evenly across the switch-to-switch links.

  Depending on the fabric and the application, 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. Therefore, 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.



8.10.2 PSM3 Multi-IP Usage

The system administrator sets up a multi-IP system by first configuring multiple IP addresses per Intel® Ethernet Fabric NIC per node. The administrator then configures the edge switches of the fabric so that different groups of IP addresses take different routes through the fabric.

By default, Intel PSM3 will select only the first IP address on each NIC. At runtime, however, MPI application users can specify that their application use multiple IP addresses per NIC to improve performance. In addition, if the fabric is configured as a true multi-rail environment as well as a multi-IP environment, PSM3 can be configured to distribute traffic across all the IP addresses across all rails.

It is important that each rank be able to communicate with every other rank in the application. Depending on the design of the fabric, this may depend on how the switches are configured. On a multi-plane fabric, in order to use a single NIC per process, Intel recommends that one or more of the mechanisms in NIC and Address Filtering be used to limit the NICs used to a single plane. As needed, PSM3_NIC_SELECTION_ALG may then be used to control the algorithm that selects 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.

When using multi-IP, PSM3 does not require that each IP address on a NIC be on a separate Ethernet IP subnet. However, configuring switches to distribute traffic across different routes per IP address may require unique IP subnets per IP address. When using a multi-plane fabric, the IP addresses of NICs on different planes should not share a subnet because PSM3 may assume that NICs on the same Ethernet IP subnet can communicate. For more information, refer to the *Intel® Ethernet Fabric Performance Tuning Guide*, MPI Performance. In cases where different subnets can actually communicate, PSM3_ALLOW_ROUTERS can be used to tell PSM3 that communications across IP subnets are possible.

8.10.3 PSM3 Multi-IP Environment Variables

NOTE

The mechanisms outlined in NIC and Address Filtering may be used to limit which NICs and IP addresses PSM3 will consider using.

The following environment variables can be set:

- `PSM3_ADDR_PER_NIC=n`, where `n` can be a value between 1 and 32.

  `PSM3_ADDR_PER_NIC` controls the multi-IP capability. The default value is 1, which disables multi-IP. Note that while the maximum value is 32, PSM3 currently supports a total of 32 IP addresses across all NICs being used. This means that if a node has four NICs, PSM3 will not support more than eight IP addresses per NIC.

- `PSM3_MULTIRAIL=n`, where `n` can be a value between -1 and 4.

  `PSM3_MULTIRAIL` controls how PSM3 selects, load balances, and stripes messages across NICs on the system. See PSM3 Multi-Rail Support for information on configuring Multi-Rail.

- `PSM3_MULTIRAIL_MAP=rail,rail,rail,...`

  Specifies the NIC and address(es) to use for each rail. Multiple `rail` are separated by a comma. Each `rail` may be specified as an RDMA device name or device unit number (starting at 0) and optionally an address index.

  `PSM3_MULTIRAIL_MAP` also allows specification of unique sets of rails per process on a given node, see PSM3_MULTIRAIL_MAP for more information.

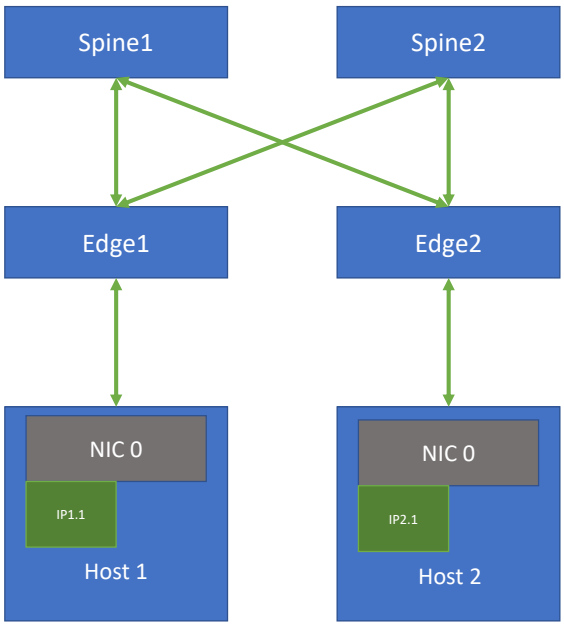
- `PSM3_ALLOW_ROUTERS=n`, where `n` can be 0 or 1. Normally, PSM3 assumes any local or remote NICs with distinct Ethernet IP subnets cannot communicate. When this value is 1, PSM3 assumes routers are present such that all Ethernet IP subnets can communicate with each other. Depending on how the fabric switches are configured, enabling `PSM3_ALLOW_ROUTERS` may be required to be 1 when using multi-IP.

8.10.4 PSM3 Multi-IP Configuration Examples

This section contains examples of multi-IP and multi-rail/multi-IP in single and multiple planes.

Single plane, single rail, single IP

This is the simplest configuration. Each node has a single NIC, and each NIC will use a single IP address (`PSM3_ADDR_PER_NIC` is not set or is set to 1).

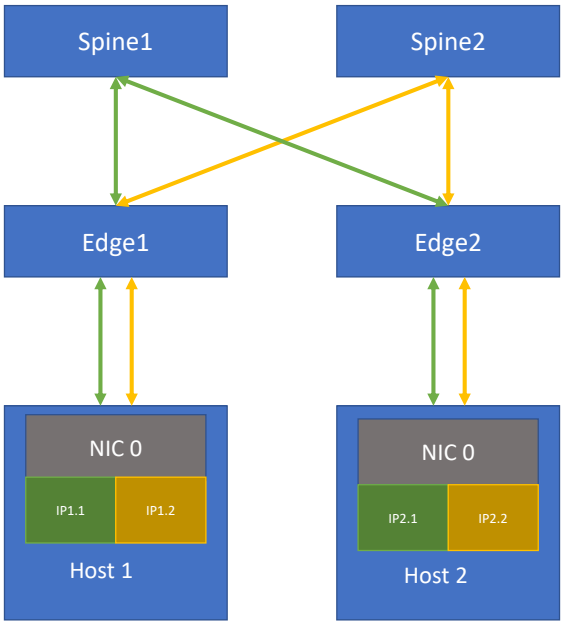


Example Environment Variables

- `PSM3_ALLOW_ROUTERS` will need to be set to 1 if the IP addresses assigned to the NICs are in different subnets.

Single plane, single rail, multiple IP addresses per NIC

Each node has a single NIC, and each NIC will use more than one IP address (`PSM3_ADDR_PER_NIC > 1`).

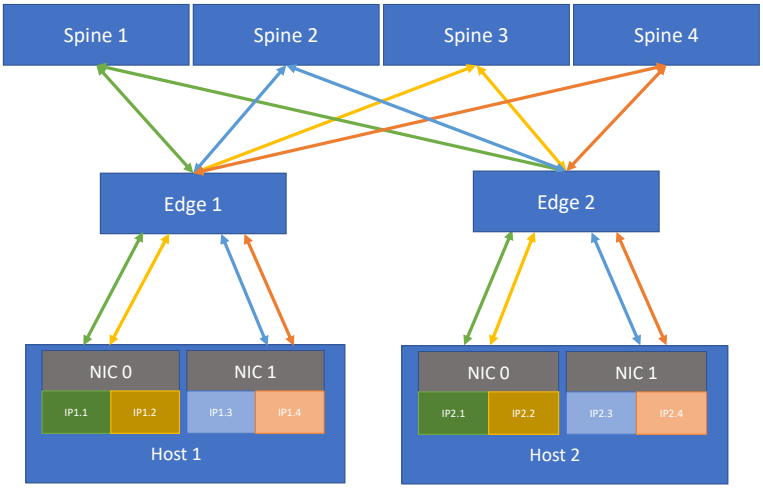


Example Environment Variables

- `PSM3_MULTIRAIL=0` or is not set. PSM3 will pick a single NIC and IP address per process. The PSM3 provider will present a unique OFI fabric interface for each NIC IP address. The first interface will be an `autoselect_one` interface permitting PSM3 to automatically select among the NICs and IP addresses, with different choices per process. Middleware above PSM3 may alternatively use any of the other presented fabric interfaces to open a specific NIC IP address. Such middleware may also choose to open more than one specific NIC and IP address and perform load balancing in the middleware. Depending on application traffic patterns, all IP addresses on all NICs may communicate with each other. `PSM3_ALLOW_ROUTERS` may need to be set to 1 if any of the IP addresses assigned to the NICs are in different subnets.
- `PSM3_MULTIRAIL=1`, `2`, `3`, or `4`. Each process will use all IP addresses on the NIC. Communications will only occur between IP.1 on each NIC and between IP.2 on each NIC. `PSM3_ALLOW_ROUTERS` will need to be set to 1 if these communicating IP addresses are in different subnets.
- `PSM3_MULTIRAIL=1` or `2` and `PSM3_MULTIRAIL_MAP` is specified. Each process will use the specified IP addresses on the NIC. Communications will only occur between the first selected IP on each NIC, between the second selected IP on each NIC, and so on. `PSM3_ALLOW_ROUTERS` will need to be set to 1 if these communicating IP addresses are in different subnets.

Single plane, multi-rail, multi-IP

Each node has two or more NICs, and each NIC will use more than one IP address (`PSM3_ADDR_PER_NIC > 1`), but all NICs are connected to a single fabric plane.

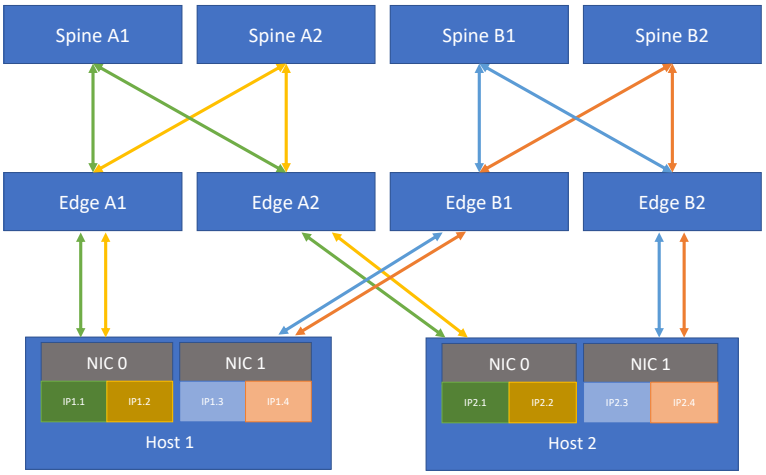


Example Environment Variables

- `PSM3_MULTIRAIL=0` or is not set. PSM3 will pick a single NIC and IP address per process. The PSM3 provider will present a unique OFI fabric interface for each NIC IP address. The first interface will be an `autoselect_one` interface permitting PSM3 to automatically select among the NICs and IP addresses, with different choices per process. Middleware above PSM3 may alternatively use any of the other presented fabric interfaces to open a specific NIC IP address. Such middleware may also choose to open more than one specific NIC and IP address and perform load balancing in the middleware. Depending on application traffic patterns, all IP addresses on all NICs may communicate with each other. `PSM3_ALLOW_ROUTERS` may need to be set to 1 if any of the IP addresses assigned to the NICs are in different subnets.
- `PSM3_MULTIRAIL=1`. Each process will use all IP addresses on all NICs. Communications will only occur between IP.1 on NIC 0, between IP.2 on NIC 0, between IP.1 on NIC 1, and between IP.2 on NIC 1. `PSM3_ALLOW_ROUTERS` will need to be set to 1 if these communicating IP addresses are in different subnets.
- `PSM3_MULTIRAIL=2`, `3`, or `4`. Each process will select one or more close NICs. Each process will use all IP addresses on the selected NICs. Communications will only occur between IP.1 on each NIC and between IP.2 on each NIC. `PSM3_ALLOW_ROUTERS` will need to be set to 1 if these communicating IP addresses are in different subnets.
- `PSM3_MULTIRAIL=1` or `2` and `PSM3_MULTIRAIL_MAP` is specified. Each process will use the specified IP addresses on the selected NICs. Communications will only occur between the first rail (NIC and IP address(es)) on each server, between the second rail (NIC and IP address(es)) on each server, and so on. `PSM3_ALLOW_ROUTERS` will need to be set to 1 if these communicating IP addresses are in different subnets.

Multi-plane, multi-rail, multi-IP

Each node has two or more NICs, and each NIC will use more than one IP address (`PSM3_ADDR_PER_NIC > 1`). Each NIC is connected to one of multiple fabric planes. The IP addresses assigned to NICs in each plane are in different subnets.



Example Environment Variables

- `PSM3_MULTIRAIL=0` or is not set. PSM3 may not work because there are multiple planes. NIC 1 on first host has no connection to NIC 2 on second host.
- `PSM3_MULTIRAIL=1`. Each process will use all IP addresses on all NICs. Communications will only occur between IP.1 on NIC 0, between IP.2 on NIC 0, between IP.1 on NIC 1, and between IP.2 on NIC 1. `PSM3_ALLOW_ROUTERS` will need to be set to 1 if these communicating IP addresses are in different subnets.
- `PSM3_MULTIRAIL=2`, `3`, or `4`. 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` or `2` and `PSM3_MULTIRAIL_MAP` is specified. Each process will use the specified IP addresses on the selected NICs. Communications will only occur between the first rail (NIC and IP address(es)) on each server, between the second rail (NIC and IP address(es)) on each server, and so on. `PSM3_ALLOW_ROUTERS` will need to be set to 1 if these communicating IP addresses are in different subnets.

8.11 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 (which is done at the receiver) involves matching a sent message's send tag with the tag and tag selector attached to every pre-posted 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 that are buffered by the PSM3 provider until a receive buffer that matches the unexpected message is provided. With both 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 therefore 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 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 acknowledgments) and makes forward progress.

8.12 PSM3 Verbs RDMA Modes and Rendezvous Module

PSM3's `verbs` HAL (see PSM3 Architecture and Hardware Abstraction Layer) 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 PSM3 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_GPU_THRESH_RNDV for Intel GPU and NVIDIA CUDA GPU 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 the window size selected by PSM3_RNDV_NIC_WINDOW or PSM3_GPU_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 queue pairs (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 an `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 reliable connected (RC) QPs must be created for each connection.
- Each RC QP must have Work Requests (Work Queue Entries, 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`) may be 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 Intel GPU drivers to manage pinned GPU memory for use in Direct GPU Copy, Direct GPU Send DMA, and Direct GPU RDMA.
- 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 Unreliable Datagram (UD) QPs. This offers low latency and the best scalability. However, single process bandwidth for large messages may be lower bandwidth 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. A user space MR cache or 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

thousands. 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 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 1, the rendezvous module MR cache is always used. For mode 2 and 3, use of a user space MR cache or the rendezvous module cache is controlled by PSM3_MR_CACHE_MODE. The user space or rendezvous module MR caches 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 (the rendezvous module uses the MMU notifier mechanism, and the user MR cache uses the userfaultfd mechanism) so that any buffers that 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, which may be overridden by PSM3_RV_MR_CACHE_SIZE. The size of the user space MR cache per process is controlled via PSM3_MR_CACHE_SIZE_MB. Values for `mr_cache_size`, PSM3_RV_MR_CACHE_SIZE, and PSM3_MR_CACHE_SIZE_MB 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. When a user space MR cache is used, this table is implicitly built into the cache. This table is sized via PSM3_MR_CACHE_SIZE.

In all modes, 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. In mode 2 and 3, PSM3 reestablishes disrupted user space RC QPs and automatically retransmits lost eager and RDMA messages. This allows PSM3 to ride through disruptions that 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_RECONNECT_TIMEOUT for more information.

NOTE

For non-GPU applications, RDMA mode 0 does not use the rendezvous module. However, when Direct GPU access or GPUDirect is enabled, all RDMA modes require the appropriate GPU-enabled rendezvous module. See PSM3 Support for Direct Intel GPU Access and PSM3 Support for NVIDIA GPUDirect

NOTE

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

NOTE

PSM3 build options control which HALs are included in the PSM3 binary as well as which data movement protocols are available within each included HAL. See Building the PSM3 RPM.

8.13 PSM3 Sockets Modes

PSM3's `sockets` HAL (see PSM3 Architecture and Hardware Abstraction Layer) supports two Sockets Modes. The modes allow trade-offs between performance and memory footprint. Within the PSM3 sockets protocol, there are three basic types of communications:

- Control Messages: Messages that facilitate PSM3 internal operations such as connection establishment, credit exchange, and acknowledgments. They carry no application data.
- Eager Messages: The majority of application messages are classified as eager.
- Rendezvous Messages: Some application messages that request a synchronous end-to-end acknowledgment and larger messages for GPU jobs.

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)/Clea 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.

The PSM3_SOCKETS environment variable selects the sockets mode to be used for a given job.

Comparing the PSM3_SOCKETS options:

- `0` – Use TCP/IP (with a few special situations during disconnect where UDP/IP may be used). This offers low latency and takes advantage of TCP/IPs reliability, flow control, and congestion handling protocols. Many larger scale switching environments are well tuned for TCP/IP's characteristics so this mode may perform best in many environments.
- `1` – Use UDP/IP. This offers low latency and uses PSM3's reliability and flow control protocols. It uses far fewer sockets per process than TCP/IP and thus may scale better when there are high numbers of PSM3 processes per node. However, to get good performance out of this mode, DCB/PFC must be enabled in the network (similar to how RDMA protocols require DCB/PFC).

When using mode 1, PSM3 is extremely resilient to packet loss and fabric disruptions.

NOTE

By default, the sockets HAL only uses rendezvous for synchronous application messages and larger messages for GPU oneAPI or CUDA jobs. While rendezvous may be enabled for other messages via PSM3_MQ_RNDV_NIC_THRESH, due to sockets' lack of RDMA, it offers no benefits and is generally discouraged.

NOTE

PSM3 build options control which HALs are included in the PSM3 binary as well as which data movement protocols are available within each included HAL. See Building the PSM3 RPM.

8.14 HAL and Protocol-Specific Configuration Controls

Many of the PSM3 configuration controls (PSM3 Environment Variables) are applicable to all modes of PSM3. However, a number of configuration options are only applicable to a subset of Hardware Abstraction Layers (HALs) and or protocols within a HAL (see PSM3 Architecture and Hardware Abstraction Layer). When specified, if an option is not applicable to the HAL and/or protocol selected, it will be silently ignored.

The following configuration options are only applicable to the verbs HAL (see PSM3 Verbs RDMA Modes and Rendezvous Module):

- All values for `PSM3_RDMA`: PSM3_NUM_RECV_CQES, PSM3_NUM_RECV_WQES, PSM3_NUM_SEND_WQES, PSM3_SEND_REAP_THRESH
- All values for `PSM3_RDMA` which use RDMA (e.g. `PSM3_RDMA` 1, 2, or 3): PSM3_MR_CACHE_SIZE, PSM3_NUM_SEND_RDMA, PSM3_QP_TIMEOUT, PSM3_RDMA_SENDSESSIONS_MAX, PSM3_RECONNECT_TIMEOUT
- `PSM3_RDMA=1` (UD/RV) only: PSM3_IB_SERVICE_ID, PSM3_RV_HEARTBEAT_INTERVAL, PSM3_RV_MR_CACHE_SIZE, PSM3_RV_Q_DEPTH, PSM3_RV_QP_PER_CONN
- `PSM3_RDMA=2` (UD/RC) only: PSM3_MR_CACHE_MODE, PSM3_QP_RETRY
- `PSM3_RDMA=3` (RC) only: PSM3_MR_CACHE_MODE, PSM3_QP_RETRY

The following configuration options are only applicable to the sockets HAL (see PSM3 Sockets Modes):

- All values for `PSM3_SOCKETS`: PSM3_UDP_RCVBUF, PSM3_UDP_SNDBUF.
- `PSM3_SOCKETS=0` (TCP/IP) only: PSM3_TCP_PORT_RANGE, PSM3_TCP_RCVBUF, PSM3_TCP_SKIPPOLL_COUNT, PSM3_TCP_SNDBUF, PSM3_TCP_SNDPACING_THRESH.
- `PSM3_SOCKETS=1` (UDP/IP) only: PSM3_UDP_GSO.

The following configuration options are only applicable when PSM3_GPUDIRECT is enabled: PSM3_GPU_THRESH_RNDV, PSM3_RV_GPU_CACHE_SIZE. Also see: PSM3 Support for Intel GPUs, PSM3_ONEAPI_ZE, PSM3 Support for NVIDIA GPUs, and PSM3_CUDA.

NOTE

PSM3 build options control which HALs are included in the PSM3 binary as well as which data movement protocols are available within each included HAL. See Building the PSM3 RPM.

8.15 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 Intel GPU drivers to manage pinned GPU memory for use in Direct GPU Copy, Direct GPU Send DMA, and Direct GPU RDMA.
- 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 PSM3_RDMA modes 1, 2, and 3, the rendezvous module may also provide MR caching. For mode 1, the rendezvous module MR cache is always used. For mode 2 and 3, use of the rendezvous module cache is controlled by PSM3_MR_CACHE_MODE. 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 that the application frees will be automatically purged from the cache.

To reduce latency for registration of kernel MRs in PSM3_RDMA mode 1, a Fast-Registration Pool may be used by the rendezvous module. This pool maintains pre-created MRs that can be quickly registered against a given application buffer address. The Fast-Registration Pool is enabled by default and controlled via `fr_batch_size` and `fr_pool_wm_lo` (discussed below).

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_ONEAPI_ZE or 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 Direct GPU RDMA or 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 Direct GPU RDMA or 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, Direct GPU Send DMA, GPUDirect Send DMA, or PSM3_RDMA mode 2 and 3 when PSM3_MR_CACHE_MODE is 1. Default is 0.
- `fr_batch_size` - Fast-registration pool batch allocation size for kernel MRs, which are used by kernel RC QPs. When it is set to 0, the pool will not be allocated and used. Minimum is 64 and default is 256.
- `fr_pool_wm_lo` - Fast-Registration Pool lower watermark. When the number of free MRs in the pool is equal to or lower than this watermark, a worker will be queued to allocate `fr_batch_size` more kernel MRs. If it is set to 0 or greater than `fr_batch_size`, it will be set to `fr_batch_size/4`. The minimum is 1 and the default is 64.

When PSM3_MR_CACHE_MODE is 1, rendezvous module caching of user mode MRs can be used for:

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

NOTE

For more information on RDMA see PSM3 Verbs RDMA Modes and Rendezvous Module. For more information on GPU Support and Direct GPU Access, see PSM3 and GPU Support.

8.15.1 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.

8.16 PSM3 and GPU Support

PSM3 supports optimized GPU buffer transfers via Direct GPU Copy, Direct GPU Send DMA, and Direct GPU RDMA. This support is integrated in conjunction with a GPU-enabled rendezvous kernel module (rv). To use this feature, both PSM3 and the rendezvous kernel module must be GPU-enabled and present in the system. When enabled, PSM3 helps accelerate transfers of GPU memory buffers. You must enable this feature at runtime.

PSM3's GPU support includes the following capabilities:

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

NOTES

- Ensure the systems are installed and configured properly by following the *Intel® Ethernet Fabric Suite Software Installation Guide*.
- Since there are additional checks in software critical paths (and such checks may have a minor performance impact), Intel recommends that you only enable this feature if you need GPU support.
- There are no performance penalties for using a GPU-enabled rendezvous module (rv) during a PSM3 job that has not enabled PSM3 GPU support.

By default, GPU support is disabled. To enable it at runtime, refer to `PSM3_ONEAPI_ZE`, `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 GPU-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.

Sizing the GPU Registration Cache

When using any of the PSM3 Direct GPU access 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 that are frequently used for PSM3 IOs involving any of the Direct GPU access features. The size of this per process cache is controlled 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.

To determine the GPU registration cache's behaviors during a given run, use PSM3_PRINT_STATS and review its output files. If a non-zero value is observed for the `rv_gpu_failed_pin` counter, this indicates the GPU 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` statistic will indicate the maximum memory (in megabytes) that 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 one process (or rank) is assigned to a given GPU. If more than one process (or rank) will be run per GPU, the per process GPU registration cache size must also be correspondingly reduced (for example, if four 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 its runtime support for GPUs via the `FI_HMEM` OFI (libfabric) capability flag. This flag is only reported when a GPU-enabled version of PSM3 is used in conjunction with runtime enablement of GPU features via `PSM3_ONEAPI_ZE`, `PSM3_CUDA` and/or `PSM3_GPUDIRECT`. When `FI_HMEM` is reported, PSM3 checks the location of all IO buffers. 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 GPU 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. Therefore, some high-level middleware GPU support may be necessary if implementing support for collectives.

8.16.1 PSM3 and Intel GPU Support

PSM3 supports GPU buffer transfers through oneAPI Level Zero via Direct GPU Copy, Direct GPU Send DMA, and Direct GPU RDMA. This support is integrated in conjunction with a oneAPI Level Zero (oneapieze) enabled rendezvous kernel module (rv). To use this feature, both PSM3 and the rendezvous kernel module must be enabled for oneAPI Level Zero and present in the system. When enabled, PSM3 helps accelerate transfers of GPU memory buffers by using the Linux kernel's dma_buf interface.

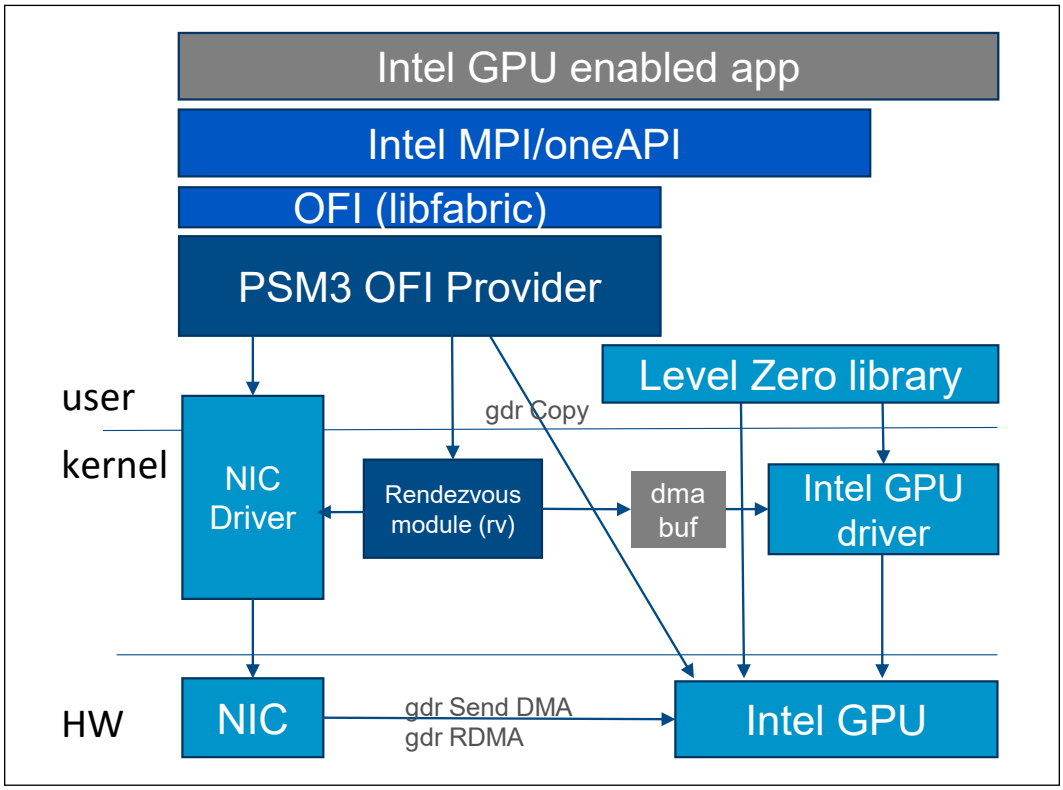
By default, PSM3 GPU support is disabled. To enable it at runtime, refer to `PSM3_ONEAPI_ZE` and `PSM3_GPUDIRECT`. These environment variables must be set before the application is launched. Additionally, if an MPI or middleware application is used, then both the MPI and middleware typically need to be GPU-enabled.

When running Intel GPU applications, it may be necessary to increase the maximum allowed open files via the Linux `ulimit -n limit` command or by editing `/etc/security/limits.conf`, and increasing the `nofile` limit. Depending on the applications and hardware configuration, values as large as 524288 may be required.

When running an Intel GPU application with Intel MPI, the following additional environment variable settings are recommended:

```
EnableImplicitScaling=0
NEOReadDebugKeys=1
I_MPI_OFFLOAD=1
I_MPI_OFFLOAD_PIPELINE=0
I_MPI_OFFLOAD_RDMA=1
I_MPI_OFFLOAD_RDMA_THRESHOLD=0
```

Figure 6. PSM3 Intel GPU Architecture



The PSM3 OFI (libfabric) provider supports Intel GPUs for both user space transfers via the oneAPI Level Zero Library as well as for optimized Direct GPU Access with the assistance of the PSM3 Rendezvous Kernel Module (rv).

As shown in the previous figure, the rendezvous module makes use of the standard Linux kernel dma_buf APIs to obtain direct access to GPU memory and create verbs Memory Regions (MRs) and/or mmap GPU memory into user space. Once access has been established, Direct GPU Copy (gdr Copy) operations can be performed by PSM3 via the mmap'ed GPU memory. Additionally, Direct GPU Send DMA (gdr Send DMA) and Direct GPU RDMA (gdr RDMA) operations can be performed by PSM3, and the NIC will make peer-to-peer requests across PCIe to directly read and write GPU memory.

NOTE

There are no performance penalties for using a oneAPI Level Zero-enabled rendezvous module (rv) during a PSM3 job that has not enabled PSM3 GPU support.

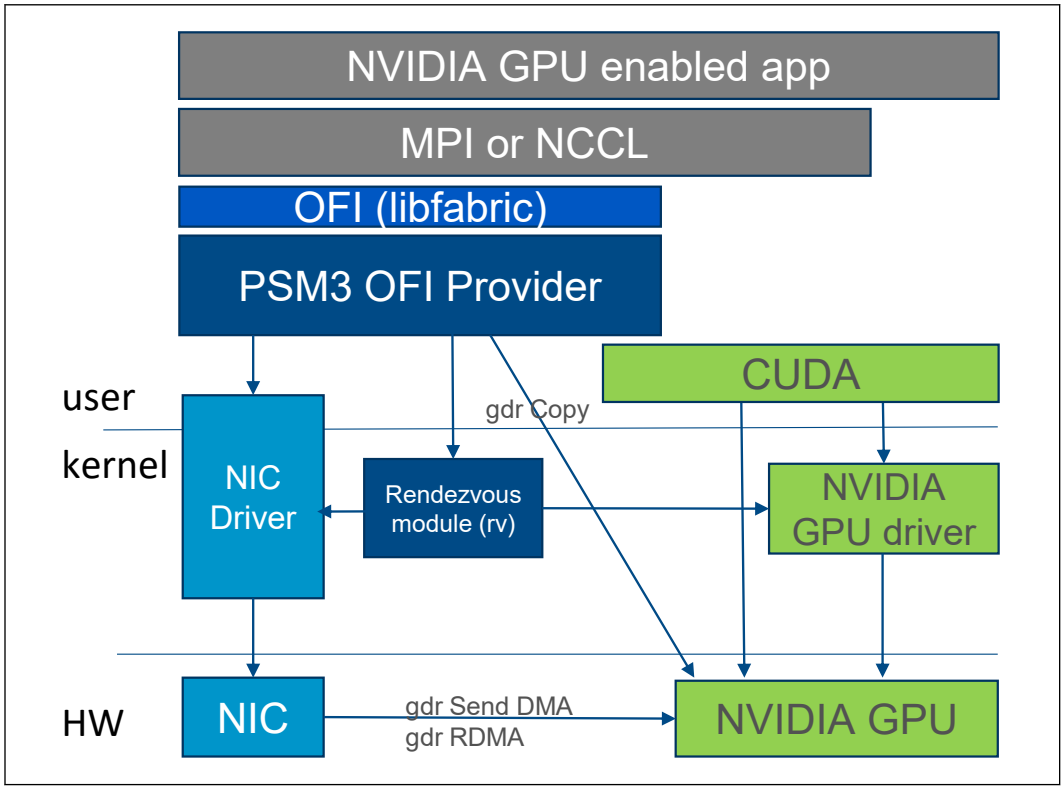
The upper bound of memory pinning for Intel GPUs can be found by looking at the PCIe `Region 2` BAR size as reported by `lspci -vvv` for the GPU device.

8.16.2 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.

By default, PSM3 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.

Figure 7. PSM3 NVIDIA GPU Architecture



The PSM3 OFI (libfabric) provider supports NVIDIA GPUs for both user space transfers via the CUDA Library as well as for optimized GPU Direct Access with the assistance of the PSM3 Rendezvous Kernel Module (rv).

As shown in the previous figure, the rendezvous module directly calls NVIDIA driver APIs to obtain direct access to GPU memory and create verbs Memory Regions (MRs) and/or mmap GPU memory into user space. Once access has been established, GPUDirect Copy (gdr Copy) operations can be performed by PSM3 via the mmap'ed

GPU memory. Additionally, GPUDirect Send DMA (gdr Send DMA) and GPUDirect RDMA (gdr RDMA) operations can be performed by PSM3. In this case, the NIC will make peer-to-peer requests across PCIe to directly read and write GPU memory.

NOTE

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

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, Intel recommends that you initialize the CUDA runtime before the OFI or `MPI_Init()` call.

The upper bound of memory pinning for NVIDIA GPUs 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.

8.17 PSM3 Data Streaming Accelerator Support

The Data Streaming Accelerator (DSA) is a high-performance data copy and transformation accelerator integrated into Intel® Xeon® Processors starting with the 4th Generation Intel® Xeon® Scalable Processors. PSM3 may be enabled to take advantage of DSA to optimize intra-node communications which use PSM3's `shm` device (see PSM3 Architecture and Hardware Abstraction Layer).

In order for PSM3 to take advantage of DSA, PSM3 must be told a set of DSA work queues to use via PSM3_DSA_WQS. These work queues must be pre-created by the sysadmin and associated with specific DSA hardware devices within the CPU. Depending on the CPU model used, the number of available DSA resources may vary.

The DSA hardware resources consist of one or more DSA hardware devices per CPU socket, each of which may have one or more DSA engines. The sysadmin may create DSA work queues. Each work queue is associated with a single DSA hardware device and may use a group of one or more DSA engines. When a DSA work queue is associated with more than one DSA engine, the DSA hardware will automatically load balance DSA operations in the work queue across all the assigned engines. Each engine can perform a single data copy or transformation at a time.

Each DSA work queue may be dedicated (allowing only one process in the system to use it at a time) or shared (allowing more than one process to use it at a time). The work queues assigned to PSM3 must be created by the sysadmin and may be dedicated or shared work queues. Each DSA engine within the work queue can only do one operation at a time, so when using shared work queues, some processes may incur delays waiting for work from other processes to finish.

Via PSM3_DSA_WQS, a user can assign one or more DSA work queues per PSM3 process. If more than one DSA work queue is assigned to a given PSM3 process, then separate threads in the process will be load balanced by PSM3 across the assigned

work queues. For applications using multiple threads via PSM3 Multi-Endpoint Functionality, the assignment of multiple DSA work queues per process may provide performance advantages.

PSM3_DSA_MULTI determines how the work queues specified in PSM3_DSA_WQS are assigned to processes.

NOTE

A sample dsa_setup script is included in Intel® Ethernet Fabric Suite that can assist a sysadmin in creating and removing DSA work queues for use by PSM3.

NOTE

For more information on DSA and how to enable it within the CPU, BIOS and Linux kernel, see https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, https://www.intel.com/content/www/us/en/products/docs/ondemand/overview.html, and https://cdrdv2.intel.com/v1/dl/getContent/759709.

8.18 PSM3 Performance Statistics

PSM3 tracks over 400 internal performance statistics per process in a job. These statistics can be reported periodically during a job, or once at the end of the job by setting PSM3_PRINT_STATS. The output files generated by PSM3_PRINT_STATS are designed for easy parsing by scripts as well as easy importing into tools such as spreadsheets. The groups of statistics to report can be controlled via PSM3_PRINT_STATSMASK. Help text for each statistic, including explanations of various groups of statistics and the PSM3 protocols they track, can be generated via PSM3_PRINT_STATS_HELP. PSM3_PRINT_STATS_PREFIX can control the location where the statistics and statistics help files are placed as well as specify a filename prefix to be used for all statistics and statistics help files generated during the run.

NOTE

PSM3 retains and reports statistics per endpoint. If an application or middleware is using multiple endpoints, each reporting interval will report statistics separately per endpoint. Such statistics must be analyzed and tabulated to get a complete understanding of the application's behaviors.

Many MPI applications have only one application thread per process and only open one PSM3 endpoint per process. However, some applications may take advantage of PSM3 multi-endpoint in which case the application may open more than one PSM3 endpoint and may have multiple application threads.

Multiple PSM3 endpoints may also be opened by PSM3 itself, the middleware, or the application when using multi-rail. Use of multi-endpoint and middleware load balancing across multiple NICs is typical in applications using oneCCL, where each oneCCL worker may open its own PSM3 endpoint.

Refer to: Running oneCCL on Network Interface Cards, PSM3 Multi-Endpoint Functionality, PSM3 Multi-Rail Support, and PSM3 Multi-IP Support

NOTE

Separate statistics files are generated per process in a given job. To get a full understanding of application performance and behaviors, it will be necessary to review some or all of the generated files. In many cases, an application may have different communications usage patterns for different processes in the job. For some applications, the communications of rank 0 may significantly differ from other processes in the job.

When PSM3_PRINT_STATSMASK bit 0x40 is enabled, the statistics file will include a one-time output of job launch information including cmdline, full environment, all PSM3_ and FI_PSM3_ settings as they are parsed, and PSM3_IDENTIFY information as it is generated. For example (in this example .... is used to represent content omitted in this example for brevity):

```
cmdline: imb/IMB-MPI1 pingpong
environ:
USER=root
PWD=/root/mpi_apps-impi
HOME=/root
....
--------------------------------------------------------------------------------
FI_PSM3_UUID=61f00000-d047-c29f-3efa-050042e2e220
FI_PSM3_DELAY=0
FI_PSM3_INJECT_SIZE=32768
FI_PSM3_LOCK_LEVEL=0
PSM3_PRINT_STATS=1
PSM3_PRINT_STATSMASK=0xfffff
....
myhost21:rank0 PSM3_IDENTIFY PSM3 v3.0 built for IEFS X.Y
myhost21:rank0 PSM3_IDENTIFY location /usr/lib64/libfabric/libpsm3-fi.so
....
```

A line of dashes is used to separate the output of the full environment, from the values for PSM3_ and FI_PSM3_ settings and PSM3_IDENTIFY information. The PSM3_IDENTIFY information may be intermingled with lines showing PSM3_ and FI_PSM3_ settings.

The files generated by PSM3 will have one or more sections showing a time stamp followed by one or more groups of statistics. For example (in this example .... is used to represent content omitted in this example for brevity):

```
....
Time Delta 1 seconds Tue Dec 27 15:47:07 2022
....
Time Delta 2 seconds Tue Dec 27 15:47:08 2022
 MPI_Statistics_Summary id  0xc0a8018e00000174:17:0 tid 5812
    COMM_WORLD_Rank 0 (0)
    Total_count_sent 133876 (52332)
    Total_bytes_sent 51853452 (50476672)
    Overall_avg_msg_size_sent 387 (371)
    Eager_count_sent 133876 (52332)
    Eager_bytes_sent 51853452 (50476672)
....
 PSM_low-level_protocol_stats id 0xc0a8018e00000174:17:0 irdma1 tid 5812
    ud_sbuf_free 4028 (4028)
....
 RcvThread_statistics
    intrthread_schedule_count 9 (9)
....
```

The timestamp shows both an absolute date and time in the local time zone as well as a time delta relative to when PSM3 was initialized (typically at job start). For each group of statistics selected by PSM3_PRINT_STATSMASK, the name of the group is shown (`MPI_Statistics_Summary`, `PSM_low-level_protocol_stats` and `RcvThread_statistics` in this example), followed by the statistics in the group (`COMM_WORLD_Rank`, `Total_count_sent`, etc). For each statistic, a running total is shown followed by the delta since the last reported timestamp in the process. So in this example, `Total_count_sent` indicates a total of `133876` messages have been sent since job start, of which `52332` have been sent between Time Delta 1 and 2 seconds.

Statistics groups that report statistics only for a single PSM3 endpoint will show the endpoint ID and the thread ID that created the endpoint (which is typically the parent thread, not the actual thread using the endpoint). In this example, we see that `MPI_Statistics_Summary` and `PSM_low-level_protocol_stats` are applicable to endpoint ID `0xc0a8018e00000174:17:0`, which was created by thread ID `5812`. We can also see that `RcvThread_statistics` is reporting statistics applicable to the entire process, due to its lack of a endpoint ID. Statistics for an endpoint associated with a specific NIC device will also show the NIC name, `irdma1` in this example. If the given process has more than one endpoint, the endpoint-specific statistics for each endpoint would be shown within each time delta.

NOTE

When PSM3_PRINT_STATS is configured to only output statistics at the end of the job, statistics will be shown only when endpoints are closed and at the final close of PSM3 by the middleware. This may result in a few timestamp sections in the statistics file, but is necessary in case a middleware were to repeatedly open and close PSM3 endpoints.

Following are just a few of the more common insights into application performance that PSM3 performance statistics can provide.

Tuning PSM3 for optimal performance with a given application or middleware often begins by examining the high-level message passing statistics (`MPI_Statistics_Summary`). This group of statistics can provide insight into how many messages were sent and received, how many bytes were transferred, which basic mechanisms were used (Eager or Rendezvous), whether the messages used intra-node (shm) protocols or inter-node (nic), whether receive buffers were posted prior to message receipt (expected) or not (unexpected), and whether the buffers were in CPU or GPU memory.

Within the high-level statistics, some things to look for include:

- Does the application transfer a lot of data per second?

  Applications with relatively low amounts of data transferred and relatively few messages per second will typically not be bottlenecked on communications, and performance optimizations may have to focus on other areas such as computational algorithms and libraries.

- What are the average message sizes used by the application?

  Applications with relatively small message sizes will tend to be more latency-sensitive, while those with larger message sizes may be more bandwidth-sensitive. Applications with larger message sizes may also be more susceptible to switching fabric bottlenecks or congestion. Applications that use larger messages

will tend to scale and perform better as well as making more efficient use of the network. Applications with small message sizes may benefit from other middleware or application options, or even application implementation improvements to utilize larger messages.

- Does the application have a high number of messages arriving before the receiver buffer has been posted (`unexpected` messages)?

  If so, PSM3 may be incurring extra overheads to land the incoming message in temporary bounce buffers (`sysbuf`), or rendezvous may be delayed waiting for the receive buffer to be posted. While many well-tuned applications still have some modest degree of `unexpected` messages, it is generally best to design an application to post receive buffers as early as possible to get more efficiency.

- Is the application doing a large number of intra-node communications (shm)?

  If so, tuning of intra-node communications or selection of optimized mechanisms such as the PSM3 Data Streaming Accelerator Support may help improve application performance.

For applications with a high degree of inter-node communications, the `PSM_low_level_protocol_stats` should be reviewed. Some things to look for include:

- Are reliability protocol events occurring?

  These typically indicate packet loss and may imply incorrect configuration of network flow control mechanisms, such as Priority Pause Control (PFC).

- Is a GPU application being run, but PSM3 is not seeing any messages sent or received using GPU buffers?

  If so, ensuring a GPU-enabled middleware and PSM3 binary is being used, and enabling PSM3 GPU support may improve performance. See PSM3 Support for GPUs, PSM3_IDENTIFY, PSM3_ONEAPI_ZE and PSM3_CUDA.

- Is a significant amount of data or messages being send or received using GPU buffers?

  If so, mechanisms like Direct GPU transfers may improve performance. See PSM3 Support for GPUs and PSM3_GPUDIRECT.

- Is a large amount of data being transferred via the rendezvous `Long Data` protocol (`rndv_long`)?

  If so, use of RDMA, Direct GPU transfers, and/or the PSM3 Rendezvous Kernel Module may improve performance. Also see: PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Rendezvous Kernel Module, PSM3 Support for GPUs, and PSM3_GPUDIRECT

- Is a lot of data being transferred via CPU send DMA (`dma_cpu_send`) or rendezvous CPU RDMA (`rndv_rdma_cpu`)?

  If so, the `MR_Cache_Statistics` should also be reviewed.

- Is a large amount of data being transferred via Direct GPU Send DMA (`gdr_send` and `gdr_isend`), rendezvous Direct GPU RDMA (`rndv_rdma_gdr`), or via Direct GPU Copy (`gdrcopy`)?

  If so, the `MR_Cache_Statistics` and `MR_GPU_Cache_Statistics` should also be reviewed.

- Are most messages using synchronous (`_send`) or asynchronous (`_isend`) sends?

Applications and middleware designed or configured to take advantage of asynchronous sends may perform better.

- When using TCP sockets (see PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes), review `partial_write_cnt`.

  If this value is large and growing over time, PSM3 may be sending data faster than the network or receiving node can handle. In this case, performance may be improved by reducing PSM3_TCP_SNDPACING_THRESH or monitoring the network for congestion or bottlenecks and correcting them.

For applications with a high degree of Send DMA or rendezvous RDMA for CPUs or GPUs, the `MR_Cache_Statistics` should be reviewed. Some things to look for include:

- Review the cache `limit` as compared to `inuse` and `max_inuse` for the User Space MR Cache and Kernel RV Cache.

  The cache maximum inuse should rarely if ever reach the limit. If the cache inuse is hitting its limit, performance may be improved by growing the limit to permit more concurrent IOs (see PSM3_MR_CACHE_SIZE).

- Review the User Space MR Cache `limit` as compared to `nelems` and `max_nelems`.

  Similarly, review User Space Cache `rejected` and `full` events. The cache maximum elements should rarely if ever reach the limit and become full or reject new entries. If the cache size is hitting its limit, performance may be improved by growing the limit to permit more concurrent IOs (see PSM3_MR_CACHE_SIZE).

- Unless the User Space MR Cache is enabled (PSM3_MR_CACHE_MODE=2), disregard the User Space MR Cache `hit` and `miss` statistics. Since in other modes the User Space MR Cache only holds MRs that are currently in use, it will be rare for hits to occur. The exception may be bandwidth-oriented micro-benchmarks that concurrently use send or receive buffers for multiple IOs or selected middleware collective algorithms such as `Broadcast`, which may concurrently use send buffers for multiple IOs.

  When the User Space MR Cache is enabled, applications that show a high number of miss events and a low overall hit rate (`hit_%`) may have poor reuse of IO buffers. Such applications may also show a high rate of removal of cache entries due to buffer free (`umrc_remove`). Such applications should be reviewed for configuration options or design for better application reuse of IO buffers. Such reuse is essential to getting good performance out of Send DMA and RDMA. If no application options are available, the application may perform better by disabling CPU RDMA (see PSM3 Verbs RDMA Modes and Rendezvous Module).

- If the User Space MR Cache is enabled, review the `limit_bytes` and `limit_entries` as compared to `registered_bytes`, `max_registered_bytes`, `nelems`, and `max_nelems` for the User Space MR Cache.

  If the cache is hitting its limit and evicting entries (`evict`), performance may be improved by growing the limit (see PSM3_MR_CACHE_SIZE and PSM3_MR_CACHE_SIZE_MB). This may also improve the User Space MR Cache hit rate.

- Review the Kernel RV Cache `rv_hit` and `rv_miss` statistics.

Applications that show a high number of miss events and a low overall hit rate (`rv_hit_%`) may have poor reuse of IO buffers. Such applications may also show a high rate of removal of cache entries due to buffer free (`rv_remove`). Such applications should be reviewed for configuration options or design for better application reuse of IO buffers. Such reuse is essential to getting good performance out of Send DMA and RDMA. If no application options are available, the application may perform better by disabling CPU RDMA and/or GPU Direct RDMA (see PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Support for GPUs and PSM3_GPUDIRECT).

- Review the `rv_limit` as compared to `rv_size` and `rv_max_size` for the Kernel RV Cache.

  If the cache is hitting its limit and evicting entries (`rv_evict`), performance may be improved by growing the limit (see PSM3_RV_MR_CACHE_SIZE). This may also improve the Kernel RV Cache hit rate.

For applications with a high degree of Direct GPU Copy, Direct GPU Send DMA, or rendezvous Direct GPU RDMA, the `MR_GPU_Cache_Statistics` should be reviewed. Some things to look for include:

- Review the Kernel RV GPU Cache `rv_gpu_limit` as compared to `rv_gpu_inuse` and `rv_gpu_max_inuse`.

  The cache maximum inuse should rarely if ever reach the limit. If the cache inuse is hitting its limit, performance may be improved by growing the limit to permit more concurrent IOs (see PSM3_RV_GPU_CACHE_SIZE).

- Review the Kernel RV GPU Cache `rv_gpu_hit` and `rv_gpu_miss` statistics.

  Applications that show a high number of miss events and a low overall hit rate (`rv_gpu_hit_%`) may have poor reuse of IO buffers. Such applications may also show a high rate of removal of cache entries due to buffer free (`rv_gpu_remove`). Such applications should be reviewed for configuration options or design for better application reuse of IO buffers. Such reuse is essential to getting good performance out of Direct GPU access for GPU Copy, Send DMA, and RDMA. If no application options are available, the application may perform better by disabling Direct GPU access (see PSM3 Support for GPUs and PSM3_GPUDIRECT).

- Review the `rv_gpu_limit` as compared to `rv_gpu_size` and `rv_gpu_max_size` for the Kernel RV GPU Cache.

  If the cache is hitting its limit and evicting entries (`rv_gpu_evict`), performance may be improved by growing the limit (see PSM3_RV_GPU_CACHE_SIZE). This may also improve the Kernel RV GPU Cache hit rate.

For applications using RDMA via the Kernel RV module, the `RV_Shared_Conn_RDMA_Statistics` should be reviewed. Some things to look for include:

- Are RV CM Connection Recovery events occurring?

  These typically indicate QP instability due to packet loss and may imply incorrect configuration of network flow control mechanisms (such as Priority Pause Control (PFC)).

8.19 Building the PSM3 RPM

The Intel® Ethernet Fabric Suite host software includes a source RPM for PSM3 as well as pre-built binary RPM versions of PSM3 for Intel GPU environments using oneAPI Level Zero, NVIDIA GPU environments using NVIDIA CUDA, and non-GPU environments. These builds have all the commonly-used PSM3 features enabled, including support for both the verbs and sockets Hardware Abstraction Layers (HAL) and the kernel rendezvous module (see PSM3 Architecture and Hardware Abstraction Layer and PSM3 Rendezvous Kernel Module).

If desired, the PSM3 source RPM can be rebuilt. The following command will rebuild PSM3 with its default build options and install the package:

```
rpmbuild --rebuild libpsm3-fi-<version>.src.rpm
```

NOTE

This section only covers the basics of building the PSM3 RPM. For more information on `rpmbuild` and rpms, see the Linux man pages for rpmbuild(8) and rpm(8).

When building PSM3 with `rpmbuild`, the following `configopts` are available. The following command is an example of specifying a `configopts` setting (in this example, we omit the sockets HAL from this build via `--disable-psm3-sockets`):

```
rpmbuild --rebuild --define "configopts --disable-psm3-sockets" libpsm3-fi-
<version>.src.rpm
```

The majority of `configopts` take one of two forms:

- `--enable-X`: This form can enable or disable feature `X`. It may be specified as `--enable-X` or `--disable-X` or `--enable-X=check`. For example, `--enable-psm3-dsa`, `--disable-psm3-dsa`, or `--enable-psm3-dsa=check`. The `check` variation will enable the given feature only if the necessary build prerequisites (typically headers and/or libraries) are found.
- `--with-X`: This form can enable the build to use another component (typically headers and/or libraries) during the build. It may be specified as `--with-X`, `--without-X`, or `--with-X=PATH`. When specified, `PATH` indicates where `X` can be found on the current system. When `PATH` is not specified, a standard set of locations for component `X` will be checked. If `X` is not found, it will be omitted and disabled in the build.

The following table summarizes the rpmbuild `configopts` specific to PSM3 and typically of interest:

Table 7. PSM3 configopt Options

Option	Default	Description
`--enable-psm3-verbs`	enabled	Includes the verbs Hardware Abstraction Layer in the build. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.
`--enable-psm3-sockets`	enabled	Includes the sockets Hardware Abstraction Layer in the build. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

continued...

Option	Default	Description
`--enable-psm3-udp`	disabled	Enables UDP support in all applicable HALs (only applicable to sockets HAL at this time). When disabled, the sockets HAL will only have support for TCP/IP. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.
`--enable-psm3-rc`	check	Enables user space RC QP support in all applicable HALs (only applicable to verbs HAL at this time). When disabled, the verbs HAL will only have support for user space UD QPs and optionally the rendezvous module (see `--with-psm3-rv` below, which can enable support for kernel RC QPs). Only available if `--enable-psm3-verbs`. The `check` option will set this to match the `--enable-psm3-verbs` option above. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.
`--with-psm3-rv`	enabled	Enables kernel rendezvous module support in all applicable HALs (applicable to sockets HAL when `--with-cuda` or `--with-oneapi-ze` are also enabled, always applicable to verbs HAL). See PSM3 Rendezvous Kernel Module, PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Sockets Modes, PSM3 and Intel GPU Support, and PSM3 and NVIDIA CUDA Support. *Note:* This requires that `iefs-kernel-updates-devel-<version>.rpm` is installed on the current system so that the kernel rendezvous module header is found. If installed in a non-standard location, `--with-psm3-rv=DIR` may be used to specify the parent directory. *Note:* The resulting PSM3 binary RPM may be run on systems with or without the kernel rendezvous module. When run on a system without the kernel rendezvous module, kernel rendezvous RC QPs, Direct GPU access, and GPUDirect options cannot be used. See PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 and Intel GPU Support, and PSM3 and NVIDIA CUDA Support.
`--enable-psm3-dsa`	check	Enables support for the Intel® Xeon® Processor Data Streaming Accelerator (DSA). See PSM3 Data Streaming Accelerator Support.
`--with-oneapi-ze`	disabled	Enables Intel GPU support via the oneAPI Level Zero library. See PSM3 and Intel GPU Support. *Note:* This requires that oneAPI Level Zero is installed on the current system. *Note:* The resulting PSM3 binary RPM may be run on systems with or without oneAPI Level Zero. When run on a system without oneAPI Level Zero, Intel GPU optimizations in PSM3 cannot be used. See PSM3 and Intel GPU Support and PSM3_ONEAPI_ZE.
`--with-cuda`	disabled	Enables NVIDIA GPU support via the CUDA library. See PSM3 and NVIDIA CUDA Support. *Note:* This requires that CUDA is installed on the current system. *Note:* The resulting PSM3 binary RPM may be run on systems with or without NVIDIA CUDA. When run on a system without CUDA, NVIDIA GPU optimizations in PSM3 cannot be used. See PSM3 and NVIDIA CUDA Support and PSM3_CUDA.
`--enable-psm3-hwloc`	check	Includes use of `hwloc` in the build. The hwloc library is required for GPU to NIC distance calculations during GPU jobs using NIC selection modes which consider GPU locality. When hwloc is not found or not enabled, these modes will not be available. See PSM3 Multi-Rail Support, PSM3_MULTIRAIL, and PSM3_NIC_SELECTION_ALG.
`--enable-psm3-umr-cache`	check	Includes use of `userfaultfd` in the build. The linux userfaultfd feature is required by the PSM3 user mode MR cache. When userfaultfd is not found or not enabled, this mode will not be available. See PSM3 Verbs RDMA Modes and Rendezvous Module and PSM3_MR_CACHE_MODE.

NOTE

When possible, Intel recommends that you use the pre-built binary RPMs packaged in the Intel® Ethernet Fabric Suite host software. The pre-built RPMs have been built using the latest Intel® oneAPI DPC++/C++ Compiler (`icx`) and distro-specific development tools, and may perform better than a PSM3 rebuilt with the default tools included with the distro.

NOTE

When building PSM3, the current system must have the appropriate set of development tools and libraries installed. The resulting binary RPM may then be installed on systems that lack the development tools. However, such systems will require the runtime versions of the equivalent libraries that were used by rpmbuild.

Building With The Intel® Compiler

By default, rebuilding PSM3 will use the `gcc` compiler unless the `CC` environment is set to select a different compiler. For example, to build PSM3 using the Intel® oneAPI DPC ++/C++ Compiler:

```
export CC=icx
rpmbuild --rebuild libpsm3-fi-<version>.src.rpm
```

NOTE

Make sure that the compiler's install location is included in `PATH`. This may require running `source` for the relevant `vars.sh` file that came with the given Intel compiler.

See the *Intel® Ethernet Fabric Suite Software Release Notes* for the list of supported and recommended compilers.

8.20 Running with Multiple PSM3 Variations

Depending on the environment, it may be beneficial to have multiple variations of PSM3 installed. The PSM3 libraries can be copied from `/usr/lib64/libfabric` into other directories, where switching between PSM3 variations can be accomplished by setting the `FI_PROVIDER_PATH` environment variable to the appropriate directory.

An example use case would be to have PSM3 variations to support different GPU environments such as support for Intel GPU and NVIDIA GPU in a OS image used across both types of hardware configurations.

```
export FI_PROVIDER_PATH=~/mypsm3-oneapi_ze/
```

or

```
export FI_PROVIDER_PATH=~/mypsm3-cuda/
```

8.21 PSM3 Environment Variables

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

8.21.1 PSM3 Config File

In many cases, it will be desirable for most or perhaps all jobs run on a given server to have specific PSM3 configuration settings. While this could be accomplished by always exporting the desired environment variables, this can be error prone, so PSM3 allows an optional, per server persistent configuration file (`/etc/psm3.conf`) to contain such settings.

`/etc/psm3.conf` may be used to control any of the PSM3 parameters specified in PSM3 Environment Variables. If a given parameter is specified more than once in `/etc/psm3.conf`, the last value specified for the given parameter will be used. If a given parameter is specified both in `/etc/psm3.conf` and the environment, the value in the environment will be used. If the value specified in the environment is empty, the value in `/etc/psm3.conf` will be ignored and the PSM3 internal default for the parameter will be used.

The syntax permitted in `/etc/psm3.conf` is similar to other configuration files. However, it is not a shell script, so only simple configuration value assignments may be specified. The syntax rules are as follows:

- Parameter assignments appear of the form `parameter_name=value` with no spaces between the `parameter_name` and the equals sign.
- The # (pound) synbol may be used to specify comment text. The # and all text after it on the given line are ignored.
- Any whitespace immediately after the equals sign is treated as part of the `value`. Intel recommends that you not include such whitespace.
- Any whitespace at the start of a line is ignored.
- Any whitespace at the end of a line or between the parameter value and any # for a comment are ignored.
- Only a single parameter assignment may be specified per line.
- If a parameter is assigned an empty value, the default value will be used unless the parameter is also specified in the environment.
- There is no quoting or special character escape mechanism. Special characters may be used directly and, when appropriate, whitespace may appear at the start or middle of a parameter value. The need for whitespace in a parameter value should be rare, unless filenames specified by a parameter happen to have whitespace in the middle of the name.

The following is a sample `/etc/psm3.conf` file:

```
# Sample /etc/psm3.conf file
PSM3_ALLOW_ROUTERS=1   # needed due to unique IP subnet per server in this cluster
PSM3_NIC_SPEED=+(100000|200000) # only consider 100g or 200g NICs
#PSM3_RDMA=1  # commented out parameter assignment, ignored
```

NOTE

The values specified in `/etc/psm3.conf` only affect jobs run on the given host. For many PSM3 parameters (such as PSM3_ONEAPI_ZE, PSM3_RDMA, etc), the same value must be used for all processes in a given job.

NOTE

PSM3_VERBOSE_ENV may be set in `/etc/psm3.conf`. When enabled, it and all subsequent, syntactically-valid, parameter assignments will be logged, regardless of what value for PSM3_VERBOSE_ENV is specified in the environment. Enabling this may be helpful when debugging or testing edits to `/etc/psm3.conf`. If the goal is only to test `/etc/psm3.conf`, it may be useful to return this setting to off later in `/etc/psm3.conf` so that the value in the environment (or the default of 0) is used for the remainder of the job. For example:

```
# sample use of PSM3_VERBOSE_ENV in /etc/psm3.conf
PSM3_VERBOSE_ENV=1
PSM3_VERBOSE_ENV=0 # restore default, env controls output once /etc/psm3.conf
parsed
PSM3_ALLOW_ROUTERS=1
PSM3_NIC_SPEED=+(100000|200000)
# PSM3_RDMA=1  # commented out assignments not shown
```

In this case, the output will include lines such as:

```
mynode:rank0: env /etc/psm3.conf: parsed PSM3_VERBOSE_ENV='1'
mynode:rank0: env /etc/psm3.conf: parsed PSM3_VERBOSE_ENV='0'
mynode:rank0: env /etc/psm3.conf: parsed PSM3_ALLOW_ROUTERS='1'
mynode:rank0: env /etc/psm3.conf: parsed PSM3_NIC_SPEED='+(100000|200000)'
```

NOTE

The parameter values specified in `/etc/psm3.conf` will not change the actual environment seen by any other libraries or sub-processes.

NOTE

`/etc/psm3.conf` cannot control environment variables used by other components, such as MPI or oneCCL. While it can control environment variables prefixed with `FI_PSM3_`, it cannot control other libfabric environment variables, such as those simply prefixed with `FI_`.

NOTE

The PSM3_DISABLE_MMAP_MALLOC parameter cannot be controlled by `/etc/psm3.conf`. When a non-default value is desired, this variable must be set in the environment.

NOTE

Any value specified for PSM3_TRACEMASK will not take effect until after `/etc/psm3.conf` is fully parsed. If it is desired to get debug output while parsing `/etc/psm3.conf`, use PSM3_VERBOSE_ENV.

NOTE

Some parameters may be overridden by the Intel® MPI Library. To see the actual values being used, set `PSM3_VERBOSE_ENV=1:` (see PSM3_VERBOSE_ENV). See Environment Variables for Intel® MPI Library Jobs for more information.

8.21.2 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.

NOTE

This parameter's default may be overridden by the Intel® MPI Library. To see the actual value being used, set `PSM3_VERBOSE_ENV=1:` (see PSM3_VERBOSE_ENV). See Environment Variables for Intel® MPI Library Jobs for more information.

8.21.3 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 variable 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. Therefore, its use is discouraged.

NOTE

This parameter's default may be overridden by the Intel® MPI Library. To see the actual value being used, set `PSM3_VERBOSE_ENV=1:` (see PSM3_VERBOSE_ENV). See Environment Variables for Intel® MPI Library Jobs for more information.

8.21.4 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 hexidecimal digits separated by dashes at the exact points shown in the default.

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

NOTE

This parameter's default may be overridden by the Intel® MPI Library. To see the actual value being used, set `PSM3_VERBOSE_ENV=1:` (see PSM3_VERBOSE_ENV). See Environment Variables for Intel® MPI Library Jobs for more information.

8.21.5 PSM3_ADDR_FMT

Specifies the type of network address(es) to consider for use within each NIC.

Options:

- `0` – Consider all address formats (IPv4, IPv6, and InfiniBand). For Ethernet ports, the first IPv4 address whose subnet matches PSM3_SUBNETS is used. Otherwise, the first IPv6 address whose subnet matches PSM3_SUBNETS is used. For InfiniBand ports, the first GID whose subnet matches PSM3_SUBNETS is used.
- `3` – Consider only InfiniBand ports. Use the first GID whose subnet matches PSM3_SUBNETS.
- `4` – Consider only Ethernet ports with an IPv4 address. Use the first IPv4 address whose subnet matches PSM3_SUBNETS.
- `6` – Consider only Ethernet ports with an IPv6 address. Use the first IPv6 address whose subnet matches PSM3_SUBNETS.

Default: `0`

This is just one of the filters applied to select a NIC and address within a NIC. See NIC and Address Filtering for more information.

NOTE

Even if a NIC is filtered out due to lack of an address with the proper format, it is still assigned a unit number based on an alphabetic sort by name among the NICs supported by a given HAL. As such, unit numbers remain constant within a given HAL regardless of which NICs have been filtered out. Unit numbers may be used in environment variables such as PSM3_NIC and PSM3_MULTIRAIL_MAP. However, those variables must select a unit that has not been filtered out.

NOTE

Addresses are considered in the order the GIDs are shown in `ibv_devinfo -v`. For Ethernet ports, IPv4 addresses appear in `ibv_devinfo -v` of the form `0000:0000:0000:0000:0000:ffff:`*`xxxx:xxxx`* where *`xxxx:xxxx`* is the 32-bit IPv4 address. All other forms are treated as IPv6 addresses.

NOTE

The NIC(s) and addresses selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the NIC and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

8.21.6 PSM3_ADDR_PER_NIC

Indicates whether PSM3 should use multiple IP addresses per NIC.

Values between 1 and 32 are valid.

Default: `1`

If a NIC is configured with more than one IP address, it is possible to achieve higher performance by using multiple IP addresses, in conjunction with appropriate configuration of the fabric switches, to spread PSM3 traffic across multiple routes in a fabric or plane. To do this, set `PSM3_ADDR_PER_NIC` to the number of IP addresses to use on each NIC.

NOTE

Only the first `PSM3_ADDR_PER_NIC` unfiltered IP addresses of a given type (IPv4 or IPv6) will be used. For example, if each NIC has four unfiltered IPv4 addresses defined, but `PSM3_ADDR_PER_NIC=2`, then only the first two IPv4 addresses on each NIC will be used.

NOTE

As discussed in NIC and Address Filtering, only NICs that have at least `PSM3_ADDR_PER_NIC` unfiltered addresses defined will be considered for use. If no NICs have at least `PSM3_ADDR_PER_NIC` unfiltered addresses defined, PSM3 will report no NICs are available.

NOTE

See PSM3 Multi-IP Support for more information on possible ways to take advantage of multiple IP addresses per NIC.

NOTE

The Intel PSM3 implementation has a limit of a total of 32 possible NIC IP addresses per process. (i.e., number of unfiltered NICs multiplied by `PSM3_ADDR_PER_NIC` must be <= 32).

NOTE

Care must be taken when combining PSM3_MULTIRAIL, PSM3_QP_PER_NIC, and PSM3_ADDR_PER_NIC as the PSM3 limit of 32 QPs per endpoint can be easily exceeded.

8.21.7 PSM3_ALLOW_ROUTERS

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

- `0` – Consider endpoints with different IP subnets unaccessible.
- `1` – Consider all Ethernet endpoints accessible, even if they have different IP subnets, provided they have the same address type (IPv4 versus IPv6).

Default: `0`

The IP subnet is defined by taking the IP address for a NIC and masking it with the IP subnet mask. When set to `1`, PSM3 will assume IP routers are configured in the network such that all Ethernet endpoints using the same address type (IPv4 or IPv6) can communicate with each other regardless of IP subnet. When set to `0`, PSM3 will assume endpoints with different IP subnets are separated and unable to communicate. See PSM3 Multi-Rail Support for more details about multi-subnet configurations. See PSM3 Multi-IP Support for more details about multi-IP address per NIC configurations.

If set incorrectly, jobs that require NICs in different IP subnets to communicate may fail to execute. This variable can influence the association of NICs to processes for PSM3_MULTIRAIL and PSM3_NIC_SELECTION_ALG.

NOTE

Two subnets are considered equivalent only if both the subnet and the subnet prefix length (i.e., the netmask) are equal. As such `192.168.128.0/24` and `192.168.128.0/20` are considered different subnets.

NOTE

It is always assumed that endpoints with different types of addresses (IPv4 versus IPv6 versus InfiniBand) cannot communicate. While such communications may be possible in some configurations, these configurations are atypical in high performance networks and therefore excluded.

NOTE

The NIC(s) and addresses selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the NIC and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

8.21.8 PSM3_CONNECT_TIMEOUT

Overrides the endpoint 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 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) seconds.

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

8.21.9 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

NOTE

A given PSM3 library build and a given kernel rendezvous module can only support one vendor's GPUs.

8.21.10 PSM3_CUDA_THRESH_RNDV

This variable has been deprecated. Use PSM3_GPU_THRESH_RNDV instead.

8.21.11 PSM3_DEBUG_FILENAME

Controls where additional debug output, which has been selected by PSM3_TRACEMASK, will be placed. The filename specified may use the markers `%h` and `%p` that 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.

NOTE

When specifying a PSM3_TRACEMASK setting which may generate large amounts of output, it can be beneficial to use `PSM3_DEBUG_FILENAME` to control where debug output is placed.

8.21.12 PSM3_DEVICES

Selects which PSM3 communications subsystems will be enabled and the order in which they are considered for communications within a job: See PSM3 Architecture and Hardware Abstraction Layer. The valid subsystems are:

- `self` - Allows a process to send messages to itself.
- `shm` - Allows a process to send messages to other processes on the same host via a variety of mechanisms including: Linux shared memory (shm), direct CPU process to CPU process copies, direct GPU to GPU transfers, and/or the Data Streaming Accelerator (DSA).
- `nic` - Allows a process to send messages to other processes via the NIC.

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

For jobs that do not require PSM3 `shm` style communications, `PSM3_DEVICES` can be specified as `self,nic`. For example, if the middleware above PSM3 is handling such communications itself or the job has only 1 process per node.

Similarly, for single node, shared-memory only jobs, `PSM3_DEVICES` can be specified as `self,shm`.

You must ensure that the endpoint included in a job does not require a subsystem that has been explicitly disabled (i.e., omitted). In some instances, enabling only the subsystems that are required may improve performance.

NOTE

For jobs using GPUs, inclusion of the `nic` subsystem may be required to take advantage of Direct GPU Copy or GPUDirect Copy (see PSM3_GPUDIRECT).

NOTE

Jobs that do not specify `nic` in `PSM3_DEVICES` will not open any NICs and will run using a special loopback HAL. In this use case, NIC and Address Filtering is not applicable as no NIC is required.

NOTE

For some platforms or workloads with multiple processes per host, use of the `nic` subsystem instead of the `shm` subsystem for communications between processes on the same host may provide performance advantages. In which case, `PSM3_DEVICES=self,nic,shm` will use the NIC when possible and fallback to `shm` as needed. When using a PSM3_RDMA mode which uses kernel RC QPs in the rendezvous module, use of the NIC for communications within the host only occurs between processes using different sets of NICs. See PSM3 Verbs RDMA Modes and Rendezvous Module.

8.21.13 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.

NOTE

This parameter cannot be specified in the PSM3 configuration file (`/etc/psm3.conf`) as discussed in PSM3 Config File.

8.21.14 PSM3_DSA_MULTI

Tells PSM3 how to associate work queues specified in PSM3_DSA_WQS with processes in the job.

Options:

- `0` – By NUMA. Each colon-separated set of work queues in PSM3_DSA_WQS is associated with a NUMA domain. The NUMA domain of the process at PSM3 initialization time is used as an index to select the set of work queues in PSM3_DSA_WQS.
- `1` – By local rank. Each colon-separated set of work queues in PSM3_DSA_WQS is associated with a local rank; using the local rank as an index to select the set of work queues in PSM3_DSA_WQS.
- `2` – Auto. Auto selects mode 0 or 1. Mode 0 is selected if sufficient sets of DSA work queues are listed in PSM3_DSA_WQS to cover all CPU NUMA domains and either the work queues specified are all shared work queues or the number of local processes is <= number of CPU NUMA domains.

Default: `1`

When using dedicated DSA work queues with more than one process per NUMA domain, `PSM3_DSA_MULTI` must be 1.

NOTE

Actual CPU process pinning is not confirmed at job launch time by `PSM3_DSA_MULTI=2`. When using this mode with dedicated work queues, ensure processes are distributed one process per CPU NUMA domain.

NOTE

See PSM3 Data Streaming Accelerator Support and PSM3_DSA_WQS for more information.

8.21.15 PSM3_DSA_WQS

Tells PSM3 which DSA work queues to use for each process.

Options: `wq0,wq1;wq2,wq3;...` where each `wq` must be a unique DSA work queue `/dev` file (such as `/dev/dsa/wq2.0`). Each process may be given a list of DSA work queues, separated by commas. Multiple process specifications are separated by semicolons. In some cases, extraneous whitespace may cause parse errors, so whitespace should be avoided.

Each `wq` specified must be a unique, valid DSA dedicated or shared work queue. All work queues must be created by the sysadmin before the application initializes PSM3. PSM3_DSA_MULTI controls how processes are associated to work queues. By default, local process ranks in a given PSM3 job will be associated with the listed work queues in the order shown.

By default `PSM3_DSA_WQS` is empty and DSA will not be used.

Some example uses of PSM3_DSA_WQS when PSM3_DSA_MULTI is defaulted or mode 1 is selected (per process selection):

- PSM3_DSA_WQS=/dev/dsa/wq0.0;/dev/dsa/wq5.0 - Allow a job with up to two processes per node. Assign /dev/dsa/wq0.0 to the first process on the node and /dev/dsa/wq5.0 to the second process on the node. Within a given process, all threads will share the same DSA work queue.
- PSM3_DSA_WQS=/dev/dsa/wq0.0,/dev/dsa/wq2.0;/dev/dsa/wq5.0,/dev/dsa/wq7.0 - Allow a job with up to two processes per node. Assign /dev/dsa/wq0.0 and /dev/dsa/wq2.0 to the first process on the node. Assign /dev/dsa/wq5.0 and /dev/dsa/wq7.0 to the second process on the node. Within a given process, PSM3 will load balance the assignment of threads to DSA work queues.
- PSM3_DSA_WQS=/dev/dsa/wq0.0,/dev/dsa/wq2.0,/dev/dsa/wq3.0,/dev/dsa/wq4.0;/dev/dsa/wq5.0,/dev/dsa/wq7.0,/dev/dsa/wq8.0,/dev/dsa/wq9.0 - Allow a job with up to two processes per node. Assign four work queues to each process. Within a given process, PSM3 will load balance the assignment of threads to DSA work queues.
- PSM3_DSA_WQS=/dev/dsa/wq0.0;/dev/dsa/wq2.0;/dev/dsa/wq3.0;/dev/dsa/wq4.0;/dev/dsa/wq5.0;/dev/dsa/wq7.0;/dev/dsa/wq8.0;/dev/dsa/wq9.0 - Allow a job with up to eight processes per node. Assign one work queue to each process. Within a given process, all threads will share the same DSA work queue.

The following describes some example uses of PSM3_DSA_WQS when PSM3_DSA_MULTI mode 0 is selected (per NUMA selection). In these examples, if more than one process is run per CPU NUMA domain, the specified DSA work queues must be shared DSA work queues:

- PSM3_DSA_WQS=/dev/dsa/wq0.0;/dev/dsa/wq5.0 - Allow a job that uses up to two CPU NUMA domains per node. Assign /dev/dsa/wq0.0 to processes on the first CPU NUMA domain and /dev/dsa/wq5.0 to processes on the second CPU NUMA domain. Within a given process, all threads will share the same DSA work queue.
- PSM3_DSA_WQS=/dev/dsa/wq0.0,/dev/dsa/wq2.0;/dev/dsa/wq5.0,/dev/dsa/wq7.0 - Allow a job that uses up to two CPU NUMA domains per node. Assign /dev/dsa/wq0.0 and /dev/dsa/wq2.0 to processes on the first CPU NUMA domain. Assign /dev/dsa/wq5.0 and /dev/dsa/wq7.0 to processes on the second CPU NUMA domain. Within a given process, PSM3 will load balance the assignment of threads to DSA work queues.
- PSM3_DSA_WQS=/dev/dsa/wq0.0,/dev/dsa/wq2.0,/dev/dsa/wq3.0,/dev/dsa/wq4.0;/dev/dsa/wq5.0,/dev/dsa/wq7.0,/dev/dsa/wq8.0,/dev/dsa/wq9.0 - Allow a job that uses up to two CPU NUMA domains per node. Assign four work queues to each CPU NUMA domain. Within a given process, PSM3 will load balance the assignment of threads to DSA work queues.
- PSM3_DSA_WQS=/dev/dsa/wq0.0;/dev/dsa/wq2.0;/dev/dsa/wq5.0;/dev/dsa/wq7.0 - Allow a job with up to four CPU NUMA domains per node. Assign one work queue to each CPU NUMA domain. Within a given process, all threads will share the same DSA work queue.

NOTE

Only a single process per node may use a given DSA dedicated work queue. When using dedicated work queues, if more than one PSM3 job is running concurrently on a given node, each job must be assigned a unique set of DSA work queues. Similarly, while jobs are running using PSM3, dedicated work queues assigned to PSM3 jobs within a given node cannot be concurrently used by any other applications or software on the given node.

NOTE

The DSA work queues assigned to a given PSM3 process do not have to be on the same CPU socket as the process. For some applications, there may be performance benefits to pinning ranks on a given node to a core in the same CPU socket as the DSA devices that it will use. However, in cases where there is only one process per CPU socket, all DSA copies will be crossing NUMA domains, and there may be no benefit to NUMA locality of the DSA devices used.

NOTE

If the nodes in a given job have different sets of DSA work queues (perhaps due to different CPU hardware or different DSA hardware capabilities), it may be necessary to provide unique values for `PSM3_DSA_WQS` to different servers. If only one PSM3 job will be run at a time per server, one way to accomplish this is via `/etc/psm3.conf`. See PSM3 Config File for more information.

NOTE

The DSA work queues selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the DSA work queue selection per thread can be shown by enabling bit `0x100` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

NOTE

See PSM3 Data Streaming Accelerator Support and PSM3_DSA_MULTI for more information.

8.21.16 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 recovery from packet loss 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 via retries versus 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 that 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).

8.21.17 PSM3_FLOW_CREDITS

The number of concurrent unacknowledged packets (credits) permitted per flow. A flow is a stream of packets between a specific pair of endpoints. When a flow is experiencing packet loss or delays, lower flow credits are preferred to reduce network pressure. However, using too low a value for flow credits may impact overall message rate and bandwidth. The syntax is:

```
PSM3_FLOW_CREDITS=min[:max[:adjust]]
```

Default value: For RC verbs (i.e., the verbs HAL is selected and PSM3_RDMA is `3`, see PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module) `PSM3_FLOW_CREDITS=768:960`. For UD verbs (i.e., the verbs HAL is selected and PSM3_RDMA is `0` or `2`) and UDP (i.e., the sockets HAL is selected and PSM3_SOCKETS is `1`), `PSM3_FLOW_CREDITS=32:128:16`.

Fields:

- `min`
- `max`

  The values of `min` and `max` set the range of credits for a flow. For UD verbs, the credits for a flow will decrease if packet loss or delays are observed, and will increase if packets are flowing reliably. For RC verbs, data transition pauses when unacknowledged packets beyond `max`, and resumes when unacknowledged packets is less than `min`. `min` and `max` are positive numbers, and `max` must be larger than or equal to `min`.

- `adjust`

  For UD verbs, `adjust` controls how aggressively the flow credit is dynamically adjusted. It defines the amount by which credits are increased or decreased to adapt to network behaviors. For RC verbs, `adjust` is ignored.

If only `min` is specified, a single fixed value will be used and will not be dynamically adjusted. If the `adjust` field is not specified, its default value will be used.

NOTE

When using TCP (i.e., the sockets HAL is selected and PSM3_SOCKETS is `0`, see PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes), `PSM3_FLOW_CREDITS` is ignored. See PSM3_TCP_SNDPACING_THRESH for TCP send pacing.

NOTE

When using UDP (i.e., the sockets HAL is selected and PSM3_SOCKETS is `1`, see PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes), the amount of concurrent unacknowledged data will also automatically be constrained by the amount of UDP socket send buffering available.

8.21.18 PSM3_FORCE_SPEED

Tells PSM to ignore the speed reported by each NIC in `/sys/class/` and use the specified value instead. Speed is specified in units of megabits per second (Mbps). The value specified will be reported by PSM3 as part of the libfabric `fi_info` for each of the available NICs and may influence middleware behaviors, such as MPI collective algorithm selections. If set to `0`, the actual NIC speed is obtained from `/sys/class/` and reported.

Default: `PSM3_FORCE_SPEED=0`

Some example uses:

- `PSM3_FORCE_SPEED=0` – Use actual NIC speed as reported in `/sys/class`.
- `PSM3_FORCE_SPEED=100000` – Report a NIC speed of 100 Gbps (100,000 Mbps) for all NICs found.

NOTE

When `PSM3_FORCE_SPEED` is non-zero, all NICs will report the specified speed for the purposes of PSM3_NIC_SPEED NIC selection filtering. For more information, see NIC and Address Filtering.

NOTE

This environment variable should only be used if one or more of the NICs lack the relevant file in `/sys/class` or have an inaccurate value in the relevant file. The main situations where this may occur are virtualized environments or environments where NICs are purposely throttled to a lower speed. For verbs, the relevant file is `/sys/class/infiniband/<DEVICE_NAME>/ports/<PORT_NUMBER>/rate`. For sockets, the relevant file is `/sys/class/net/<DEVICE_NAME>/speed`. For more information, see PSM3 Architecture and Hardware Abstraction Layer.

NOTE

On servers where this is required, it can be useful to add this setting to the PSM3 configuration file (`/etc/psm3.conf`) as discussed in PSM3 Config File.

8.21.19 PSM3_GPUDIRECT

GPU Direct Access and GPUDirect are technologies that enable a direct path for data exchange between a graphics processing unit (GPU) and a third-party peer device using standard features of PCI Express.

When set for Intel GPUs, this enables Direct GPU Copy, Direct GPU Send DMA, and Direct GPU RDMA. When set for NVIDIA GPUs, this enables GPUDirect Copy, GPUDirect Send DMA, and GPUDirect RDMA support. These allow increased performance through direct data exchange between GPU and NIC. When enabled, the rendezvous driver is required with support for the appropriate vendor's GPU. 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_ONEAPI_ZE=1` or `PSM3_CUDA=1`

NOTE

When using the sockets Hardware Abstraction Layer (HAL), only Direct GPU Copy or GPUDirect Copy are available. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

For Intel GPUs, see: PSM3 and Intel GPU Support and PSM3_ONEAPI_ZE

For NVIDIA GPUs, see: PSM3 and NVIDIA CUDA Support and PSM3_CUDA. For more information, see the NVIDIA CUDA documentation: https://developer.nvidia.com/gpudirect and http://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

NOTE

A given PSM3 library build and a given kernel rendezvous module can only support one vendor's GPUs.

NOTE

This parameter's default may be overridden by the Intel® MPI Library. To see the actual value being used, set `PSM3_VERBOSE_ENV=1`: (see PSM3_VERBOSE_ENV). See Environment Variables for Intel® MPI Library Jobs for more information.

8.21.20 PSM3_GPUDIRECT_RDMA_RECV_LIMIT

Sets the upper bound for receiver use of Direct GPU RDMA or GPUDirect RDMA when the verbs HAL is selected. Messages being received into a GPU buffer, which are larger than `PSM3_GPUDIRECT_RDMA_RECV_LIMIT` bytes, will not use Direct GPU RDMA nor GPUDirect RDMA. However, depending on other settings, RDMA may still be used to a host-based buffer which will then be copied to the GPU, and Direct GPU RDMA or GPUDirect RDMA may still be used on the sender.

Options:

- Any value between 0 and 4 GB (4294967295). Smaller values may disable the use of GPUDirect RDMA on the receiver entirely.
- `max` - Selects the largest valid value (4294967295).

Default is `PSM3_GPUDIRECT_RDMA_RECV_LIMIT=1` when Intel GPU support is enabled (i.e., PSM3_ONEAPI_ZE is enabled). Default is `PSM3_GPUDIRECT_RDMA_RECV_LIMIT=max` (4294967295) when NVIDIA GPU support is enabled (i.e., PSM3_CUDA is enabled).

See PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3_GPU_THRESH_RNDV, PSM3 Architecture and Hardware Abstraction Layer, and PSM3_GPUDIRECT_RDMA_SEND_LIMIT for more details.

NOTE

This setting is only used when PSM3_GPUDIRECT is enabled.

8.21.21 PSM3_GPUDIRECT_RDMA_SEND_LIMIT

Sets the upper bound for sender use of Direct GPU RDMA or GPUDirect RDMA when the verbs HAL is selected. Messages being sent from a GPU buffer, which are larger than `PSM3_GPUDIRECT_RDMA_SEND_LIMIT` bytes, will not use Direct GPU RDMA nor GPUDirect RDMA. However, depending on other settings, RDMA may still be used after copying GPU data to a host-based buffer, and Direct GPU RDMA or GPUDirect RDMA may still be used on the receiver.

Options:

- Any value between 0 and 4 GB (4294967295). Smaller values may disable the use of GPUDirect RDMA on the sender entirely.
- `max` - Selects the largest valid value (4294967295).

Default: `PSM3_GPUDIRECT_RDMA_SEND_LIMIT=max` (4294967295).

See PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3_GPU_THRESH_RNDV, PSM3 Architecture and Hardware Abstraction Layer, and PSM3_GPUDIRECT_RDMA_RECV_LIMIT for more details.

NOTE

This setting is only used when PSM3_GPUDIRECT is enabled.

8.21.22 PSM3_GPU_RNDV_NIC_WINDOW

Sets the window size in bytes for messages from or to GPU buffers. This controls how large messages are split for transmission.

Larger window sizes may reduce CPU loading. Smaller window sizes may provide better distribution of bandwidth in workloads with many simultaneous destinations like an MPI collective operation, but will slightly increase CPU loading.

When PSM3_MULTIRAIL is active or PSM3_QP_PER_NIC is > 1 or PSM3_RV_QP_PER_CONN is > 1, this value controls the granularity at which messages are striped across multiple NICs and/or QPs, respectively.

When GPU copy pipelines are used in conjunction with RDMA, the window size also controls the size of each pipelined copy for send data pre-fetching or receive data pipelining. In which case, the proper window size may allow overlap of copy and data transmission resulting in better pipeline performance. See PSM3 and GPU Support, PSM3_GPUDIRECT, PSM3_GPUDIRECT_RDMA_RECV_LIMIT, and PSM3_GPUDIRECT_RDMA_SEND_LIMIT for more information.

Specified as a list of the form: `window_size:limit,window_size:limit,window_size:limit,...`, where `window_size` selects the actual transmission size as a value between 1 and 4194304 bytes inclusive and `limit` specifies the largest sized message which will use the given `window_size`. `limit` is specified as a value between 1 and 4294967295 inclusive, where 4294967295 can also be specified as `max`.

Some example uses of `PSM3_GPU_RNDV_NIC_WINDOW`:

- `PSM3_GPU_RNDV_NIC_WINDOW=131072` - A transmission size of 131072 is used for all message sizes. This is equivalent to `PSM3_GPU_RNDV_NIC_WINDOW=131072:4294967295` or `PSM3_GPU_RNDV_NIC_WINDOW=131072:max`.
- `PSM3_GPU_RNDV_NIC_WINDOW=131072:524287,262144:1048575,524288` - A transmission size of 131072 is used for message sizes up to 524287 bytes. Messages of 524288 to 1048575 bytes will use a transmission size of 262144 bytes and messages 1048576 bytes or larger will use a transmission size of 524288 bytes.

The `limit` specified for a given entry in the list, must be larger than the `limit` for the prior entry. When a `limit` is not specified for a given entry, it defaults to `max` (4294967295). For the last `window_size` in the list, a `limit` of `max` (4294967295) is always used and need not be specified. Multiple `window_size` specifications are separated by commas. A trailing comma will be ignored. In some cases, extraneous whitespace may cause parse errors, so whitespace should be avoided.

The default is `PSM3_GPU_RNDV_NIC_WINDOW=131072:524287,262144:1048575,524288` when Intel GPU support is enabled (i.e., PSM3_ONEAPI_ZE is enabled). The default is `PSM3_GPU_RNDV_NIC_WINDOW=2097152` when NVIDIA GPU support is enabled (i.e., PSM3_CUDA is enabled).

NOTE

Each `window_size` specified will rounded up to be a multiple of the CPU page size.

NOTE

This setting is only used when PSM3_GPUDIRECT is enabled.

Also see PSM3_RNDV_NIC_WINDOW.

See PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Multi-Rail Support, PSM3 Support for Intel GPUs, and PSM3 and GPU Support for more details.

8.21.23 PSM3_GPU_THRESH_RNDV

Sets the eager-to-rendezvous switchover threshold in bytes for messages sent from a GPU buffer. Messages larger than `PSM3_GPU_THRESH_RNDV` will use rendezvous while those equal or smaller than `PSM3_GPU_THRESH_RNDV` will use eager. Typically, rendezvous is used for larger messages. As outlined below, the usage depends on the selected HAL PSM3 Architecture and Hardware Abstraction Layer.

When the verbs HAL is selected (see PSM3_HAL, PSM3 Verbs RDMA Modes and Rendezvous Module) and PSM3_RDMA=1, 2, or 3, rendezvous 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_GPU_RNDV_NIC_WINDOW.

When the sockets HAL is selected (see PSM3_HAL, PSM3 Sockets Modes), rendezvous may permit some pipelining of GPU to CPU copies to improve both latency and bandwidth.

Options:

- Any value between 0 and 4 GB (4294967295). 4 GB will disable rendezvous entirely.
- `max` - Selects the largest valid value (4294967295) (disables rendezvous entirely).

Default : `PSM3_GPU_THRESH_RNDV=8000` when the verbs HAL is selected. `PSM3_GPU_THRESH_RNDV=4294967295` when the sockets HAL is selected.

See PSM3 Verbs RDMA Modes and Rendezvous Module and PSM3 Sockets Modes for more details.

Also see PSM3_MQ_RNDV_NIC_THRESH, PSM3_GPU_RNDV_NIC_WINDOW, PSM3_GPUDIRECT_RDMA_RECV_LIMIT, and PSM3_GPUDIRECT_RDMA_SEND_LIMIT.

NOTE

This setting is only used when PSM3_GPUDIRECT is enabled.

NOTE

This environment variable is applicable to both Intel GPUs and NVIDIA GPUs.

NOTE

The environment variable is a new name for PSM3_CUDA_THRESH_RNDV, which has been deprecated.

8.21.24 PSM3_HAL

Specifies the Hardware Abstraction Layer (HAL) to use for the process. See PSM3 Architecture and Hardware Abstraction Layer and Building the PSM3 RPM.

Options:

- `any` – Consider all available HAL layers, select the first that has an acceptable NIC (verbs is checked first, then sockets).
- `verbs` – Consider only the verbs HAL. See PSM3 Verbs RDMA Modes and Rendezvous Module.
- `sockets` – Consider only the sockets HAL. See PSM3 Sockets Modes.

Default: `any`

The HAL selected controls the details of what hardware API and lower-level PSM3 protocol will be used for data movement. The `verbs` HAL takes advantage of the verbs API and may use kernel bypass and RDMA mechanisms to efficiently move data. The `verbs` HAL is typically the best performing HAL on a given RDMA capable NIC.

The `sockets` HAL makes use of the sockets API and TCP/IP. Most NICs will support sockets, but sockets lacks kernel bypass and often depends on interrupts and additional data copies to move data. However, in systems without an RDMA capable NIC, this may be the best choice for running applications.

HAL selection occurs early during process launch, and exactly one HAL will be selected for all PSM3 node-to-node communications by a given process. All the processes in a job must use the same HAL. When `any` is specified (the default), all HALs are considered by applying the NIC and Address Filtering rules against all available NICs that can support the given HAL. If at least one acceptable NIC is found, the given HAL will be selected. The `verbs` HAL is considered first, as it will typically offer the best performance, followed by the `sockets` HAL.

NOTE

Unit numbers are assigned among all the NICs that a given HAL can support. The use of NIC names or patterns, as opposed to unit numbers, is generally recommended in PSM3_NIC, especially when `PSM3_HAL=any`. PSM3_MULTIRAIL_MAP is evaluated only after a HAL has been selected, but may similarly benefit from use of NIC names as opposed to unit numbers.

NOTE

The HAL selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the HAL, NIC, and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

NOTE

PSM3 build options control which HALs are included in the PSM3 binary as well as which data movement protocols are available within each included HAL. See Building the PSM3 RPM.

NOTE

This parameter's default may be overridden by the Intel® MPI Library. To see the actual value being used, set `PSM3_VERBOSE_ENV=1:` (see PSM3_VERBOSE_ENV). See Environment Variables for Intel® MPI Library Jobs for more information.

8.21.25 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 that use the same PSM3_FI_UUID (as defaulted, set explicitly, or set via the MPI middleware) must use the same service id.

NOTE

This setting is only used when the verbs HAL is selected and the rendezvous module is used for RDMA (i.e., PSM3_RDMA is `1`). Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, and PSM3 Rendezvous Kernel Module.

8.21.26 PSM3_IDENTIFY

Enable verbose output of process PSM3 software version identification including library location, build date, GPU support, rendezvous module API version (if rendezvous module is being used, see PSM3 Verbs RDMA Modes and Rendezvous Module), process rank IDs, total ranks per node, total ranks in the job, CPU core, NIC(s) selected, the address selected within each NIC, and the DSA work queues selected (if DSA is being used, see PSM3 Data Streaming Accelerator Support).

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 # is the relative process number in the job. If MPI runtime and the job scheduler have not indicated the rank to PSM3, the label will be of the form `hostname:pid#`, where # 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 a large amount of output especially when all processes are selected in a high process count job.

NOTE

If the job does not output any `PSM3_IDENTIFY` lines, either PSM3 was not used in the job or `PSM3_IDENTIFY` was not properly specified and exported. In this case, review the parameters used for job launch.

The following provides sample alternatives for a given portion of the `PSM3_IDENTIFY` output:

- The type of PSM3 provider build and version is shown.

  In the `PSM3_IDENTIFY` output, oneAPI Level Zero enablement of the PSM3 OFI Provider is indicated via `oneapi-ze` after the PSM protocol version, such as:

  ```
  PSM3_IDENTIFY PSM3 v3.0 onezpi-ze built for IEFS X.Y
  ```

  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 X.Y
  ```

Non-GPU enabled PSM3 OFI Providers appear such as:

```
PSM3_IDENTIFY PSM3 v3.0 built for IEFS X.Y
```

- The location of the PSM3 provider shared library is also indicated.

  If the selected PSM3 provider was separately installed, such as via the Intel® Ethernet Fabric Suite installation or a standalone PSM3 RPM, the location will appear as:

  ```
  PSM3_IDENTIFY location /usr/lib64/libfabric/libpsm3-fi.so
  ```

  If the selected PSM3 provider is built into the Intel® MPI Library, the location may appear such as:

  ```
  PSM3_IDENTIFY location /software/oneAPI/2023.0/mpi/2021.8.0/libfabric/lib/
  prov/libpsm3-fi.so
  ```

  If the selected PSM3 provider was built into libfabric itself, the location will appear as:

  ```
  PSM3_IDENTIFY location /lib64/libfabric.so.1
  ```

- The HAL selected (see PSM3 Architecture and Hardware Abstraction Layer) and how it was built is also reflected.

  If the selected PSM3 HAL was built without rendezvous module (rv) support, the `PSM3_IDENTIFY` output will indicate it, such as:

  ```
  PSM3_IDENTIFY HAL: verbs (RDMA Verbs)
              or
  PSM3_IDENTIFY HAL: sockets (TCP Sockets)
              or
  PSM3_IDENTIFY HAL: sockets (Sockets)
              or
  PSM3_IDENTIFY HAL: loopback (loopback)
  ```

  If the selected PSM3 HAL was built with rendezvous module (rv) support, the `PSM3_IDENTIFY` output will indicate it, such as:

  ```
  PSM3_IDENTIFY HAL: verbs (RDMA Verbs) built against rv interface v1.1
  ```

  If PSM3 was built with rendezvous module and Intel GPU oneAPI Level Zero support, the rendezvous GPU API version is shown, such as:

  ```
  PSM3_IDENTIFY HAL: verbs (RDMA Verbs oneapi-ze) built against rv interface
  v1.1 gpu v1.0 oneapi-ze
              or
  PSM3_IDENTIFY HAL: sockets (TCP Sockets oneapi-ze) built against rv interface
  v1.1 gpu v1.0 oneapi-ze
              or
  PSM3_IDENTIFY HAL: sockets (Sockets oneapi-ze) built against rv interface
  v1.1 gpu v1.0 oneapi-ze
  ```

If PSM3 was built with rendezvous module and CUDA support, the rendezvous GPU API version is shown, such as:

```
PSM3_IDENTIFY HAL: verbs (RDMA Verbs cuda) built against rv interface v1.1
gpu v1.0 cuda
            or
PSM3_IDENTIFY HAL: sockets (TCP Sockets cuda) built against rv interface v1.1
gpu v1.0 cuda
            or
PSM3_IDENTIFY HAL: sockets (Sockets cuda) built against rv interface v1.1 gpu
v1.0 cuda
```

- If GPU support was enabled (see PSM3 Support for GPUs), the `PSM3_IDENTIFY` output will indicate the version of the runtime library PSM3 is using and what interface version PSM3 was built against.

  When Intel GPU support is enabled (i.e., PSM3_ONEAPI_ZE is enabled), the oneAPI Level Zero library version is reported, such as:

  ```
  PSM3_IDENTIFY Level-Zero Runtime 1.6 (v1.11.0) built against interface 1.6
  ```

  When NVIDIA GPU support is enabled (i.e., PSM3_CUDA is enabled) the NVIDIA CUDA library version is reported, such as:.

  ```
  PSM3_IDENTIFY CUDA Runtime 12.2 built against interface 12.0
  ```

- 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
  ```

  When `user_mr` is present, it indicates `enable_user_mr` is enabled within the rendezvous module, such as:

  ```
  PSM3_IDENTIFY run-time rv interface v1.1 user_mr
  ```

  If both PSM3 and rendezvous module support Intel GPUs via oneAPI Level Zero, the rendezvous GPU API version is also shown such as:

  ```
  PSM3_IDENTIFY run-time rv interface v1.1 user_mr gpu v1.0 oneapi-ze
  ```

  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 user_mr gpu v1.0 cuda
  ```

  If the rendezvous module supports Intel GPUs via oneAPI Level Zero, but PSM3 does not, this will be indicated such as:

  ```
  PSM3_IDENTIFY run-time rv interface v1.1 user_mr oneapi-ze
  ```

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 user_mr cuda
```

See PSM3 Rendezvous Kernel Module.

- The `PSM3_IDENTIFY` output will show the NIC(s) selected and their network addresses. Ethernet addresses are shown in the Classless Inter-Domain Routing (CIDR) notation. For example, an IPv4 address of `192.168.100.77` with a netmask of `255.255.255.0` is shown as `192.68.100.77/24` reflecting the full address, and that the IPv4 subnet is the first 24 bits of the address (`192.168.100.0`).
- When DSA is used, the `PSM3_IDENTIFY` output will show the list of DSA work queues being used by the given process. For more information, see PSM3 Data Streaming Accelerator Support.

NOTE

A given PSM3 library build and a given kernel rendezvous module can only support one vendor's GPUs (cuda or oneapi-ze).

8.21.27 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`

8.21.28 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`

8.21.29 PSM3_MQ_RNDV_NIC_THRESH

Sets the eager-to-rendezvous switchover threshold in bytes for messages sent from a CPU buffer. Typically, rendezvous is used for larger messages. As outlined below, the defaults and usage depends on the selected HAL, see PSM3 Architecture and Hardware Abstraction Layer.

When the verbs HAL is selected (see PSM3 Verbs RDMA Modes and Rendezvous Module) and PSM3_RDMA=1, 2, or 3, rendezvous 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_RNDV_NIC_WINDOW.

When the sockets HAL is selected (see PSM3 Sockets Modes), by default rendezvous is only used for synchronous application messages and larger messages for GPU jobs (see PSM3_GPU_THRESH_RNDV). While rendezvous may be enabled for other messages via `PSM3_MQ_RNDV_NIC_THRESH`, due to sockets' lack of RDMA, it offers no benefits and is generally discouraged.

Options:

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

Default : `PSM3_MQ_RNDV_NIC_THRESH=64000` for verbs and disabled for sockets (`PSM3_MQ_RNDV_NIC_THRESH=4294967299`)

See PSM3 Verbs RDMA Modes and Rendezvous Module and PSM3 Sockets Modes for more details.

Also see PSM3_GPU_THRESH_RNDV.

8.21.30 PSM3_MQ_RNDV_NIC_WINDOW

This variable has been deprecated. Use PSM3_RNDV_NIC_WINDOW and PSM3_GPU_RNDV_NIC_WINDOW instead.

8.21.31 PSM3_MQ_RNDV_SHM_GPU_THRESH

Sets the threshold (in bytes) for shared memory eager-to-rendezvous switchover for messages sent from a GPU buffer.

Default is 127 when Intel GPU support is enabled (i.e., PSM3_ONEAPI_ZE is enabled). Default is 63 when NVIDIA GPU support is enabled (i.e., PSM3_CUDA is enabled).

NOTE

For platforms with GPU to GPU connectivity within a given server, rendezvous will take advantage of such connectivity resulting in direct GPU to GPU transfers. For platforms without GPU to GPU connectivity, larger values, such as 4096, may perform better.

NOTE

Messages larger than PSM3_MQ_RNDV_SHM_THRESH will always use rendezvous. Typically `PSM3_MQ_RNDV_SHM_GPU_THRESH` is smaller than PSM3_MQ_RNDV_SHM_THRESH.

8.21.32 PSM3_MQ_RNDV_SHM_THRESH

Sets the threshold (in bytes) for shared memory eager-to-rendezvous switchover. Threshold is applicable to CPU and GPU messages. See PSM3_MQ_RNDV_SHM_GPU_THRESH.

Default: `PSM3_MQ_RNDV_SHM_THRESH=16000`

8.21.33 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`

8.21.34 PSM3_MR_CACHE_MODE

Controls the user space MR cache as follows:

- `0` – No MR caching.
- `1` – Use rendezvous module for MR caching.
- `2` – User space 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_RNDV_NIC_WINDOW and PSM3_GPU_RNDV_NIC_WINDOW). When a user space MR cache is used, this table is implicitly built into the cache.

Use of the user space MR cache or 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 MR cache will retain some MRs after they are done with their current transfer, so they may be reused by future transfers. Such buffer reuse is common in many applications and in many implementations of middleware collective algorithms such as MPI_AllReduce.

In general, the rendezvous module can provide lower memory and CPU overhead for MR caching and has less risk of CPU jitter. The key difference being how stale MRs are detected and removed from the cache. The rendezvous module makes use of the kernel MMU notifier mechanism. This mechanism efficiently indicates when an application page is going to be freed. The user space MR cache makes use of the userfaultfd mechanism. This mechanism also provides an indication when an applicable page is being freed. However, to process these indications, PSM3 must establish an additional thread and maintain its own table of which address ranges

represent currently pinned MRs. This stems from the original design intent of userfaultfd, providing a mechanism for user space hypervisors to implement page fault handling for the virtual machines they manage.

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 PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3_MR_CACHE_SIZE, PSM3_MR_CACHE_SIZE_MB, and PSM3_RV_MR_CACHE_SIZE.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.

8.21.35 PSM3_MR_CACHE_SIZE

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

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

MR Cache Default: 16384.

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

See PSM3 Verbs RDMA Modes and Rendezvous Module and PSM3_MR_CACHE_MODE.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.

8.21.36 PSM3_MR_CACHE_SIZE_MB

Sets the maximum amount of CPU memory, in megabytes, to be pinned per process by the user space CPU MR cache.

Default: `1024`

Use of 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 user space MR cache will retain some MRs after they are done with their current transfer, so they may be reused by future transfers. Such buffer reuse is common in many applications and in many implementations of middleware collective algorithms such as MPI_AllReduce. For some applications, performance may be improved by growing this value, however values that result in `number_of_processes*PSM3_MR_CACHE_SIZE_MB` 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) * (the largest window size specified by PSM3_RNDV_NIC_WINDOW)

NOTE

This setting is only used when the verbs HAL is selected, PSM3_MR_CACHE_MODE selects user space MR cache mode (2), and PSM3_RDMA selects mode 2 or 3. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Rendezvous Kernel Module, PSM3_MR_CACHE_MODE, and PSM3_RV_GPU_CACHE_SIZE.

8.21.37 PSM3_MTU

Sets upper bound on maximum PSM3 payload per transmission during eager messages. As outlined below, the defaults and usage depends on the selected HAL. See PSM3 Architecture and Hardware Abstraction Layer.

For UD verbs (i.e., the verbs HAL is selected and PSM3_RDMA is 0 or 2, see PSM3 Verbs RDMA Modes and Rendezvous Module) and UDP (i.e., the sockets HAL is selected and PSM3_SOCKETS is 1, see PSM3 Sockets Modes), the maximum MTU is controlled by the NIC driver. In these cases, this variable may only be used to decrease the value selected by the driver. Input values larger than the largest PSM3 payload the NIC MTU permits will be ignored.

For RC verbs (i.e., the verbs HAL is selected and PSM3_RDMA is 3, see PSM3 Verbs RDMA Modes and Rendezvous Module), a value up to PSM3_MQ_RNDV_NIC_THRESH may be used. It may be larger (or smaller) than the NIC MTU.

For TCP (i.e., the sockets HAL is selected and PSM3_SOCKETS is 0, see PSM3 Sockets Modes), a value up to 262076 may be used. For TCP, the value specified may be larger (or smaller) than the NIC MTU.

Valid values are:

- <= 0 – Use the MTU selected by NIC driver (typically, 4096 minus PSM header size for verbs or 8196 minus UDP/IP and PSM header sizes for UDP or TCP sockets).
- 1 – 256
- 2 – 512
- 3 – 1024
- 4 – 2048
- 5 – 4096
- 6 – 8192 (This value is only valid for RC verbs or sockets.)
- 7 – 10240 (This value is only valid for RC verbs or TCP.)
- 256
- 512
- 1024
- 2048
- 4096

- `8192` (This value is only valid for RC verbs or sockets.)
- For RC verbs, any value from 1024 to PSM3_MQ_RNDV_NIC_THRESH may be specified.
- For TCP, any value from 256 to 262072 may be specified.

Default: -1 for UD/RC verbs and UDP, 65536 for TCP.

NOTE

Typically, larger values tend to perform better and reduce communications protocol overheads. However, larger values may increase memory consumption. See *Intel® Ethernet Fabric Performance Tuning Guide*.

NOTE

For UD verbs, values must be a power of 2 between 256 and 4096. 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. For RC verbs, values are rounded down to the nearest multiple of 64. The maximum payload for RDMA rendezvous is separately controlled via PSM3_RNDV_NIC_WINDOW and PSM3_GPU_RNDV_NIC_WINDOW. See PSM3 Verbs RDMA Modes and Rendezvous Module.

NOTE

Values less than 1024 may have insufficient room for PSM3 to issue connection establishment messages and may cause the job to fail to launch.

NOTE

Actual transmission sizes (i.e., packets for verbs or UDP) may be larger to accommodate space for PSM3 header information per transmission.

8.21.38 PSM3_MULTI_EP

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

NOTE

This parameter has been deprecated. It has been removed from PSM3_VERBOSE_ENV help text, and it is recommended that the default be used. Some middleware will require this be enabled (the default) to obtain the best performance.

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.

NOTE

This parameter's default may be overridden by the Intel® MPI Library. To see the actual value being used, set `PSM3_VERBOSE_ENV=1:` (see PSM3_VERBOSE_ENV). See Environment Variables for Intel® MPI Library Jobs for more information.

8.21.39 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:

- `-1` – No NIC autoselection nor multi-rail within PSM3. PSM3 will present all the unfiltered physical NICs. PSM3 will not present an `autoselect_one` fabric interface.
- `0` – Single NIC per process configuration. PSM3 will present an `autoselect_one` fabric interface as the default as well as all the unfiltered physical NICs. If a process opens the `autoselect_one` fabric interface, PSM3 will select a NIC based on PSM3_NIC_SELECTION_ALG.
- `1` – Enable multi-rail capability and each process will use all available NIC(s) in the system. PSM3 will only present an `autoselect` fabric interface.
- `2` – Enable multi-rail capability and limit NIC(s) used by a given process to NIC(s) within a single NUMA socket. PSM3 will only present an `autoselect` fabric interface.

  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.
- `3` – Enable multi-rail capability and limit NIC(s) used by a given process to NIC(s) within a single NUMA socket and equally close to the process's GPU. PSM3 will only present an `autoselect` fabric interface.

  PSM3 looks for at least one available NIC in the same NUMA socket on which you pin the task and then further limits selection to those NICs equally close to the process's GPU. If no such NUMA local NICs are found, PSM3 will simply limit selection to those NICs equally close to the process's GPU. If the job is not using a

GPU and no NUMA local 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. The GPU aspects of this option are only applicable for GPU jobs with PSM3_CUDA=1 or PSM3_ONEAPI_ZE=1.

- `4` – Enable multi-rail capability and limit NIC(s) used by a given process to NIC(s) closest to the process's GPU. PSM3 will only present an `autoselect` fabric interface.

  PSM3 looks for the NIC equally close to the process's GPU. If multiple such NICs are found, PSM3 further limits selection to those in the same NUMA socket as the process. If none of the NICs are in the same NUMA socket, only GPU closeness is considered. 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. This option is only available when PSM3_CUDA=1 or PSM3_ONEAPI_ZE=1.

Default: `0`

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

NOTE

PSM3_QP_PER_NIC QPs are created per address within each rail. Care must be taken when combining PSM3_MULTIRAIL, PSM3_QP_PER_NIC, and PSM3_ADDR_PER_NIC as the PSM3 limit of thirty-two (32) QPs per endpoint can be easily exceeded.

NOTE

Some multi-GPU platforms are designed with PCIe Switches to allow NICs to be placed close to each GPU. This can help optimize GPU communications performance. On such platforms, the `PSM3_MULTRAIL=4` option may offer the best performance. On multi-GPU platforms without PCIe switches, the `PSM3_MULTIRAIL=3` option may offer the best performance.

NOTE

When using NVIDIA GPUs, if more than 1 GPU is visible to the current process, at the time of PSM3 endpoint initialization PSM3 will identify the GPU being used by the process via `cuCtxGetCurrent` and `cuCtxGetDevice`. To limit the GPUs visible to a given process, the NVIDIA environment variable `CUDA_VISIBLE_DEVICES` can be exported with a list of GPU device numbers.

For more detail on this feature, see PSM3 Multi-Rail Support. See also PSM3_MULTIRAIL_MAP and PSM3_ALLOW_ROUTERS.

8.21.40 PSM3_MULTIRAIL_MAP

Tells PSM3 which NIC and address(es) to use for each rail. If only one rail is specified, it is equivalent to a single-rail use 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 -1, 0, 3 or 4. For details, see PSM3 Multi-Rail Support.

Options: `rail,rail,rail,...;rail,rail,rail,...`, where *rail* can be `unit-addr_index` or simply `unit`. When an addr_index is not specified for a given rail, it defaults to `all` and the rail load balances across all PSM3_ADDR_PER_NIC addresses. Multiple rail specifications are separated by commas. Multiple sets of rails are separated by semicolons. In some cases, extraneous whitespace may cause parse errors, so whitespace should be avoided.

The `unit` may be specified as an explicit RDMA or sockets device name (as shown in `ibv_devices` or `ifconfig`) or a Device Unit number. When a unit number is specified, it is relative to the alphabetic sort of the RDMA or sockets device names applicable to the selected HAL. Unit 0 is the first name.

The `addr_index` may be specified as follows:

- `integer` - The integer may be from 0 to PSM3_ADDR_PER_NIC-1. This will select a specific address within the selected unit.
- `all` - The given rail will load balance across all PSM3_ADDR_PER_NIC addresses.
- `any` - PSM3 will select a single address among the PSM3_ADDR_PER_NIC addresses for the selected unit. Each process on a given server makes its own selection.

If only one set of rails is specified, it will be used for all processes. If more than one set of rails is specified, PSM3_MULTIRAIL controls how the set for a given process is chosen:

- `PSM3_MULTIRAIL=1` - The local rank number of the process is used to select the set of rails. Local rank 0 will use the first set, local rank 1 will use the second set, etc.
- `PSM3_MULTIRAIL=2` - The local CPU NUMA of the process is used to select the set of rails. All processes on CPU NUMA 0 will use the first set, all processes on CPU NUMA 1 will use the second set, etc.

Some example uses of `PSM3_MULTIRAIL_MAP`:

- `PSM3_MULTIRAIL_MAP=irdma0,irdma1` - Load balance across all PSM3_ADDR_PER_NIC addresses on `irdma0` and `irdma1` NICs. This is equivalent to `PSM3_MULTIRAIL_MAP=irdma0-all,irdma1-all`.
- `PSM3_MULTIRAIL_MAP=irdma0,irdma1-0` - Load balance across all PSM3_ADDR_PER_NIC addresses on `irdma0` and the first unfiltered address on `irdma1`.
- `PSM3_MULTIRAIL_MAP=irdma0-0,irdma1-0` - Load balance across the first unfiltered address on `irdma0` and the first unfiltered address on `irdma1`. This is equivalent to `PSM3_MULTIRAIL_MAP=irdma0,irdma1` with `with PSM3_ADDR_PER_NIC=1`.

- `PSM3_MULTIRAIL_MAP=irdma0-any,irdma1-any` - Load balance across a single PSM3 selected address on each NIC.
- `PSM3_MULTIRAIL=1 PSM3_MULTIRAIL_MAP=irdma0;irdma1;irdma0;irdma1` - Local ranks 0 and 2 will use `irdma0` and local ranks 1 and 3 will use `irdma1`.
- `PSM3_MULTIRAIL=2 PSM3_MULTIRAIL_MAP=irdma0;irdma1` - All processes on CPU NUMA 0 will use `irdma0` and all processes on CPU NUMA 1 will use `irdma1`.

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=irdma0,irdma0,irdma1`, a total of four QPs will be created on NIC irdma0 and two QPs on NIC irdma1. They will be scheduled as NIC irdma0 first QP, NIC irdma0 third QP, NIC irdma1 first QP, NIC irdma0 second QP, NIC irdma0 fourth QP, NIC irdma1 second QP.

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

- `PSM3_MULTIRAIL_MAP=irdma0,irdma0` and `PSM3_QP_PER_NIC=1` is equivalent to `PSM3_MULTIRAIL_MAP=irdma0` and `PSM3_QP_PER_NIC=2`.
- `PSM3_MULTIRAIL_MAP=irdma0,irdma1,irdma0,irdma1` and `PSM3_QP_PER_NIC=1` is equivalent to `PSM3_MULTIRAIL_MAP=irdma0,irdma1` and `PSM3_QP_PER_NIC=2`.

NOTE

The first `rail` specified in a given set of rails 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 NICs selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY.

NOTE

The specification of a unique set of rails per local rank or per CPU NUMA is typically only required in asymmmetric or atypical platforms with a high NIC count, such as some GPU platforms when only a subset of NICs are installed. In most environments, automatic NIC selection via PSM3_MULTIRAIL (without using `PSM3_MULTIRAIL_MAP`), PSM3_NIC_SELECTION_ALG (when only 1 NIC per process or thread is desired), or selection of a uniform set of NICs for all processes (via a single set of rails in `PSM3_MULTIRAIL_MAP`) will be sufficient. See Multi-Rail Configuration Examples.

NOTE

The PSM3 implementation has a limit of 32 NICs per node and 32 QPs per endpoint.

NOTE

PSM3_QP_PER_NIC QPs are created per address within each rail. Care must be taken when combining multi-rail, PSM3_QP_PER_NIC, and PSM3_ADDR_PER_NIC as the PSM3 limit of 32 QPs per endpoint can be easily exceeded.

NOTE

If one or more of the units selected have been filtered out or do not exist, it is considered a fatal error. See NIC and Address Filtering.

NOTE

Specification of units by name is recommended. Support for specification by Device Unit number may be removed in a future PSM3 release. Be aware that the unit number of a given hardware NIC is often different within each HAL.

8.21.41 PSM3_NIC

Specifies the RDMA and/or sockets device name(s) (as shown in `ibv_devices` or `ifconfig`) or a Device Unit number. When a device name is specified, it may be specified explicitly or as an extended glob pattern. When a unit number is specified, it is relative to the alphabetic sort of the RDMA or sockets device names applicable to the corresponding HAL, see PSM3 Architecture and Hardware Abstraction Layer. Unit 0 is the first name.

Default: `PSM3_NIC=any`

NOTE

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

Some example uses:

- `PSM3_NIC=irdma1` – Select `irdma1`.
- `PSM3_NIC=irdma*` – Select all NICs whose name starts with `irdma`, such as `irdma0`, `irdma1`, and so on. On typical systems, this would consider only Intel® Ethernet RDMA NICs.
- `PSM3_NIC=irdma[01]` – Select `irdma0` and `irdma1` NICs, if present. Other NICs, such as `irdma2`, will not be considered.
- `PSM3_NIC=+(irdma1|eth1)` – Select `irdma1` and/or `eth1` NICs, if present. Other NICs, such as `irdma0`, will not be considered.
- `PSM3_NIC=+(irdma*|eth*)` – Select all NICs whose name starts with `irdma` or `eth`, such as `irdma0`, `irdma1`, `eth0`, `eth1`, and so on.
- `PSM3_NIC=0` – Select the first RDMA or sockets NIC (alphabetically by name within `ibv_devices` or `ifconfig`) found in the system.

When `PSM3_NIC=any` or `PSM3_NIC` is a pattern that matches more than one NIC, which passes all the other filter criteria for the selected HAL (see NIC and Address Filtering), the NIC for each process is selected based on PSM3_MULTIRAIL and PSM3_NIC_SELECTION_ALG.

NOTE

As shown in examples such as `PSM3_NIC=+(irdma1|eth1)`, `PSM3_NIC` can be defined such that it will match NICs in more than one HAL. In this example, `irdma1` may be an RDMA NIC considered while evaluating applicable NICs for use by the verbs HAL, while `eth1` may be a sockets NIC considered only while evaluating applicable NICs for use by the sockets HAL. While powerful, such selections must be used with care as some processes may end up using `irdma1` and the verbs HAL while others may end up using `eth1` and the sockets HAL, which will result in job launch errors. Such a situation may occur if the `irdma1` NIC is down or not found on some of the nodes in the job. However, a powerful use case may be that if the job is run on a set of nodes that all lack an `irdma1` device, `eth1` may be used, allowing the job to run, albeit at lower performance due to the use of sockets versus verbs.

NOTE

PSM3 detects all RDMA and sockets devices, so if multiple types of RDMA or sockets 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 32 NICs per node.

NOTE

When PSM3_MULTIRAIL is 1 or 2, `PSM3_NIC` is still processed. In this case, it should be set to `any` or a pattern which will match more than one NIC in the given HAL.

NOTE

Specification of NICs by RDMA and/or sockets device name or pattern is recommended as opposed to specification by unit number. Support for specification by Device Unit number may be removed in a future PSM3 release. Be aware that the unit number of a given hardware NIC is often different within each HAL.

NOTE

If the NIC selected has been filtered out or does not exist, PSM3 will report that it has no units, and the middleware may select a different provider or transport protocol, such as TCP/IP. See NIC and Address Filtering.

NOTE

The HAL, NIC(s), and addresses selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the HAL, NIC, and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

8.21.42 PSM3_NIC_SELECTION_ALG

Specifies the algorithm to use for selecting the NIC per process when NIC and Address Filtering selects more than one NIC and PSM3_MULTIRAIL is 0.

For each of the algorithms below, only NICs that pass all the filtering criteria will be considered. See NIC and Address Filtering.

Options:

- `RoundRobin` or `rr` – Selects a NIC on the same NUMA socket as the process. If more than one NIC is on the process' NUMA socket, those NICs will be distributed evenly across all processes on the given NUMA socket. Processes that have no NIC on their NUMA socket will simply be distributed across all available NICs.
- `Packed` or `p` – All processes will use the first available NIC.
- `RoundRobinAll` or `rra` – The processes will be distributed across all available NICs without considering NUMA locality of the NIC versus process.
- `CpuRoundRobin` or `crr` – Selects a NIC on the same NUMA socket as the process. If more than one NIC is on the process's NUMA socket, the NIC closest to the process's GPU is selected. If more than one NIC passes this criteria, those NICs will be distributed evenly across all processes on the given NUMA socket and GPU. Processes that have no NIC on their NUMA socket will simply be distributed across all available NICs which are equally close to the process's GPU. The GPU aspects of this option are only applicable for GPU jobs with PSM3_CUDA=1 or PSM3_ONEAPI_ZE=1.
- `GpuRoundRobin` or `grr` – Selects a NIC closest to the process's GPU. If more than one NIC is close to the process's GPU, the NIC on the process's NUMA socket is selected. If more than one NIC passes this criteria, those NICs will be distributed evenly across all processes on the GPU and given NUMA socket. This option is only available when PSM3_CUDA=1 or PSM3_ONEAPI_ZE=1.

Default:

- If all NICs are on subnets that can access each other (see PSM3_ALLOW_ROUTERS) - `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

The Intel PSM3 implementation has a limit of 32 NICs per node.

NOTE

The NIC(s) and addresses selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the NIC and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

NOTE

Some multi-GPU platforms are designed with PCIe Switches to allow NICs to be placed close to each GPU. This can help optimize GPU communications performance. On such platforms, the `PSM3_NIC_SELECTION_ALG=GpuRoundRobin` option may offer the best performance. On multi-GPU platforms without PCIe switches, the `PSM3_NIC_SELECTION_ALG=CpuRoundRobin` option may offer the best performance.

NOTE

When using NVIDIA GPUs, if more than 1 GPU is visible to the current process, at the time of PSM3 endpoint initialization PSM3 will identify the GPU being used by the process via `cuCtxGetCurrent` and `cuCtxGetDevice`. To limit the GPUs visible to a given process, the NVIDIA environment variable `CUDA_VISIBLE_DEVICES` can be exported with a list of GPU device numbers.

8.21.43 PSM3_NIC_SPEED

Specifies the links speed(s) to consider. NICs that do not match the specified speed(s) will be filtered out and not considered for use. The speed may be specified as `max`, `any`, as an explicit speed, or as an extended glob pattern for comparison to each NIC's speed. Speed is specified in units of megabits per second (Mbps). Filtering by speed is applied after port status, PSM3_NIC, PSM3_ADDR_FMT, and PSM3_SUBNETS have potentially filtered out some of the NICs. As such, the `PSM3_NIC_SPEED=max` setting is only applied among NICs not otherwise filtered (see NIC and Address Filtering).

Default: `PSM3_NIC_SPEED=max`.

NOTE

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

Some example uses:

- `PSM3_NIC_SPEED=max` – Examine all NICs within the given HAL that are not otherwise filtered out from consideration and identify the fastest speed among them. Then consider only NICs that match that speed.
- `PSM3_NIC_SPEED=any` – Ignore speed when selecting a NIC.

- `PSM3_NIC_SPEED=100000` – Consider only NICs whose links are running at 100 Gbps (100,000 Mbps).
- `PSM3_NIC_SPEED=+(100000|200000)` – Consider only NICs whose links are running at 100 or 200 Gbps (100,000 or 200,000 Mbps).
- `PSM3_NIC_SPEED=*00000` – Consider only NICs whose links are running at ≥100 Gbps (≥ 100,000 Mbps).

This is just one of the filters applied to select a NIC. See NIC and Address Filtering for more information.

When `PSM3_NIC_SPEED` matches more than one NIC within the given HAL that passes all the other filter criteria (see NIC and Address Filtering), the NIC for each process is selected based on PSM3_MULTIRAIL and PSM3_NIC_SELECTION_ALG.

NOTE

Even if a NIC is filtered out due to its current speed, it is still assigned a unit number based on an alphabetic sort by name among the NICs supported by a given HAL. As such, unit numbers remain constant within a given HAL regardless of which NICs have been filtered out. These unit numbers may be used in environment variables such as PSM3_NIC and PSM3_MULTIRAIL_MAP. However, those variables must select a unit that has not been filtered out.

NOTE

PSM3 detects all RDMA and sockets devices, so if multiple types of RDMA or sockets 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

If the none of the NICs match the selected speed or all have been filtered out, PSM3 will report it has no units and the middleware may select a different provider or transport protocol, such as TCP/IP. See NIC and Address Filtering.

NOTE

The HAL, NIC(s), and addresses selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the HAL, NIC, and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

8.21.44 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 PSM3 Verbs RDMA Modes and Rendezvous Module for more details.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer.

8.21.45 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) that 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 PSM3 Verbs RDMA Modes and Rendezvous Module for more details.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer.

8.21.46 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 PSM3 Verbs RDMA Modes and Rendezvous Module for more details.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer.

8.21.47 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 PSM3 Verbs RDMA Modes and Rendezvous Module for more details.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer.

8.21.48 PSM3_ONEAPI_ZE

Enables Intel GPU Support in PSM3 when set. Requires the PSM3 provider to be compiled with ONEAPI_ZE support.

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

NOTE

If GPU buffers are used in the workloads and `PSM3_ONEAPI_ZE` is not set to 1, undefined behavior will result.

Default: `PSM3_ONEAPI_ZE=0`

See also: PSM3_GPUDIRECT

NOTE

A given PSM3 library build and a given kernel rendezvous module can only support one vendor's GPUs.

8.21.49 PSM3_PRINT_STATS

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

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_PRINT_STATS_PREFIX`psm3-perf-stat-[hostname]-pid-[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.

Options:

- `0` – Disables statistics output.

- `-1` – Performs statistics output once at the successful completion of a job upon the first close of a PSM3 endpoint.
- `>0` – Specifies the frequency of statistics output in seconds.

Default: `0`

After the numeric value, an optional colon may be specified, optionally followed by a pattern. This behaves as follows:

- `value` – Specified statistics output frequency is enabled on all processes.
- `value:` – Specified statistics output frequency is enabled only on rank 0 (abbreviation for `PSM3_PRINT_STATS=value:*:rank0`). All other processes will use the default of 0 (disabled).
- `value:pattern` – Specified statistics output frequency is enabled only on processes whose label matches the extended glob pattern. All other processes will use the default of 0 (disabled).

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 `#` is the relative process number in the job. If the MPI runtime and the job scheduler have not indicated the rank to PSM3, the label will be of the form `hostname:pid#`, where `#` 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_PRINT_STATS=1:*:rank0` – Enable statistics output at a frequency of once a second for rank 0. When the user only needs to see statistics output for one process, this can provide a more concise output.
- `PSM3_PRINT_STATS=1:myhost047:*` - Enable statistics output at a frequency of once a second on myhost047. All processes on that host will provide statistics output. This can be helpful if only a single host's behavior is suspect.
- `PSM3_PRINT_STATS=1:+(*:rank0|*:rank1)` – Enable statistics output at a frequency of once a second for 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.

For more information see PSM3 Performance Statistics.

NOTE

Help text describing the statistics can be generated by using PSM3_PRINT_STATS_HELP.

8.21.50 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 group of statistics:

- `0x000001` – High level message passing statistics
- `0x000002` – oneAPI Level Zero call statistics
- `0x000002` – CUDA call statistics
- `0x000040` – Process launch information including command line, full environment, each `PSM3_` or `FI_PSM3_` setting as parsed (even if PSM3_VERBOSE_ENV is not set), and `PSM3_IDENTIFY` information (even if PSM3_IDENTIFY is not set).
- `0x000100` – Receive progress thread statistics
- `0x000200` – Inter-node "nic" 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`

For more information, see PSM3 Performance Statistics.

NOTE

Help text describing the statistics and the mask value for each statistics group can be generated by using PSM3_PRINT_STATS_HELP.

8.21.51 PSM3_PRINT_STATS_HELP

Generates a help file on rank 0 describing the statistics available with PSM3_PRINT_STATS.

Options:

- `0` – Disables statistics help output.
- `1` – Performs statistics help output on rank 0.

Default: `0`

When enabled, statistics help is output by rank 0 in a job to a local file whose name is of the form: PSM3_PRINT_STATS_PREFIX`psm3-perf-stat-help-[hostname]-pid-[pid]`. The help includes a description of each statistic group, its mask bit in PSM3_PRINT_STATSMASK, and a description of each statistic.

For more information, see PSM3 Performance Statistics.

NOTE

The list of statistics shown in the help may be affected by options such as PSM3_DEVICES, PSM3_HAL and PSM3_RDMA. Therefore, Intel recommends that you run a job with the desired options so the appropriate statistics help text is included.

8.21.52 PSM3_PRINT_STATS_PREFIX

Selects the prefix for the filenames used for PSM3_PRINT_STATS and PSM3_PRINT_STATS_HELP output. This may be used to control both the directory and a prefix to the filename in the directory.

Default: `./`

Examples:

- `PSM3_PRINT_STATS_PREFIX=logs/` – put statistics and statistics help output in the `logs` directory
- `PSM3_PRINT_STATS_PREFIX=logs/myrun` – put statistics and statistics help output in the `logs` directory with `myrun` at the start of every statistics and statistics help filename
- `PSM3_PRINT_STATS_PREFIX=/home/myuser/logs/myrun` – put statistics and statistics help output in the `/home/myuser/logs` directory with `myrun` at the start of every statistics and statistics help filename

For more information, see PSM3 Performance Statistics.

NOTE

Using a unique value for `PSM3_PRINT_STATS_PREFIX` per job can make it easier to organize and separate the statistics output from various jobs.

NOTE

Using `PSM3_PRINT_STATS_PREFIX` to select a directory within a shared filesystem mounted by all hosts can simplify subsequent management and analysis of statistics output.

8.21.53 PSM3_QP_PER_NIC

Sets the number of user space queues (e.g., UD QPs or sockets) per local endpoint. In addition for the verbs HAL, when using 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 queues may increase performance, especially for large messages. However, this represents a multiplier to the amount of communications memory needed as each queue requires its own set of resources (e.g., transport state, WQEs, receive buffers, etc.).

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

NOTE

The Intel PSM3 implementation has a limit of 32 NICs per node and 32 queues per endpoint.

NOTE

`PSM3_QP_PER_NIC` QPs are created per address within each rail. Care must be taken when combining PSM3_MULTIRAIL, PSM3_QP_PER_NIC and PSM3_ADDR_PER_NIC as the PSM3 limit of 32 QPs per endpoint can be easily exceeded.

NOTE

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

For more details, see PSM3 Verbs RDMA Modes and Rendezvous Module and PSM3 Multi-Rail Support.

8.21.54 PSM3_QP_RETRY

Sets the retry limit for user space 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.

NOTE

This setting is only used when the verbs HAL is selected and PSM3_RDMA is `2 or 3`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.

8.21.55 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.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.

8.21.56 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.

8.21.57 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

continued...

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 this 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 that infrequently call into MPI. However, if run too frequently, the receiver thread can introduce host CPU jitter and overheads, especially for applications that 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.

8.21.58 PSM3_RDMA

Controls the use of RC QPs and RDMA when the verbs Hardware Abstraction Layer (HAL) is selected.

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: `1` when rendezvous module is available or `PSM3_GPUDIRECT` is enabled. Default is 0 when rendezvous module is unavailable.

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 PSM3 Verbs 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.

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.

NOTE

PSM3 build options control which HALs are included in the PSM3 binary as well as which data movement protocols are available within each included HAL. See Building the PSM3 RPM.

8.21.59 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 the number of simultaneous sends and may be less than this value. See also PSM3_NUM_SEND_RDMA.

If a sender needs an 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`

NOTE

This setting is only used when the verbs HAL is selected and RDMA is enabled (e.g., PSM3_RDMA is `1`, `2`, or `3`). Otherwise it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.

8.21.60 PSM3_RECONNECT_TIMEOUT

Sets the RC QP minimum re-connection timeout in seconds for PSM3_RDMA mode 1, 2 or 3. If a value of 0 is specified, then the RC QP connection recovery feature is disabled for the job.

Default is 30 for PSM3_RDMA mode 1. Default is 0 for PSM3_RDMA mode 2 or 3.

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_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.

When using PSM3_RDMA mode 1, all jobs that are using the same FI_PSM3_UUID (as defaulted, set explicitly, or set via the MPI middleware) must use the same timeout.

When enabled and using PSM3_RDMA mode 1, the re-connect timeout selected must be set larger than PSM3_RV_HEARTBEAT_INTERVAL.

NOTE

This setting is only used when the verbs HAL is selected and RC QPs are used for RDMA (i.e., PSM3_RDMA is 1, 2 or 3). Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, and PSM3 Rendezvous Kernel Module.

NOTE

The environment variable is a new name for PSM3_RV_RECONNECT_TIMEOUT, which has been deprecated.

8.21.61 PSM3_RNDV_NIC_WINDOW

Sets the window size in bytes for messages from and to CPU buffers. This controls how large messages are split for transmission.

Larger window sizes may reduce CPU loading. Smaller window sizes may provide better distribution of bandwidth in workloads with many simultaneous destinations like an MPI collective operation, but will slightly increase CPU loading.

When PSM3_MULTIRAIL is active or PSM3_QP_PER_NIC is > 1 or PSM3_RV_QP_PER_CONN is > 1, this value controls the granularity at which messages are striped across multiple NICs and/or QPs, respectively.

Specified as a list of the form:
`window_size:limit,window_size:limit,window_size:limit,...`, where `window_size` selects the actual transmission size as a value between 1 and 4194304 bytes inclusive and `limit` specifies the largest sized message which will use the given `window_size`. `limit` is specified as a value between 1 and 4294967295 inclusive, where 4294967295 can also be specified as `max`.

Some example uses of `PSM3_RNDV_NIC_WINDOW`:

- `PSM3_RNDV_NIC_WINDOW=131072` - A transmission size of 131072 is used for all message sizes. This is equivalent to `PSM3_RNDV_NIC_WINDOW=131072:4294967295` or `PSM3_RNDV_NIC_WINDOW=131072:max`.
- `PSM3_RNDV_NIC_WINDOW=131072:524287,262144:1048575,524288` - A transmission size of 131072 is used for message sizes up to 524287 bytes. Messages of 524288 to 1048575 bytes will use a transmission size of 262144 bytes and messages 1048576 bytes or larger will use a transmission size of 524288 bytes.

The `limit` specified for a given entry in the list, must be larger than the `limit` for the prior entry. When a `limit` is not specified for a given entry, it defaults to `max` (4294967295). For the last `window_size` in the list, a `limit` of `max` (4294967295) is always used and need not be specified. Multiple `window_size` specifications are separated by commas. A trailing comma will be ignored. In some cases, extraneous whitespace may cause parse errors, so whitespace should be avoided.

Default: `PSM3_RNDV_NIC_WINDOW=131072`.

NOTE

Each `window_size` specified will rounded up to be a multiple of the CPU page size.

Also see PSM3_GPU_RNDV_NIC_WINDOW.

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

8.21.62 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)

8.21.63 PSM3_RV_FR_PAGE_LIST_LEN

Sets the fast-registration (FR) page list length for each pre-allocated memory region in the FR MR cache of the rendezvous module for PSM3_RDMA mode `1`. If a value of `0` is specified, then the rendezvous module's `fr_page_list_len` parameter controls the selection.

Default: `1024`

For some applications, use of larger `PSM3_RV_FR_PAGE_LIST_LEN` may increase performance for large messages. For jobs with small messages or running with many ranks per node, this parameter should be set to a lower value to save NIC resources.

All jobs that are using the same FI_PSM3_UUID (as defaulted, set explicitly, or set via the MPI middleware) must use the same page list length.

NOTE

This setting is only used when the verbs HAL is selected and the rendezvous module is used for RDMA (i.e., PSM3_RDMA is `1`). Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, and PSM3 Rendezvous Kernel Module.

8.21.64 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 Direct GPU RDMA or GPUDirect RDMA is enabled, (e.g., PSM3_GPUDIRECT is enabled, the verbs HAL is selected and PSM3_RDMA mode >=1), the cache is used for Direct GPU Copy, Direct GPU Send DMA, Direct GPU RDMA, GPUDirect Copy, GPUDirect Send DMA, and/or 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 Direct GPU access or GPUDirect is enabled without Direct GPU RDMA or GPUDirect RDMA, (e.g., PSM3_GPUDIRECT is enabled, the verbs HAL is selected, and PSM3_RDMA mode is 0), the cache is used for Direct GPU Copy, Direct GPU Send DMA, GPUDirect Copy, and/or 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).
- When Direct GPU access or GPUDirect is enabled for sockets, (e.g., PSM3_GPUDIRECT is enabled and the sockets HAL is selected), the cache is used for Direct GPU Copy or GPUDirect Copy only. 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 that result in `number_of_processes*PSM3_RV_GPU_CACHE_SIZE` near or exceeding the total GPU memory (or the limit that 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 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) * (the largest window size specified by PSM3_GPU_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 PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Sockets Modes, PSM3 Rendezvous Kernel Module, PSM3 and Intel GPU Support, PSM3 and NVIDIA CUDA Support, and PSM3_RV_MR_CACHE_SIZE.

NOTE

This setting is only used when PSM3_GPUDIRECT is enabled.

NOTE

A given PSM3 library build and a given kernel rendezvous module can only support one vendor's GPUs.

8.21.65 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 that 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 that 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_RECONNECT_TIMEOUT.

NOTE

This setting is only used when the verbs HAL is selected and the rendezvous module is used for RDMA (i.e., PSM3_RDMA is `1`). Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, and PSM3 Rendezvous Kernel Module.

8.21.66 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 have completed 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 that 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) * (the largest window size specified by PSM3_RNDV_NIC_WINDOW)

NOTE

This setting is only used when the verbs HAL is selected and the rendezvous module is used for MR caching in PSM3_RDMA `1`, `2` or `3` mode. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, PSM3 Rendezvous Kernel Module, PSM3_MR_CACHE_MODE, and PSM3_RV_GPU_CACHE_SIZE.

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

NOTE

This setting is only used when the verbs HAL is selected and the rendezvous module is used for RDMA (e.g., PSM3_RDMA is `1`). Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, and PSM3 Rendezvous Kernel Module.

8.21.68 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 that 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.

NOTE

This setting is only used when the verbs HAL is selected and the rendezvous module is used for RDMA (i.e., PSM3_RDMA is `1`). Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer, PSM3 Verbs RDMA Modes and Rendezvous Module, and PSM3 Rendezvous Kernel Module.

8.21.69 PSM3_RV_RECONNECT_TIMEOUT

This variable has been deprecated. Use PSM3_RECONNECT_TIMEOUT on page 145 instead.

8.21.70 PSM3_SEND_REAP_THRESH

Sets the number of outstanding send WQEs before reaping CQEs.

Default: `256`

NOTE

This setting is only used when the verbs HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Verbs RDMA Modes and Rendezvous Module.

8.21.71 PSM3_SOCKETS

Controls the data movement mechanism used when the sockets Hardware Abstraction Layer (HAL) is selected.

Options:

- `0` – Use TCP/IP (with a few special situations during disconnect where UDP/IP may be used).
- `1` – Use UDP/IP.

Default: `0`

In all modes, a UDP socket is created per endpoint.

As the job size and number of processes (ranks) per node increases, the memory required for these sockets can be significant, especially for mode 0.

NOTE

To get good performance out of mode 1 (UDP/IP), DCB/PFC must be enabled in the network (similar to how RDMA protocols require DCB/PFC).

NOTE

This setting is only used when the sockets HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

NOTE

PSM3 build options control which HALs are included in the PSM3 binary as well as which data movement protocols are available within each included HAL. See Building the PSM3 RPM.

8.21.72 PSM3_SUBNETS

Specifies a comma-separated list of subnets which will be considered. Addresses that do not match any of the entries will not be considered for use by PSM3. Each entry in the list is an extended glob pattern. For Ethernet ports, the pattern must match the entire subnet (in the Classless Inter-Domain Routing (CIDR) notation). For InfiniBand ports, the pattern must match the InfiniBand subnet prefix shown as 16 hexidecimal digits with a leading `0x`. Leading and trailing whitespace in the pattern will be ignored. This environment variable is most useful to filter and select subnets when ports have more than one valid address or there is more than one NIC per server. The filtering is applied after PSM3_ADDR_FMT has potentially filtered out which types of addresses should be considered.

Options: `pattern, pattern, pattern`

A given pattern may be preceded with ^, in which case subnets that match the pattern are excluded. Comparison of a given subnet against the list is performed in the order of the entries, and comparison completes on first match.

Default: `PSM3_SUBNETS=^fe[89ab]?:*/*,*`

NOTE

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

Some example uses:

- `PSM3_SUBNETS=^fe[89ab]?:*/*,*` – IPv6 link-local subnets are excluded (`^fe[89ab]?:*/*`); all other subnets (`*`) are allowed.
- `PSM3_SUBNETS=^fe[89ab]?:*/*,192.168.100.0/24,fd*` – IPv6 link-local subnets are excluded; the IPv4 subnet `192.168.100.0/24` is allowed as well as any IPv6 subnet starting with `fd` (such as `fd57:1234::/64`).
- `PSM3_SUBNETS=^fe[89ab]?:*/*,^192.168.100.0/24,192.168.200.0/24,fd*` – IPv6 link-local subnets are excluded; the IPv4 subnet `192.168.100.0/24` is excluded; the IPv4 subnet `192.168.200.0/24` is allowed as well as any IPv6 subnet starting with `fd` (such as `fd57:1234::/64`).
- `PSM3_SUBNETS=^fe[89ab]?:*/*,^192.168.100.0/24,192.168.1??.0/24` – IPv6 link-local subnets are excluded; the IPv4 subnet `192.168.100.0/24` is excluded; any other IPv4 subnet matching `192.168.1??.0/24` is allowed (`192.168.101.0/24`, `192.168.102.0/24`, etc).
- `PSM3_SUBNETS=^fe[89ab]?:*/*,^0xfe80000000001000,*` – IPv6 link-local subnets are excluded; the InfiniBand subnet prefix `0xfe80000000001000` is excluded; all other subnets are allowed.

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

The IPv6 standards define a IPv6 link-local subnet as one whose top 10 bits are `1111 1110 10`. A link-local address' subnet prefix length may be 10 or more (these 10 bits may not be used to start any other type of IPv6 address), as such `fe8?::/*`, `fe9?::/*`, `fea?::/*` and `feb?::/*` are all link-local subnets. Typically, all ethernet ports are assigned a default link local IPv6 address, and by default the link-local address may have limited connectivity. As such, it is desirable to exclude this subnet from consideration in the PSM3 default.

This is just one of the filters applied to select a NIC and address within a NIC. See NIC and Address Filtering for more information.

NOTE

Even if a NIC is filtered out due to lack of an address with a matching subnet, it is still assigned a unit number based on an alphabetic sort by name among the NICs supported by a given HAL. Unit numbers remain constant within a given HAL regardless of which NICs have been filtered out. Unit numbers may be used in environment variables such as PSM3_NIC and PSM3_MULTIRAIL_MAP. However, those variables must select a unit that has not been filtered out.

NOTE

Addresses are considered in the order the GIDs are shown in `ibv_devinfo -v`. In the `ibv_devinfo -v` output, IPv4 addresses appear of the form `0000:0000:0000:0000:0000:ffff:`*`xxxx:xxxx`* where *`xxxx:xxxx`* is the 32-bit IPv4 address. All other forms are treated as IPv6 addresses. However, the `PSM3_SUBNETS` variable will use the CIDR notation to represent all IPv4 and IPv6 addresses.

NOTE

In CIDR notation, IPv4 addresses will be shown in decimal with dots, such as `192.168.100.0/24`, while IPv6 addresses will be shown in hexadecimal with colons, such as `fd57:1234::/64`. InfiniBand addresses are purposely shown in a unique format with a leading `0x`. Due to these formatting differences, a typical pattern will only match one type of address format.

NOTE

The NIC(s) and addresses selected for each process in a given job can be displayed at job start by enabling PSM3_IDENTIFY. Further details about the NIC and address selection process can be shown by enabling bit `0x2` in PSM3_TRACEMASK. See PSM3_TRACEMASK for more details.

8.21.73 PSM3_TCP_BIND_SRC

Indicates whether to bind to the source address before connecting. Useful when you must connect from source address.

Options:

- `0` – Do not bind to source address before connecting.
- `1` – Bind to source address before connecting.

Default: `1`

NOTE

This setting is only used when the sockets HAL is selected and PSM3_SOCKETS is `0`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.74 PSM3_TCP_PORT_RANGE

Sets the range of TCP port numbers that PSM3 will use to listen for incoming PSM3 TCP connections. The syntax is:

```
PSM3_TCP_PORT_RANGE=low:high
```

Default: `PSM3_TCP_PORT_HIGH_RANGE=0:0`

When `PSM3_TCP_PORT_HIGH_RANGE=0:0`, PSM3 will let the OS pick a TCP port number per PSM3 queue. When a non-zero range is specified, PSM3 will randomly select a unique listener port per PSM3 queue within the specified range (inclusive).

NOTE

When a range is specified, it must be large enough to accommodate a unique listener port number for every queue, in every endpoint, in every rail, in every PSM3 process, within the given host in the job. There is a limit of 32 queues per process so a range of `32 * processes_per_node` could be large enough for all possible configurations of a given job. See PSM3 Multi-Rail Support, PSM3 Multi-Endpoint Functionality, and PSM3_QP_PER_NIC,

NOTE

This setting is only used when the sockets HAL is selected and PSM3_SOCKETS is `0`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.75 PSM3_TCP_RCVBUF

Sets the maximum TCP socket receive buffering (in bytes) requested by PSM3 per TCP socket.

Default: `PSM3_TCP_RCVBUF=0`

When `PSM3_TCP_RCVBUF=0`, PSM3 will not request a specific amount of buffering. Instead, the OS sockets configuration settings will control how much buffering is assigned to each PSM3 TCP socket for each process. When a non-zero value is specified, PSM3 will use the Linux `setsockopt` API to set `SO_RCVBUF` to the specified value. See the Linux socket man page for socket(7) for more information.

NOTE

This setting is only used when the sockets HAL is selected and PSM3_SOCKETS is `0`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.76 PSM3_TCP_SKIPPOLL_COUNT

Allows PSM3 to skip polling all the open sockets in some situations. The syntax is:

```
PSM3_TCP_SKIPPOLL_COUNT=inactive_polls[:active_polls]
```

`inactive_polls` controls how many polls to skip after a poll where PSM3 finds no incoming socket data. `active_polls` controls how many polls to skip after a poll where PSM3 finds some sockets had incoming data, but there was not yet a posted application receive for the given tag(s). The `inactive_polls` value must be greater than or equal to `active_polls`.

Default: `PSM3_TCP_SKIPPOLL_COUNT=20:10`

When `PSM3_TCP_SKIPPOLL_COUNT=0:0`, the skip polls feature is turned off.

NOTE

By skipping polls, PSM3 overhead may be reduced, which may reduce communications CPU overhead or improve message rate or bandwidth for some applications. In some cases, this may also improve intra-node communications performance. However, for some applications it may increase latency.

NOTE

This setting is only used when the sockets HAL is selected and PSM3_SOCKETS is `0`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.77 PSM3_TCP_SNDBUF

Sets the maximum TCP socket send buffering (in bytes) requested by PSM3 per TCP socket.

Default: `PSM3_TCP_SNDBUF=0`

When `PSM3_TCP_SNDBUF=0`, PSM3 will not request a specific amount of buffering. Instead, the OS sockets configuration settings will control how much buffering is assigned to each PSM3 TCP socket for each process. When a non-zero value is specified, PSM3 will use the Linux `setsockopt` API to set `SO_SNDBUF` to the specified value. See the Linux socket man page for socket(7) for more information.

NOTE

This setting is only used when the sockets HAL is selected and PSM3_SOCKETS is `0`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.78 PSM3_TCP_SNDPACING_THRESH

Sets the PSM3 level send pacing threshold in bytes. When the socket send buffer depth is beyond the threshold, PSM3 will pause sending data until the depth is below the threshold.

Default: PSM3_MTU

When `PSM3_TCP_SNDPACING_THRESH=0`, PSM3 send pacing is turned off.

NOTE

This setting is only used when the sockets HAL is selected and PSM3_SOCKETS is `0`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.79 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 and PSM3_VERBOSE_ENV if enabled) are printed. This value should be considered a minimum.
- `0x2` – High-level debug information, mostly during initialization and shutdown.
- `0x20` – PSM3 connection establishment and disconnection.
- `0x40` – Detailed debug output regarding packet movement. This is a significant amount of output since multiple messages are output per packet.
- `0x80` – Detailed debug output regarding packet contents. This is a significant amount of output since multiple messages are output per packet.
- `0x100` – PSM3 process startup and shutdown.
- `0x200` – Detailed debug output regarding rendezvous exchanges and memory registration. This is a significant amount of output since multiple messages are output per `nic` or `shm` message using rendezvous .
- `0x400` – PSM3 environment variables. Also see PSM3_VERBOSE_ENV.

Default: `PSM3_TRACEMASK=0x1`

After the numeric value, an optional colon may be specified, optionally followed by a pattern. This behaves as follows:

- `value` – Specified TRACEMASK value is enabled on all processes.
- `value:` – Specified TRACEMASK value is enabled only on rank 0 (abbreviation for `PSM3_TRACEMASK=value:*:rank0`). All other processes will use the default TRACEMASK of 0x1.
- `value:pattern` – Specified TRACEMASK value is enabled only on processes whose label matches the extended glob pattern. All other processes will use the default TRACEMASK of 0x1.

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 # is the relative process number in the job. If the MPI runtime and the job scheduler have not indicated the rank to PSM3, the label will be of

the form `hostname:pid#`, where # 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_TRACEMASK=0x3:*:rank0` – Use TRACEMASK 0x3 for rank 0. When the user only needs to see additional output for one process, this can provide a more concise output.
- `PSM3_TRACEMASK=0x3:myhost047:*` - Use TRACEMASK 0x3 for processes on myhost047. All processes on that host will provide additional output. This can be helpful if only a single host's behavior is suspect.
- `PSM3_TRACEMASK=0x3:+(*:rank0|*:rank1)` – Use TRACEMASK 0x3 for 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

When specifying a `PSM3_TRACEMASK` setting which may generate large amounts of output, it can be beneficial to use PSM3_DEBUG_FILENAME to control where debug output is placed.

The following are some useful sample settings:

- `0x3` – Basic startup and finalization messages, including HAL selection, NIC filtering, and NIC selection (NIC and Address Filtering) are added to the output.
- `0x101` – Startup and finalization messages are added to the output.
- `0x123` – Details of startup and shutdown.
- `0x23` – High-level information about startup, shutdown, and connection establishment and disconnection.
- `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 a large amount of debug output.
- `0x23:` – High-level information about startup, shutdown, and connection establishment and disconnection provided only for rank 0.
- `0` – Silence PSM3 informative messages (warnings, PSM3_VERBOSE_ENV, and so on). Only PSM3_IDENTIFY output (if enabled) and errors will be reported.

Also see PSM3_IDENTIFY, PSM3_PRINT_STATS, and PSM3_VERBOSE_ENV.

NOTE

Any value specified for PSM3_TRACEMASK will not take effect until after the PSM3 configuration file (`/etc/psm3.conf`) is fully parsed. If it is desired to get debug output while parsing `/etc/psm3.conf`, use PSM3_VERBOSE_ENV. See PSM3 Config File for more information.

NOTE

When using `PSM3_TRACEMASK` to debug job failures during startup (such as failure to find an acceptable NIC during NIC and Address Filtering), it may be necessary to omit the colon and pattern. Some middlewares will terminate all processes after the first process exits, in which case the process with extra debug logging will be slower and may be terminated before providing any useful output.

NOTE

In addition to debugging information from PSM3, additional high-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.

8.21.80 PSM3_UDP_GSO

Enables UDP Segmentation Offload for UDP sending of application messages. This option allows PSM3 to send multiple messages in a single sockets API call. This may improve performance in some environments. This can be disabled by setting `PSM3_UDP_GSO=0`.

Default: `PSM3_UDP_GSO=65535`

When Segmentation Offload is enabled, PSM3 will send messages larger than one MTU using the sockets UDP `sendmsg` API call with up to `PSM3_UDP_GSO` bytes, but no smaller than one single MTU.

NOTE

For backward compatibility `PSM3_UDP_GSO=1` will set a limit of 65535.

NOTE

Some NICs may not fully implement the GSO feature of sockets. In this case messages such as `UDP GSO not supported` or `UDP GSO send failed` will be reported. If this occurs, the option should be disabled (set to `0`).

NOTE

This setting is only used when the sockets HAL is selected and PSM3_SOCKETS is `1`. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.81 PSM3_UDP_RCVBUF

Sets the maximum UDP socket receive buffering (in bytes) requested by PSM3 per UDP socket.

Default: `PSM3_UDP_RCVBUF=0`

When `PSM3_UDP_RCVBUF=0`, PSM3 will not request a specific amount of buffering. In this case, the OS sockets configuration settings will control how much buffering is assigned to the PSM3 UDP socket(s) for each process. When a non-zero value is specified, PSM3 will use the Linux `setsockopt` API to set `SO_RCVBUF` to the specified value. See the Linux socket man page for socket(7) for more information.

NOTE

A UDP socket is used in all PSM3_SOCKETS modes.

NOTE

This setting is only used when the sockets HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.82 PSM3_UDP_SNDBUF

Sets the maximum UDP socket send buffering (in bytes) requested by PSM3 per UDP socket.

Default: `PSM3_UDP_SNDBUF=0`

When `PSM3_UDP_SNDBUF=0`, PSM3 will not request a specific amount of buffering. In this case, the OS sockets configuration settings will control how much buffering is assigned to the PSM3 UDP socket(s) for each process. When a non-zero value is specified, PSM3 will use the Linux `setsockopt` API to set `SO_SNDBUF` to the specified value. See the Linux socket man page for socket(7) for more information.

NOTE

A UDP socket is used in all PSM3_SOCKETS modes.

NOTE

This setting is only used when the sockets HAL is selected. Otherwise, it is ignored. See PSM3 Architecture and Hardware Abstraction Layer and PSM3 Sockets Modes.

8.21.83 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 `/etc/psm3.conf` or the environment are output. No help text.
- `1:` – Enabled only on rank 0 (abbreviation for `PSM3_VERBOSE_ENV=1:*:rank0`). Only variables found in `/etc/psm3.conf` or the environment are output. No help text.
- `1:pattern` – Enabled only on processes whose label matches the extended glob pattern. Only variables found in `/etc/psm3.conf` or 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 # is the relative process number in the job. If the MPI runtime and the job scheduler have not indicated the rank to PSM3, the label will be of the form `hostname:pid#`, where # 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 you only need the help text or are confident that the same PSM3 environment variables and `/etc/psm3.conf` settings 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 or `/etc/psm3.conf` file 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. This variable can generate a large amount of output especially when all processes are selected in a high process count job. When help text output is desired, it's recommended to use `PSM3_VERBOSE_ENV=2:` to limit the output to rank 0.

NOTE

Only variables used by PSM3 (e.g., names prefixed with `PSM3_` or `FI_PSM3_`) are included in this output. Variables applicable to libfabric itself or other components in the software stack for the job (such as MPI or oneCCL) are not shown.

NOTE

Variables that 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.

NOTE

Output only occurs if PSM3_TRACEMASK has enabled informative messages (the default).

NOTE

There is special handling of `PSM3_VERBOSE_ENV` when parsing the PSM3 configuration file (`/etc/psm3.conf`) as discussed in PSM3 Config File.

NOTE

When PSM3_PRINT_STATS is enabled with PSM3_PRINT_STATSMASK bit `0x000040` set, the resulting statistics files will also include the values specified for any PSM3 env variables explicitly set as well as the full environment for the process, regardless of the `PSM3_VERBOSE_ENV` setting.

9.0 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, you can find 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

9.1 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 that 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 RDMA 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 RDMA interconnect.

Another example using the `lsof` command:

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

This command displays a list of processes using the RDMA interconnect.

Run the following command to terminate all processes using the RDMA 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 that are using the PSM3 Rendezvous Kernel Module. 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. Care must be taken 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.

9.2 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, use the ethshmcleanup tool (as root) on each node. Either log on to the node, or run remotely using `ethcmdall`, `pdsh`, or `ssh`.

Alternatively, 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.* /dev/shm/sem.psm3_nic_affinity* /dev/shm/
psm3_nic_affinity*
```

10.0 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 that 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 for URL.

10.1 Confirming the PSM3 Provider is Selected

A first step in any troubleshooting effort is to ensure the PSM3 provider is actually being used and selected.

The first step is to enable PSM3_IDENTIFY output. This will cause each process using PSM3 to indicate more details about the PSM3 version being used, some basic information about its capabilities and what NICs are being used.

```
PSM3_IDENTIFY=1
```

If a given process in the job does not report any `PSM3_IDENTIFY` output, this means PSM3 is not being selected for use by that process. In which case the parameters given to the application or middleware should be checked to ensure libfabric is being used and that `psm3` is specified as the provider to use.

NOTE

Even after a job is working properly, it can be useful to specify `PSM3_IDENTIFY=1`: so that the first rank in the job reports its `PSM3_IDENTIFY` information. This can help quickly confirm if PSM3 is being used when expected.

If PSM3 is not being selected, there could be a PSM3 configuration error, such as specification of an unavailable NIC in PSM3_NIC or the inability for PSM3 to find an acceptable NIC for a given process.

To further debug, Intel recommends that you to repeat the job with these additional variables exported:

```
PSM3_TRACEMASK=0x3
PSM3_IDENTIFY=1
FI_LOG_INFO=debug
```

These settings will cause additional debug output showing which libfabric providers were considered, which ones could not be loaded, and if PSM3 was considered, it will show the details of NICs and NIC addresses PSM3 evaluated), which ones were ruled out, and why. To reduce output, it may be desirable to run the job with only the subset of nodes having issues and with a reduced number of processes per node.

Also see: MPI Job Failures Due to Initialization Problems.

10.2 BIOS Settings

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

10.3 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

10.3.1 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.

10.3.2 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.

10.3.3 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
```

10.3.4 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, a non-existent NIC is selected, the existing NICs do not have an acceptable address, the existing NICs do not have an acceptable subnet, or that a hardware error occurred.

To further debug, Intel recommends that you to repeat the job with these additional variables exported:

```
PSM3_TRACEMASK=0x3
PSM3_IDENTIFY=1
FI_LOG_INFO=debug
```

These settings will cause additional debug output showing which libfabric providers were considered, which ones could not be loaded, and if PSM3 was considered, it will show the details of NICs and NIC addresses PSM3 evaluated), which ones were ruled out, and why. To reduce output, it may be desirable to run the job with only the subset of nodes having issues and with a reduced number of processes per node.

10.4 System Administration Troubleshooting

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

10.4.1 Flapping/Unstable NIC Links

Although PSM is designed to withstand temporary link outages, there may be NIC link instabilities that 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 administrator 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 administration-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. At 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).

10.4.2 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 a 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*.

10.5 Intel GPU Application Failures

If an Intel GPU application fails to launch or segfaults early in its execution, check the following:

- The MPI used and/or all middleware is oneAPI-enabled.
- The PSM3 OFI provider being used is oneAPI Level Zero-enabled. (See PSM3_IDENTIFY.)
- The PSM3_ONEAPI_ZE and/or PSM3_GPUDIRECT options are enabled.
- If PSM3_GPUDIRECT is enabled, verify that the Intel GPU oneAPI Level Zero-enabled rendezvous module is loaded. (See PSM3_IDENTIFY.)

If messages such as the following are observed, it may be necessary to increase the maximum allowed open files.

```
OneAPI Level Zero failure: zeMemGetIpcHandle() ... (ERROR_OUT_OF_HOST_MEMORY)
Error returned from OneAPI Level Zero function zeMemGetIpcHandle.
```

See PSM3 Support for Direct Intel GPU Access and PSM3 and Intel GPU Support.

10.6 CUDA Application Failures

If a CUDA application fails to launch or segfaults early in its 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 PSM3 Support for NVIDIA GPUDirect and PSM3 and NVIDIA CUDA Support

10.7 Performance Issues

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

11.0 Recommended Reading

This section contains lists of reference material for further reading.

11.1 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

11.2 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

11.3 Reference and Source for SLURM

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

https://slurm.schedmd.com/

11.4 OpenFabrics Alliance

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

http://www.openfabrics.org

11.5 Clusters

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

11.6 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

11.7 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/

12.0 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*.

12.1 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 with Intel® Ethernet Fabric Suite installed. However, they can also be directly used on an individual host.

12.1.1 dsa_setup

The Data Streaming Accelerator (DSA) is a high-performance data copy and transformation accelerator integrated into Intel® Xeon® Processors starting with the 4th Generation Intel® Xeon® Scalable Processors. PSM3 may be enabled to take advantage of DSA to optimize intra-node communications that use PSM3's `shm` device.

`/usr/share/eth-tools/samples/dsa_setup` is provided as a sample script to create DSA work queues in `/dev/dsa` for use by PSM3 jobs. This sample script should be copied to `/usr/local/bin/` and then edited as appropriate for the system. The resulting script must be run as `root` to configure DSA work queues each time the system reboots or immediately prior to and after each job which will use PSM3 with DSA enabled. To configure `dsa_setup` to be run at boot time, copy `/usr/share/eth-tools/samples/dsa.service` to `/etc/systemd/system/` and then edit `/etc/systemd/system/dsa.service` and follow the instructions in the file.

When configuring DSA work queues, `dsa_setup` will remove all existing DSA work queues, so if run per job, it should only be used when no other applications are using DSA. If the DSA configuration is to be selected per job, `dsa_setup` may be used in post job processing with the `-w none` or `-w restart` options to remove DSA resources after the job finishes. Then at the start of the next job, the appropriate `-w workload` option can be provided.

The use of `restart` is only required on some older distros, such as RHEL 8.6 and RHEL 9.0, to fully clear out DSA resources. Be aware that the use of restart may affect other applications that are using any of the CPU accelerators managed by the `idxd` kernel driver.

Syntax

```
dsa_setup [-u user] [-w workload] [-T timelimit]
```

or

```
dsa_setup --list
```

or

```
dsa_setup --help
```

Options

`--help` Produces full help text.

`--list` Shows DSA resources and configuration.

`-w workload` Configures DSA work queues for specified workload. Default is `ai`. When run to configure DSA work queues, must be run as root. Workloads may be added by adding `setup_all_WORKLOAD` functions. Valid workload values are: `ai`, `hpc`, `shared`, `none`, and `restart`.

`-u user` Specifies the owner for DSA work queue devices. Default is `root`. Specified as `[owner][:[group]]` similar to `chown` command.

NOTES

- If `:` is not specified, then only the user is granted read/write (`rw`) access.
- If `:` is specified, then the queues are granted group and user `rw` access for the specified group.
- If `:` is specified, but no group is specified, then the user's group is used.
- If `all` is specified, then everyone is granted `rw` access.

`-T timelimit` Specifies the seconds to wait for DSA device discovery. Default is 0. Sometimes during boot, a non-zero timeout is needed to allow time for the idxd kernel driver to discover and enumerate the devices.

Examples

```
dsa_setup --help
dsa_setup --list
dsa_setup
dsa_setup -u myname -w ai
dsa_setup -u myname: -w ai
dsa_setup -u myname:mygroup -w hpc
dsa_setup -u :mygroup -w hpc
dsa_setup -w none
dsa_setup -w restart
```

NOTE

For more information on DSA and how to enable it within the CPU, BIOS and Linux kernel, see https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, https://www.intel.com/content/www/us/en/products/docs/ondemand/overview.html, and https://cdrdv2.intel.com/v1/dl/getContent/759709.

12.1.2 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.

12.1.3 ethbw

`ethbw` reports the total data moved per RDMA NIC over each interval (default of 1 second). The bandwidth reported for each interval is in units of MB (1,000,000 bytes) over the interval. Both transmit (xmt) and receive (rcv) bandwidth counters are monitored. `ethbw` also monitors Intel NICs for any RDMA retransmit or input packet discards, in which case, the xmt or rcv, respectively, is shown as red. The data is gathered via data movement counters in `/sys/class/infiniband`.

The following cases may present the need to improve PFC tuning:

1. Retransmits can represent packet loss or corruption in the network and may indicate opportunities to improve PFC tuning or high bit error rates (BER) on some cables or devices.
2. Input packet discards indicate packets the NIC itself dropped upon receipt. This can represent opportunities to improve PFC tuning but can also be normal for some environments. Retransmits at the remote NICs that are communicating with this NIC are a more powerful indicator of PFC or BER causes for packet loss.

Syntax

```
ethbw [-i seconds] [-d seconds] [nic ... ]
```

Options

`--help`	Produces full help text.
`-i/--interval` *seconds*	Specifies the interval at which bandwidth will be shown. Values of 1-60 allowed. Defaults to 1.
`-d/--duration` *seconds*	Specifies the duration to monitor. Default is `infinite`.
nic	Specifies an RDMA NIC name. If no NICs are specified, all RDMA NICs will be monitored.

Examples

```
ethbw
ethbw irdma1 irdma3
ethbw -i 2 -d 300 irdma1 irdma3
```

12.1.4 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. The following utilities allow specifying a network interface. If no interface is specified, the action will apply to all available interfaces. If more than one interface is specified, only first interface is used. `LFC`

```
FW_DCB

SW_DCB_Willing/SW_DCB_Unwilling
```

Example .

```
ethsystemconfig --disable LFC
ethsystemconfig --enable SW_DCB_Willing en785f0
```

12.1.5 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 GPU drivers installed), either `NVIDIA_GPU_DIRECT=<DIR>` or `INTEL_GPU_DIRECT=<DIR>` must be in env.
`-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 Installation Prerequisites for the list of components by supported OS. Supported components include: `eth_tools psm3 eth_module fastfabric eth_roce 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*):

`PFC_MODE`	PFC mode (0-Off, 1-Software DCB Willing, 2-Software DCB Unwilling, 3-Firmware DCB Willing)
`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 RoCE
   3) Reconfigure Driver Autostart
   4) Generate Supporting Information for Problem Report
   5) FastFabric (Host/Admin)
```

```
   6) Uninstall Software

   X) Exit
```

12.1.6 ethcapture

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

NOTE

The resulting host capture file can require significant amounts of space on the host. While the actual size varies, 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*, obtains basic fabric information using `ethreport`.
`3` (Analysis)	In addition to *Fabric*, obtains `ethallanalysis` results. If `ethallanalysis` has not yet been run, it is run as part of the capture.

NOTES

- 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
```

12.1.7 ethshmcleanup

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. This script is intended to remove unused files related to PSM3.

The unused files that are removed include:

- `/dev/shm/psm3_shm*`
- `/dev/shm/sem.psm3_nic_affinity*`
- `/dev/shm/psm3_nic_affinity*`

Syntax

```
ethshmcleanup
```

Options

`--help` Produces full help text.

Examples

```
ethshmcleanup
```