Cornelis® Omni-Path Express® Fabric Performance Tuning

User Guide

February 2025
Doc. No. H93143, Rev. 29.0

Table of Contents

Revision History

Date	Rev	Description
Feb 2025	29.0	Updates include: • Additional environment variables for MPI and the OPX Provider. • Additional recommendations and tunings for the use of NVIDIA and AMD GPUs.
Jan 2025	28.0	Updates include: • Added Intel Emerald Rapids and AMD Turin tunings. • Updated AMD IOMMU recommendations. • Added recommendations regarding performance impacts due to SRSO mitigations.
Dec 2024	27.0	Several updates throughout, including: • Updated to RHEL/Rocky version sections.
Jul 2024	26.0	Several updates throughout, including: • Removed references to the Intel Xeon Phi processor. • Updated the MPI Library sections. • Updated the MPI Applications Performance Tuning table. • Other cosmetic updates, such as changing IBM Storage Scale from IBM Spectrum Scale.
Mar 2024	25.0	• Updated usage of Intel MPI. • Updated for 4th Generation Intel (Sapphire Rapids) and AMD (Genoa) CPUs.
Nov 2022	24.0	Minor updates throughout.
Feb 2022	23.0	• Cornelis Branding updates as follows: – "Omni-Path Architecture" is now "Omni-Path Express." – "OPA" is now "OPX." – "Intel Fabric Suite" or "Cornelis Fabric Suite" is now "Omni-Path Express Fabric Suite." – "IFS" is now "OPXS."
Apr 2021	22.0	• Updated “Intel Xeon Scalable Processor”to include new BIOS setting for 3rd Generation Intel Xeon Scalable Processors. • Added new section, “AMD EPYC Processor” . • Updated “Adjusting UD Mode MTU Size” to specify RHEL release information. • Updated “Tuned Utility Latency-Performance Profile” to include 3rd Generation Intel Xeon Scalable Processors.

For previous releases, refer to Appendix A “Older Revisions”.

Preface

This guide is part of the documentation set for the Omni-Path Express Fabric, which is an end-to-end solution consisting of Cornelis Omni-Path Express Host Fabric Interface Adapters (HFIs), Cornelis Omni-Path Express Edge Switches, Cornelis Omni-Path Express Director Class Switches, and fabric management and development tools.

The Cornelis Omni-Path Express 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 large-scale HPC clusters.

Both the Omni-Path Express Fabric and standard InfiniBand (IB) can send Internet Protocol (IP) traffic over the fabric, or *IPoFabric*. In this document it may also be referred to as *IP over IB* or *IPoIB*. From a software point of view, IPoFabric behaves the same way as IPoIB, and in fact uses an ib_ipoib driver to send IP traffic over the ib0/ib1 ports.

Intended Audience

This document is intended for system administrators and other personnel with similar qualifications.

Documentation Library

All Cornelis Networks product documentation may be found in the Release Library located in the Cornelis Customer Center.

Refer to the "Document Library table" in the *Cornelis Omni-Path Express Fabric Quick Start Guide*.

Document Conventions

The following conventions are standardized across all Cornelis Omni-Path Express 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 and underlined 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 Cornelis Networks.

- Text in **bold** indicates user interface elements such as menu items, buttons, check boxes, key names, keystrokes, 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`.
- Preformatted text in `Courier` font with a gray background indicates a block of code.

```
Welcome to Cornelis Networks shell
Use 'tab' for autocomplete and up/down arrow to see command history.
```

- Text in *italics* indicates terms, emphasis, variables, or document titles. For example:
  - Refer to *Cornelis Omni-Path Express Fabric Software Installation Guide* for details.
  - In this document, the term *chassis* refers to a managed switch.
- Most of the acronyms in this document link to the glossary.

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

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

Cornelis Omni-Path Express Fabric Design Generator for Cornelis Omni-Path Express Fabric

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

To access the Fabric Design Generator for Omni-Path Express Fabric, go to Cornelis® Omni-Path® Fabric Design Generator.

License Agreements

Cornelis software and firmware are 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 agreed to the terms and conditions of the license agreement(s). By loading or using the software, you agree to the terms of the license agreement(s). If you do not wish to so agree, do not install or use the software.

Technical Support

Technical support for Cornelis products is available 24 hours a day, 365 days a year:

- Website: Cornelis Networks Customer Support
- Email: support@cornelisnetworks.com

1. Introduction

The Cornelis Omni-Path Express product family is designed for excellent out-of-the-box performance. However, you may be able to further tune the performance to better meet the needs of your system.

This document describes the BIOS settings and parameters that have been shown to improve performance, or make performance more consistent, on Omni-Path Express. If you are interested in benchmarking the performance of your system, these tips may help you obtain better performance.

1.1. OPX Libfabric Provider

The OPX Libfabric Provider (hereafter called the OPX Provider) is written to take full advantage of the libfabric acceleration features while running over existing and future Omni-Path Express hardware.

The OPX Provider delivers excellent latency characteristics and message rates at smaller message sizes (under 16K message length). Bulk transfer using SDMA is present in the OPX Provider giving some added performance for large messages. RDMA support is implemented in the OPX Provider using the environment variable `FI_OPX_EXPECTED_RECEIVE_ENABLE=1`.

Use `FI_LOG_LEVEL=trace FI_LOG_SUBSYS=core` to print the file location info and OPX Provider library file in use. The output will look similar to:

```
Using opx Provider: Library file location is *.so file location
```

1.2. Terminology

The table below lists the abbreviations and acronyms used in this document.

Table 1. Terminology

Term	Description
ACPI	Advanced Configuration and Power Interface
AIP	Accelerated IPoFabric
BIOS	Basic Input/Output System
CPU	Central Processing Unit
FM	Fabric Manager
GCC	GNU Compiler Collection
GUPS	Giga-Updates per Second
HFI	Host Fabric Interface
HPC	High-Performance Computing
HPL	High-Performance LINPACK

Term	Description
HT	Hyper Threading
IMB	Intel MPI Benchmarks
IO	Input/Output
IP	Internet Protocol
IPoFabric	Internet Protocol over Fabric
IPoIB	Internet Protocol over InfiniBand
IRQ	Interrupt Request
MPI	Message Passing Interface
MTU	Maximum Transmission Unit
NUMA	Non-Uniform Memory Access
OFI	OpenFabrics Interface
OMB	OSU Micro Benchmarks
OPX Provider	Omni-Path Express Libfabric Provider
OS	Operating System
OSU	Ohio State University
PPN	Processes per Node
PSM2	Performance Scaled Messaging 2
QCD	Quantum Chromodynamics
QP	Queue Pair
RDMA	Remote Direct Memory Access
RPS	Receive Packet Steering
SDMA	Send Direct Memory Access
SMP	Symmetric Multiprocessing
TBB	Threading Building Blocks
TCP	Transmission Control Protocol
THP	Transparent Huge Pages
TID	Token ID
UD	Unreliable Datagram
VL	Virtual Lane
VM	Virtual Machine
VT	Virtualization Technology

1.3. Omni-Path Fabric Performance Tuning Quick Start

The table below outlines the most important tunings for Omni-Path performance, and is sorted by most important tunings. Separate columns are shown for MPI/PSM2/OPX, Verbs, and IPoFabric performance tunings.

This is only a rough guide and individual clusters may require other tunings, discussed in other sections of this guide.

Table 2. Highest Priority Tunings

MPI/PSM2/OPX	**Verbs**	**IPoFabric**
Set BIOS settings. (See Section 2 "BIOS and Platform Settings".)		
Enable processor turbo mode, if possible. Enable "Performance Governor" with either ACPI or Intel P-State frequency driver: `#cpupower -c all frequency-set -g performance`		
Set irqbalance. See Section 3.1 "irqbalance".		
Make sure the MPI is using PSM2 or OPX. See Section 5.2 "Intel MPI Library Settings". Use the latest available Intel OneAPI for optimized application performance.	Set `sge_copy_mode=2`. If the performance is low with the default of `krcvqs=2`, slowly increase `krvqs` to no more than one half of the number of physical cores per socket. See Section 6.2 "Parallel File System Concurrency Improvement".	For best-possible bandwidth, Cornelis recommends using Datagram Mode with Accelerated IPoFabric (AIP). In this mode, the MTU size can be adjusted to a maximum of 10236 bytes to achieve better bandwidth. See Section 7.2.2 "Adjusting UD Mode MTU Size". If Connected Mode is used, set a 64KB MTU (See Section 7 "IPoFabric Performance"), and disable the AIP driver module parameter (`ipoib_accel=0`). Connected mode and AIP are mutually exclusive.
Set MPI affinity. See Section 5.9 "MPI Affinity and HFI Selection".	Apply IBM Storage Scale (formerly GPFS) tuning if needed. See Section 6.5 "IBM Storage Scale (aka GPFS)".	Set `krcvqs=3` or `krcvqs=5` (using an odd number may improve performance). See Section 7.3 "krcvqs Tuning for IPoFabric Performance".
Adjust `rcvhdrcnt` and `num_user_contexts` if beneficial for application performance. See Section 5.11 "MPI Applications Performance Tuning".	Use Lustre version 2.10 or newer. See Section 6.4 "Lustre".	Apply Receive Packet Steering (RPS) and turn off Generic Segmentation Offload (GSO). See Section 7.5 "RPS and GSO Tuning for IPoFabric Performance".

The following tuning options are of lower priority, but we recommend that you explore using them for advanced performance tuning:

- Enable C-state for better Verbs/IPoFabric latency. See Section 6.3.2 "Verbs Latency".

- Enable Accelerated RDMA to improve Verbs BW. See Section 6.1 “Accelerated RDMA”.
- Reduce memory fragmentation. See Section 3.5 “Memory Fragmentation”.
- Increase Address Resolution Protocol (ARP) cache on large fabrics. See Section 3.6 “Disable IPv6 and Adjust Address Resolution Protocol Thresholds on Large Fabrics”.
- Reduce system jitter for large-scale MPI/PSM applications. See Section 5.17 “Reducing System Jitter”.

2. BIOS and Platform Settings

Setting the system BIOS is an important step in configuring a cluster to provide the best mix of application performance and power efficiency. In this chapter, we specify settings that can maximize the Omni-Path Express Fabric and application performance. Optimally, settings similar to these should be used during a cluster bring-up and validation phase in order to show that the fabric is performing as expected. Once bring-up and validation is completed, you may want to set the BIOS to provide more power savings, even though that may reduce overall application and fabric performance to some extent.

2.1. Intel Xeon Processor E5 v3 and v4 Families

The performance-relevant BIOS settings on a server with Intel Xeon Processor E5 V3 and V4 Family CPUs, recommended for all-around performance with an Omni-Path Express fabric, are shown in the table below:

Table 3. Recommended BIOS Settings for Intel Xeon Processor E5 v3 and v4 Families

BIOS Setting	Value
CPU Power and Performance Policy	Performance or Balanced Performance[1]
Workload Configuration	Balanced
Uncore Frequency Scaling	Enabled
Performance P-limit	Enabled
Enhanced Intel SpeedStep Technology	Enabled
Intel Configurable TDP	Disabled
Intel Turbo Boost Technology	Enabled
Intel VT for Directed I/O (VT-d)	Disabled
Energy Efficient Turbo	Enabled
CPU C-State	Enabled
Processor C3	Disabled
Processor C6	Enabled
Intel Hyper-Threading Technology	No recommendation (Test with your applications to see if a benefit occurs.)
IOU Non-posted Prefetch	Disabled (where available)[2]
Cluster-on-Die	Disabled
Early Snoop	Disable
NUMA Optimized	Enable[3]
MaxPayloadSize	Auto or 256B[4]
MaxReadReq	4096B[4]

BIOS Setting	Value
Snoop Holdoff Count	9[5]

Notes:

1. To get the most consistent Turbo mode performance for demanding workloads, set this to "Performance." Either Performance or Balanced Performance will result in good Omni-Path Express Fabric performance.
2. Available in the Intel Xeon Processor E5 v4 BIOS version R016.
3. Also known as `Memory.SocketInterleave=NUMA` in some BIOSes.
4. PCIe Max Payload Size and Max Read Req settings are sometimes not in the BIOS. In that case, the Omni-Path Express driver (hfi1) parameter `pcie_caps=0x51` setting (which implies setting `MaxPayLoad` to 256B and `MaxReadReq` to 4096B) can be made as described in Section 4 "HFI1 Driver Module Parameters".
5. Also known as `Snooped Response Wait Time for Posted Prefetch` in some BIOSes.

2.2. Intel Xeon Scalable Processor

For Intel Xeon Scalable Processor CPUs, Cornelis recommends the following:

1. Install the latest BIOS version available from your vendor with these or similar settings.

Table 4. Recommended BIOS Settings for Intel Xeon Scalable Processor

BIOS Setting	Value
Sub_NUMA Cluster[a]	Disabled[b]
Snoop Holdoff Count/Snoop Timer Hold Off	See Note c.
C states (System profile)	Enabled
Uncore Frequency Scaling	Enabled
WFR Uncore GV Rate Reduction[d]	Disabled (where available)
MADT Core Enumeration[e]	Linear

BIOS Setting	Value
Notes: a. Also known as `SNC` or `Cluster-on-Die` in some BIOSes. b. "Disabled" is recommended for most applications. "Enabled" should be tested with your application to determine if it provides better performance. c. For 1st and 2nd Generation Intel Xeon Scalable Processors: • Recommended value is 0x9 • Also known as `Snooped Response Wait Time for Posted Prefetch` in some BIOSes. For 3rd Generation Intel Xeon Scalable Processors: • Recommended value is 0xa For 4th and 5th generation Intel Xeon Scalable Processors: • Suggested Value is 0xb d. May be named differently in some BIOSes. e. Applies to 4th Generation Intel Xeon Scalable Processors only.	

2. Use the default settings, including Turbo Boost Technology= Enabled.

   Settings for Intel Xeon Processor E5 V3 and V4 Family CPUs are listed in Table 3 "Recommended BIOS Settings for Intel Xeon Processor E5 v3 and v4 Families". Cornelis recommends the same values for these settings be used on Intel Xeon Scalable Processor CPUs, where the setting is available.

3. Enable Turbo speeds as specified in Section 3.2 "CPU Frequency Scaling Drivers".

For best performance with the Intel Server System S9200WK Family, memory DIMMs must be installed in multiples of eight. Follow the layout prescribed in the *Intel Server System S9200WK Product Family Setup and Service Guide*.

2.3. AMD EPYC Processor

For AMD EPYC Processors, Cornelis recommends that you refer to AMD's High Performance Tuning Guide relevant to your generation of EPYC Processor:

- 2nd Generation EPYC Processors
- 3rd Generation EPYC Processors
- 4th Generation EPYC Processors
- 5th Generation EPYC Processors

NOTE

Customers should experiment with different tunings based on their workload to determine the best performance.

Install the latest BIOS version available from your vendor with these or similar settings.

Table 5. Recommended BIOS Settings for AMD EPYC Processors

BIOS Setting	Value	Description
System Profile	Performance	Typically controls turbo boost enable/ disable, C States enable/disable, and power management settings.
NPS (NUMA per socket)	NPS=4[a]	NPS=4 had significant impact on MPI Message Rate measurements.
PCIe Preferred I/O	Enabled[b]	Can be used to configure an HFI in a state to get preferential treatment.
Simultaneous Multi-Threading (SMT)	Disabled	Typically, for HPC workloads, best performance is seen when SMT is disabled.
APBDIS	Enabled	Can improve small message latency, bandwidth, and message rate performance.
IOMMU	Pass-Through	Improves stability with high core count systems.

Notes:

1. Depending on the number of cores available on the CPU, the default mapping for AIP interrupts may be suboptimal and could impact IPoFabric BW without manual interrupt tunings.
2. For 2nd Generation AMD EPYC Processors:
   - For single HFI systems, this has been shown to give the best Verbs performance. For dual HFI systems, there is a tradeoff for performance for the two HFIs because only one HFI bus can be specified.

   For 3rd Generation AMD EPYC Processors:
   - For dual HFI systems, change the Root Complex LCLK Frequency corresponding to the HFI PCIe locations from auto to 593 MHz.

   For 4th Generation AMD EPYC Processors:
   - Neither of the above options are available.

When using two Milan CPUs in a system with a single HFI the system `ulimit` values may need to be modified from default. See Section 3.7 "Configuring ulimit Values".

2.4. GPUDirect Requirements

For GPUDirect to function properly, NVIDIA recommends disabling setting PCIe Access Control Services (ACS), also known as IO virtualization, VT-d, or IOMMU, to pass-through mode. If left enabled, unpredictable behavior such as application failures may be experienced. Refer to the NVIDIA documentation, PCI Access Control Services, for more information regarding setting the pass-through mode.

2.5. AMD GPU Requirements

To optimize AMD GPU, AMD recommends:

1. Enable large bar addressing in the BIOS to support peer-to-peer GPU memory access.
2. Verify SR-IOV is enabled, if needed.
3. Disable ACS.

   ACS forces peer-to-peer transactions through the PCIe root complex.

For more information, refer to the following AMD documentation: Single-Node and Multi-Node Network Configuration documentation and System Optimization.

3. Linux Settings

Cornelis recommends the following settings to enable consistent performance measurements on the Linux distributions supported with Omni-Path Express Host Software.

3.1. irqbalance

The purpose of irqbalance is to distribute hardware interrupts across processors on a multiprocessor system in order to increase performance. Omni-Path Express uses the `irqbalance --policyscript` parameter to configure irq affinity to work with the receive and SDMA interrupt algorithms in the HFI1 driver.

To implement the irqbalance setting, perform the following steps using root or sudo permissions.

1. Install the irqbalance package, if not already installed:

```
# yum install irqbalance
```

2. Add the following line to the `/etc/sysconfig/irqbalance` file, if it is not already there:

```
IRQBALANCE_ARGS="--policyscript=/etc/sysconfig/opa/hintpolicy_exact_hfi1.sh”
```

3. After the HFI1 driver is loaded, restart the irqbalance service:

```
/bin/systemctl restart irqbalance.service
```

NOTE

For detailed tuning recommendations using IRQ affinity assignments to distribute CPU workloads, see Section 8 “Driver IRQ Affinity Assignments”.

NOTE

If you need to unload and reload the HFI1 driver (to make driver configuration changes, for example), you must first stop irqbalance, unload and reload the driver, then start irqbalance. This is required to prevent improper assignments that can occur when you unload and load the HFI1 driver while irqbalance is running, and then restart irqbalance.

3.2. CPU Frequency Scaling Drivers

Methods for power saving on CPUs can impact performance inversely. By reducing the CPU clock frequency based on sustained demand and thermal conditions, CPUs reduce power

consumption. This can result in substantial savings on power and cooling requirements. However, this can reduce the performance or make performance measurements more variable. Thermal conditions are not predictable, resulting in a run-to-run variation in CPU performance.

The default scaling driver for Intel processors in RHEL 8.x and 9.x is the Intel P-State (`intel_pstate`) driver. An alternative driver called the Advanced Configuration and Power Interface (ACPI) CPUfreq (`acpi_cpufreq`) is also available and is utilized by default for AMD processors. Both have their advantages and disadvantages, but only one can be active at a time. In this section we describe how to use each driver for consistent, best-effort performance measurements. Setting your frequency scaling driver for maximum performance is advisable during cluster/fabric bring-up when trying to determine if all components of the cluster are performing up to their full capabilities.

For long-run operation of a production cluster/supercomputer, settings other than those described in the following sections may be desired to scale up for performance when loaded, and to scale down for energy savings when idle.

NOTE

The information and recipes in the following subsections only apply to Intel processors.

For AMD processors, it is recommended to use either the default ACPI CPUfreq driver or the newer AMD P-State `amd_pstate` driver if Collaborative Processor Performance Control (CPPC) is supported on your platform.

3.2.1. Using the Intel P-State Driver

The Intel P-state driver is the default driver for RHEL 8.x and 9.x, and no additional setup is required. A detailed description of the design and features available with Intel P-state drivers is available here. In general, no customization beyond the default is required for the best fabric performance, other than ensuring that the turbo frequencies are enabled and the performance governor is enabled.

The following settings are sysfs entries that can be controlled by the system administrator in real time, and a reboot is not required in order to take effect. However, due to the nature of Intel P-state, it is not always straightforward to monitor the core frequencies and confirm your settings are in effect. For example, a command such as `grep MHz /proc/cpuinfo` will return a wide range of clock frequencies at any given time, unlike ACPI, which would return a consistent value in a format like "2X00000" or "2X01000," if Turbo mode is enabled. Cornelis recommends confirming and monitoring the clock frequencies using a kernel tool such as turbostat.

By default, the CPU will vary in frequency and power state depending on workload, resulting in sub-optimum performance. To avoid this set, as root, set the minimum frequency to 100% as shown below.

```
echo 100 > /sys/devices/system/cpu/intel_pstate/min_perf_pct
```

To run the CPU at its maximum turbo frequency, in the BIOS set the following values:

- Set **Intel Turbo Boost Technology → Enabled**
- If it is in your BIOS, set **Advanced → Advanced Power Management Configuration → CPU P State Control → Turbo mode**
- `echo 0 > /sys/devices/system/cpu/intel_pstate/no_turbo`
- Set the cpufreq policy to "performance": `cpupower frequency-set -g performance`

  For information about the CPU frequency driver you are running and other frequency information, use the command:

  `cpupower frequency-info`

It is possible to enforce a slower clock frequency for benchmarking or validation purposes with the Intel P-state frequency driver. To do this, first disable Turbo mode, then set `min_perf_pct` and `max_perf_pct` such that `[min/max]_perf_pct = ceiling(target_clock/base_clock*100)`. For example, if we want to enforce a clock frequency of 1.8 GHz on a processor with a 2.1 GHz base frequency, we would set `[min/max]_perf_pct = ceiling(1.8/2.1*100)= 86`.

- `echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo`
- `echo 86 > /sys/devices/system/cpu/intel_pstate/min_perf_pct`
- `echo 86 > /sys/devices/system/cpu/intel_pstate/max_perf_pct`

If you have previously disabled the P-state driver, you must re-enable it *before* applying the tunings listed above. To re-enable the P-state driver:

1. In `/etc/default/grub`, remove `intel_pstate=disable` from the GRUB_CMDLINE_LINUX command line.
2. For RHEL/Rocky versions 9.2 and prior, apply the change using:

```
if [ -e /boot/efi/EFI/redhat/grub.cfg ]; then
 GRUB_CFG=/boot/efi/EFI/redhat/grub.cfg
 elif [ -e /boot/grub2/grub.cfg ]; then
 GRUB_CFG=/boot/grub2/grub.cfg
 fi
 grub2-mkconfig -o $GRUB_CFG
```

3. For RHEL/Rocky versions 9.3 and later, apply the change using:

```
grub2-mkconfig --update-bls-cmdline -o /etc/grub2.cfg
```

NOTE

The code example above is for Red Hat. Other OSes may require a different method for modifying grub boot parameters.

4. Reboot.

For more information on controlling and tuning the behavior of the Intel P-state driver, consult here.

3.2.2. Using the ACPI CPUfreq Driver and cpupower Governor

NOTE

If you are satisfied with the behavior of your system when using the P-State driver, you do not need to set up the acpi_cpufreq driver.

The ACPI CPUfreq (acpi_cpufreq) driver, in conjunction with cpupower, can be used to set a consistent CPU clock rate on all CPU cores. The method to enable ACPI varies depending on whether the server is Intel or AMD based.

To enable the ACPI CPUfreq driver (Intel CPUs only):

1. Disable intel_pstate in the kernel command line:

   Edit `/etc/default/grub` by adding `intel_pstate=disable` to `GRUB_CMDLINE_LINUX`.

   For example:

   ```
   GRUB_CMDLINE_LINUX=vconsole.keymap=us console=tty0
   vconsole.font=latarcyrheb-sun16 crashkernel=256M
   console=ttyS0,115200 intel_pstate=disable
   ```

2. For RHEL/Rocky versions 9.2 and prior, apply the change using:

   ```
   if [ -e /boot/efi/EFI/redhat/grub.cfg ]; then
    GRUB_CFG=/boot/efi/EFI/redhat/grub.cfg
    elif [ -e /boot/grub2/grub.cfg ]; then
    GRUB_CFG=/boot/grub2/grub.cfg
    fi
    grub2-mkconfig -o $GRUB_CFG
   ```

3. For RHEL/Rocky versions 9.3 and later, apply the change using:

   ```
   grub2-mkconfig --update-bls-cmdline -o /etc/grub2.cfg
   ```

   NOTE

   The code example above is for Red Hat. Other OSes may require a different method for modifying grub boot parameters.

4. Reboot.

   When the system comes back up with `intel_pstate` disabled, the acpi_cpufreq driver is loaded.

To reduce run-to-run performance variations during benchmarking, you may want to pin the CPU clock frequency to a specific value and use the `Performance` setting of the CPU power governor.

To set the CPU clock frequency and power governor:

1. Set the clock frequency values and governor using the command line below.

```
 sudo cpupower -c all frequency-set --min <value> --max <value>
\ -g Performance
```

   Where <*value*> is a valid number and unit (GHz) for min and max settings. Note the values can be the same.

For example, the following command will set the frequency of all cores to a value of 2.3 GHz and `Performance` governor, when using the acpi-cpufreq driver.

```
sudo cpupower -c all frequency-set --min 2.3GHz --max 2.3GHz \
-g Performance
```

NOTE

The power savings will diminish and the heat dissipation will increase in the server chassis if the above scheme is used.

To get the maximum advantage from Turbo mode:

1. Ensure that Turbo mode is set to Enabled in the BIOS (as recommended in Section 2 "BIOS and Platform Settings").
2. Set the frequencies appending "01" to the clock rate. This will enable the Turbo advantage.

For example, if running on an Intel Xeon Processor E5-2699 v3 (nominal 2.3 GHz clock rate), then the corresponding command option would be:

```
sudo cpupower -c all frequency-set --min 2.301GHz --max 2.301GHz \
-g Performance
```

3.3. Setting IOMMU to Passthrough

IOMMU is enabled in most modern Linux distributions by default but can hurt verbs and MPI performance. In most instances, it is recommended to set IOMMU to passthrough rather than disabling IOMMU entirely.

1. Navigate to `/etc/default/grub`.
2. For AMD processors, perform the following: Change the `iommu=on` flag to `iommu=pt`, or add the flag if not present.
3. For RHEL/Rocky versions 9.2 and prior, apply the change using:

```
if [ -e /boot/efi/EFI/redhat/grub.cfg ]; then
 GRUB_CFG=/boot/efi/EFI/redhat/grub.cfg
 elif [ -e /boot/grub2/grub.cfg ]; then
 GRUB_CFG=/boot/grub2/grub.cfg
 fi
 grub2-mkconfig -o $GRUB_CFG
```

4. For RHEL/Rocky versions 9.3 and later, apply the change using:

```
grub2-mkconfig --update-bls-cmdline -o /etc/grub2.cfg
```

NOTE

The code example above is for Red Hat. Other OSes may require a different method for modifying grub boot parameters.

5. Reboot.

3.4. Transparent Huge Pages

Transparent Huge Pages is set to “always”. It is enabled in RHEL 7.2 and later by default. Note that changing this setting to “never” will hurt large message bandwidth (above 64 MB) significantly.

If the default is set, this file should show the following output:

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

If the default "always" is not set on each node, you can set it by using the following command:

```
echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

3.5. Memory Fragmentation

When a Linux system has been running for a while, memory fragmentation, which depends heavily on the nature of the applications that are running on it, can increase. The more processes that request the kernel to allocate and free physical memory, the quicker the physical memory becomes fragmented. If that happens, performance on applications can suffer significantly. Over time, the performance of benchmarks and applications can decrease because of this issue.

Cluster/system administrators and users can take steps to address the memory fragmentation issue as described below. Note that users will not be able to apply their settings until the system administrators have applied theirs first.

3.5.1. System Administrator Settings

The following settings are performed by system administrators.

1. Enable THP to "always" as per Section 3.4 "Transparent Huge Pages".
2. As an alternative to THP, reserve huge pages with the sysfs entries, `nr_hugepages` or `nr_overcommit_hugepages`.
3. To better ensure that the system will allocate 2M pages to the job, set the cluster's job submission system to drop the caches and compact memory before each user job with these commands:

```
echo 3 >/proc/sys/vm/drop_caches
echo 1 >/proc/sys/vm/compact_memory
```

4. Setting a smaller number of User Contexts, using driver parameter `num_user_contexts`, can allocate a larger number of TID descriptors per context, which can deal better with the effects of memory fragmentation. See section 5.3 for guidelines on how to adjust HFI1 driver parameters. Note that reducing this setting to less than the number of physical cores on the system will require PSM2 context sharing if the number of MPI ranks exceeds num_user_contexts.

3.5.2. User Settings

The following settings are performed by users.

1. Assuming that the system administrator has enabled THP (described in Section 3.5.1 "System Administrator Settings", Step1), the user can align larger MPI buffers on 2M boundaries and pad the total size to a multiple of 2M.

   You can use `posix_memalign` or Intel's `_mm_malloc` to cause the OS to try to allocate 2 MB pages.
2. Assuming that the system administrator has enabled the alternative to THP (described in Section 3.5.1 "System Administrator Settings", Step 2), the user can explicitly allocate huge pages using mmap, Threading Building Blocks (TBB) malloc with `TBB_MALLOC_USE_HUGE_PAGES=1`, or libhugetlbfs.

3.6. Disable IPv6 and Adjust Address Resolution Protocol Thresholds on Large Fabrics

Cornelis recommends that you disable IPv6 on IPoFabric interfaces if they are not in use, which is typical. Most clusters only require IPv6 for communication to external networks over Ethernet, and disabling IPv6 for IPoFabric significantly reduces the overhead associated with multicast groups. This can be done globally on the entire node, or for just the specific IPoFabric interface.

To disable IPv6 for just the IPoFabric interface (ib0 in this example), add this line to `/etc/sysctl.conf`:

```
net.ipv6.conf.ib0.disable_ipv6 = 1
```

NOTE

If the interface is bonded, IPv6 can be disabled by using the bonded interface name such as `net.ipv6.conf.`**`bond0`**`.disable_ipv6=1`.

To disable globally for all interfaces, add this line to `/etc/sysctl.conf`:

```
net.ipv6.conf.all.disable_ipv6 = 1
```

To **load the settings from `/etc/sysctl.conf`**, issue the following command (or reboot):

```
# sysctl -p
```

For more information on using `sysctl`, see the Linux `sysctl.conf(5)` man page.

3.6.1. ARP Threshold Variables

For large fabrics with greater than 128 hosts, or networks with heavy IP traffic, it may be beneficial to increase the kernel's internal Address Resolution Protocol (ARP) cache size. The following ARP threshold variables are used by the Linux kernel:

- For IPv4 and IPv6 (if IPv6 was not disabled as previously recommended):
  - `net.ipv[4,6].neigh.default.gc_thresh1` (Default: 128)
  - `net.ipv[4,6].neigh.default.gc_thresh2` (Default: 512)
  - `net.ipv[4,6].neigh.default.gc_thresh3` (Default: 1024)

If you notice messages in the system log file containing `kernel: Neighbour table overflow`, this indicates that the ARP table on the server is full and needs to be expanded to avoid overflow, which causes addresses to get dropped from the ARP cache.

3.6.2. Modifying ARP Threshold Values

The instructions below provide you with the basic steps for modifying ARP thresholds.

The values shown below are examples only. You will need to experiment with different settings to find the best thresholds for your fabric size.

NOTE

The instructions below pertain to IPv4. For IPv6, replace each occurrence of `ipv4` with `ipv6`.

1. To list the current ARP threshold values (IPv4):

```
# cat /proc/sys/net/ipv4/neigh/default/gc_thresh1
# cat /proc/sys/net/ipv4/neigh/default/gc_thresh2
# cat /proc/sys/net/ipv4/neigh/default/gc_thresh3
```

2. Increase the ARP threshold levels:

   To increase the ARP threshold levels (IPv4) using root or sudo, add these lines to `/etc/sysctl.conf`:

```
net.ipv4.neigh.default.gc_thresh1 = 16384
net.ipv4.neigh.default.gc_thresh2 = 32768
net.ipv4.neigh.default.gc_thresh3 = 65536
```

NOTE

Settings should be consistent throughout the cluster.

3. Load the settings as described in Section 3.6.1 “ARP Threshold Variables”.

3.6.3. Increase the ARP Garbage Collection Interval

IPoIB ages its pathrecords based on `net.ipv4.neigh.default.gc_interval`, which defaults to 30 seconds. Pathrecords that are unused for `2*gc_interval` are released and subsequent interactions with the given remote node will require a Pathrecord query to the SA (but not necessarily an ARP). This query adds overhead, especially on clusters of thousands of nodes. Cornelis recommends that you increase this interval value to a very large value such as `net.ipv4.neigh.default.gc_interval=2000000`.

Implement this change in the same manner as the ARP threshold values were changed in the previous section.

3.7. Configuring ulimit Values

Some MPI applications may require an unlimited stack size to be set by the user. To achieve this, place `ulimit -s unlimited` in `~/.bashrc`, and log out and back in. Without this setting, segmentation faults and other unexpected behavior may result.

4. HFI1 Driver Module Parameters

Default settings for HFI1 Driver Module parameters currently achieve the best performance. However to further tune your performance, you can modify specific parameters as described in this document.

This chapter describes:

- The list of HFI1 driver module parameters
- How to set the parameters
- How to activate the changes

This chapter also includes information on tuning for dual/multi rail configurations.

4.1. Listing the Driver Parameters

To get a listing and brief description of the HFI1 driver module parameters, enter the following command:

```
$ modinfo hfi1
```

Results:

```
modinfo hfi1
filename:       /lib/modules/5.14.0-427.13.1.el9_4.x86_64/extra/ifs-kernel-updates/hfi1.ko
version:        10.14.4.0
description:    Cornelis Omni-Path Express driver
license:        Dual BSD/GPL
firmware:       hfi1_pcie.fw
firmware:       hfi1_sbus.fw
firmware:       hfi1_fabric.fw
firmware:       hfi1_dc8051.fw
rhelversion:    9.4
srcversion:     C756AD6BC8BBBD4B7DC1D08
alias:          pci:v00008086d000024F1sv*sd*bc*sc*i*
alias:          pci:v00008086d000024F0sv*sd*bc*sc*i*
depends:        rdmavt,ib_core,i2c-algo-bit
retpoline:      Y
name:           hfi1
vermagic:       5.14.0-427.13.1.el9_4.x86_64 SMP preempt mod_unload modversions
parm:           lkey_table_size:LKEY table size in bits (2^n, 1 <= n <= 23) (uint)
parm:           max_pds:Maximum number of protection domains to support (uint)
parm:           max_ahs:Maximum number of address handles to support (uint)
parm:           max_cqes:Maximum number of completion queue entries to support (uint)
parm:           max_cqs:Maximum number of completion queues to support (uint)
parm:           max_qp_wrs:Maximum number of QP WRs to support (uint)
parm:           max_qps:Maximum number of QPs to support (uint)
parm:           max_sges:Maximum number of SGEs to support (uint)
parm:           max_mcast_grps:Maximum number of multicast groups to support (uint)
parm:           max_mcast_qp_attached:Maximum number of attached QPs to support (uint)
parm:           max_srqs:Maximum number of SRQs to support (uint)
```

```
parm:           max_srq_sges:Maximum number of SRQ SGEs to support (uint)
parm:           max_srq_wrs:Maximum number of SRQ WRs support (uint)
parm:           piothreshold:size used to determine sdma vs. pio (ushort)
parm:           sge_copy_mode:Verbs copy mode: 0 use memcpy, 1 use cacheless copy, 2 adapt
based on WSS (uint)
parm:           wss_threshold:Percentage (1-100) of LLC to use as a threshold for a
cacheless copy (uint)
parm:           wss_clean_period:Count of verbs copies before an entry in the page copy
table is cleaned (uint)
parm:           sdma_comp_size:Size of User SDMA completion ring. Default: 128 (uint)
parm:           cache_size:Send and receive side cache size limit (in MB) (ulong)
parm:           sdma_descq_cnt:Number of SDMA descq entries (uint)
parm:           sdma_idle_cnt:sdma interrupt idle delay (ns,default 250) (uint)
parm:           num_sdma:Set max number SDMA engines to use (uint)
parm:           desct_intr:Number of SDMA descriptor before interrupt (uint)
parm:           qp_table_size:QP table size (uint)
parm:           pcie_caps:Max PCIe tuning: Payload (0..3), ReadReq (4..7) (int)
parm:           pcie_target:PCIe target speed (0 skip, 1-3 Gen1-3) (uint)
parm:           pcie_force:Force driver to do a PCIe firmware download even if already at
target speed (uint)
parm:           pcie_retry:Driver will try this many times to reach requested speed (uint)
parm:           pcie_pset:PCIe Eq Pset value to use, range is 0-10 (uint)
parm:           pcie_ctle:PCIe static CTLE mode, bit 0 - discrete on/off, bit 1 -
integrated on/off (uint)
parm:           num_user_contexts:Set max number of user contexts to use (default: -1 will
use the real (non-HT) CPU count) (int)
parm:           krcvqs:Array of the number of non-control kernel receive queues by VL
(array of uint)
parm:           rcvarr_split:Percent of context's RcvArray entries used for Eager buffers
(uint)
parm:           eager_buffer_size:Size of the eager buffers, default: 8MB (uint)
parm:           rcvhdrcnt:Receive header queue count (default 2048) (uint)
parm:           hdrq_entsize:Size of header queue entries: 2 - 8B, 16 - 64B, 32 - 128B
(default) (uint)
parm:           user_credit_return_threshold:Credit return threshold for user send
contexts, return when unreturned credits passes this many blocks (in percent of allocated
blocks, 0 is off) (uint)
parm:           max_mtu:Set max MTU bytes, default is 10240 (uint)
parm:           cu:Credit return units (uint)
parm:           cap_mask:Bit mask of enabled/disabled HW features
parm:           num_vls:Set number of Virtual Lanes to use (1-8) (uint)
parm:           rcv_intr_timeout:Receive interrupt mitigation timeout in ns (uint)
parm:           rcv_intr_count:Receive interrupt mitigation count (uint)
parm:           link_crc_mask:CRCs to use on the link (ushort)
parm:           loopback:Put into loopback mode (1 = serdes, 3 = external cable (uint)
parm:           aspm:PCIe ASPM: 0: disable, 1: enable, 2: dynamic (uint)
```

NOTE

ipoib_accel is not available on some older distros and is no longer required on the latest distros as the feature is enabled by default.

NOTE

kdeth_qp is no longer available.

For systems with NVIDIA GPUs installed, two additional `hfi1` parameters will be listed in the `modinfo` output:

```
parm:           gpu_cache_size:GDRCopy device Nvidia buffer cache size limit (in MB) (ulong)
parm:           nvidia_cache_size:SDMA Nvidia buffer pin cache size limit (in MB) (ulong)
```

For systems with AMD GPUs installed, these `hfi1` parameters will be listed in the `modinfo` output:

```
softdep:        pre: amdgpu
parm:           amd_cache_size:Per-context AMD pin cache size limit (in MB) (ulong)
parm:           amd_use_cache:Enabled: use user SDMA ROCm VA:DMA cache when handling
user SDMA requests. Disabled: do not use ROCm VA:DMA cache; do rdma_get_pages()/
rdma_put_pages() for each (iovec,VA) in user SDMA request. (uint)
parm:           amd_use_mmu:Enabled: use mmu_notifier to maintain user SDMA ROCm VA:DMA
cache; not applicable when amd_use_cache=0. Disabled: do not use mmu_notifier to maintain
user SDMA ROCm VA:DMA cache; do not use with amd_use_cache=1. (uint)
```

4.2. Current Values of Module Parameters

To list the current values for the module parameters, run the following short script:

```
grep . /sys/module/hfi1/parameters/*
```

Output from script (these are the default values):

```
/sys/module/hfi1/parameters/aspm:0
/sys/module/hfi1/parameters/cache_size:256
/sys/module/hfi1/parameters/cap_mask:0x4c09a08cb9a
/sys/module/hfi1/parameters/cu:1
/sys/module/hfi1/parameters/desct_intr:64
/sys/module/hfi1/parameters/eager_buffer_size:8388608
/sys/module/hfi1/parameters/hdrq_entsize:32
/sys/module/hfi1/parameters/ifs_sel_mode:0
/sys/module/hfi1/parameters/ipoib_accel:1
/sys/module/hfi1/parameters/krcvqs:2
/sys/module/hfi1/parameters/link_crc_mask:3
/sys/module/hfi1/parameters/lkey_table_size:16
/sys/module/hfi1/parameters/loopback:0
/sys/module/hfi1/parameters/max_ahs:65535
/sys/module/hfi1/parameters/max_cqes:3145727
/sys/module/hfi1/parameters/max_cqs:131071
/sys/module/hfi1/parameters/max_mcast_grps:16384
/sys/module/hfi1/parameters/max_mcast_qp_attached:16
/sys/module/hfi1/parameters/max_mtu:10240
/sys/module/hfi1/parameters/max_pds:65535
/sys/module/hfi1/parameters/max_qps:32768
```

```
/sys/module/hfi1/parameters/max_qp_wrs:16383
/sys/module/hfi1/parameters/max_sges:96
/sys/module/hfi1/parameters/max_srqs:1024
/sys/module/hfi1/parameters/max_srq_sges:128
/sys/module/hfi1/parameters/max_srq_wrs:131071
/sys/module/hfi1/parameters/num_sdma:0
/sys/module/hfi1/parameters/num_user_contexts:-1 [number of physical CPU cores]
/sys/module/hfi1/parameters/num_vls:8
/sys/module/hfi1/parameters/pcie_caps:0
/sys/module/hfi1/parameters/pcie_ctle:3
/sys/module/hfi1/parameters/pcie_force:0
/sys/module/hfi1/parameters/pcie_pset:255
/sys/module/hfi1/parameters/pcie_retry:5
/sys/module/hfi1/parameters/pcie_target:3
/sys/module/hfi1/parameters/piothreshold:256
/sys/module/hfi1/parameters/port_reorder:N
/sys/module/hfi1/parameters/qp_table_size:256
/sys/module/hfi1/parameters/rcvarr_split:25
/sys/module/hfi1/parameters/rcvhdrcnt:2048
/sys/module/hfi1/parameters/rcv_intr_count:16
/sys/module/hfi1/parameters/rcv_intr_timeout:840
/sys/module/hfi1/parameters/sdma_comp_size:128
/sys/module/hfi1/parameters/sdma_descq_cnt:2048
/sys/module/hfi1/parameters/sdma_idle_cnt:250
/sys/module/hfi1/parameters/sge_copy_mode:0
/sys/module/hfi1/parameters/user_credit_return_threshold:33
/sys/module/hfi1/parameters/wss_clean_period:256
/sys/module/hfi1/parameters/wss_threshold:80
```

NOTE

num_user_contexts value can be found in `/sys/class/infiniband/hfi1_0/nctxts`.

For systems with GPUDirect components installed, `gpu_cache_size` will also be listed in the output:

```
/sys/module/hfi1/parameters/gpu_cache_size:256
```

For systems with AMD GPUs installed, `amd_Cache_size` and `amd_use_Cache` will be listed:

```
cat /sys/module/hfi1/parameters/amd_cache_size 18446744073709551615 cat /sys/module/hfi1/
parameters/amd_use_cache 1
```

Unless otherwise specified, Cornelis recommends to use the default values for the HFI1 parameters. The following parameters are discussed throughout this document in order to improve the performance.

Parameters	Sections
pcie_caps	Section 2.1 “Intel Xeon Processor E5 v3 and v4 Families”
num_user_contexts	Section 3.5 “Memory Fragmentation”

Parameters	Sections
rcvhdrcnt	Section 5.11 "MPI Applications Performance Tuning"
cap_mask	Section 6.1 "Accelerated RDMA"
sge_copy_mode	Section 6.2 "Parallel File System Concurrency Improvement"
krcvqs	Section 6.2 "Parallel File System Concurrency Improvement"
	Section 7.3 "krcvqs Tuning for IPoFabric Performance"
num_sdma	Section 8.4.3 "Changing SDMA Engines"
gpu_cache_size	Section 5.14 "GPUDirect RDMA Tuning for MPI Benchmarks and Applications"
amd_cache_size	Section 5.15 "AMD GPU (ROCm)"
amd_use_cache	Section 5.15 "AMD GPU (ROCm)"

4.3. Setting HFI1 Driver Parameters

NOTE

The settings in this section are examples only and not recommended for general use.

To set or change the HFI1 driver module parameters, *as root* perform the following:

1. Create `hfi1.conf`, if it does not already exist. This file will include all options and parameters for the driver set by the user. These can be set either using an editing tool such `vim` or using `export`. After modifying this file, print its contents to verify the desired options were set properly.

   Example:

   ```
   $ cat /etc/modprobe.d/hfi1.conf
   options hfi1 pcie_caps=0x51 krcvqs=3
   ```

2. Determine if dracut needs to be run:
   - If the following sequence happens, run the dracut command as described in Step 3.
     a. At the start of boot, initramfs is all that is visible.
     b. The hfi1 driver is loaded while only the initramfs is visible.
     c. The `hfi1.conf` file **within the initramfs** is used.

     NOTE

     If you are using SLES Linux, dracut must be run.

- If one of the following happens, then the dracut command is not needed. Skip to Step 4.
  - If you reload the driver while the OS is running, the initramfs is not used.
  - If the hfi1 driver is loaded after the initramfs stops being used, then the initramfs is not used.

3. Run the `/usr/bin/dracut -f` command to force `/etc/modprobe.d/hf1.conf` into the initramfs image.

```
$ /usr/bin/dracut -f
```

4. Reboot the system.

After the system comes up from the reboot, you can run the script listed in Section 4.2 "Current Values of Module Parameters"to see if your changes to `hfi1.conf` took effect.

The `krcvqs=3` setting only affects the first virtual lane (VL). To set a krcvqs value of 3 in case eight VLs were set in the Fabric Manager, the krcvqs values would look like (note that this is not typical):

```
options hfi1 krcvqs=3,3,3,3,3,3,3,3
```

4.4. Dual/Multi-Rail Tuning

This section provides tuning guidance for users who are configured for dual/multi-rail use cases.

NOTE

For more information on the dual/multi-rail feature, refer to *Cornelis Omni-Path Express Fabric Software Installation Guide*, *Multi-Rail Overview*.

4.4.1. General Discussion

The HFI1 driver assigns SDMA and `krcvq` interrupts per HFI installed in a server. Therefore, it may be beneficial to tune `num_sdma` and `krcvqs` driver parameters based on the following:

- The number of CPU cores
- How many HFIs are installed
- On which NUMA nodes the HFIs are installed

Refer to Section 8 "Driver IRQ Affinity Assignments" for more details.

In general, the HFI1 driver avoids overlapping CPU cores for all `krcvqs`, then overlaps the remaining CPU cores for the SDMA interrupts. You cannot always avoid overlap, but should where possible.

4.4.2. NUMA Location of HFIs

Cornelis recommends that you install each HFI on a separate NUMA node, if possible. Since the default behavior of the HFI driver is to use only the NUMA-local CPU cores for the HFI interrupts, more cores will be used for driver interrupt processing.

For some latency-sensitive MPI applications, having HFIs installed on separate NUMAs allows PSM ranks to communicate directly into the fabric without crossing the inter-socket link (UPI/QPI), therefore reducing latency.

4.4.3. Tuning of krcvqs and num_sdma

Increasing `krcvqs` above the default of `2` (up to `4`, for example) is a popular tuning strategy to improve receive performance, especially for storage servers with verbs protocols. However, this tuning must be considered carefully if more than one HFI are on the same NUMA node. It is possible that the default setting of `krcvqs=2` on a dual rail system will provide similar performance to `krcvqs=4` on a single rail system (four cores used for receive in both scenarios). You should select the total number of `krcvqs` based on the total network demand of traffic flowing into the server. Also, when two HFIs are installed on the same NUMA, reducing `num_sdma` to `8` may prevent overlap of any SDMA interrupts and improve performance of traffic being sent out of the server.

NOTE

The overlapping of interrupts discussed in this section occurs less or is not applicable when the HFIs are installed on separate CPU sockets.

The following table shows an example of the interrupt mapping differences between single and dual rail for a 16 core CPU (0-15). A single HFI is installed on NUMA 0 and the default driver parameters `num_sdma=16` and `krcvqs=2`. The default driver behavior would be to use core 0 for the general interrupt, core 1-2 for `krcvqs`, and core 3-15 for SDMA interrupts, then wrapping back to 3-5 for the remaining SDMA interrupts. In this scenario, three cores are recycled for SDMA usage. This limited wrapping will not likely be problematic for performance.

If two HFIs are installed on the same NUMA node, the interrupt allocation would begin to overlap CPU cores more aggressively. Since there is no overlap for `krcvqs`, only 11 unique cores are available to wrap all 32 SDMA interrupts.

Table 6. Single and Dual HFI Layout on a 16-Core CPU

	Single HFI	Two HFIs on NUMA 0	
core	hfi1_0	hfi1_0	hfi1_1
0	kctxt0	kctxt0	kctxt0
1	kctxt1	kctxt1	
2	kctxt2	kctxt2	
3	sdma0, sdma13		kctxt1

	Single HFI	Two HFIs on NUMA 0	
4	sdma1, sdma14		kctxt2
5	sdma2, sdma15	sdma0,sdma11	sdma6
6	sdma3	sdma1,sdma12	sdma7
7	sdma4	sdma2,sdma13	sdma8
8	sdma5	sdma3,sdma14	sdma9
9	sdma6	sdma4,sdma15	sdma10
10	sdma7	sdma5	sdma0,sdma11
11	sdma8	sdma6	sdma1,sdma12
12	sdma9	sdma7	sdma2,sdma13
13	sdma10	sdma8	sdma3,sdma14
14	sdma11	sdma9	sdma4,sdma15
15	sdma12	sdma10	sdma5

In the next table, the configuration decreases `num_sdma` to `8` leaving five cores unused in the single rail case. This is not a good use of resources. However, for the dual rail case, it does reduce the amount of overlap and may provide a performance benefit.

Table 7. Single and Dual HFI Layout on a 16-Core CPU with num_sdma=8

	Single HFI	Two HFIs on NUMA 0	
core	hfi1_0	hfi1_0	hfi1_1
0	kctxt0	kctxt0	kctxt0
1	kctxt1	kctxt1	
2	kctxt2	kctxt2	
3	sdma0		kctxt1
4	sdma1		kctxt2
5	sdma2	sdma0	sdma3
6	sdma3	sdma1	sdma4
7	sdma4	sdma2	sdma5
8	sdma5	sdma3	sdma6
9	sdma6	sdma4	sdma7
10	sdma7	sdma5	
11		sdma6	
12		sdma7	
13			sdma0
14			sdma1

	Single HFI	Two HFIs on NUMA 0	
15			sdma2

Alternatively, if `krcvqs=4` is specified as shown in the following table, the overlap would be increased. Note how heavily overloaded cores 9-15 are for the dual rail scenario. This scenario will likely cause performance problems and is not recommended.

Table 8. Single and Dual HFI Layout on a 16-Core CPU with krcvqs=4

	Single HFI	Two HFIs on NUMA 0	
core	hfi1_0	hfi1_0	hfi1_1
0	kctxt0	kctxt0	kctxt0
1	kctxt1	kctxt1	
2	kctxt2	kctxt2	
3	kctxt3	kctxt3	
4	kctxt4	kctxt4	
5	sdma0,sdma11		kctxt1
6	sdma1,sdma12		kctxt2
7	sdma2,sdma13		kctxt3
8	sdma3,sdma14		kctxt4
9	sdma4,sdma14	sdma0,sdma7,sdma14	sdma5,sdma12
10	sdma5	sdma1,sdma8,sdma15	sdma6,sdma13
11	sdma6	sdma2,sdma9	sdma0,sdma7,sdma14
12	sdma7	sdma3,sdma10	sdma1,sdma8,sdma15
13	sdma8	sdma4,sdma11	sdma2,sdma9
14	sdma9	sdma5,sdma12	sdma3,sdma10
15	sdma10	sdma6,sdma13	sdma4,sdma11

Now, consider a 22-core CPU example as shown in the next table. With a single HFI, no interrupts overlap; this is the ideal scenario. For two HFIs on the same NUMA with all default driver parameters, core 5-17 are servicing SDMA for both `hfi1_0` and `hfi1_1`. However, if `num_sdma` is reduced to `8` (the minimum supported by hfi1), all overlap is completely avoided and it is possible that the driver will operate more efficiently and improve performance.

Table 9. Single and Dual HFI Layout on a 22-Core CPU

	Single HFI	Two HFIs on NUMA 0, num_sdma=16 (default)		Two HFIs on NUMA 0, num_sdma=8	
core	hfi1_0	hfi1_0	hfi1_1	hfi1_0	hfi1_1
0	kctxt0	kctxt0	kctxt0	kctxt0	kctxt0
1	kctxt1	kctxt1		kctxt1	

	Single HFI	Two HFIs on NUMA 0, num_sdma=16 (default)		Two HFIs on NUMA 0, num_sdma=8	
2	kctxt2	kctxt2		kctxt2	
3	sdma0		kctxt1		kctxt1
4	sdma1		kctxt2		kctxt2
5	sdma2	sdma0	sdma1	sdma0	
6	sdma3	sdma1	sdma2	sdma1	
7	sdma4	sdma2	sdma3	sdma2	
8	sdma5	sdma3	sdma4	sdma3	
9	sdma6	sdma4	sdma5	sdma4	
10	sdma7	sdma5	sdma6	sdma5	
11	sdma8	sdma6	sdma7	sdma6	
12	sdma9	sdma7	sdma8	sdma7	
13	sdma10	sdma8	sdma9		sdma0
14	sdma11	sdma9	sdma10		sdma1
15	sdma12	sdma10	sdma11		sdma2
16	sdma13	sdma11	sdma12		sdma3
17	sdma14	sdma12	sdma13		sdma4
18	sdma15	sdma13			sdma5
19	<empty>	sdma14			sdma6
20	<empty>	sdma15			sdma7
21	<empty>		sdma0		<empty>

4.5. Monitoring HFI Usage

In multi-HFI installations, it is sometimes desirable to ensure that both HFIs are utilized evenly. There may be scenarios in which there is an HFI usage imbalance, causing reduced performance. There are two ways to monitor HFI usage:

1. opatop
   - Run `opatop`.
   - Press (0) to select ALL.
   - Press (p) to select Group Performance.
   - Press (d) to enter the detailed view. This will show HFI utilization for the entire fabric in sorted order, similar to the Linux top. Look for the utilization values of `hfi1_0` and `hfi1_1` for the nodes of interest.
2. opainfo

- Run `opainfo` on the specific hosts of interest. Record the values of `Xmit Data` and `Recv Data` for both `hfi1_0` and `hfi1_1`.
- Run the desired benchmark.
- Rerun `opainfo`. Compare the new `Xmit Data` and `Recv Data` values for both `hfi1_0` and `hfi1_1` to those recorded before the benchmark. The difference is the amount of data sent/received by that HFI.

5. MPI Performance

MPI libraries are a key type of middleware for building HPC applications. This chapter discusses how to load an MPI into your user environment as well as how to run simple MPI benchmarks to measure fabric performance. While users may chose to use an alternative MPI such as the Intel MPI Library bundled with Intel OneAPI, the Omni-Path Express Fabric Software package includes builds of Open MPI and MVAPICH2.

5.1. Selecting Open MPI or MVAPICH2

Open MPI and MVAPICH2 are located in subdirectories of the `/usr/mpi` directory. They can be located using the `ls -r` command as shown below:

```
/usr/mpi] $ ls -r *
gcc:
openmpi-<version>-hfi mvapich2-<version>-hfi
```

The **gcc** directory shows the Gnu Compiler Collection (GCC) used to build the MPI library.

For best performance, run MPIs over the OPX provider or the PSM2 library included with the Host Software.

To run MPIs over the OPX provider or PSM2 library:

1. Use the MPIs with **hfi** in their name.
2. Source a `mpivars.sh` file from the `bin` directory of one of the MPIs from your Linux shell's startup scripts.

   For example, include the following statement in a startup script such as `.bashrc`:

   ```
   source /usr/mpi/gcc/openmpi-<version>-hfi/bin/mpivars.sh
   ```

   This will set the PATH and LD_LIBRARY_PATH and MANPATH variables for this MPI version.
3. Use the options in your `mpirun` command to specify the use of OPX or PSM2. For example, to use the Open MPI version indicated in step 2 with PSM2, use:

   ```
   mpirun -mca pml cm -mca btl self,vader -mca mtl psm2 …
   ```

   And to specify the use of OPX, use:

   ```
   mpirun -mca mtl ofi -x FI_PROVIDER=opx -mca btl self,vader ...
   ```

NOTE

If you use the MVAPICH2 library with hfi in the name of its root directory (such as `/usr/mpi/gcc/mvapich2-x.y-hfi`), then no special `mpirun` options are needed to use the PSM2 library. Currently the version of MVAPICH packaged with OPXS only supports PSM2.

5.2. Intel MPI Library Settings

NOTE

The information in this section assumes the use of MPI Library, unless otherwise specified.

For best performance, Cornelis recommends that you use the OPX provider or PSM2. This is accomplished using the Open Fabrics Interface (OFI) MPI fabric setting `-genv I_MPI_FABRICS=shm:ofi` and ensure that `FI_PROVIDER=` is set to either OPX or PSM2.

For more details on the `I_MPI_FABRICS` values, refer to the *Intel MPI Library Developer Reference for Linux OS* found at https://software.intel.com/en-us/mpi-developer-reference-linux. To ensure that the Intel MPI fabric or provider is what you expect (especially that PSM2 is the provider for OFI, use `-genv I_MPI_DEBUG=5` option to view the debug output.

5.3. Verification of Fabric Selection

In order to validate the MPI is using the desired fabric, set `PSM2_IDENTIFY=1` and confirm via standard output which PSM2 library is chosen. Alternatively, for the opx provider, set `FI_LOG_LEVEL=trace` and confirm the requested libfabric provider (either opx or psm2) is used.

5.4. Enabling Explicit Huge Pages for Shared Memory Communication with Intel MPI Library

When performing shared memory (shm) communication with theIntel MPI Library, users may see a drop in bandwidth at large message sizes such as 32MB and higher. As of Intel MPI Library 2019 update 1 , assuming the system administrator has enabled a huge page filesystem (see https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt), users can request that the Intel MPI Library use huge pages with one of the following environment variables:

```
export I_MPI_SHM_FILE_PREFIX_2M=/path/to/2Mhugepages
```

Other size huge pages are also supported, such as `I_MPI_SHM_FILE_PREFIX_4K` and `I_MPI_SHM_FILE_PREFIX_1G`. For more detail, refer to the Intel MPI Library documentation.

5.5. MPI Benchmark Fundamentals

Two common types of benchmarks are used to measure MPI performance: OSU Micro-Benchmarks (OMB) (https://mvapich.cse.ohio-state.edu/benchmarks/) and Intel MPI Benchmarks (IMB) (https://github.com/intel/mpi-benchmarks). In general, the goal of these benchmarks is to measure point-to-point performance (latency, bandwidth, and message rate) between two nodes. Additionally, MPI collectives performance can be measured using a large group of nodes.

For simplicity, this section demonstrates how to run the Intel MPI Benchmarks using Open MPI as packaged with OPX-OPXS to measure latency, bandwidth, and message rate. We also use the version of IMB-MPI1 that is packaged with Omni-Path Express Fabric Suite. For more information, refer to the *Intel MPI Benchmarks User Guide*.

To begin, load Open MPI into the environment:

```
source /usr/mpi/gcc/openmpi-<version>-hfi/bin/mpivars.sh
```

NOTE

You must have password-less ssh enabled between all nodes where you want to run benchmarks.

5.5.1. MPI Latency

MPI latency is measured between two nodes using one core (MPI rank) per node.

```
mpirun -np 2 --map-by ppr:1:node -host hostA,hostB
/usr/mpi/gcc/openmpi-<version>-hfi/tests/IMB-4.0/IMB-MPI1 Pingpong
...
#---------------------------------------------------
# Benchmarking PingPong
# #processes = 2
#---------------------------------------------------
#bytes #repetitions t[usec] Mbytes/sec
...
```

The resulting output in the third column (`t[usec]`) is latency as a function of message size. Typically, 8-byte latency is used for performance analysis. The analogous benchmark with OMB is called `osu_latency`. Sometimes, the MPI rank needs to be pinned to a certain CPU socket in order to achieve the best latency (see Section 5.9 "MPI Affinity and HFI Selection"). Note that the bandwidth returned is from a single buffer (mpi_send/recv) and does not fully stress the throughput capability of the network.

5.5.2. MPI Bandwidth

MPI bandwidth is measured between two nodes using one or more ranks per node. As you use more ranks per node, the aggregate bandwidth increases for lower message sizes. We use the `Uniband` and `Biband` benchmarks for MPI bandwidth measurements because

they perform many simultaneous, non-blocking sends and are able to *stream* messages continuously and saturate the network.

The following is an example of one rank per node for uni-directional bandwidth:

```
mpirun -np 2 --map-by ppr:1:node -host hostA,hostB
/usr/mpi/gcc/openmpi-<version>-hfi/tests/IMB-4.0/IMB-MPI1 Uniband
...
#---------------------------------------------------
# Benchmarking Uniband
# #processes = 2
#---------------------------------------------------
#bytes #repetitions Mbytes/sec Msg/sec
...
```

The third column (`Mbytes/sec`) reports MPI bandwidth. The fourth column (`Msg/sec`) is message rate (discussed in the next section).

To run a bidirectional bandwidth test, replace `Uniband` with `Biband` in the example above.

The analogous benchmarks in OMB are `osu_bw` and `osu_bibw`.

The following example shows how to run both Uniband and Biband simultaneously, using four MPI ranks per node:

```
mpirun -np 8 --map-by ppr:4:node -host hostA:4,hostB:4
/usr/mpi/gcc/openmpi-<version>-hfi/tests/IMB-4.0/IMB-MPI1 Uniband
Biband -npmin 8
```

NOTE

The `npmin 8` flag is required to ensure that exactly four communicating pairs are running, the first four ranks on nodeA and the second four ranks on nodeB.

The analogous benchmark in OMB is `osu_mbw_mr`. There is no equivalent bidirectional benchmark.

5.5.3. MPI Message Rate

Message rate is also measured with `Uniband` and `Biband` benchmarks, but using as many ranks per node as there are cores on the node. The message rate is the total number of MPI messages (typically eight bytes) sent between the two nodes. This is a derived quantity that can be calculated from the bandwidth output.

If we have nodes with 32 physical cores per node,

```
mpirun -np 64 --map-by ppr:32:node -host hostA:32,hostB:32
/usr/mpi/gcc/openmpi-<version>-hfi/tests/IMB-4.0/IMB-MPI1 Uniband
-npmin 64
...
#---------------------------------------------------
# Benchmarking Uniband
# #processes = 64
```

```
#-----------------------------------------------
#bytes #repetitions Mbytes/sec Msg/sec
...
```

The fourth column (`Msg/sec`) is the message rate and is typically quoted for eight bytes. The same method can be used to measure bidirectional message rate with the Biband benchmark.

Maximum MPI message rate may be influenced by an OPX parameter controlling the frequency of HFI receive header queue register updates. Unfortunately, the best value may vary between CPU architectures. This parameter can currently be adjusted by re-compiling the OPX provider with the `DFI_OPX_HFI1_HDRQ_UPDATE_MASK` directive. The new default is `DFI_OPX_HFI1_HDRQ_UPDATE_MASK=64`, which is changed by previous defaults of `-DFI_OPX_HFI1_HDRQ_UPDATE_MASK_1024`.

5.5.4. MPI Collectives

Performance can be measured for a variety of collectives such as Allreduce. These benchmarks can be run between many nodes. For example, a 128 node, 32 rank per node Allreduce can be run with the following command:

```
mpirun -np $((128*32)) --map-by ppr:32:node -hostfile 128hosts
/usr/mpi/gcc/openmpi-<version>-hfi/tests/IMB-4.0/IMB-MPI1 Allreduce
-npmin $((128*32))
...
#---------------------------------------------------------------- #
Benchmarking Allreduce # #processes = 4096
#----------------------------------------------------------------
#bytes #repetitions t_min[usec] t_max[usec] t_avg[usec]
...
```

Typically, `t_avg` latency gives a good idea of the performance of the system. Sometimes, large deviations between `t_min` and `t_max` can indicate suboptimal performance and perhaps system-jitter effects (see Section 5.17 “Reducing System Jitter”).

5.6. MPI Collective Tunings

Cornelis recommends using the latest Intel MPI Librarywhen possible for optimized MPI collectives performance. The following table is a collection of additional tuning recommendations.

Collective	MPI	Tuning	Notes
96 ppn Alltoall, 4 nodes	Open MPI (3.1.4, 4.0.3, ...)	`-mca coll_tuned_alltoall_algorithm 3` `-mca coll_tuned_use_dynamic_rules 1`	Significantly improves Alltoall performance for 512 to 2048 bytes.

5.7. Tuning for the OFI Fabric

Omni-Path Express V10.5 or newer will, by default, install OFI (also known as libfabric) on your nodes. libfabric is evolving. Therefore, you may get better performance from the latest version found at https://ofiwg.github.io/libfabric/.

Intel MPI Library are packaged with self-contained libfabric and should be used without any additional required flags. For earlier versions of Intel MPI or libfabric, the following two environment variables may be required to improve performance or stability:

- `FI_PSM2_LOCK_LEVEL=1`
- `FI_PSM2_DELAY=0`

If you are running applications or benchmarks (such as IMB-RMA) that use one-sided operations compliant with the MPI-3 standard, the direct RMA mode for OFI will improve performance of MPI_Put and MPI_Get operations. To turn direct RMA mode on, use the `I_MPI_OFI_DIRECT_RMA=on` environment variable.

The PSM2 Multi-Endpoint (Multi-EP) feature in Intel MPI Library enables multiple threads to be active in the MPI library, allowing a single MPI rank to use multiple threads through `MPI_THREAD_MULTIPLE`. For applications not intended to use Multi-EP, if using libfabric 1.6 or newer, included with Omni-Path Express V10.8, set `PSM2_MULTI_EP=0` to enable context sharing. Typically this is required when trying to use more than 1 MPI rank/PSM context per core.

5.8. Scalable Endpoints with Open MPI

The Open Fabrics Interface (OFI) specification defines Scalable Endpoints (SEP) as a communication portal that supports multiple transmit and receive contexts in a single process. This allows scaling transmit/receive side processing by using multiple queues for data transfer. Intel MPI Library also supports SEP with a feature known as Multi-EP. This section explains how to take advantage of SEP in a similar method using Open MPI.

Using the master branch of Open MPI (https://github.com/open-mpi/ompi) as well as the libfabric release (https://github.com/ofiwg/libfabric/releases/tag/v1.7.1), performance gains have been shown using the SEP feature with Intel MPI Benchmarks 2019 IMB-MT benchmarks. Open MPI needs to be compiled with the `--enable-mpi-thread-multiple` option. When running the IMB-MT benchmark or any other application written to take advantage of SEP, include the following Open MPI parameters during runtime:

```
-x OMP_NUM_THREADS=$nt -mca mtl_ofi_enable_sep 1 -mca mtl_ofi_thread_grouping 1 -mca
opal_max_thread_in_progress $((nt+1))
-mca mtl_ofi_num_ctxts $((nt+1))
```

where `$nt` is the desired number of threads per MPI rank.

For example, to run the IMB-MT Uniband benchmark to measure unidirectional bandwidth for the 1MB message size, with one thread as a baseline:

```
mpirun -np 2 --map-by ppr:1:node -host host1,host2 -mca pml cm -mca mtl ofi
-mca btl ^openib,ofi -mca mtl_ofi_provider_include psm2 -x OMP_NUM_THREADS=1 -mca
```

```
mtl_ofi_enable_sep 1 -mca mtl_ofi_thread_grouping 1 -mca opal_max_thread_in_progress 2
-mca mtl_ofi_num_ctxts 2 ./IMB-MT UnibandMT
-thread_level multiple -datatype char -count 1048576
```

And using eight threads, we divide 1048576 into eight chunks of 128KB:

```
mpirun -np 2 --map-by ppr:1:node -host host1,host2 -mca pml cm -mca mtl ofi
-mca btl ^openib,ofi -mca mtl_ofi_provider_include psm2 -x OMP_NUM_THREADS=8 -mca
mtl_ofi_enable_sep 1 -mca mtl_ofi_thread_grouping 1 -mca opal_max_thread_in_progress 9
-mca mtl_ofi_num_ctxts 9 ./IMB-MT UnibandMT
-thread_level multiple -datatype char -count 131072
```

More detail can be found here: https://github.com/open-mpi/ompi/blob/master/ompi/mca/mtl/ofi/README.md

5.9. MPI Affinity and HFI Selection

In multi-HFI systems, where each unit is connected to a different CPU socket, the choice of the HFI with respect to the affinity of the MPI process has a measurable impact on performance. For example, latency-sensitive applications that use the HFI on the remote NUMA node will incur a performance cost related to memory/cache locality of the MPI process and an additional delay related to inter-NUMA interconnect traffic.

To minimize the effects described above, PSM2 defaults to using the HFI that is local to the MPI process. For example, in a compute node with two 14-core CPUs, (cores 0-13 on socket 0, 14-27 on socket 1), a typical `mpirun` command line that specifies the MPI process affinity is represented below as:

```
mpirun -mca pml cm -mca mtl psm2 -H node01,node02 taskset -c 14 ./osu_latency
```

Here the utility affinitizes the MPI process to the first core on socket 1. The command line above will default to using the HFI in socket 1. However, you can still choose the HFI connected to socket 0 using the environment variable `HFI_UNIT`.

```
mpirun -mca pml cm -mca mtl psm2 -H node01,node02 -x HFI_UNIT=0 taskset -c 14 ./osu_latency
```

5.9.1. Using MPI Multiple Endpoints with Intel MPI

The PSM2 Multi-Endpoint (Multi-EP) feature in Intel MPI enables multiple threads to be active in the MPI library. This allows a single MPI rank to use multiple threads through MPI_THREAD_MULTIPLE.

The Multiple-Endpoint feature of the Intel MPI Library can be initialized with the following:

```
source $I_MPI_ROOT/intel64/bin/mpivars.sh release_mt
export I_MPI_THREAD_SPLIT=1
export I_MPI_THREAD_RUNTIME=openmp
export OMP_PLACES=cores
export PSM2_MULTI_EP=1
```

Then run an example benchmark packaged with the Intel MPI specifically designed to use the Multi-EP feature (IMB-MT) as shown in the following example.

```
export size=1048576
export omp=4
mpirun -np 2 -ppn 1 -host hostA,hostB -genv I_MPI_FABRICS=shm:ofi -genv
OMP_NUM_THREADS=$omp $I_MPI_ROOT/intel64/bin/IMB-MT -thread_level multiple -datatype char
-count $((size/omp))
```

In the previous example, a total message size of 1048576 bytes is sent. Each of the four threads sends a size of 262144 bytes per thread (1048576 divided by 4).

For further guidance on using the IMB-MT binary, refer to the Intel MPI documentation.

For applications not intended to use Multi-EP: If you are using libfabric 1.6 or newer (included with Omni-Path Express V10.8), set PSM2_MULTI_EP=0 to enable context sharing. Typically, this is required when trying to use more than one MPI rank/PSM context per core.

5.10. Tuning for High-Performance LINPACK Performance

High-Performance LINPACK (HPL) is a software package that solves a uniformly random, dense system of linear equations in double precision (64-bits) on distributed-memory computers. HPL reports time and floating-point execution rate. The HPL implementation of the HPC LINPACK Benchmark is portable and freely available (see http://www.netlib.org/benchmark/hpl/).

Intel Parallel Studio XE contains pre-compiled versions of HPL (referred to as the Distribution for LINPACK Benchmark) that are optimized using Math Kernel Library tools for best performance. The pre-compiled binaries are also processor- and platform-aware; they automatically choose the most optimal core pinning and other environment variables at runtime.

Visit https://software.intel.com/en-us/mkl-windows-developer-guide-intel-distribution-for-linpack-benchmark for more details and documentation, including performance optimization techniques.

5.10.1. Expected Levels of Performance

The peak flops for a node is given by the simple formula:

```
Rpeak = Ncores * GHz(base) * (operations/cycle)
```

Intel Xeon Scalable Processor deliver 32 operations/cycle and Intel Xeon Processor v3 and v4 families deliver 16, double-precision floating-point operations/cycle per CPU core. The expected level of performance is only a percentage of the peak flops, and this efficiency varies depending on the exact processor.

IMPORTANT

Before doing any multi-node runs, be sure that individual node performance is evaluated and is in the expected range for the processor. The multi-node HPL calculation is throttled by the lowest performing node.

5.10.2. Selection of HPL Binary and MPI

Cornelis recommends that you use the pre-compiled version of HPL and scripts contained within Parallel Studio XE, known as the Distribution for LINPACK Benchmark. Currently, these are the versions located in the `<path to latest compilers_and_libraries installation>/linux/mkl/benchmarks/mp_linpack/` directory tree.

Be sure to use the matching MPI Library.

To load the library into your environment:

```
source /opt/intel/<path to latest parallel studio>/bin/psxevars.sh
```

5.10.3. MPI Flags and Proper Job Submission Parameters/Syntax

No Omni-Path Express-specific MPI or PSM2 parameters are required to achieve optimized HPL performance onOmni-Path Express. Refer to Section 5.2 "Intel MPI Library Settings" for the recommended OFI settings to ensure PSM2 is being used. Note that as of Intel MPI Library2019, only OFI fabric is supported. It is also important to use the scripts provided with the binaries for proper CPU pinning and NUMA-aware optimizations. See https://software.intel.com/en-us/mkl-windows-developer-guide-intel-distribution-for-linpack-benchmark for details.

Typically, the Intel-provided binary and scripts provide good performance by automatically choosing optimal MPI and thread affinity. However, you still must execute the binary and example scripts requesting the appropriate number of MPI ranks and MPI ranks per node.

- On dual socket Intel Xeon Processor systems, the recommendation is to use one MPI rank per NUMA node.

During an HPL run, if you run the Linux top command on the node, you should see multiple cores used within one MPI process. On a dual socket, 16-core Intel Xeon Processor system, you would see two HPL processes each consuming 1600% of CPU resources. This implies that the threads are active within each MPI task.

NOTE

It is strongly recommended that you do not run performance benchmarks, including HPL, *as root*. This is because clean-up after improperly terminated runs can be cumbersome. Cornelis recommends running these benchmarks with a regular *user* account. If it is necessary to clean up rogue or zombie processes, a simple `pkill -u` *user* command can be issued across the cluster, unlike if root was running the benchmarks.

5.10.4. HPL.dat Input File

Detailed tuning of the HPL.dat file is not generally required to achieve the bulk of the expected performance and is out of the scope of this document.

NOTE

An online tool can be used to generate an example HPL.dat file.

The main parameters in HPL.dat file are:

- The "problem size" *N* dictates the solution of an *NxN* matrix, which determines the total amount of memory used.
  - To estimate the total amount of memory for a given *N*, use this formula:

    `GB = (N*N*8)/2`^{30}

    where

    `GB` is the total amount of memory required in GibiBytes

    `8` is due to the size of an 8 byte word,

    2^{30} is the number of bytes in Gibibytes.
  - Conversely, to determine *N* for a desired memory usage, use this formula:

    `N= SQRT((GB*` 2^{30}`)/8)`
  - Typically, we use 60 to 90 % of the system memory (`%M = 0.6-0.9`) for the most optimal HPL score. Then, for a multi-node (*C* nodes) where NODEMEM is the total memory per node in Gibibytes, the cluster-level calculation is:

    `N = ROUND(SQRT((C * %M * NODEMEM *` 2^{30}`)/8))`

    where `ROUND` is a function that rounds to a nearby integer.
- The "block size" *NB*: An optimal block size that typically will return the best performance and no adjustment is necessary. Optimal block sizes are defined for each processor below:
  - Intel Xeon Processor E5 v3 Family and v4 Family: 192
  - Intel Xeon Scalable Processor: 384
- "*Ps* and *Qs*": *P* and *Q*. *P* and *Q* govern the dimension of the problem matrix.

  Selecting *P* and *Q* are somewhat of a "fine-tuning" but always make sure `P ~ Q` and `P * Q = number of MPI ranks`.

  The best P and Q are determined by the balance of computation and network performance. If higher computational performance is required, a larger P is required. If the network becomes the bottleneck, then a larger Q is required to reduce vertical communication in the matrix. For example, if a node has add-in coprocessor cards, the best performance may be seen if `P > Q`.

Very large-scale calculations on Intel Xeon Phi Processors have shown that the best scaling occurs where `Q ~ 4*P`. For instance, for a 1024-node calculation using one MPI rank per node, set `P=16` and `Q=64`.

You can use the FastFabric tool, `/usr/src/opa/mpi_apps/hpl_dat_gen`, to generate a starting HPL.dat file. The TUI command, when executed, prompts for the number of nodes, cores per node, memory per node, and memory pressure between 30 and 90%. The output may need to be adjusted further for fine-tuning and peak performance.

5.10.5. Recommended Procedure for Achieving Optimized HPL Performance

The following procedure is recommended for running multi-node HPL:

1. Follow the previous sections to set up binaries, run scripts, and HPL.dat file for the target node count. Note that you can pass arguments after the binary so you can adjust parameters on the fly, but HPL.dat still needs to be present in the directory. Make sure you save the log files for all runs for future reference if required.
2. Perform a single node HPL run on each node you plan to use in the multi-node run.

   If you are able to choose from a group of nodes greater than the target multi-node count and eliminate the lowest performing nodes, you may achieve a greater multi-node score. For example, if you want to perform a 128-node HPL run and have 144 nodes available to test, remove the lowest 16 performing nodes for your final multi-node run.

   Run at least ten sequential single node HPL for every node in order to identify any potential problems with a particular node. If the performance drops more than 5% for even one of the ten runs, eliminate that node from the multi-node run even if it recovered in subsequent single node runs. To save time, you can run individual instances of HPL on all of the nodes simultaneously.

   The score in GFlops is reported in the right-most column of the program output. You should check this to make sure the performance is as expected for the particular CPU you are using. A general rule of thumb is ±5% as an acceptable performance range. If nodes fall below this range, they should not be used in the multi-node run.

   Note the run time increases appreciably with increasing *N*, especially for multi-node studies, so it is important to weigh the benefit of increasing *N* for both score and runtime allowance.
3. Perform a multi-node run with all of the selected hosts.

   For very large scale multi-node runs (up to thousands), the efficiency may be up to 5–10% lower than the lowest single node efficiency used in the run. For smaller cluster runs, the fabric impact on efficiency should be less. Between every multi-node run, ensure there are no zombie processes or other unwanted processes running on the nodes. If time allows, it is also a good idea to revisit the single node HPL study described above (which takes minutes), between large scale tests (which may take multiple hours).

5.11. MPI Applications Performance Tuning

Certain non-default settings for either HFI1 driver parameters or PSM2 , OPX Provider, or MPI environment variables may improve HPC applications performance. The following tunings have been tested on the indicated processor types with positive results.

Table 10. Omni-Path Express Fabric Suite Software Release 10.5 or Newer

Application	Processor	Tuning Parameters
ANSYS Mechanical 18.0, Case=bga	Intel Xeon Processor E5 v4 Family	`I_MPI_ADJUST_GATHERV=3` [1]
LSTC's LS-DYNA, test models: Neon, Car2Car, and 3Cars	Intel Xeon Processor E5 v4 Family Intel Xeon Scalable Processor	`rcvhdrcnt=8192` [2]
OpenFOAM	Intel Xeon Scalable Processor	`I_MPI_FABRICS=ofi` [1]
OpenFOAM: motorbike42m, OpenMPI	3rd Gen AMD EPYC Processor	`-mca coll_tuned_use_dynamic_rules true -mca coll_tuned_allreduce_algorithm`[5]
115.fds4[3], Fire Dynamics Simulator 4	Intel Xeon Scalable Processor	`rcvhdrcnt=4096` [2]
		`MPIR_CVAR_CH4_OFI_ENABLE_DATA=0` Required for Intel OneAPI. May provide 35% or more performance improvement on Scalable processors.
ANSYS Fluent	Intel Xeon Scalable Processor	`-mpi=intel -pib.infinipath` [4] Allocate an extra node as host node to avoid the dirty file buffer issue associated with large input files.
GROMACS	Intel Xeon Processor E5 v4 Family	`PSM2_BOUNCE_SZ=4096` [5]
SPEC MPI2007	Intel Xeon Processor E5 v4 Family Intel Xeon Scalable Processor	`I_MPI_COMPATIBILITY=3` [6]
Specfem3d Global: Regional Sgloberani, Intel MPI	3rd Generation Intel Xeon Scalable Processor	`I_MPI_FABRICS=ofi`
Weather Research & Forecasting Model	3rd Generation Intel Xeon Scalable Processor	`FI_OPX_SDMA_BOUNCE_BUF_THRESHOLD=1048576` `FI_OPX_RZV_THRESHOLD=131072`
Quantum ESPRESSO	3rd Generation Intel Xeon Scalable Processor	`FI_OPX_RZV_THRESHOLD=131072`

Application	Processor	Tuning Parameters
TeaLeaf: bm5	3rd Generation Intel Xeon Scalable Processor	`I_MPI_FABRICS=ofi`
QCD Applications	Intel Xeon Scalable Processor	• `PSM2_RTS_CTS_INTERLEAVE=1` [7] • `PSM2_AVX512=0` [8] • Refer to Section 3.5 "Memory Fragmentation" for additional tips that can aid QCD performance.
MPI Message Rate	Intel Xeon Scalable Processor	`PSM2_AVX512=0` has been shown to improve MPI message rate for high core count CPUs.
MPI Bandwidth	3rd Generation Intel Xeon Scalable Processor	`PSM2_MAX_PENDING_SDMA_REQS=127` may result in more consistent peak MPI bandwidth in benchmarks such as `osu_bw` and `osu_bibw` or IBM Uniband and Biband. This variable does not appear to have a significant impact on application performance.

	5th Generation Intel Xeon Scalable Processor	`OPX: FI_OPX_RZV_MIN_PAYLOAD_BYTES=16384`
		`FI_OPX_SDMA_MIN_PAYLOAD_BYTES=8192`
		`PSM2: PSM2_MAX_PENDING_SDMA_REQS=64`
		`PSM2_MQ_RNDV_HFI_WINDOW=1048576`
		`PSM2_MQ_RNDV_HFI_THRESH=64000`
		`PSM2_MQ_EAGER_SDMA_SZ=16384 PSM2_BOUNCE_SZ=65536`

	3rd Gen AMD EPYC Processor	Bandwidth may be sensitive to Speculative Return Stack Overflow mitigations. Disabling these mitigations may improve performance.[10]

	5th Gen AMD EPYC Processor	`OPX: FI_OPX_RZV_MIN_PAYLOAD_BYTES=4096`
		`FI_OPX_SDMA_MIN_PAYLOAD_BYTES=4096`
		`PSM2: PSM2_MAX_PENDING_SDMA_REQS=24`
		`PSM2_MQ_RNDV_HFI_WINDOW=1048576`
		`PSM2_MQ_RNDV_HFI_THRESH=64000`
		`PSM2_MQ_EAGER_SDMA_SZ=Default (Set it to 8 for sizes > 8192) PSM2_BOUNCE_SZ=256`

MPI Collectives	3rd Generation Intel Xeon Scalable Processor	`FI_OPX_DELIVERY_COMPLETION=1000000`[9] FI_OPX_DELIVERY_COMPLETION_THRESHOLD Integer. Will be deprecated. Please use FI_OPX_SDMA_BOUNCE_BUF_THRESHOLD.

Application	Processor	Tuning Parameters

Notes:

1. Parameters that begin with `I_MPI_` are Intel MPI environment variables.
2. To set the `rcvhdrcnt` driver parameter, refer to Section 4.3 "Setting HFI1 Driver Parameters".
3. This application is part of the SPEC MPI2007 Medium suite. This tuning has no negative effects on the other 12 codes in the suite, so can be used for a base run.
4. Fluent options to force using Omni-Path Express Fabric Suite.
5. The optimal value is rank count dependent. The suggested value needs to be adjusted with varying rank counts.
6. If running with MPI Library earlier than version 2019, contact support for recommended tunings. With Intel OneAPI, only this compatibility flag is required.
7. This PSM2 setting can significantly help some QCD tests that use large messages (16 MB or larger).
8. This PSM2 setting can give approximately 10% performance benefit for some OpenQCD workloads.
9. Maintaining the lower default value of `FI_OPX_DELIVERY_COMPLETION` is beneficial for bandwidth benchmarks.
10. Older generations of AMD EPYC processors may also be affected. See the mitigation documentation for more details.

5.12. OPX Provider Environment Variables

The following tunings are specific to the OPX provider. The default values have been found to provide positive performance across a wide spectrum of applications; however, specific applications may benefit from other values.

Table 11. General OPX Provider Environment Variable Recommendations

Variable	Description	Range	Default	GPU-Specific?
`FI_OPX_RELIABILITY_USEC_MAX`	The number of microseconds between pings for un-ACK'ed packets.	`-1 : INT_MAX`	500	No
`FI_OPX_RELIABILITY_MAX_UNCONGESTED_PINGS`	The maximum number of pings sent in a single timer iteration when the network link is uncongested.	1 : 65535	128	No
`FI_OPX_RELIABILITY_MAX_CONGESTED_PINGS`	The maximum number of pings sent in a single timer iteration when the network link is congested.	1 : 65535	4	No

Variable	Description	Range	Default	GPU-Specific?
FI_OPX_RELIABILITY_SERVICE_PRE_ACK_RATE	The number of packets to receive before sending an ACK. (Must be a power of 2).	0 : 32768	64	No
FI_OPX_RZV_MIN_PAYLOAD_BYTES	The number of packets to receive before sending an ACK. (Must be a power of 2).	64 : 65536	16385	No
FI_OPX_SDMA_MIN_PAYLOAD_BYTES	The number of packets to receive before sending an ACK. (Must be a power of 2).	64 : INT_MAX	16385	No
FI_OPX_EXPECTED_RECEIVE_ENABLE	Enable/disable expected receive (i.e., TID).	0, 1	0	No
FI_OPX_DEV_REG_SEND_THRESHOLD	Maximum value for device-registered sends when using a GPU buffer (i.e., GDRCopy).	0 : 8192	4096	Yes
FI_OPX_DEV_REG_RECV_THRESHOLD	Maximum value for device-registered receives when using a GPU buffer (i.e., GDRCopy).	0 : 8192	8192	Yes

5.13. GPU Specific MPI Environment Variables

The following environment variables are specific to systems that utilize either AMD or NVIDIA GPUs:

```
MPIR_CVAR_CH4_OFI_ENABLE_HMEM: Enables HMEM capabilities for MPI operations, 0 or 1
MPIR_CVAR_ENABLE_GPU: Enables GPU support for MPI operations, 0 or 1
```

5.14. GPUDirect RDMA Tuning for MPI Benchmarks and Applications

NOTE

GPUDirect Remote Direct Memory Access (RDMA) was formerly known as GPUDirect v3.

From the NVIDIA CUDA Toolkit Documentation:

> GPUDirect RDMA is a technology that enables a direct path for data exchange between the GPU and third-party peer devices using standard features of PCI Express. Examples of third-party devices include network interfaces.

The Omni-Path Express HFI is an example of a network interface device that supports GPUDirect RDMA. This section discusses how to best use this capability on Omni-Path Express-enabled compute nodes.

GPUDirect RDMA is available on various families of NVIDIA GPUs: Volta, Tesla, and Quadro.

IMPORTANT

For GPUDirect to function properly, NVIDIA recommends disabling setting PCIe Access Control Services (ACS), also known as IO virtualization, VT-d, or IOMMU, to pass-through mode. If left enabled, unpredictable behavior such as application failures may be experienced. Refer to the NVIDIA documentation, PCI Access Control Services, for more information regarding setting the pass-through mode.

5.14.1. Prerequisites

The following prerequisites are needed to obtain maximum bandwidth and best latency between CUDA-enabled GPU devices:

1. Connect the GPU and Omni-Path Express HFA to the same CPU (socket) via PCIe buses. As stated in the CUDA documentation, the two devices must share the same upstream PCI Express root complex.
2. Use a CUDA-aware MPI, such as the Open MPI provided as part of the Omni-Path Express Software 10.4 (or newer) release, installed by default at `/usr/mpi/gcc/openmpi-<version>-cuda-hfi`.
3. Use a CUDA-enabled application or benchmark, such as OMB 5.3.2 configured and built with `--enable-cuda` and other settings documented in the OMB README file; and, run using `-d cuda D D` at runtime.
4. If using PSM2 , set environment variables `PSM2_CUDA=1` and `PSM2_GPUDIRECT=1`. For example, Open MPI mpirun command line options `-x PSM2_CUDA=1 -x PSM2_GPUDIRECT=1` would propagate these variable settings to the compute nodes running the job.

   If using OPX Provider, set the environment variables `OFI_NCCL_CUDA_FLUSH_ENABLE=1` and/or `OFI_NCCL_GDR_FLUSH_DISABLE=1`. It is also recommended to set `FI_HMEM_CUDA_USE_GDRCOPY=1`.
5. Increase the gpu_cache_size on each node.
   a. Edit `/etc/modprobe.d/hfi1.conf` (see Section 4.3 “Setting HFI1 Driver Parameters”).
   b. Add the following hfi1 driver parameter setting to the "options hfi1" line.

      `gpu_cache_size=X`

Where X is the send and receive side GPU buffer cache size limit (in MB). The default value is 256 for compatibility with older GPU cards. Newer GPUs show performance benefits with increased buffer cache size. It is recommended to set gpu_cache_size to the size of GPU memory on your cards, at a minimum. Experiment with this parameter to determine optimal performance for your applications.

c. Restart the hfi1 driver to activate the setting as described in Section 4.3 "Setting HFI1 Driver Parameters".

5.14.2. Use Cases

Usage tips and examples depend on the number of Omni-Path Express HFIs used per node. Four cases include: 1) single-HFI systems; 2) multi-HFI systems where each MPI rank uses one HFI; 3) multi-HFI systems where each MPI rank uses multiple HFIs to increase single-rank bandwidth); and, 4) single-HFI systems where the HFI and GPU are connected to different CPU sockets (that is, Section 5.14.1 "Prerequisites" #1 was not possible).

1. Single HFI per node.

   This is the most common situation. The HFI adapter and GPU are connected to the same socket. First assume that the compute nodes have the HFI adapter connected to socket 0. An example mpirun command line that follows these recommendations is:

```
/usr/mpi/gcc/openmpi-<version>-cuda-hfi/bin/mpirun -mca pml cm \
 -mca mtl psm2 -H node01,node02 -x PSM2_CUDA=1 \
 -x PSM2_GPUDIRECT=1 ./osu_bw –d cuda D D
```

   A similar well constructed, well performing run command using Intel MPI is as follows:

```
I_MPI_OFI_ISEND_INJECT_THRESHOLD=8 FI_PROVIDER=psm2 PSM2_CUDA=1
PSM2_GPUDIRECT=1 I_MPI_OFFLOAD_RDMA=1 \ mpirun -np 2 -ppn 1 -host
host1,host2 IMB-MPI1-GPU Uniband -mem_alloc_type device
```

   A similar run commands may be executed using the OPX provider rather than PSM2. An example using Open MPI is as follows:

```
/usr/mpi/gcc/openmpi-<version>-cuda-hfi/bin/mpirun --mca btl_ofi_disable_sep 1 --mca
mtl_ofi_enable_sep 0 --mca osc ^ucx --mca pml
^ucx --mca mtl ofi --mca btl self,vader -x FI_OPX_UUID=$RANDOM -x FI_PROVIDER=opx -x
MPIR_CVAR_CH4_OFI_ENABLE_AV_TABLE=0 -x
MPIR_CVAR_CH4_OFI_ENABLE_MR_SCALABLE=0 -x
MPIR_CVAR_CH4_OFI_ENABLE_ATOMICS=1 -x MPIR_CVAR_CH4_OFI_ENABLE_RMA=1 -
x MPIR_CVAR_ENABLE_GPU=0 -x MPIR_CVAR_CH4_OFI_ENABLE_HMEM=0 -x
FI_OPX_EXPECTED_RECEIVE_ENABLE=1 -x FI_HMEM_CUDA_USE_GDRCOPY=0 -host
opx-node3:1,opx-node4:1 -np 2 -N 1 osu_bw
```

NOTE

Typically, the full pathname to `mpirun` is not required if the bin directory is in your path. However, in the command line above, we wanted to emphasize that the CUDA-enabled MPI, that is `openmpi-<version>-cuda-hfi`, should be used for GPU workloads.

2. Dual HFI per node where each MPI rank running on the node uses one HFI.

   If the GPU device is connected to the second CPU socket (socket 1) and a second HFI on the system also is connected to socket 1 (HFI_UNIT=1), then a good-performing `mpirun` command line would be:

```
mpirun -mca pml cm -mca mtl psm2 -H node01,node02 -x HFI_UNIT=1 \
 -x PSM2_CUDA=1 -x PSM2_GPUDIRECT=1 ./osu_bw –d cuda D D
```

   The command line above leaves the placement of the osu_bw process to the OS scheduler. If the compute nodes had two 14-core CPUs, and you wanted more control over which core and socket on which the benchmark process ran, the following command with `taskset` could be employed:

```
mpirun -mca pml cm -mca mtl psm2 -H node01,node02 -x PSM2_CUDA=1 \
 -x PSM2_GPUDIRECT=1 taskset –c 20 ./osu_bw –d cuda D D
```

   Alternatives to `taskset -c 20` include: `taskset -c 14-27` or `numactl -N 1` to place the task on any core of socket 1 (also known as NUMA node 1) and potentially allow movement between cores on NUMA node 1. Note that we did not specify which HFI in the command above.

   If there is more than one HFI on same socket, PSM2 spreads the ranks uniformly across the HFIs within the socket (default behavior).

   You can use Round-Robin algorithm to consider all HFIs in the system (and not restrict to just the socket) by setting the environment variable `HFI _SELECTION_ALG="Round Robin All"` (note that this is not recommended for GPU workloads).

3. Dual HFIs per node where each MPI rank uses both HFIs, when possible.
   - For each MPI rank NOT running GPU workloads, you may set the environment variable, `PSM2_MULTIRAIL=1` to stripe large messages across both HFIs.
   - For the MPI ranks running GPU workloads, you have the following two cases:
     a. Both HFIs are connected to the same socket as the GPU.

        In this case, you can set `PSM2_MULTIRAIL=2` to stripe large messages across both HFIs. If both HFIs were on socket 1 (of 0,1), then the following would be a good `mpirun` command:

```
mpirun -mca pml cm -mca mtl psm2 -H node01,node02 -x PSM2_MULTIRAIL=2 \
-x PSM2_CUDA=1 -x PSM2_GPUDIRECT=1 numactl –N 1 ./osu_bw –d cuda D D
```

b. If the two HFIs are connected to different sockets, you should **not** use PSM2_MULTIRAIL. For the GPU workload, you should specify to use only the single HFI on the same socket as the GPU. For example, if both the GPU and hfi1_1 were on NUMA node 1 (of 0,1), you could use the following `mpirun` command to run a bandwidth test:

```
mpirun -mca pml cm -mca mtl psm2 -H node01,node02 -x HFI_UNIT=1 \
 -x PSM2_CUDA=1 -x PSM2_GPUDIRECT=1 numactl –N 1 ./osu_bw –d cuda D D
```

4. HFI adapter and GPU connected to different sockets.

   The HFI adapter and GPU are PCIe-connected to different CPUs/sockets. A loss of performance occurs in this situation (as compared to having both connected to the same socket), but most of it can be gained back by setting the environment variable `PSM2_GPUDIRECT_RECV_THRESH` to `4096` as per the example below:

```
/usr/mpi/gcc/openmpi-<version>-cuda-hfi/bin/mpirun -mca pml cm \
-mca mtl psm2 -H node01,node02 -x PSM2_CUDA=1 -x PSM2_GPUDIRECT=1 \
-x PSM2_GPUDIRECT_RECV_THRESH=4096 ./osu_bw –d cuda D D
```

NOTE

When using CUDA-enabled Open MPI, user applications that rely on GPU buffers may segfault without the use of the `smcuda BTL`. Users may need to specify the `smcuda btl`, for example: `-mca btl self,vader,smcuda`.

5.15. AMD GPU (ROCm)

Prerequisites

To achieve optimal bandwidth and the lowest latency for ROCm-enabled GPU devices, ensure the following:

1. Hardware configuration:

   Connect the AMD GPU and Omni-Path Express HFI to the same CPU socket or behind a PCIe switch. (Follow the instructions in section 2.5)

2. MPI configuration:

   Use an MPI implementation compatible with ROCm, such as OpenMPI.

3. Compile OpenMPI with ROCm support:

   Include the ROCm path during OpenMPI compilation. Example of the configure command:

```
./configure --with-rocm=/opt/rocm
--with-rccl=/opt/rocm --with-cuda=no --enable-orterun-prefix-by-default
```

```
LDFLAGS=-Wl,--enable-new-dtags --with-ofi=/usr --with-ofi-libdir=/usr/lib64
--with-libfabric=/usr
```

4. After installation, ensure the following paths are updated:

   Add `openmpi/bin` to your `PATH` environment variable.

   Include `openmpi/lib` and `rocm/lib` in your `LD_LIBRARY_PATH`.

5. Application and benchmark setup:

   Use a ROCm-enabled benchmark or application, such as OMB (OSU Micro-Benchmarks) version 7.5.

   Ensure the application is built with `--enable-rocm` and any additional settings specified in the application's README documentation.

Use Cases

The specific configurations depend on the number of Omni-Path Express HFIs utilized per node.

NOTE

Note: Add the environment variable `MPIR_CVAR_CH4_OFI_ENABLE_HMEM=1` to all `mpirun` commands to enable HMEM support.

1. Single HFI per node

   This is the most common setup, where the HFI adapter and GPU are connected to the same socket. For example, if the compute nodes have the HFI adapter on socket 0, the following command can be used:

```
mpirun -x FI_OPX_HFI_SELECT=0 -x HIP_VISIBLE_DEVICES=1
-x MPIR_CVAR_CH4_OFI_ENABLE_RMA=1 -x MPIR_CVAR_ENABLE_GPU=1 -x
MPIR_CVAR_CH4_OFI_ENABLE_HMEM=1 -mca mtl ofi -x FI_PROVIDER=opx -mca pml cm -np 2
-host node1,node2 ./osu_bw D D
```

2. Dual HFI per node (One HFI per MPI rank)

   In this setup, each MPI rank utilizes a dedicated HFI adapter. To use multiple HFIs on a node, you need to use multiple processes per node. Depending on the application, each process may select to use a unique GPU, otherwise you can control with the `HIP_VISIBLE_DEVICES` environment variable. For each process, the opx provider will select the HFI that is on the same NUMA node as each process. Use the following command:

```
mpirun-x
MPIR_CVAR_CH4_OFI_ENABLE_RMA=1 -x MPIR_CVAR_ENABLE_GPU=1 -x
MPIR_CVAR_CH4_OFI_ENABLE_HMEM=1 -mca mtl ofi -x FI_PROVIDER=opx -mca pml cm -np
2-np 4 --map-by ppr:2:node -host node1,node2 ./osu_bwosu_mbw_mr D D
```

These steps ensure maximum performance for MPI benchmarks and applications with AMD GPUs and Omni-Path Express HFIs.

5.16. Assigning Virtual Lanes to MPI Workloads

Grouping MPI applications into unique Virtual Lanes (VLs) can help minimize inter-application contention resulting in better system performance. VLs provide a mechanism to implement multiple logical flows over a physical link. For each VL, independent buffering resources are provided and link level flow control is enabled. In addition, Quality of Service (QoS) policies can be used to assign bandwidth distributions to the VL in order to reduce latency jitter and improves system throughput.

This section describes a method to assign MPI jobs to a specific Service Level (SL), which identifies flows within a subnet using virtual fabrics. The subnet manager programs the SL-to-VL mappings and the VL arbitration tables to support appropriate forwarding of each class of flows. After the virtual fabrics have been configured in the Fabric Manager,Omni-Path Express tools can be used to extract information about the SL and propagate it to all the ranks of an MPI job. For more information, refer to *Cornelis Omni-Path Express Fabric Suite Fabric Manager User Guide*, Integrating Job Schedulers with Virtual Fabrics.

To select a specific VL with MPI, use the environment variable `HFI_SL`. For Open MPI, this needs to be set at runtime with `-x HFI_SL=<desired SL>`. With Intel MPI and MVAPICH2, this can be specified by exporting the environment variable with `-genv HFI_SL=<desired SL>` at runtime.

NOTE

When using CUDA-enabled Open MPI, user applications that rely on GPU buffers may segfault without the use of the `smcuda btl`. Users may need to specify the `smcuda btl`, for example: `-mca btl self,vader,smcuda`.

5.17. Reducing System Jitter

System noise can have a negative impact on application performance, especially when running at scale. The following tunings may reduce system noise and improve application performance.

1. Add `nohz_full=1-xx` to boot options. If a CPU has only one runnable task, there is no need of scheduling-clock interrupts because there is no other task to switch to. The `nohz_full=` boot parameter can be used to specify the adaptive-tick CPUs. With this setting, the kernel will avoid sending scheduling-clock interrupts to adaptive-tick CPUs. For example, on the 40 core system with HT enabled, add `nohz_full=1-39, 41-79` to the boot option; with this setting CPUs 1-39, and 41-79 are adaptive-tick CPUs that will not get scheduling-clock interrupts. Note at least one non-adaptive-tick CPU must remain online to handle timekeeping tasks. Further benefit can be achieved by assigning application processes to only adaptive-tick CPUs. However, be aware that this tuning will negatively impact IPoFabric performance, see Section 7.7 “Kernel Boot Parameters to Avoid”.

2. Set `I_MPI_THREAD_YIELD=off` when running with Intel MPI. This feature is available in the Intel MPI Library 2018 update or later. This variable controls thread yield behavior during MPI busy wait time.

5.18. 1 GB Huge Pages

In some scenarios, you may want to allocate 1 GB huge pages to get better throughput for very large messages. For example, certain MPI collective algorithms generate internal point-to-point transfers that are large, such as 512 MB to 1 GB. Without 1 GB huge pages, mild bandwidth drops may be seen for these larger message sizes.

The following provides example instructions for setting up 1 GB huge pages with `libhugetlbfs`.

NOTE

Your implementation may vary based on your preferences.

1. On every compute node, perform the following steps:
   a. Install the following `libhugetlbfs` packages from the OS distribution.

      NOTE

      The following files are from the RHEL distribution.

      - `libhugetlbfs-2.16-12.el7.x86_64`
      - (optionally) `libhugetlbfs-devel-2.16-12.el7.x86_64`
      - (optionally) `libhugetlbfs-utils-2.16-12.el7.x86_64`

      For more information, see https://github.com/libhugetlbfs/libhugetlbfs/blob/master/HOWTO.

   b. Request a large number of 1 GB pages.

      NOTE

      Only a smaller number will actually be allocated due to free memory.

      ```
      echo 128 >
      /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
      ```

      The number of 1 GB pages created depends on the amount of free system memory and most likely will vary from node to node.

      ```
      actual1Gpages=`grep ./sys/devices/system/node/node0/hugepages/
      hugepages-1048576kB/nr_hugepages`
      ```

You can also use `hugeadm --pool-list` to check that 1 GB pages were allocated (requires `libhugetlbfs-utils-2.16-12.el7.x86_64`).

c. Mount a temporary file system.

```
mkdir /mnt/hugepages
mount -t hugetlbfs -o pagesize=1024M,size=$((1024*actual1Gpages))M none /mnt/
hugepages
```

d. Change permissions so group members can read/write.

For example:

```
chmod 775 /mnt/hugepages
```

NOTE

Ensure the target user is part of group ownership.

2. Launch the MPI job with the following additions:

```
mpirun ... -genv LD_PRELOAD=libhugetlbfs.so -genv HUGETLB_MORECORE=1073741824 ...
```

If you do not properly have 1 GB huge pages reserved, the application may return a message such as:

```
libhugetlbfs: WARNING: ... Cannot allocate memory
```

You can also enable verbose logging with the environment variable `HUGETLB_VERBOSE=99`.

6. Storage and Verbs Performance

This section describes settings that may improve the performance of the Verbs protocol, as well as guidance on using the perftest benchmark to measure RDMA/Verbs performance on Omni-Path Express and performance tunings for storage systems.

6.1. Accelerated RDMA

Accelerated RDMA is a Verbs protocol extension to improve the performance of RDMA write and RDMA read operations on Omni-Path Express hardware. This extension improves the efficiency of large message transfers to provide performance benefits for storage protocols and other Verbs-based protocols. The performance benefits include increased achievable bandwidth with reduced CPU utilization. This feature accelerates the OpenFabrics Alliance (OFA) Verbs API with no changes required to API consumers. Since the acceleration technique is performed by the host driver, the application running over the OFA Verbs API does not need to make any code change. However, it does need to meet certain criteria to take advantage of the feature (described below).

NOTE

Accelerated RDMA has no impact to MPI performance.

Accelerated RDMA works by removing buffer copies on the receive side. The receiving HFI will place packet payloads directly into the application buffer without CPU involvement after the initial setup of the Accelerated RDMA connection.

The conditions below must be true to enable and engage Accelerated RDMA (it is not enabled by default).

To enable Accelerated RDMA, perform the following steps:

1. Add the `cap_mask=0x4c09a09cbba` setting to the `/etc/modprobe.d/hfi1.conf` file (see Section 4.3 “Setting HFI1 Driver Parameters”).
2. Restart the hfi1 driver to activate the setting as described in Section 4.3 “Setting HFI1 Driver Parameters”.

To use Accelerated RDMA, ensure that your application or middleware (such as file system software) meets the following conditions:

1. Use a payload size of at least 256 KB.
2. Set the payload size as a multiple of 4 KB.
3. Align the data buffers to 4 KB page boundaries.

Accelerated RDMA may have benefits when:

- Other verbs tuning is resulting in high CPU load due to interrupt handling.
- The specific conditions listed above are met.

NOTE

For Omni-Path Express Fabric Suite versions 10.9.3.1.1 and earlier, Accelerated RDMA only works properly if `krcvqs` is set to either 2 or 4. If `krcvqs` is set to any other value, performance and/or functional problems may occur. For Omni-Path Express Fabric Suite version 10.10 and later, this issue has been resolved.

NOTE

It is not necessary to enable Accelerated RDMA on all nodes in a cluster. However, the performance effects of enabling Accelerated RDMA on a subset of nodes has not been characterized. Accelerated RDMA is only active between two nodes when both nodes have the feature enabled.

NOTE

Accelerated RDMA is not currently compatible with Congestion Control Architecture (CCA). For details on CCA, see the *Cornelis Omni-Path Express Fabric Suite Fabric Manager User Guide*.

6.2. Parallel File System Concurrency Improvement

The Accelerated RDMA setting should be a preferred setting for high concurrency file system traffic.

NOTE

As of Omni-Path Express Host Software version 10.9 and later, memory optimizations have significantly decreased the required memory to establish verbs queue pairs with Accelerated RDMA. Large clusters that previously could not use Accelerated RDMA due to memory restrictions should re-enable Accelerated RDMA and confirm reduced memory utilization particularly on the server nodes that need to establish connections with many client nodes.

If the Accelerated RDMA setting does not work well with your workload, as an alternative you can turn on the Adaptive Cache Memcpy, by setting the parameter `sge_copy_mode=2` similar to other driver parameters as discussed in Section 4.3 “Setting HFI1 Driver Parameters”.

Another driver parameter setting that can improve parallel file (storage) system scalability is krcvqs (kernel receive queues).

On a compute node that is a storage system client node, you may want to try values larger than the default of `krcvqs=2`.

1. Edit `/etc/modprobe.d/hfi1.conf file` (see Section 4.3 "Setting HFI1 Driver Parameters").
2. Modify the `krcvqs` value as shown in the example below:

```
options hfi1 sge_copy_mode=2 krcvqs=5
```

The above setting assumes one active VL. If, for example, there were three active VLs (00, 01, 02) and the parallel file system traffic were on VL 01, then a good krcvqs setting might be `krcvqs=3,5,3`, to avoid using too many resources on the non- storage VLs.

On a storage system server node, typically there is no competing MPI traffic or floating point compute activity, so more of the cores can be used for kernel receive queues. In this case, using settings such as `krcvqs=5` up to `krcvqs=9` may provide the best throughput and IO operations per second for the file system. Increasing krcvqs dedicates cores for this work and therefore reduces the number of cores available for other purposes such as SDMA. It is not recommended to blindly increase krcvqs without careful examination of the return in performance. Typically there is diminishing return with increasing krcvqs. Only increase krcvqs up to a value that provides improved performance, and no further.

6.3. Perftest

Perftest is an open-source benchmark from OFA for verbs performance. It is a set of microbenchmarks written over user-level verbs to measure latency, and uni- and bi-directional bandwidth.

When running perftest, the IPoIB IP address must be used to connect the client and server. It is also recommended to use the RDMA Connection Manager (RDMA_CM). This will make use of FM PathRecord queries to establish connections with the best possible MTU.

NOTE

Using different versions of perftest between test nodes may cause unexpected failures. If you are using the OS-provided version of these tools between two test nodes with different OS levels, ensure that the version of perftest is the same.

6.3.1. Verbs Bandwidth

The best perftest to measure verbs RDMA bandwidth performance is the `ib_write_bw` test with the default connection type set to `Reliable Connection`. Note that you can view the available options by running `ib_write_bw -h`. This section highlights optimizations for the `ib_write_bw` test, but can be applied to other perftest benchmarks as well.

InfiniBand supports MTU sizes of 256 bytes, 512 bytes, 1024 bytes, 2048 bytes, and 4096 bytes only. Additionally, Omni-Path Express can support MTU sizes from 2048 bytes (2 KB) up to 8192 bytes (8 KB) for verbs traffic.

To enable the largest possible MTU size of 8 KB:

1. Specify the `-R` parameter to connect Queue Pairs (QPs) with `rdma_cm`.
2. Use the address for the server node's ib0 port to specify the IPoIB interface.

The sequence of execution of the `ib_write_bw` test is:

1. `ib_write_bw -F -R -s 1048576` (on server node)
2. `ib_write_bw -F -R -s 1048576 <server's IPoIB address>` (on client node)

NOTE

The `-F` parameter is used to prevent the test from failing when the cpufreq_ondemand module is used. Refer to Section 3.2 "CPU Frequency Scaling Drivers" for more information.

6.3.2. Verbs Latency

You can use the `ib_write_lat` test to measure verbs RDMA write latency. Note that you can view the available options by running `ib_write_lat -h`.

To improve the `ib_write_lat` small message latency in some environments, boot parameters can be modified or the Tuned utility can be run in real-time without needing to modify boot parameters.

- **Option 1: Modify Boot Parameters**
  1. In the `/etc/default/grub` file, add `processor.max_cstate=1 intel_idle.max_cstate=0` to the GRUB_CMDLINE_LINUX command line.
  2. For RHEL/Rocky versions 9.3 and later, apply the change using:

```
if [ -e /boot/efi/EFI/redhat/grub.cfg ]; then
GRUB_CFG=/boot/efi/EFI/redhat/grub.cfg
elif [ -e /boot/grub2/grub.cfg ]; then
GRUB_CFG=/boot/grub2/grub.cfg
fi
grub2-mkconfig -o $GRUB_CFG
```

NOTE

The code example above is for Red Hat. Other OSes may require a different method for modifying grub boot parameters.

  3. Reboot.

- **Option 2: Tuned**

  Tuned is a utility (for example, `tuned-2.10.0-15.el8.noarch`) that allows for dynamic and adaptive tuning of CPU behavior. It has a variety of profiles and the default on RHEL 8 systems is *throughput-performance*. Changing the policy to *latency-performance* has been shown to significantly reduce verbs latency in certain environments.

– To determine what policy is currently running:

```
tuned-adm profile
```

– To change the policy to latency-performance:

```
tuned-adm profile latency-performance
```

The latency-performance profile may have negative impacts on other areas of performance such as MPI message rate, so care should be taken to select the profile that works best for overall performance.

NOTE

The improvement you may see in verbs and IPoFabric small message latency must be weighed against the *likely increase in energy consumption* from these nodes due to the reduction in scaling to lower CPU frequencies that these settings will cause.

To reduce run to run variation in verbs latency, sometimes it helps to pin the ib_write_lat process to cores not processing driver work. Use a command such as `dmesg | grep hfi1_0 | grep IRQ` (described in Section 8 "Driver IRQ Affinity Assignments", Method 2) to view the CPUs to which these IRQs are assigned.

In this example, assume that core 24 was determined to not service any krcvqs or sdma engines. Then, `taskset -c 24` will pin the processes to core 24. This will prevent benchmark processes from running on the CPUs being used by the hfi1 driver's receive contexts and SDMA engines, providing more consistent performance results.

- `taskset -c 24 ib_write_lat --ib-dev=hfi1_0 -a -R` (on server node)
- `taskset -c 24 ib_write_lat <server's IPoIB address> --ib-dev=hfi1_0 -a -R` (on client node)

6.4. Lustre

Lustre Version 2.10 or newer is recommended. Lustre Version 2.10 includes a performance improvement relevant to Omni-Path Express and other fabrics, particularly for 1-client to 1-server tests. The Lustre module load process will automatically load the necessary tunings for optimal Omni-Path Express performance.

6.4.1. Lustre Multi-Rail Support with Omni-Path Express

It is possible to use more than one Omni-Path Express HFI per Lustre client or server. As of Lustre version 2.10, the LNet Multi-Rail feature of Lustre, by default, load balances traffic across the active NIDs. Lustre version 2.11 and newer has Dynamic Discovery that automatically detects and sets up multi-rail peers. If you are using an earlier version of Lustre, some manual setup is required.

Enabling a multi-rail peer in Lustre is relatively easy. In general, you configure a host (Lustre client or server) with two Omni-Path Express HFIs that have two active IPoFabric interfaces, such as ib0 and ib1. Then using either the static configuration method or the dynamic discovery method, you add the host to the LNet configuration. The steps below describe this general setup process.

1. On the host with two Omni-Path Express HFIs, update lustre.conf to define both IPoFabric interfaces on the same LNet network:

   ```
   cat /etc/modprobe.d/lustre.conf
   options lnet networks=o2ib0(ib0,ib1)
   ```

   Reboot the host.

2. On all other Lustre hosts, add the configured host to the LNet configuration using one of the following methods:

   - **Using the Static Configuration Method**

     Set up the definition for the multi-rail peer using either CLI or YAML configuration files.

     – CLI:

     ```
     lnetctl peer add --prim_nid 192.168.237.1@o2ib --nid 192.168.237.2@o2ib
     ```

     – YAML:

     Use `lnetctl import < config.txt` to add the multi-rail peer definition contained in config.txt .

     Example contents of config.txt can be found in `/etc/lnet.conf`. For example,

     ```
     cat config.txt

     peer:
     - primary nid: 192.168.237.1@o2ib
     Multi-Rail: True
     peer ni:
     - nid: 192.168.237.1@o2ib
     - nid: 192.168.237.2@o2ib
     ```

     In the above file, `192.168.237.[1,2]` are the IP addresses of ib0 and ib1, respectively. Executing `lnetctl import < config.txt` on all Lustre hosts will add this multi-rail peer to their configuration.

     

     NOTE

     These changes are not persistent on reboot. In order to make them persistent, you must edit `/etc/lnet.conf` to include the contents of config.txt (and more for other peers), and have the Lnet service running. Refer to Lustre documentation for more guidance.

- **Using the Dynamic Configuration Method**

  Use `lnetctl discover <primary_nid>` to automatically search the network and add a multi-rail enabled peer.

  For example:

```
[~]# lnetctl discover 192.168.237.1@o2ib
discover:
- primary nid: 192.168.237.1@o2ib
Multi-Rail: True
peer ni:
- nid: 192.168.237.1@o2ib
- nid: 192.168.237.2@o2ib
```

Note that the formats between Static and Dynamic configurations are very similar.

LNet automatically load balances across all available Omni-Path Express HFI in the system. This procedure can be applied to any combination of dual/single rail clients and servers. In the above example, if LNET_selftest was executed using two or more single-rail client nodes writing to this dual-rail server, the aggregate throughput would approach the line rate of two Omni-Path Express HFI, or 200 Gbps. For more details and an example LNET_selftest script, see https://wiki.whamcloud.com/display/LNet/Multi-Rail+Configuration+Post+2.11.

In addition to using multi-rail to increase throughput and reliability for bulk data transfers, Lustre often uses the IPoIB address to perform initial connection, maintenance, and management tasks, even though it uses RDMA/Verbs for the bulk data transfers. To increase the reliability of a dual rail storage server, consider enabling IPoIB bonding on the two interfaces and use the bonded address when adding the server to the Lustre configuration. Refer to *Cornelis Omni-Path Express Fabric Host Software User Guide*, "IPoIB Bonding" section.

6.5. IBM Storage Scale (aka GPFS)

The Accelerated RDMA setting is the preferred setting for high concurrency file system traffic, such as a cluster with an IBM Storage Scale parallel file system.[1]The Accelerated RDMA setting must be made on all server and client nodes.

Use the following Storage Scale parameter settings (tunings) to engage Omni-Path Express's Accelerated RDMA, to set the MTU to 8KB, and to otherwise improve GPFS performance:

```
NSD Server:
mmchconfig verbsRdmaQpRtrPathMtu=8192, verbsRdmaMaxSendBytes=1024k,
scatterBufferSize=1024k, verbsRdmaMaxSendSge=40, maxMBpS=32000
```

```
Client/compute node:
mmchconfig verbsRdmaQpRtrPathMtu=8192, verbsRdmaMaxSendBytes=1024k,
```

[1]IBM Storage Scalewas previously known as IBM Spectrum Scale and General Parallel File System, GPFS.

```
scatterBufferSize=1024k, verbsRdmaMaxSendSge=40, pagepool=4096M,
prefetchPct=50, maxMBpS=16000
```

NOTE

The `pagepool=` parameter should be set during initial installation of the nodes. Changing it after the file system is created may have no effect. Consult the IBM Storage Scale documentation to confirm when this parameter can be set or changed.

Pertaining to Accelerated RDMA conditions:

- For the first condition—`verbsRdmaQpRtrPathMtu`—Storage Scale V4.2.1, or later is required.
- For the third condition—data buffer is 4K page aligned—this is always in effect for Storage Scale RDMA traffic, except when in GPFS Native Raid (GNR) mode. See the note below.

An additional tuning parameter recommended by IBM for GPFS is to increase `WorkerThreads` from the default of `48` to `512`. This parameter controls a few other parameters and improves performance when file systems have very high-performance capability. For more information, consult the IBM Storage Scale documentation.

If any of the nodes involved with the Storage Scale file system traffic have two Omni-Path Express HFI adapters, and you want both to support the file system traffic, then for each such node, use the `mmchconfig` command to set the verbsPorts parameter as:

```
verbsPorts="hfi1_0 hfi1_1"
```

In addition to using two Omni-Path Express adapters to increase throughput and reliability for bulk data transfers, GPFS often uses the IPoIB address to perform initial connection, maintenance, and management tasks, even though it uses RDMA/Verbs for the bulk data transfers. To increase the reliability of a dual rail storage server, consider enabling IPoIB bonding on the two interfaces and use the bonded address when adding the server to the GPFS configuration. Refer to *Cornelis Omni-Path Express Fabric Host Software User Guide*, "IPoIB Bonding" section.

The following Intel Xeon Processor E5 v3 and v4 families BIOS tunings, found in Table 3 "Recommended BIOS Settings for Intel Xeon Processor E5 v3 and v4 Families", were found to be particularly important for a Spectrum Scale file system workload:

```
IOU Non-posted Prefetch= Disabled
NUMA Optimized= Enable (aka Memory.SocketInterleave=NUMA)
Snoop Holdoff Count=9
```

IBM recommends using Datagram Mode for IPoIB control messages. See Section 7.2 "IPoFabric Datagram Mode" for details on configuring Datagram Mode on your cluster. IBM Storage Scale bulk data transfers are done using RDMA and the tunings listed above.

NOTE

Some implementations such as GPFS Native Raid (GNR) generate messages that are not exactly 4K aligned. In this scenario, Accelerated RDMA is not used and should be left disabled in order to avoid unnecessary processing overhead.

6.5.1. GPFS Settings for Large Clusters

When usingOmni-Path Express for RDMA with the Verbs protocol (verbsRdma=enable), GPFS still uses TCP/IP communications for administrative commands, daemon communication, node health monitoring, etc. This TCP/IP communication can be configured to use a separate ethernet interface but more typically uses the Omni-Path Express IPoIB interface.

On large clusters approaching hundreds or thousands of nodes, there can be contention between this TCP/IP GPFS management traffic and the bulk RDMA transfers that are occurring. In addition, packets less than 8KB (configurable with verbsRdmaMinBytes) also use the IPoFabric interface. There are generally two approaches to avoiding GPFS timeouts due to contention between these two different types of traffic.

6.5.1.1. Separation of the IPoIB and Verbs/RDMA Traffic into Two Unique Virtual Fabrics (Preferred)

Refer to Mapping IBM Storage Scale (GPFS) Traffic to a Virtual Fabric in Appendix J of the *Cornelis Omni-Path Express Fabric Suite Fabric Manager User Guide*.

This methodology provides significantly reduced latency for the IPoIB traffic because it eliminates the Head-of-Line blocking of the IPoIB traffic by the bulk Verbs RDMA traffic when both traffics are on the same VFabric. When separating the two, the HFI schedules access to the fabric equally between the two VFabrics and based on arbitration rules configured by the FM (see Virtual Fabrics Overview of the *Cornelis Omni-Path Express Fabric Suite Fabric Manager User Guide*). This methodology has been shown to virtually eliminate TCP/IP delay on large fabrics.

6.5.1.2. Increase GPFS Timeout Parameters

With Omni-Path Express, performing the VFabric separation outlined above is most likely sufficient. In some scenarios where QoS is not desired or for non-Omni-Path Express networks where QoS is not functional or available, you may be required to adjust GPFS timeout settings. The following settings were found to be beneficial for clusters of around 4000-node scale; slightly lower values may be acceptable on smaller clusters.

- `failureDetectionTime = 60` (default is -1)
- `minMissedPingTimeout = 60` (default is 3)
- `maxMissedPingTimeout = 180` (default is 60)
- `leaseRecoveryWait = 180` (default is 35)

7. IPoFabric Performance

The traditional term for sending IP traffic over the InfiniBand fabric is IP over IB or IPoIB. The Omni-Path Express Fabric does not implement InfiniBand. Instead, the Cornelis implementation for Omni-Path Express is known as IP over Fabric or *IPoFabric*. From the software point of view, IPoFabric behaves the same way as IPoIB, and in fact uses an ib_ipoib driver to send IP traffic over the ib0 and/or ib1 ports. Therefore, we will primarily refer to this traffic as IPoFabric, but will continue to use the term ib_ipoib and refer to the ib0/ib1 ports, and measure performance with traditional IP-oriented benchmarks such as qperf and iperf.

For RHEL, the `ifcfg-ib*` files are located in `/etc/sysconfig/network-scripts/`. To keep these settings persistent across boot, add `ONBOOT=yes` to the `ifcfg-ib*` files. For SLES, the `ifcfg-ib*` files are located in `/etc/sysconfig/network/`. To keep these settings persistent across boot, add `STARTMODE=auto` to the `ifcfg-ib*` files.

NOTE

Previous versions of the tuning guide have recommended connected mode for best throughput. However, since Omni-Path Express Fabric Suite 10.9 now contains "Accelerated IPoFabric" (AIP; see Section 7.2 “IPoFabric Datagram Mode”), users should check again to see if Connected Mode is preferred over Datagram Mode. AIP now enables line-rate performance between two nodes with Datagram Mode and is enabled by default. AIP also provides IPoFabric latency improvements relative to standard Datagram Mode without AIP.

7.1. IPoFabric Connected Mode

For Connected Mode IPoFabric bandwidth benchmarks, a prerequisite for the best possible performance is having 64KB MTU enabled on the fabric.

7.1.1. Configuring IPoFabric Connected Mode

To enable the 64KB MTU:

1. Edit the `ifcfg-ib0` file to modify the following settings:

   For RHEL:

   ```
   MTU=65520
   CONNECTED_MODE=yes
   ```

   For SLES:

   ```
   MTU=65520
   IPOIB_MODE=connected
   ```

2. Run the following commands:

```
# ifdown ib0
# modprobe -r ib_ipoib
# modprobe ib_ipoib
# ifup ib0
```

3. Verify that connected mode is enabled:

```
# cat /sys/class/net/ib0/mode
connected
```

4. If you have twoOmni-Path Express ports in your system, perform the above steps for the `ifcfg-ib1` file and `ib1` interface.

NOTE

Settings must be applied on each node in the fabric.

7.2. IPoFabric Datagram Mode

Although Connected Mode is used to provide the best performance in general, Datagram mode with a newly introduced Accelerated IP (AIP) may provide improved performance (both throughput and latency) compared to Connected Mode when supported. To see whether AIP is supported in your system, check if the `ipoib_accel` parameter exists as listed in Section 4 "HFI1 Driver Module Parameters". AIP is supported by default, and it may be disabled by setting the `hfi1 cap_mask` parameter to `hfi1 cap_mask=0x4c09a00cb9a`. Refer to Section 4.3 "Setting HFI1 Driver Parameters" for more details.

NOTE

UD mode only uses one Queue Pair (QP) per node, so it will have a lower memory footprint than Connected Mode, which has one QP for each destination node for which IPoFabric communications are desired.

NOTE

Due to the high-performance of UD mode with AIP, you may experience delays or slow responses for other traffic in the node when happening concurrently with high IPoFabric loads. Consider using Virtual Fabrics or Quality of Service Policies described in the *Cornelis Omni-Path Express Fabric Suite Fabric Manager User Guide*.

7.2.1. Configuring IPoFabric UD Mode

To use UD mode:

1. RHEL: Verify that `MTU=` and `CONNECTED_MODE=` do *NOT* exist in `ifcfg-ib0`.

   SLES: Verify that `MTU=` and `IPOIB_MODE=` do *NOT* exist in `ifcfg-ib0`.

2. Run the following commands:

```
# ifdown ib0
# modprobe -r ib_ipoib
# modprobe ib_ipoib
# ifup ib0
```

3. Verify that datagram mode is enabled:

```
# cat /sys/class/net/ib0/mode
datagram
```

4. If you have twoOmni-Path Express ports in your system, perform the above steps for the `ifcfg-ib1` file and `ib1` interface.

NOTE

An alternative method to configure datagram mode is to specify `CONNECTED_MODE=no` (RHEL) or `IPOIB_MODE=datagram` (SLES) in `ifcfg-ib*`, and verify that `MTU=` does not exist. Follow steps 2, 3, and 4 above after modifying `ifcfg-ib*`.

NOTE

Settings must be applied on each node in the fabric.

7.2.2. Adjusting UD Mode MTU Size

When you view the IP MTU size for ib0, you will see that it gets set to 2044 bytes:

```
# cat /sys/class/net/ib0/mtu
2044
```

You can increase the size of the UD mode MTU to nearly 10K bytes if AIP is supported in your system, or 4K bytes (without AIP) in order to improve throughput.

NOTE

- If network booting overOmni-Path Express fabric, ensure UEFI has been upgraded to version 10.9 or higher before enabling multicast group size larger than 4096 bytes.
- If using UD mode without AIP, ensure the MTU is set to no greater than 4092 bytes.

To increase the size of MTU in UD mode:

1. Change the FM's configuration file to allow larger MTU for multicast.

```
vi /etc/opa-fm/opafm.xml
```

   Change this:

```
      <MulticastGroup>
        <Create>1</Create>
        <MTU>2048</MTU>
```

   to this for UD mode with AIP (MTU updated to 10K):

```
      <MulticastGroup>
        <Create>1</Create>
        <MTU>10240</MTU>
```

   or to this for UD mode without AIP (MTU updated to 4K):

```
      <MulticastGroup>
        <Create>1</Create>
        <MTU>4096</MTU>
```

2. For RHEL 7.x, allow the 'ifup-ib' script to accept the larger MTU size. This script does not exist on SLES or RHEL 8.x; skip this step if using those distributions.

```
vi /etc/sysconfig/network-scripts/ifup-ib
```

   Change this:

```
    else
        echo datagram > /sys/class/net/${DEVICE}/mode
        # cap the MTU where we should based upon mode
        [ -z "$MTU" ] && MTU=2044
        [ "$MTU" -gt 4092 ] && MTU=4092
    fi
```

   to this (comment out the two lines using #):

```
    else
        echo datagram > /sys/class/net/${DEVICE}/mode
        # cap the MTU where we should based upon mode
#     [ -z "$MTU" ] && MTU=2044
#     [ "$MTU" -gt 4092 ] && MTU=4092
    fi
```

3. Stop ib_ipoib on all hosts, restart the Fabric Manager, and then restart ib_ipoib as shown below:

```
modprobe -r ib_ipoib
systemctl stop opafm
```

```
systemctl start opafm
opainfo | grep PortState
   PortState:     Active
modprobe ib_ipoib
ifup ib0
ifconfig ib0
ib0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 10236
        inet 10.228.216.150  netmask 255.255.255.0  broadcast 10.255.255.255
        inet6 fe80::211:7501:165:b0ec  prefixlen 64  scopeid 0x20<link>
…
```

The example above shows ib0 interface set to UD mode with AIP, MTU set to 10236 bytes as a result of the FM change. Make sure to check whether your MTU is set to no greater than 4092 for UD mode without AIP.

4. If you have twoOmni-Path Express ports in your system, make sure to bring up the ib1 interface with `ifup ib1`, in addition to the commands listed above.

7.3. krcvqs Tuning for IPoFabric Performance

When a large number of clients accesses one or more servers, such as parallel or NFS file systems, through IPoFabric (IPoIB), Cornelis recommends that you use an odd number of kernel receive queues. The number of kernel receive queues can be configured with the `krcvqs` parameter in the hfi1 driver. Odd number values, such as `krcvqs=3`, `5`, or `7`, have shown to be advantageous when all or most of the storage traffic is over IPoFabric in Connected Mode (CM).

NOTE

As an alternative to IPoFabric, verbs RDMA can be used to communicate from clients to storage servers. However, no advantage is reported for an odd number value of the `krcvqs` parameter for verbs RDMA.

Using krcvqs=5 and 3 provides good IPoFabric CM performance for single and multiple HFIs, respectively. However, it is highly recommended that you test to see which value for `krcvqs` works best for your application scenarios. Keep in mind that each of these receive contexts will be assigned/pinned to a different CPU core. Thus, if you have multiple HFIs active on a node:

Total number of kernel receive queues = Number of krcvqs x Number of active HFI*s*

For example, in an IP Router node with two HFIs connected to the first socket with `krcvqs=3`, 2 x 3 CPU cores (numbered 1–6), all on socket 0 (NUMA node 0), are assigned to receive context interrupts.

NOTE

`krcvqs` values larger than 3 could be used with multiple HFIs, but be aware of the multiplicative effect of multiple HFIs on this parameter.

NOTE

UD mode with AIP uses dedicated CPUs for receive contexts. Therefore, setting krcvqs=1 may be advantageous, particularly on lower core count processors and where IPoFabric UD traffic is the only or predominant type of traffic used on the node.

To set `krcvqs` or other driver parameters, refer to Section 4.3 "Setting HFI1 Driver Parameters" for details on using the `/etc/modprobe.d/hfi1.conf` file.

7.4. IPoIB Module Parameter Tuning

For clusters of larger scale or with certain nodes under heavy IPoFabric load, Cornelis recommends that you increase values of the IPoIB module parameters `recv_queue_size` and `send_queue_size`. The default values are 256 and 128, respectively, both configurable up to a maximum of 8192.

If you do not increase from the defaults of these parameters, you may experience large packet drops and reduced throughput.

To determine the existing `*queue_size` values, enter the following command:

```
grep . /sys/module/ib_ipoib/parameters/*queue_size
```

To change these values persistently, perform the following steps:

1. Enter the following command, specifying the new values:

   ```
   echo 'options ib_ipoib recv_queue_size=8192 send_queue_size=8192' >> /etc/modprobe.d/
   ib_ipoib.conf
   ```

2. Reboot the system for changes to take effect.

7.5. RPS and GSO Tuning for IPoFabric Performance

Receive Packet Steering (RPS) tuning may improve performance when a larger number of client nodes access services on one or more servers using IPoFabric transport in connected mode. In datagram mode with Accelerated IPoFabric enabled, RPS may not offer much improvement because AIP implements a form of receive side scaling. In terms of logic, RPS is a software implementation of receive-side scaling. Being in the software, it is necessarily called later in the datapath. RPS selects the CPU to perform protocol processing above the interrupt handler.

RPS requires a kernel compiled with the `CONFIG_RPS kconfig` symbol (on by default for SMP). Even when compiled, RPS remains disabled until explicitly configured. The list of CPUs to which RPS may forward traffic can be configured for each receive queue using a sysfs file entry:

```
/sys/class/net/<dev>/queues/rx-<n>/rps_cpus
```

Generic Segmentation Offload (GSO) uses the TCP or UDP protocol to send large packets. GSO performs segmentation/fragmentation operations, bypassing the NIC hardware. This is achieved by delaying segmentation until as late as possible, for example, when the packet is processed by the device driver. This became default behavior in RHEL 7.4.

7.5.1. RPS Tuning

Experience has shown that a good way to configure Receive Packet Steering (RPS) is to use all of the CPU cores local to the HFI. An example of how to do this is:

```
cat /sys/class/net/<dev>/device/local_cpus > /sys/class/net/<dev>/queues/rx-<n>/rps_cpus
```

where <dev> is the IPoIB port name.

This recommendation applies to Connected Mode only and does not apply with AIP in datagram mode.

To make the above tuning persist after reboot, refer to the next section that describes using the `/etc/rc.local` script file.

7.5.2. Persisting GSO and RPS Tuning

Perform the following steps using root privileges:

1. Ensure the Omni-Path Express hfi1 driver and ipoib driver are configured to autostart.
2. Add the following two lines to the file `/etc/rc.local`:

   ```
   ethtool -K <dev> gso off
   cat /sys/class/net/<dev>/device/local_cpus > /sys/class/net/<dev>/queues/rx-0/rps_cpus
   ```

   where <dev> is the IPoIB port name.
3. Make sure `/etc/rc.local` script file is executable by issuing:

   ```
   # chmod +x /etc/rc.local
   ```

4. Reboot to activate the changes.

7.6. TCP Parameter Tuning for IPoFabric Performance

The default TCP parameters supplied by Red Hat and SUSE operating systems perform reasonably well with the Omni-Path Express Fabric's IPoFabric, so *typically no tuning is required*.

If you have nodes where memory usage is not a concern, or that communicate with only a few other nodes, and where there is bi-directional (simultaneous read/write) traffic, the following tunings could improve total IPoFabric bi-directional throughput up to 10%.

To improve IPoFabric traffic performance, apply the following settings:

- For Intel Xeon Processors, set the following:

```
sudo sysctl -w net.ipv4.tcp_rmem="16384 349520 16777216"
sudo sysctl -w net.ipv4.tcp_wmem="16384 349520 16777216"
sudo sysctl -w net.core.rmem_max=16777216
sudo sysctl -w net.core.wmem_max=16777216
sudo sysctl -w net.core.somaxconn=2048 sudo sysctl -w net.ipv4.tcp_mtu_probing=1 sudo
sysctl -w net.core.netdev_max_backlog=250000
```

NOTE

The above commands will set these parameters and affect performance only until the next node reboot.

To make the above changes persistent, perform the following:

1. Edit the `/etc/sysctl.conf` file to add the appropriate settings listed above, in a format such as: `net.ipv4.tcp_rmem="16384 349520 16777216"`.
2. Run `sysctl -p` in a system start-up script, such as `/etc/rc.local`, to apply the tunings.

7.7. Kernel Boot Parameters to Avoid

If IPoFabric is used heavily for data transport for compute nodes to access file systems, then the following kernel parameters can significantly reduce your IPoFabric throughput:

- `idle=halt`
- `nohz_full=<range of CPU #s>`

Remove these parameters from the `/etc/default/grub` file, and refer to the instructions in Section 3.2.1 “Using the Intel P-State Driver” for the changes to take effect.

7.8. Tuned Utility Latency-Performance Profile

On some platforms, implementing the Tuned utility latency_performance profile has been shown to significantly improve IPoFabric latency. See Section 6.3.2 “Verbs Latency” for instructions on how to implement the Tuned latency-performance profile.

7.9. IPoFabric Benchmarks

This section provides information about IPoFabric benchmarks.

7.9.1. qperf

qperf is a benchmark that is included in Omni-Path Express Fabric Suite or OFA. It is designed to be run on a pair of nodes. You arbitrarily designate one node to be the server and the other to be the client.

To run a qperf test:

1. Run a ping test to ensure that ib_ipoib is running.
2. If you are concerned about performance, run a quick and useful qperf pretest on both the server and the client node.

```
qperf <server ipoib addr> -m 1M –ca 7  tcp_bw
```

In the previous command line:

Option	Description
`<server ipoib addr>`	Specifies the IP address of the ib0 port of the server node.
`-ca 7`	Can be considered a tuning that will pin both the server and client-side qperf process to core 7, which is typically on the first socket. To reduce run-to-run variation, you may want to avoid specifying core 0 or other lower number cores where OS or driver interrupts will interfere with the benchmark.
`-m`	Specifies a message size. In the example above, 1M specifies a 1 megabyte message size.

3. Type `<Ctrl+C>` on the server side to stop the test.

7.9.2. iperf3

iperf is a tool for active measurements of the maximum achievable bandwidth on networks that carry IP traffic. It supports tuning of various parameters related to timing, protocols, and buffers. For each test, iperf reports the bandwidth, loss, and other parameters.

To improve Omni-Path Express's IPoIB throughput, run the following iperf command lines:

1. For server, specify:

```
iperf3 -s -1 -f G -A 6
```

2. For client, specify:

```
iperf3 -c <server ipoib addr> -f G -t 12 -O 2 --len 1M -A 6
```

In the previous command lines:

Option	Description
`-A 6`	Sets CPU affinity to core 6. The optimal CPU core may vary based on system configuration and is not necessarily core 6.
`-t 12 -O 2`	Runs the test for 12 seconds, but omits the first two seconds when calculating the bandwidth result at the bottom of the output. This typically improves performance and makes performance results more stable.
`-f G`	Indicates the output bandwidth should be in gigabyte (GB) units.

Option	Description
`--len`	Indicates the message size in bytes, in this case 1M = 1 Megabyte.
`<server ipoib addr>`	Specifies the IPoIB address of the server.

NOTE

The latest version of iperf is available for download at: http://software.es.net/iperf/.

NOTE

To show the benefit of multiple parallel streams in UD mode with AIP, multiple iperf3 clients and servers will need to be run in parallel. To do this, run each iperf3 server and client pair with a different `--port` number to listen on/connect to. Alternatively, the iperf2 version 2.0.5 (08 Jul 2010) or later, rather than the iperf3 tool, can be used with the client specific `--parallel` option to result in separate iperf threads running on separate cores.

7.9.3. iperf2

iperf2 may be used with the `--parallel/-P` option in place of manually running multiple instances of iperf3 (described in the previous section). In many scenarios, this implementation is simpler and may provide higher throughput, especially with AIP.

The following is an example showing how to run with 16 parallel threads:

```
server: iperf2 -s
client: iperf2 -c <server ipoib addr> -P16 --len 1M
```

NOTE

With high thread counts, iperf2 may have problems connecting and may return the following:

`write failed: Connection reset by peer`

If this occurs, an adjustment to the iperf2 source code should help.

In `src/Listener.cpp`, change the line `rc = listen (mSettings->mSock, 5);` to `rc = listen (mSettings->mSock, 128);`, and re-compile.

8. Driver IRQ Affinity Assignments

The HFI1 host driver uses IRQ affinity assignments to distribute the CPU workload for SDMA engine interrupts, kernel receive queue interrupts, and the general interrupt. Over-subscribing CPU cores could be detrimental for performance and should be avoided if possible.

Large messages sent by applications using PSM2 or the Verbs API use the Send DMA mechanism (SDMA) for gathering data from host memory to build packets in the HFI1 send buffer. There are 16 SDMA engines to perform this task. This allows host software to have up to 16-way concurrency when using SDMA without locking. The HFI1 host driver sets up SDMA interrupts to notify the host of SDMA packet completions.

Kernel receive queue interrupts (receive interrupts) process incoming Verbs packets. The general interrupt handler processes all interrupts that are delivered using a legacy mechanism, which is not typically used in modern processors. In addition, the general interrupt handler also processes miscellaneous interrupts such as error interrupts.

The driver makes affinity selections for these interrupt handlers at driver initialization time. This section describes how these choices are made and how they can be influenced. Note that some of the details can vary from software release to software release.

8.1. Affinity Hints

Affinity hints allow device drivers to specify to which preferred CPU cores hardware interrupts should be bound. They are effective when a userspace process copies `/proc/irq/<IRQ #>/affinity_hint` to `/proc/irq/<IRQ #>/smp_affinity`.

8.2. Role of Irqbalance

Irqbalance is a daemon for load balancing interrupts on different CPU cores. When it uses the argument `--policyscript=/etc/sysconfig/opa/hintpolicy_exact_hfi1.sh`, it applies the policy of setting the hardware interrupts to CPU core mappings exactly how the device drivers suggested. Otherwise, irqbalance dynamically tries to distribute them across CPU cores based on the PCI device class of the device that owns the interrupt.

For the HFI1 driver, irqbalance `--policyscript=/etc/sysconfig/opa/hintpolicy_exact_hfi1.sh` is strongly recommended to preserve interrupt locality and to have a dedicated CPU core for high priority interrupts such as receive interrupts.

NOTE

Verify the `IRQBALANCE_ARGS` parameter is set according to step 2 of Section 3.1 "irqbalance". See the *Cornelis Omni-Path Express Fabric Software Release Notes* for instructions on installing the Omni-Path Express Host Software to apply the `IRQBALANCE_ARGS` setting automatically during the install process.

8.2.1. Reduce TxRx Interrupts for IRQ

The APIC (Advanced Programmable Interrupt Controller) slot for each CPU has 96 slots available for driver interrupts. On systems that have the i40e Ethernet driver loaded, many of these interrupts are consumed by Ethernet TxRx interrupts, and irqbalance does not behave correctly for Omni-Path Express Host Software. For systems where the i40e driver is only used for management, and performance is not essential on the Ethernet network, the following workaround can be used until a kernel with the fix is in use:

1. Reduce the TxRx interrupts of the Ethernet device to a lower number such as 16:

```
ethtool –L <dev> combined 16
```

2. Restart irqbalance, as outlined in Section 3.1 “irqbalance”.

NOTE

This workaround should be evaluated for Ethernet performance if that is a concern, such as in an IP router.

8.3. Identifying to Which CPU Core an Interrupt is Bound

8.3.1. Method 1

```
echo "irq name core";for irq in `grep -i hfi  /proc/interrupts | grep -e sdma -e
kctxt | awk '{print $1}' | sed 's/://g'`;do echo `grep " ${irq}: " /proc/interrupts |
sed 's/nd kctxt/nd_kctxt/g' | awk '{print $((NF-1)),$NF}'` "  "  `cat /proc/irq/${irq}/
smp_affinity_list`;done
```

Example result for a 26 core CPU and default Omni-Path Express driver parameters:

```
OPA driver parameters:
irq name   core
hfi1_0 sdma0    3
hfi1_0 sdma1    4
hfi1_0 sdma2    5
hfi1_0 sdma3    6
hfi1_0 sdma4    7
hfi1_0 sdma5    8
hfi1_0 sdma6    9
hfi1_0 sdma7    10
hfi1_0 sdma8    11
hfi1_0 sdma9    12
hfi1_0 sdma10    13
hfi1_0 sdma11    14
hfi1_0 sdma12    15
hfi1_0 sdma13    16
hfi1_0 sdma14    17
hfi1_0 sdma15    18
hfi1_0 kctxt0    0
```

```
hfi1_0 kctxt1    1
hfi1_0 kctxt2    2
hfi1_0 nd_kctxt3    19
hfi1_0 nd_kctxt4    20
hfi1_0 nd_kctxt5    21
hfi1_0 nd_kctxt6    22
hfi1_0 nd_kctxt7    23
hfi1_0 nd_kctxt8    24
hfi1_0 nd_kctxt9    25
hfi1_0 nd_kctxt10    3
```

Essentially, this script finds all of the interrupt numbers in `/proc/interrupts` that are used by HFI1, and then looks up their affinity settings in `/proc/irq/$irq/smp_affinity_list`.

NOTE

The script above may need to be modified to work with the OS-specific versions of bash and awk.

8.3.2. Method 2

This is a shorter, easier-to-remember command, and produces more readable output; but, it is not guaranteed to work in all OSes, and your system administrator may have limited the verbosity level in `/var/log/messages` so that this command does not provide the indicated output.

Command line:

```
# dmesg | grep hfi1 | grep IRQ
```

Results:

```
[   10.741235] hfi1 0000:18:00.0: hfi1_0: IRQ: 366, type GENERAL  -> cpu: 0
[   10.748239] hfi1 0000:18:00.0: hfi1_0: IRQ: 367, type SDMA engine 0 -> cpu: 3
[   10.755625] hfi1 0000:18:00.0: hfi1_0: IRQ: 368, type SDMA engine 1 -> cpu: 4
[   10.763006] hfi1 0000:18:00.0: hfi1_0: IRQ: 369, type SDMA engine 2 -> cpu: 5
[   10.770391] hfi1 0000:18:00.0: hfi1_0: IRQ: 370, type SDMA engine 3 -> cpu: 6
[   10.777775] hfi1 0000:18:00.0: hfi1_0: IRQ: 371, type SDMA engine 4 -> cpu: 7
[   10.785158] hfi1 0000:18:00.0: hfi1_0: IRQ: 372, type SDMA engine 5 -> cpu: 8
[   10.792545] hfi1 0000:18:00.0: hfi1_0: IRQ: 373, type SDMA engine 6 -> cpu: 9
[   10.799926] hfi1 0000:18:00.0: hfi1_0: IRQ: 374, type SDMA engine 7 -> cpu: 10
[   10.807534] hfi1 0000:18:00.0: hfi1_0: IRQ: 375, type SDMA engine 8 -> cpu: 11
[   10.815138] hfi1 0000:18:00.0: hfi1_0: IRQ: 376, type SDMA engine 9 -> cpu: 12
[   10.822747] hfi1 0000:18:00.0: hfi1_0: IRQ: 377, type SDMA engine 10 -> cpu: 13
[   10.830442] hfi1 0000:18:00.0: hfi1_0: IRQ: 378, type SDMA engine 11 -> cpu: 14
[   10.838140] hfi1 0000:18:00.0: hfi1_0: IRQ: 379, type SDMA engine 12 -> cpu: 15
[   10.845834] hfi1 0000:18:00.0: hfi1_0: IRQ: 380, type SDMA engine 13 -> cpu: 16
[   10.853529] hfi1 0000:18:00.0: hfi1_0: IRQ: 381, type SDMA engine 14 -> cpu: 17
[   10.861225] hfi1 0000:18:00.0: hfi1_0: IRQ: 382, type SDMA engine 15 -> cpu: 18
[   10.869093] hfi1 0000:18:00.0: hfi1_0: IRQ: 383, type RCVCTXT ctxt 0 -> cpu: 0
[   10.876847] hfi1 0000:18:00.0: hfi1_0: IRQ: 384, type RCVCTXT ctxt 1 -> cpu: 1
[   10.884602] hfi1 0000:18:00.0: hfi1_0: IRQ: 385, type RCVCTXT ctxt 2 -> cpu: 2
[   15.956554] hfi1 0000:18:00.0: hfi1_0: IRQ: 386, type NETDEVCTXT ctxt 3 -> cpu: 19
```

```
[   15.985097] hfi1 0000:18:00.0: hfi1_0: IRQ: 387, type NETDEVCTXT ctxt 4 -> cpu: 20
[   15.988440] hfi1 0000:18:00.0: hfi1_0: IRQ: 388, type NETDEVCTXT ctxt 5 -> cpu: 21
[   15.991718] hfi1 0000:18:00.0: hfi1_0: IRQ: 389, type NETDEVCTXT ctxt 6 -> cpu: 22
[   15.995003] hfi1 0000:18:00.0: hfi1_0: IRQ: 390, type NETDEVCTXT ctxt 7 -> cpu: 23
[   15.998285] hfi1 0000:18:00.0: hfi1_0: IRQ: 391, type NETDEVCTXT ctxt 8 -> cpu: 24
[   16.001589] hfi1 0000:18:00.0: hfi1_0: IRQ: 392, type NETDEVCTXT ctxt 9 -> cpu: 25
[   16.004910] hfi1 0000:18:00.0: hfi1_0: IRQ: 393, type NETDEVCTXT ctxt 10 -> cpu: 3
```

In this case, it can be observed that:

- The SDMA engine interrupts were assigned by the HFI1 host driver to CPU cores starting at core 3 and ending at CPU core 18.
- CPU cores 0, 1, and 2 were skipped in the SDMA engine interrupt assignments because they are reserved for kernel receive queue interrupts and the general interrupt.
- The AIP `NETDEVCTXT` cores use the remaining available cores 19 through 25 and wrap back to core 3. The intent of these assignments are to share cores with SDMA engines.

8.3.3. Method 3

`/proc/irq/<IRQ #>/smp_affinity` is a variable containing hex values where each bit set represents a CPU. Reading this variable shows the CPUs to which a specific interrupt is bound. The IRQ number can be found by looking at `/proc/interrupts`.

Example 1:

```
cat /proc/irq/368/smp_affinity_list
4
```

IRQ 368 for SDMA engine 1 is bound to CPU core 4. This can also be expressed in hex values:

Example 2:

```
cat /proc/irq/368/smp_affinity
00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,0
0000000,00000000,00000000,00000010
```

8.4. Manually Changing IRQ Affinity

The HFI1 host driver will, by default, provide interrupt affinity hints to obtain the best performance for most computing environments. However, in some circumstances custom IRQ settings may be required to optimize performance. For example, on Intel Xeon Phi x200 Product Family processors, locating a user process and SDMA interrupt on different CPU cores that share the same L2 cache reduces latency and increases throughput due to thread synchronization as long as the user process is associated with the specific SDMA engine. This pair of CPU cores sharing the L2 cache is also referred to as a *tile*. Instructions on mapping a user process to an SDMA engine can be found in this section.

General guidelines for manually tuning IRQ affinity:

- Bind interrupts at the beginning of the CPU core range in the local NUMA node. This yields better performance than binding interrupts starting from the end of the CPU core range.
- Bind kernel receive queue interrupts to their own dedicated CPU core as they tend to be heavily utilized.
- Avoid affinitizing an SDMA engine interrupt and a kernel receive queue interrupt to the same CPU core. Line rate performance cannot be achieved with the incurred context switch overheads and an oversubscribed CPU core.
- Pin user processes to non-interrupt CPUs first if possible.
- Keep all interrupts bound to CPU cores in the local NUMA node as HFI1 is local NUMA centric.

8.4.1. Identifying and Changing the Number of VLs

The number of VLs that are used is a fabric manager parameter (see `opafm.xml`). `opaportinfo` can be used to determine the number of enabled VLs. The following output shows that one data VL is in use (VL0) as all other data VLs (not VL15) have the MTU configured to 0 bytes. This is a fairly typical configuration:

```
MTU Supported: (0x7) 10240 bytes
MTU Active By VL:
00:10240 01:    0 02:    0 03:    0 04:    0 05:    0 06:    0 07:    0
08:    0 09:    0 10:    0 11:    0 12:    0 13:    0 14:    0 15: 2048
16:    0 17:    0 18:    0 19:    0 20:    0 21:    0 22:    0 23:    0
24:    0 25:    0 26:    0 27:    0 28:    0 29:    0 30:    0 31:    0
```

The driver partitions the configured number of SDMA engines over the number of VLs. For example, 16 SDMA engines distributed over one VL means that all SDMA engines can be used by that VL.

8.4.2. Changing Kernel Receive Queues

The number of kernel receive queues defaults to 2 but can be modified using the krcvqs module parameter:

```
parm:    krcvqs:Array of the number of non-control kernel receive queues by VL (array of
uint)
```

Further information on how to set a module parameter can be found in Section 4.3 “Setting HFI1 Driver Parameters”.

Note that if the Section 8.3.1 “Method 1” script output indicates a CPU range similar to the output below, then it is likely that the irqbalance service is not running and needs to be started. For more information, refer to Section 3.1 “irqbalance”.

```
sdma0 0-13
sdma1 0-13
```

```
sdma2 0-13
sdma3 0-13
sdma4 0-13
sdma5 0-13
sdma6 0-13
sdma7 0-13
sdma8 0-13
sdma9 0-13
sdma10 0-13
sdma11 0-13
sdma12 0-13
sdma13 0-13
sdma14 0-13
sdma15 0-13
```

8.4.3. Changing SDMA Engines

The number of SDMA engines (default 16) that are used by the HFI1 host driver can be modified using the num_sdma module parameter:

```
parm:    num_sdma:Set max number SDMA engines to use (uint)
```

Further information on how to set a module parameter can be found in Section 4.3 "Setting HFI1 Driver Parameters".

8.4.4. Changing Interrupt CPU Bindings

Ensure there are no user processes running that periodically change the values for:

```
/proc/irq/<IRQ #>/smp_affinity
```

On most systems, irqbalance will be the daemon updating the smp_affinity file. Therefore, you should disable it unless it is running with the `--oneshot` option.

Determine the hex value for the CPU where the IRQ will be bound to and write it to `/proc/irq/<IRQ #>/smp_affinity`.

Example 1:

```
echo 00003fff > /proc/irq/105/smp_affinity
```

IRQ 105 is now bound to the CPU core range 0-13.

Example 2:

```
echo 8 > /proc/irq/35/smp_affinity
```

IRQ 35 is now bound to CPU 3.

8.4.5. Mapping from MPI Processes to SDMA Engines

Typically, MPI jobs are assigned to a specific service level; and, this is mapped to a service channel and virtual lane by mappings set up by the fabric manager (configured by the

opafm.xml configuration file). Each MPI process is associated with a VL, a context number and potentially a sub-context number (if context sharing is used). The VL selects the set of SDMA engines that can be used. In many cases, only one VL is configured (as shown in the output above) allowing an MPI job to use all 16 SDMA engines. The context and sub-context number (if required) are used to distribute the MPI processes over these SDMA engines. This is essentially a round-robin mapping. However, the context numbers that are assigned to MPI jobs typically do not start at 0, so there is often an offset for the first SDMA engine that is used.

Omni-Path Express Fabric Suite offers a way to control a mapping of CPU core A to SDMA engine X. Thus, any MPI process running on that Core A will get assigned to SDMA engine X. To accomplish this, the driver exposes new sysfs entries per SDMA engine. For each SDMA engine, the driver creates a new sysfs directory called sdma<N>, where N is the SDMA engine ID.

```
/sys/devices/pci…/infiniband/hfi1_<N>/sdma<N>/
```

pci… refers to several directory levels (a method for listing those directory names is provided below). Each directory will expose two new files (attributes):

- cpu_list: A read/write attribute that allows you to set up the process to SDMA engine assignments based on which CPU a process is running.
- vl: A read-only attribute that allows you to find out the existing SDMA engine to Virtual Lane mappings.

To print out the full path names to these files, use either of the following commands:

- find /sys/devices/ -name vl | grep hfi1
- find /sys/devices/ -name cpu_list | grep hfi1

Or, if you only want to print the files for the first HFI, use hfi1_0:

```
$ find /sys/devices/ -name cpu_list | grep hfi1_0
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma0/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma1/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma2/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma3/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma4/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma5/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma6/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma7/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma8/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma9/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma10/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma11/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma12/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma13/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma14/cpu_list
/sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma15/cpu_list
```

NOTE

Since the typical installation only uses one VL, number 0, and there are 16 SDMA engines for that VL, we will not deal with the VL number from this point on.

To print the CPUs that the SDMA engines and Receive contexts (# defined by krcvqs parameter), use this command:

```
$ dmesg | grep hfi1_0 | grep IRQ
[71396.873706] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 43, type GENERAL  -> cpu: 0
[71396.873736] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 44, type SDMA engine 0 -> cpu: 3
[71396.873763] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 45, type SDMA engine 1 -> cpu: 4
[71396.873789] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 46, type SDMA engine 2 -> cpu: 5
[71396.873816] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 47, type SDMA engine 3 -> cpu: 6
[71396.873843] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 49, type SDMA engine 4 -> cpu: 7
[71396.873878] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 50, type SDMA engine 5 -> cpu: 8
[71396.873905] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 51, type SDMA engine 6 -> cpu: 9
[71396.873937] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 52, type SDMA engine 7 -> cpu: 10
[71396.873963] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 53, type SDMA engine 8 -> cpu: 11
[71396.873989] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 54, type SDMA engine 9 -> cpu: 12
[71396.874015] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 55, type SDMA engine 10 -> cpu: 13
[71396.874040] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 56, type SDMA engine 11 -> cpu: 14
[71396.874066] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 57, type SDMA engine 12 -> cpu: 15
[71396.874098] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 58, type SDMA engine 13 -> cpu: 16
[71396.874125] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 59, type SDMA engine 14 -> cpu: 17
[71396.874151] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 60, type SDMA engine 15 -> cpu: 18
[71396.874403] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 61, type RCVCTXT ctxt 0 -> cpu: 0
[71396.874743] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 62, type RCVCTXT ctxt 1 -> cpu: 1
[71396.875100] hfi1 0000:01:00.0: hfi1_0: IRQ vector: 63, type RCVCTXT ctxt 2 -> cpu: 2
```

NOTE

If this `dmesg` command does not work, you can use the "Method 1 script" above to get the SDMA engine to CPU core assignment.

As an example, you can assign a single core or a range of cores to the cpu_list file as follows:

```
# echo “0-3” > /sys/devices/pci0000:00/0000:00:02.0/0000:01:00.0/infiniband/hfi1_0/sdma0/
cpu_list
```

NOTE

You need to use the "**`find /sys/devices/ -name cpu_list | grep hfi1_0`**" command shown above to find the correct pathname to use with this echo command for sdma engine 0 on hfi1_0. The above directories `pci0000:00/0000:00:02.0/0000:01:00.0`, most likely, will not exist on your system.

9. Fabric Manager Performance

In Omni-Path Express Fabric Suite 10.8 and newer, the opafm service has enhanced parallelization that greatly improves FM performance on large clusters. Still, for very large fabrics (on the scale of thousands or more nodes), a significant amount of Fabric sweep time is spent performing cable information checks. The cable info data is used by FastFabric for deployment verification and can be gathered directly by the SMA when detail is specifically needed.

For general cluster operation, it is not typically required on each sweep. Disabling these checks has been shown to reduce fabric sweep time by as much as two minutes.

To disable cable info gathering, perform the following steps:

1. Set the following line in `opafm.xml`:

```
<CableInfoPolicy>none</CableInfoPolicy>
```

2. Restart the Fabric Manager:

```
systemctl restart opafm
```

9.1. Reducing Fabric Congestion

There are many advanced fabric management features that may be explored to fine tune and get the best performance from an Omni-Path Express Fabric. Applications that span multiple switches in a fabric and are highly bandwidth or latency sensitive may see impacts from fabric congestion. Congestion occurs when:

- The same fabric link is being used by multiple hosts communicating through the fabric and the aggregate bandwidth required is more than the available bandwidth.
- There is an imbalance in performance between the sender and receiver.

Omni-Path Express provides several methods for identifying congestion hotspots as well as many features that can be used to potentially avoid congestion in the fabric.

9.2. Routing Features

This section provides tuning information related to routing features.

9.2.1. Dispersive Routing

Dispersive routing is an advanced routing feature that attempts to avoid congestion hotspots by using multiple paths through the fabric to reach the same destination HFI port. Unlike adaptive routing, where routing tables are dynamically updated by the switch firmware as congestion is observed, dispersive routing is unaware of the congestion levels in the fabric. Dispersive routing tries to solve congestion-related issues by breaking messages

into chunks and spreading the message across the multiple predetermined paths that have been configured.

Refer to "LMC, Dispersive Routing, and Fabric Resiliency" in the *Cornelis Omni-Path Express Fabric Suite Fabric Manager User Guide* and the "Dispersive Routing" section of the *Cornelis Omni-Path Express Fabric Host Software User Guide* for a more detailed explanation of dispersive routing.

To enable dispersive routing, you can simply modify the Lid Mask Control (LMC) parameter in the Fabric Manager configuration file (`opafm.xml`), and restart the opafm service. Currently, dispersive routing is disabled (`LMC=0`) in the `opafm.xml` file by default. The FM assigns 2^LMC unique LIDs per end node port.

9.2.1.1. Recommendation

Cornelis recommends that dispersive routing be enabled to take advantage of the potentially significant performance improvements, especially in highly subscribed fabrics. The recommended LMC value we suggest using is either 2 or 3, depending on the size of the fabric.

When `LMC=3`, 8 LIDs will be assigned to each end node port which could, for larger fabrics (over 4k nodes), result in the Fabric Manager using an excessive number of LIDs. We recommend using an LMC value of 3 for fabrics less than 4k nodes, and 2 for fabrics larger than 4k nodes.

The use of dispersive routing can be fine-tuned at the application level via PSM2 environment variables. While enabling dispersive routing in the fabric is done at the system administrator/global level, the following workflow is recommended to enable it only for users who specifically request it. Other users will by default use static routing:

- LMC should be set greater than 0 (recommended value is 2 or 3) prior to starting the Fabric Manager.
- System administrators should disable all users from automatically using dispersive routing by setting `PSM2_PATH_SELECTION=static_base` in every user's environment, such as with `/etc/profile`.
- Users who want to run with dispersive routing enabled should set `PSM2_PATH_SELECTION` to "adaptive" in their environment.

The performance benefits realized when using dispersive routing will depend on many factors, including message sizes of the application and contention with other applications running on the fabric.

For further fine tuning of the behavior of dispersive routing, we recommend that you test your cluster workloads and experiment with different PSM2 variables. For example, for applications that send messages smaller than the setting of the `PSM2_MQ_RNDV_HFI_WINDOW` (this is the parameter that controls the size of the chunks that a message is broken down into), only 1 DLID will be used, even when dispersive routing is enabled. If necessary, depending on the application, you can adjust this to take advantage of dispersive routing. However, take care when adjusting this variable because the default value has been carefully chosen to optimize point-to-point bandwidth.

PSM2 also has variables that determine behavior for small and large message sizes. You can use the variable `PSM2_PATH_NO_LMC_RANGE` to control the range of message sizes for which dispersive routing is not utilized. For example, setting `PSM2_PATH_NO_LMC_RANGE=524288:1048576` would disable dispersive routing for message sizes between 512KB and 1MB, inclusively.

For more information on tuning of PSM2 variables, refer to the *Cornelis Performance Scaled Messaging 2 (PSM2) Programmer's Guide*.

9.2.2. Adaptive Routing

With Medium Grained Adaptive Routing (MGAR), switches can adjust their routes dynamically on a port-to-port basis, while applications are running, in order to alleviate congestion and potentially improve bandwidth between nodes and improve application performance. MGAR has three main tuning knobs:

- Algorithm
- Threshold - the congestion threshold at which adaptations occur
- ARFrequency - the frequency at which the switch ports are monitored for congestion exceeding the thresholds

Refer to the section on Adaptive Routing in the *Cornelis Omni-Path Express Fabric Suite Fabric Manager User Guide* for more information on how to configure the parameters for Adaptive Routing. Recommendations for these parameters are discussed below.

9.2.2.1. Recommendation

If you suspect inter-switch link congestion is causing performance degradation, we recommend testing your cluster workloads with MGAR enabled and making a guided decision on whether to enable it or not. Some HPC applications may not benefit from using MGAR, and some HPC applications, even when running in isolation, are negatively impacted. However, there are some use cases, such as highly directional cluster traffic, where improved application performance could be obtained. An example is a full-bisectional bandwidth test, where each node on the cluster is sending to a neighbor node on the other section of the fabric. In this scenario, Cornelis recommends `ARFrequency=1`, `Threshold=5`, `6`, or `7`. The traffic must be sustained long enough (on the order of multiple seconds) in order for AR to properly adjust.

The combination of ARFrequency and Threshold should be selected carefully to avoid possible negative performance impacts, particularly if routes are dynamically changing too often. In general, the lower the ARFrequency setting (actually implying a higher frequency), the faster adaptive routing will adjust routes. AR Threshold is the opposite. The higher the AR Threshold setting, such as 6 or 7, there is an increase the risk of routes changing unnecessarily and causing a negative performance impact.

ARFrequency	0 (64ms)	1 (128ms)	2 (256ms)	3 (512ms)	4 (1.024s)	5 (2.048s)	6 (4.096s)	7 (8.192s)
Threshold	0	1 (100%)	2 (90%)	3 (80%)	4 (70%)	5 (65%)	6 (60%)	7 (55%)

Appendix A. Older Revisions

Date	Rev	Description
Aug 2020	21.0	Updated "Best Practices" to include pointer to "Updating the Certificate" in the *Cornelis Omni-Path Express Fabric Switches GUI User Guide*.
Jul 2020	20.0	• Added new section "MPI Benchmark Fundamentals". • Added new section "MPI Collective Tunings". • Added new section "Using MPI Multiple Endpoints with Intel MPI Library". • Added new section "1 GB Huge Pages".
Apr 2020	19.0	Added "GPUDirect Requirements".
Jan 2020	18.0	• Added new section: "Slurm Settings". • Added new section: "IPoIB Module Parameter Tuning". • Added new section: "GPFS Settings for Large Clusters". • Added new section: "Reducing Fabric Congestion".
Oct 2019	17.0	• Added "Dual/Multi-Rail Tuning". • Added new section: "Lustre Multi-Rail Support with Omni-Path Express". • Add new section: "iperf2".
Jun 2019	16.0	Added new section: "Scalable Endpoints".
Apr 2019	15.0	Added new section: "Reduce TxRx Interrupts for IRQ".
Mar 2019	14.0	Added new section: "Advanced Fabric Management Feature Training".
Dec 2018	13.0	• Added new section: "Omni-Path Express Fabric Performance Tuning Quick Start Guide" and removed the "Performance Tuning Checklist". • Added content to Using the Intel P-State Driver. • Add new Section: "Enabling Huge Pages for Shared Memory Communication with the Intel MPI Library". • Merged all Verbs content into one section ("System Settings for Verbs Performance").
Sep 2018	12.0	• Added "Reducing System Jitter". • Reorganized the sections in "IPoFabric Performance" covering IPoFabric Connected Mode and Datagram mode.

Date	Rev	Description
Apr 2018	11.0	• Added new section: "Address Resolution Protocol Thresholds on Large Fabrics". • Added new section: "SPEC MPI2007 Performance Tuning". • Added new section: "Assigning Virtual Lanes to MPI Workloads". • Added new section: "RDMA_CM Requirements". • Added "Verbs Latency". • Added new section: "krcvqs Tuning for IPoFabric Performance". • Added new section: "IP Router Performance Fix". • Added new section: "Kernel Boot Parameters to Avoid".
Oct 2017	10.0	• Updated "Tuning for High-Performance LINPACK Performance": Added new subsections: – "Expected Levels of Performance" – "Selection of HPL Binary and MPI" – "MPI Flags and Proper Job Submission Parameters/Syntax" – "HPL.dat Input File" – "Recommended Procedure for Achieving Best HPL Score" • Added new sections: "RPS Tuning and Persisting GSO" and "RPS Tuning".
Aug 2017	9.0	• Added "Terminology". • Added "Intel Xeon Scalable Processor". • Added " MPI 2017: New Support of OFI as a Fabric". • Added "IBM Storage Scale (aka GPFS) Tuning for Omni-Path Express". • Added "RPS and GSO Tuning for IPoFabric Performance".
May 2017	8.0	Added new sections: "MPI Collective Scaling Guidelines" for Large Clusters and "Driver Parameter Settings for Intel Xeon Phi x200 Product Family".
Apr 2017	7.0	• Added new section "Do Not Enable intel_iommu". • Added new section "Dealing with Memory Fragmentation". • Added new section "Switching to the Intel P-State Driver to Run Certain FastFabric Tools". • Added new section "Tuning for High-Performance LINPACK Performance". • Added new section "Tuning for Improved Performance on QCD Applications". • Added new section "GPUDirect RDMA Tuning for MPI Benchmarks and Applications". • Added new section "Accelerated RDMA".

Date	Rev	Description
Dec 2016	6.0	• Added new section “Cornelis Omni-Path Express Fabric Design Generator for Cornelis Omni-Path Express Fabric” to “Preface”. • Added new section “Tuning for MPI Performance on Nodes with Intel Xeon Phi x200 Product Family”. • Added new section “Tuning for Improved 1 KB to 8 KB Message Bandwidth at High Processes per Node”. • Added new section “TCP Parameter Tuning for IPoFabric Performance”.
Aug 2016	5.0	Added new Section “Performance Tuning Checklist”.
May 2016	4.0	• Added Section “Driver IRQ Affinity Assignments”. • Added Section “Next-Generation Processor Family”. • Added Section “Tuning for LS-DYNA Performance”.
Apr 2016	3.0	• Added Section “Using the P-State Driver”. • Added Section “Tuning for MPI Performance on Nodes with CPUs”. • Added Section “Parallel File System Concurrency Improvement”. • Added Section “Lustre Parameter Tuning for Intel Omni-Path”. • Added Section “IPoFabric Datagram (UD) Mode Configuration”.
Feb 2016	2.0	Clarified language in “Platform Settings” and “irqbalance”.
Nov 2015	1.0	Initial Document.