IBM Spectrum MPI
Version 10 Release 1.0.2

# *User's Guide*

GC27-8265-01

# IBM Spectrum MPI
Version 10 Release 1.0.2

# *User's Guide*

GC27-8265-01

**Note**
Before using this information and the product it supports, read the information in "Notices" on page 63.

This edition applies to version 10, release 1, modification 0.2 of IBM Spectrum MPI (product number 5725-G83) and to all subsequent releases and modifications until otherwise indicated in new editions.

# Contents

# Tables

# Getting started with this information

| **Disclaimer:** |
| --- |
| The functions or features found herein may not be available on all operating systems or platforms and do not indicate the availability of these functions or features within the IBM® product or future versions of the IBM product. The development, release, and timing of any future features or functionality is at IBM's sole discretion. IBM's plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. The information mentioned is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The information may not be incorporated into any contract and it should not be relied on in making a purchasing decision. |

This information explains parallel programming as it relates to IBM Spectrum™ IBM, IBM's implementation of Open MPI 2.0. It includes information about developing, running, and optimizing parallel applications for use in a distributed memory environment.

IBM Spectrum MPI is a complete MPI implementation, based on the Open MPI open source project, and is designed to comply with all the requirements of the Message Passing Interface standard, MPI: A Message-Passing Interface Standard, Version 3.1, University of Tennessee, Knoxville, Tennessee, June 4, 2015.

For information about Open MPI, and to obtain official Open MPI documentation, refer to the Open MPI web site (www.open-mpi.org).

This information assumes that one of the currently supported Linux distributions and IBM Spectrum MPI are already installed. For information about the supported Linux distributions and installing IBM Spectrum MPI see *IBM Spectrum MPI: Installation*.

**Note:** This document borrows heavily from the information that is provided on the Open MPI web site (www.open-mpi.org). In many cases, this document explains a topic at a high level, and then points users to the Open MPI website for more detailed information.

## Who should use this information

This information is intended for experienced programmers who want to develop parallel applications using the C or FORTRAN programming language. It is also intended for end users who need to run parallel programs. Some of the information covered here should also interest system administrators.

Readers of this information should know C or FORTRAN and should be familiar with Linux commands, file formats, and special files. They should also be familiar with MPI (Message Passing Interface) and Open MPI concepts. In addition, readers should be familiar with distributed-memory machines. Where necessary, background information relating to these areas is provided. More commonly, you are referred to the appropriate documentation.

## Conventions and terminology used in this information

Table 1 shows the conventions used in this information:

*Table 1. Conventions*

| Convention | Usage |
|---|---|
| **bold** | Environment variables. |
| `monospace` | Examples and information that the system displays, command line options, file names, pathnames. |
| **`bold monospace`** | Command names and parameter names. |
| *italic* | *Italic* words or characters represent variable values that you must supply.<br><br>Italics are also used for unit titles, the first use of a term, and general emphasis in text. |
| <*key*> | Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, <**Enter**> refers to the key on your terminal or workstation that is labeled with the word *Enter*. |
| \ | In command examples, a backslash indicates that the command or coding example continues on the next line. For example:<br>`mkcondition -r IBM.FileSystem -e "PercentTotUsed > 90" \`<br>`-E "PercentTotUsed < 85" -m d "FileSystem space used"` |
| {*item*} | Braces enclose a list from which you must choose an item in format and syntax descriptions. |
| [*item*] | Brackets enclose optional items in format and syntax descriptions. |
| <**Ctrl-***x*> | The notation <**Ctrl-***x*> indicates a control character sequence. For example, <**Ctrl-c**> means that you hold down the control key while pressing <**c**>. |
| *item*... | Ellipses indicate that you can repeat the preceding item one or more times. |
| \| | • In *synopsis* statements, vertical lines separate a list of choices. In other words, a vertical line means *Or*.<br>• In the margin of the document, vertical lines indicate technical changes to the information. |

## Prerequisite and related information

IBM Spectrum MPI is a member of the IBM Spectrum Computing family (www.ibm.com/systems/spectrum-computing/).

The IBM Spectrum MPI library consists of:

- *IBM Spectrum MPI: Installation*, GC27-8264-01
- *IBM Spectrum MPI: User's Guide*, GC27-8265-01

To access the most recent IBM Spectrum MPI documentation in PDF and HTML format, refer to IBM Knowledge Center (www.ibm.com/support/knowledgecenter), on the web.

The IBM Spectrum MPI books are also available in PDF format from the IBM Publication Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss), on the web.

It is easiest to locate a book in the IBM Publications Center by supplying the book's publication number. The publication number for each of the IBM Spectrum MPI books is listed after the book title in the preceding list.

IBM Platform LSF® (Load Sharing Facility) also works in conjunction with IBM Spectrum MPI. The LSF publications can be found in IBM Knowledge Center (www.ibm.com/support/knowledgecenter) and the IBM Publication Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss).

## Terminology

For terms and definitions related to IBM Spectrum MPI, see IBM Terminology (www.ibm.com/software/globalization/terminology/).

# Chapter 1. Getting started

Before using IBM Spectrum MPI, it is important to understand the environment in which you will be creating and running your applications, as well as its requirements and limitations.

This information assumes that one of the currently-supported Linux distributions is already installed. It also assumes that you have already installed IBM Spectrum MPI. For information about installing IBM Spectrum MPI, refer to *IBM Spectrum MPI: Installation*.

For information about the hardware and the operating systems that are supported by IBM Spectrum MPI, refer to the current announcement letter at IBM Offering Information (http://www.ibm.com/common/ssi/index.wss?request_locale=en).

## Introduction

IBM Spectrum MPI is a high-performance implementation of the MPI (Message Passing Interface) Standard. It is widely used in the high-performance computing (HPC) industry for developing scalable, parallel applications.

IBM Spectrum MPI supports a broad range of industry-standard platforms, interconnects, and operating systems, helping ensure that parallel applications can run almost anywhere.

IBM Spectrum MPI offers:

**Portability**
IBM Spectrum MPI allows a developer to build a single executable that can take advantage of the performance features of a wide variety of interconnects. As a result, applications have optimal latency and bandwidth for each protocol. This reduces development effort and enables applications to use the latest technologies on Linux without the need to recompile and relink applications. Application developers can confidently build and test applications on small clusters of machines, and deploy that same application to a larger cluster.

**Network optimization**
IBM Spectrum MPI supports a wide variety of networks and interconnects. This enables developers to build applications that run on more platforms, thereby reducing testing, maintenance, and support costs.

**Collective optimization**
IBM Spectrum MPI offers a library of collectives called *libcollectives*, which:

- Supports the seamless use of GPU memory buffers
- Offers a range of algorithms that provide enhanced performance, scalability, and stability for collective operations
- Provides advanced logic to determine the fastest algorithm for any given collective operation.

## Limitations

Some IBM Spectrum MPI product features are subject to certain limitations, as explained in this section.

- IBM Spectrum MPI is not ABI compatible with any other MPI implementations such as Open MPI, Platform MPI, or IBM PE Runtime Edition.
- The IBM Spectrum MPI collectives component (libcollectives) does not support intercommunicators. For intercommunicator collective support, IBM Spectrum MPI relies on Open MPI collective components.
- Dynamic process management (dynamic tasking) is not supported.
- The use of the MPI_File_set_atomicity() call is not supported.
- When creating n-dimensional topologies using MPI_Dims_create, ndims must be greater than 0.
- When running Spectrum MPI over TCP, on nodes with a virtual adapter, users must specify the correct adapter to use by using `-netaddr`, because Spectrum MPI does not ignore virbr# named devices.
- Multithreaded I/O is not supported.
- When running with PAMI, adapter affinity is enabled by default. This generally leads to better performance when CPU binding is enabled (as it is by default). This instructs each rank to use the InfiniBand adapter that is physically closest to the core where the rank is bound. However, either of the following circumstances can lead to SEGVs (if using the `-verbsbypass` option) or a hang.
  - If a user runs a job across nodes that have different numbers of InfiniBand adapters per node (for example, if some nodes have two, and other nodes only have one)
  - If a user runs a job across nodes that have one Infiniband adapter on one fabric, and another adapter on different fabric

  In either of these situations, users must disable adapter affinity by specifying the following option with **mpirun**.

  ```
  PAMI_IBV_ADAPTER_AFFINITY=0
  ```

  When running with PAMI, adapter striping is also enabled by default. However, both adapter affinity and adapter striping must be disabled before a user runs a job across a set of nodes in which one node contains a single link only. In this situation, to disable both adapter affinity and adapter striping, specify the following environment variables with **mpirun**.

  ```
  PAMI_ENABLE_STRIPING=0
  PAMI_IBV_DEVICE_NAME=mlx5_0:1
  PAMI_IBV_ADAPTER_AFFINITY=0
  ```
- At the time of this release, the MPI standard has some ambiguity about the meaning of MPI_Comm_get_info(), MPI_Win_get_info(), and MPI_File_get_info(). In this release, the *_get_info() calls return the current internal value of each info setting, which can differ from values provided in a previous MPI_Comm_set_info(), and so on. In the future, the MPI standard is likely to be changed to clarify that the get calls should return the same values that were set. So, the current behavior of MPI_*_get_info() is not compliant with the expected clarification of the MPI standard.
- If your switch network topology consists of more than one Infiniband network, IBM Spectrum MPI requires that each network be assigned a unique subnet prefix (also known as the network ID). Jobs might fail to run if disconnected networks are assigned the same network ID.

- IBM Spectrum MPI's use of libevent v2.0.22 might cause a perturbation in the random number stream generated by the libc rand() and random() functions. Applications that rely on behavior should consider using the rand_r() and random_r() APIs.
- Querying or attempting to write the value of a performance variable by using the MPI Tool information interface (MPI_T) is not supported.
- All Fortran MPI applications require the libgfortran3 runtime library to be installed on each compute node, regardless of which Fortran compiler was used to compile and link the application. This is to satisfy the libmpi_usempi.so dependency on libgfortran.so.3.
- If an application is built using the NVIDIA CUDA Toolkit, the NVIDIA CUDA Toolkit must be installed on the node from which it is launched, as well as each compute node. For Power Systems™ users, the CUDA Toolkit version 8.0 is required. For x86 users, the CUDA Toolkit version 7.5 is required.
- On a node with GPUs, it is recommended that you run the following commands shortly after the node boots and before running any GPU workload:

  ```
  nvidia-smi -pm 1  # persistence mode
  nvidia-modprobe -u -c=0  # pre-load uvm
  ```

- The Open MPI collectives components that are included with IBM Spectrum MPI do not support GPU buffers. For GPU buffer collective support, you must use libcollectives (the default).
- Support for GPU-accelerated applications is provided only if you are using the IBM Spectrum MPI PAMI backend and the IBM collective library (*libcollectives*). These are the default options for IBM Spectrum MPI, but if you choose to use a different messaging protocol or collective component, note that it will not support GPUs.
- The libcollectives component is the only collective component that is able to support CUDA buffers. As a result, when `-gpu` is specified with the **`mpirun`** command, libcollectives must be the preferred collective component. Therefore, you cannot specify **`mpirun`** `-gpu` with any of the following options:
  - `-mxm/-MXM`
  - `-mxmc/-MXMC`
  - `-tcp/-TCP`
  - `-ibv/-IBV`
  - `-ib/-IB`
  - `-openib/-OPENIB`
  - `-fca/-FCA`
  - `-hcoll/-HCOLL`
- The following MPI functions are not CUDA-aware:
  - MPI_Alltoallw
  - MPI_Ialltoallw
  - MPI_Ineighbor_allgather
  - MPI_Ineighbor_allgatherv
  - MPI_Ineighbor_alltoall
  - MPI_Ineighbor_alltoallv
  - MPI_Ineighbor_alltoallw
  - MPI_Neighbor_allgather
  - MPI_Neighbor_allgatherv
  - MPI_Neighbor_alltoall

- MPI_Neighbor_alltoallv
- MPI_Neighbor_alltoallw

## Migrating from IBM Parallel Environment Runtime Edition to IBM Spectrum MPI

The following table contains a list of basic end-user tasks, describes the method for completing those tasks with IBM PE Runtime Edition, and then shows you the equivalent method for carrying out the same tasks using IBM Spectrum MPI.

*Table 2. IBM PE Runtime Edition tasks and IBM Spectrum MPI equivalents*

| Task | IBM PE Runtime Edition method | IBM Spectrum MPI method |
|---|---|---|
| Executing programs | **poe** *program* [*args*] [*options*] | **mpirun** [*options*] *program* [*args*] |
| Compiling programs | The following compiler commands:<br>• **mpfort**, **mpic77**, **mpif90**<br>• **mpcc**, **mpicc**<br>• **mpCC**, **mpic++**, **mpicxx**<br>or the following environment variable settings:<br>• **MP_COMPILER**=xl \| gcc \| nvcc | The following compiler commands:<br>• **mpfort**<br>• **mpicc**<br>• **mpiCC**, **mpic++**, **mpicxx**<br>or the following environment variable settings:<br>• **OMPI_CC**=xl \| gcc<br>• **OMPI_FC**=xlf \| gfortran<br>• **OMPI_CXX**=xlC \| g++ |
| Determining rank before MPI_Init | The **MP_CHILD** environment variable | The **OMPI_COMM_WORLD_RANK** environment variable |
| Specifying the local rank | The **MPI_COMM_WORLD_LOCAL_RANK** environment variable | The **OMPI_COMM_WORLD_LOCAL_RANK** environment variable |
| Setting affinity | The environment variables:<br>• **MP_TASK_AFFINITY**=cpu<br>• **MP_TASK_AFFINITY**=core<br>• **MP_TASK_AFFINITY**=mcm<br>• **MP_TASK_AFFINITY**=cpu:n<br>• **MP_TASK_AFFINITY**=core:n<br>• **MP_TASK_AFFINITY**=1 | **mpirun** options:<br>• `-aff width:hwthread`<br>• `-aff width:core`<br>• `-aff width:numa`<br>• `--map-by ppr:$MP_TASKS_PER_NODE:node:pe=N --bind-to hwthread`<br>• `--map-by ppr:$MP_TASKS_PER_NODE:node:pe=N --bind-to core`<br>• `-aff none` |
| Setting CUDA-aware | The **MP_CUDA_AWARE** environment variable | The **mpirun** `-gpu` option |
| Setting FCA | The **MP_COLLECTIVE_OFFLOAD** environment variable | The **mpirun** `-FCA` and `-fca` options |
| Setting RDMA | • **MP_USE_BULK_XFER**<br>• The **MP_BULK_MIN_MSG_SIZE** environment variable | • RDMA default, when MSG_SIZE is greater than 64k |
| Controlling level of debug messages | The **MP_INFOLEVEL** environment variable | The **mpirun** `-d` option |

*Table 2. IBM PE Runtime Edition tasks and IBM Spectrum MPI equivalents (continued)*

| Task | IBM PE Runtime Edition method | IBM Spectrum MPI method |
|---|---|---|
| Setting STDIO | The environment variables:<br>• **MP_STDINMODE**<br>• **MP_STOUTMODE**<br>• **MP_LABELIO** | The **mpirun** `-stdio *` option |
| Specifying the number of tasks | The **MP_PROCS** environment variable | The **mpirun** `-np *` option |
| Specifying a host list file | The **MP_HOSTFILE** environment variable | The **mpirun** `-hostfile *` option |

For information about the processor affinity options and settings that you used for IBM PE Runtime Edition and how to achieve the same affinity settings with IBM Spectrum MPI, see "IBM PE Runtime Edition affinity equivalents" on page 50.

# Chapter 2. Understanding IBM Spectrum MPI

Because IBM Spectrum MPI is an implementation of Open MPI, its basic architecture and functionality is similar. IBM Spectrum MPI supports many, but not all of the features offered by Open MPI, and adds some unique features of its own.

## IBM Spectrum MPI code structure

IBM Spectrum MPI uses the same basic code structure as Open MPI, and is made up of the following sections:

- *OMPI* - The Open MPI API
- *ORTE* - The Open Run-Time Environment, which provides support for back-end runtime systems
- *OPAL* - The Open Portable Access Layer, which provides utility code that is used by OMPI and ORTE

These sections are compiled into three separate libraries, respectively; `libmpi_ibm.so`, `liborte`, and `libopal`. An order of dependency is imposed on these libraries; OMPI depends on ORTE and OPAL, and ORTE depends on OPAL. However, OMPI, ORTE, and OPAL are not software *layers*, as one might expect. So, despite this dependency order, each of these sections of code can reach the operating system or a network interface without going through the other sections.

IBM Spectrum MPI works in conjunction with the ORTE to launch jobs. The **mpirun** and **mpiexec** commands, which are used to run IBM Spectrum MPI jobs, are symbolic links to the **orterun** command.

For more information about the organization of the Open MPI code, refer to the Open MPI web site (www.open-mpi.org).

## MPI library support

To create a parallel program with IBM Spectrum MPI, use the API that is provided on the Open MPI web site (www.open-mpi.org). Information about the Open MPI subroutines and commands, including the various compiler script commands, is also available at this location.

It is important to note that if you used Open MPI to build your application, you must recompile and relink with IBM Spectrum MPI.

# Chapter 3. IBM Spectrum MPI supported features

IBM Spectrum MPI supports the a number of features, each of which is described in this section.

## 64-bit support

IBM Spectrum MPI can be used on 64-bit architectures and operating systems in Little Endian mode (for x86) and for IBM Power Systems servers (8335-GCA and 8335-GTA), with and without GPUs.

## Thread safety

Support for **MPI_THREAD_MULTIPLE** (multiple threads executing within the MPI library) is provided by IBM Spectrum MPI. However, note that multithreaded I/O is not supported.

## Portable Hardware Locality (hwloc)

IBM Spectrum MPI uses hwloc (Portable Hardware Locality), which is an API that navigates the hardware topology of your server. An abbreviated picture of the server's hardware can be seen by using the `--report-bindings` option. For example:

```
% mpirun -np 1 --report-bindings ./any_mpi_program.x
           [ibmgpu01:27613] MCW rank 0 bound to socket 0[core 0[hwt 0-1]], socket
           0[core 1[hwt 0-1]], socket 0[core 2[hwt 0-1]], socket 0[core 3[hwt 0-1]], socket
           0[core 4[hwt 0-1]], socket 0[core 5[hwt 0-1]], socket 0[core 6[hwt 0-1]], socket
           0[core 7[hwt 0-1]]: [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
```

In this example, the end of the output line:

```
[BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
```

indicates that the server has two sockets, each with eight cores, and that each core has two hyper-threads. This output also shows that the launched MPI process is bound to the first socket.

hwloc provides IBM Spectrum MPI with details about NUMA memory nodes, sockets, shared caches, cores and simultaneous multithreading, as well as system attributes and the locality of I/O devices. Using this information allows you to place processes, and the memory associated with them, most efficiently, and for best performance.

IBM Spectrum MPI includes hwloc version 1.11.2. For more information about hwloc, refer to the Open MPI web site (www.open-mpi.org).

## GPU support

For Power Systems servers, IBM Spectrum MPI supports running GPU-accelerated applications over CUDA-aware MPI. For x86 servers, IBM Spectrum MPI also supports running GPU-accelerated applications with NVIDIA GPUDirect RDMA. For Power Systems users, the CUDA Toolkit version 8.0 is required. For x86 users, the CUDA Toolkit version 7.5 is required.

By default, GPU support is turned off. To turn it on, use the **mpirun** `-gpu` flag:

```
mpirun -gpu
```

**Note:** For a list of limitations that apply to GPU support under IBM Spectrum MPI, see "Limitations" on page 2.

## FCA (hcoll) support

For installations that use the InfiniBand interconnect, the Mellanox Fabric Collective Accelerator (FCA), which uses Core-Direct technology, can be used to accelerate collective operations. FCA is also known as hcoll. FCA 3.0 is required.

FCA is installed into the `/opt/mellanox/fca` directory, by default. To verify that the FCA support was built correctly, use the **ompi_info** `--param` command, as follows.

```
ompi_info --param coll fca --level9|grep fca_enable
```

After FCA support is established, a list of FCA parameters is displayed.

**Note:** In order for users to be able to use FCA 3.0 (or later), the system administrator must set `/opt/mellanox/hcoll/lib` in `/etc/ld.so.conf` after MOFED is installed.

## MPI-IO

MPI has a number of subroutines that enable your application program to perform efficient parallel input-output operations. These subroutines (collectively referred to as MPI-IO) allow efficient file I/O on a data structure that is distributed across several tasks for computation, but is organized in a unified way in a single underlying file. MPI-IO presupposes a single parallel file system underlying all the tasks in the parallel job. For IBM Spectrum MPI, this parallel file system is IBM Spectrum Scale™ version 4.2.1.

For parallel I/O, IBM Spectrum MPI supports only ROMIO version 3.1.4. To understand how ROMIO was built, use the **ompi_info** command, with the highest level of verbosity. For example:

```
$ MPI_ROOT/bin/ompi_info -l 9 --param io romio314
                 MCA io: romio314 (MCA v2.1.0, API v2.0.0, Component
                         v10.1.0)
        MCA io romio314: ------------------------------------------------
        MCA io romio314: parameter "io_romio314_priority" (current value:
                         "40", data source: default, level: 9 dev/all, type:
                         int)
                         Priority of the io romio component
        MCA io romio314: parameter "io_romio314_delete_priority" (current
                         value: "40", data source: default, level: 9
                         dev/all, type: int)
                         Delete priority of the io romio component
```

```
MCA io romio314: informational "io_romio314_version" (current value:
                 "from MPICH v3.1.4", data source: default, level: 9
                 dev/all, type: string)
                 Version of ROMIO
MCA io romio314: informational "io_romio314_user_configure_params"
                 (current value: "", data source: default, level: 9
                 dev/all, type: string)
                 User-specified command line parameters passed to
                 ROMIO's configure script
MCA io romio314: informational
                 "io_romio314_complete_configure_params" (current
                 value: " FROM_OMPI=yes CC='gcc -std=gnu99'
                 CFLAGS='-DNDEBUG -m64 -O3 -Wall -Wundef
                 -Wno-long-long -Wsign-compare -Wmissing-prototypes
                 -Wstrict-prototypes -Wcomment -pedantic
                 -Werror-implicit-function-declaration
                 -finline-functions -fno-strict-aliasing -pthread

                --disable-aio --disable-weak-symbols
                 --enable-strict", data source: default, level: 9
                 dev/all, type: string)
                 Complete set of command line parameters passed to
                 ROMIO's configure scrip
```

## IBM Platform LSF

IBM Spectrum MPI supports the IBM Platform Load Sharing Facility (LSF) version 9.1.3 for launching jobs. For more information, see “Running applications with IBM Platform LSF” on page 28.

## Debugger support

IBM Spectrum MPI supports the Allinea DDT and TotalView debuggers. For more information, see Chapter 8, “Debugging and profiling applications,” on page 33.

## PMIx

IBM Spectrum MPI supports the extended version of the Process Management Interface (PMI), called PMI Exascale (PMIx) version 1.1.2. PMIx extends the PMI standard, including the existing PMI-1 and PMI-2 APIs, to support clusters of exascale size. For more information about PMIx, and to obtain the PMIx library and documentation, refer to the PMIx Programmer's Manual (http://pmix.github.io/master/)

# Chapter 4. Understanding IBM Spectrum MPI's collective library (libcollectives)

IBM Spectrum MPI provides a library of collectives called *libcollectives*. The libcollectives library provides seamless use of GPU memory buffers and includes a number of algorithms that offer excellent performance, scalability, and stability for collective operations. The libcollectives library also provides advanced logic to determine the fastest algorithm for any given collective operation.

## MCA parameters for collective communication

This section describes the MCA parameters that can be used for managing collective communications for IBM Spectrum MPI.

By default, *libcollectives* is the collective algorithm that will be used with IBM Spectrum MPI.

To see the complete list of MCA parameters that pertain to libcollectives, use the **ompi_info** command. For example:

```
ompi_info --param coll ibm -l 9 --internal
```

### MCA parameters for the libcollectives library

These parameters are for general use with libcoll.

**-mca coll_ibm_priority** *number*
Changes the priority of the libcollectives component. By default, the libcollectives component has the highest priority (a value of 95).

**Possible Values:** A number less than or equal to 100. If a negative value is specified, the component is deselected.

**Default:** 95

**-mca coll_ibm_verbose** *number*
Changes the verbosity of the collective component. This can be useful for debugging.

**Possible Values:**

**-1** Silence

**0** Error messages only (the default)

**10** Component level messages

**20** Warnings. For example, when libcollectives is skipped and the algorithm with the next highest priority should be used instead.

**40** Informational messages about algorithm availability and selection.

**60** Tracing messages related to the call stack. A message is displayed before every collection operation.

**80** Detailed debugging information.

**Default:** 0

**-mca coll_ibm_display_table** *value*
Displays a table of the algorithms that are available for each communicator (printed at the rank 0 of that communicator).

**Possible Values:**

The value is boolean, and can be any one of the following:

- 0 | f | false | disabled | no
- 1 | t | true | enabled | yes

**Default:** 0 | f | false | disabled | no

**-mca coll_ibm_tune_results** *path*
Specifies the path to the XML libcollectives tuning file that should be used. The file must be called `libcoll_tune_results.xml`.

**Possible Values:** Any path name.

**Default:** The path name of the version that was included with IBM Spectrum MPI (`etc/libcoll_tune_results.xlm`).

## Environment variables for collective communication

IBM Spectrum MPI provides a number of environment variables for managing collective communication.

By default, *libcollectives* is the collective algorithm that will be used with IBM Spectrum MPI.

When an application makes many calls to a non-synchronizing collective (for example, MPI_Bcast), it is possible to exhaust the unexpected message queue on one or more participants, causing the application to crash. To avoid this situation, Open MPI provides a *barrier* when it encounters a sequence of non-synchronizing collectives. By default,

- A barrier is issued every 128 calls for short collectives
- A barrier is issued every 16 calls for long collectives
- The boundary between short and long collectives is 262144 bytes.

IBM Spectrum MPI provides the following environment variables for customizing the barrier values.

**LIBCOLL_NUM_ASYNC_SHORT**
For short operations, specifies the number of asynchronous calls per barrier.

**LIBCOLL_NUM_ASYNC_LONG**
For long operations, specifies the number of asynchronous calls per barrier.

**LIBCOLL_NUM_ASYNC_CUTOFF**
Specifies the boundary between short and long collectives (in bytes).

# Chapter 5. Interconnect selection

This article describes a number of options that you can use for selecting interconnects.

In addition to this article, which provides options for specifying a communication method, these additional options might also be helpful:



IBM Spectrum MPI includes shortcuts for specifying the communication method that is to be used between the ranks. At the Open MPI level, point-to-point communication is handled by a PML (point-to-point message layer), which can perform communications directly, or use an MTL (matching transport layer) or BTL (byte transfer layer) to accomplish its work.

The types of PMLs that can be specified include:

**pami** IBM Spectrum MPI PAMI (Parallel Active Messaging Interface)

**yalla** Mellanox MXM (Mellanox Messaging Accelerator)

**cm** Uses an MTL layer

**ob1** Uses a BTL layer

The types of MTLs that can be specified include:

**psm** Intel PSM (Performance Scaled Messaging)

**mxm** An altnerate Mellanox MXM. However, yalla is preferred.

The types of BTLs that can be specified include:

**tcp** TCP/IP

**openib**
OpenFabrics InfiniBand

**usnic** Cisco usNIC (x86_64 only)

IBM Spectrum MPI provides the following shortcuts (**mpirun** options) that allow you to specify which PML, MTL, or BTL layer should be used. Specifying an option in uppercase letters (for example -MXM) forces the related PML, MTL, or BTL layer. Note that the lowercase options are equivalent to the uppercase options.

**-PAMI | -pami**
Specifies that IBM Spectrum MPI's PAMI should be used by way of the PML pami layer.

**-PAMI_NOIB | -pami_noib**
When running with the PAMI PML on a single node, specifies a libpami.so that does not attempt to open any IBV devices, and only communicates by way of PAMI's shared memory and cross memory attach mechanisms.

**Note:** The `-PAMI_NOIB` option is available for use with Power Systems servers only.

**-MXM | -mxm**
Specifies that Mellanox MXM should be used by way of the PML yalla layer. This is the preferred method.

**-MXMC | -mxmc**
Specifies that Mellanox MXM should be used by way of the PML cm and MTL mxm layers.

**-PSM | -psm**
Specifies that Intel PSM (formerly from QLogic) should be used by way of the PML cm and MTL psm layers.

**-TCP | -tcp**
Specifies that TCP/IP should be used by way of the PML ob1 and BTL tcp layers.

**-UNIC | -unic | -USNIC | -usnic**
Specifies that Cisco usNIC should be used by way of the PML ob1 and BTL usnic layers.

**-IB | -ib | -IBV | -ibv | -OPENIB | -openib**
Specifies that OpenFabrics InfiniBand should be used by way of the PNL ob1 and BTL openib layers.

## Using the PAMI verbs bypass

By default, PAMI uses a portable interface to the underlying `libibverbs.so` library. However, you can use a faster interface called the PAMI verbs bypass if you know the version of `libibverbs` that was installed on your system. To take advantage of the PAMI verbs bypass interface, you need to use the **mpirun** `-verbsbypass` option and specify the version of `libibverbs` that is currently installed.

For example, you could query the `libibverbs` RPM package, as follows:

```
% rpm -q libibverbs
```

If the RPM query returned `libibverbs-1.1.8mlnx1-OFED.3.3.1.5.0.32200.x86_64`, then you would determine that you are using `libibverbs` version 3.3.

Next, you would use the `-verbsbypass` option, as follows:

```
% mpirun -verbsbypass 3.3
```

The PAMI `-verbsbypass` option accepts the following values:

**3.3** Specifies `libibverbs` version 3.3

**3.4** Specifies `libibverbs` version 3.4

**auto** Determines the compatible MOFED version that is installed on the **mpirun** node. This option assumes that the same MOFED level has been installed on all the nodes of the cluster.

**off** Sets the PAMI verbs bypass to off and re-enables the default PAMI portable interface.

## Specifying use of the FCA (hcoll) library

The IBM Spectrum MPI libcollectives collectives library is used by default. However, you can enable the Mellanox hcoll library (also known as FCA 3.x) using one of the following **mpirun** command line options:

**-HCOLL | -FCA**
Specifies that the hcoll collective library should be used universally.

**-hcoll | -fca**
Specifies that the IBM Spectrum MPI libcollectives collectives library retains the highest priority, but that it is able to *fall back* to any of the hcoll collectives.

For more information about libcollectives and controlling the priority of collective algorithms, see Chapter 4, "Understanding IBM Spectrum MPI's collective library (libcollectives)," on page 13.

## Managing on-host communication

If a BTL is used for point-to-point traffic, the most commonly-used on-host communication method is the shared memory BTL called *vader*. However, there is an alternate BTL called *sm*, and it is always possible to use an off-host BTL for on-host traffic, as well. The vader BTL is likely to provide the best on-host performance, but it is possible for InfiniBand, for example, to provide higher on-host bandwidth than shared memory.

You can use the following options to specify how on-host communication should be performed. Note that these options only apply if a BTL is being used. They are not available for MXM, PSM, or PAMI.

**-intra vader | -intra shm**
Specifies that BTL=vader (shared memory) should be used for on-host traffic (only applies if the PML is already ob1).

**-intra nic**
Specifies that the off-host BTL for on-host traffic should be used.

**-intra sm**
Specifies that BTL=sm (older shared memory component) on-host should be used (only applies if the PML is already ob1).

## Specifying an IP network

If you are using TCP/IP, you can use the **mpirun** `-netaddr` option to specify the network over which traffic should be sent.

**-netaddr** *spec*,*spec*,**..**
Specifies the network to use for TCP/IP traffic. This option applies to control messages as well as the regular MPI rank traffic.

**-netaddr** *type*:*spec*,*spec*,**..**
Specifies the networks for particular types of traffic.

The *type* variable can be one of the following:

**rank** Specifies the network for regular MPI rank-to-rank traffic.

**control | mpirun**
Specifies the network for control messages (for example, launching).

The *spec* variables can be one of the following:

- An interface name. For example, eth0.
- CIDR notation. For example, 10.10.1.0/24.

## Displaying communication methods between hosts

With IBM Spectrum MPI, you can print a two-dimensional table that shows the method that is used by each host to communicate with each of the other hosts. The following options allow you to do this:

**-prot** Displays the interconnect type that is used by each host. The first rank on each host connects to all peer hosts in order to establish connections that might otherwise be on-demand.

**-protlazy**
Similar to `-prot`. Displays the interconnect type that is used by each host at MPI_Finalize. Connections to peer hosts are not established, so it is possible that many peers are unconnected.

The output from either the `-prot` or `-protlazy` options looks similar to this:

```
Host 0 [mpi01] ranks 0 - 3
Host 1 [mpi02] ranks 4 - 7
Host 2 [mpi03] ranks 8 - 11
Host 3 [mpi04] ranks 12 - 15


 host | 0    1    2    3
======|====================
    0 : shm  tcp  tcp  tcp
    1 : tcp  shm  tcp  tcp
    2 : tcp  tcp  shm  tcp
    3 : tcp  tcp  tcp  shm


Connection summary:
  on-host:  all connections are shm
  off-host: all connections are tcp
```

By default, the table only displays information for a maximum of 16 hosts (although the connection summary, which appears after the table, is not limited by size). If you have a larger cluster, you can use the **MPI_PROT_MAX** environment variable to increase the number of hosts that are displayed in the table. Note, however, that the larger this table becomes, the more difficult it is to use.

# Chapter 6. Compiling applications

IBM Spectrum MPI supports multiple compilers, for both x86 and Power Systems users.

For x86 users, IBM Spectrum MPI supports the following compilers:

- GNU compilers for C and FORTRAN, version 4.4.7 or 4.8.x (the default)
- Intel Compiler Suite, version 12.5 or later
- PGI (the Portland Group) compiler version, 16.9 for C++ and Fortran

For Power Systems users, IBM Spectrum MPI supports the following compilers:

- IBM XL C, version 13.1.4, and IBM XL Fortran, version 15.1.4 (the default)
- GNU GCC compilers for C and FORTRAN, version 4.8.x
- PGI (the Portland Group) compiler, version 16.9 for C++ and Fortran

The compiler that is used to build your programs is selected automatically by IBM Spectrum MPI. For x86 users, GNU compilers have first priority. Intel compilers have second priority, followed by other compilers. For Power® users, XL compilers have first priority. GNU compilers have second priority, followed other compilers.

When using PGI, specify the PGI compiler that you wish to use with one of the following environment variables.

- If you are using the **mpicc** C compiler wrapper, use the **OMPI_CC** environment variable to specify the C compiler.
- If you are using the **mpicxx** or **mpiCC** C++ compiler wrapper, use the **OMPI_CXX** environment variable to specify the C++ compiler.
- If you are using the **mpif77**, **mpif90**, or **mpifort** Fortran compiler wrapper, use the **OMPI_FC** environment variable to specify the Fortran compiler.

It is important to note that if you used Open MPI to build your application, you must recompile and relink with IBM Spectrum MPI.

## Using the wrapper compiler scripts

IBM Spectrum MPI includes a set of wrapper compiler scripts that read the configuration script and then build the command line options that are supplied to the underlying compiler. The wrapper scripts do not actually compile your applications; they simply invoke the compiler that is specified in the *configure* script. The wrapper scripts provide an easy and reliable way to specify options when you compile. As a result, it is strongly recommended that you use one of the wrapper compiler scripts instead of trying to compile your applications manually.

**Note:** Although you are strongly encouraged to use the wrapper compiler scripts, there might be a few circumstances in which doing so is not feasible. In this case, consult the Open MPI web site (www.open-mpi.org) FAQ for information about how to compile your application without using the wrappers.

The wrapper compiler scripts that are provided by IBM Spectrum MPI include:

*Table 3. IBM Spectrum MPI wrapper compiler scripts*

| Language | Wrapper compiler name |
|---|---|
| C | **mpicc** |
| Fortran | **mpifort** (v1.7 or later), **mpif77** and **mpif90** (for earlier versions) |

In the following example, the **mpicc** wrapper script is used to compile a C program called `hello_world_smpi.c`.

```
shell$ mpicc hello_world_smpi.c -o hello_world_smpi -g
```

To understand how the underlying compilers are invoked, you can use the various `--showme` options, which are available with all of the IBM Spectrum MPI wrapper scripts. The `--showme` options are:

**--showme**
Displays all of the command line options that will be used to compile the program.

**--showme:command**
Displays the underlying compiler command.

**--showme:compile**
Displays the compiler flags that will be passed to the underlying compiler.

**--showme:help**
Displays a usage message.

**--showme:incdirs**
Displays a list of directories that the wrapper script will pass to the underlying compiler. These directories indicate the location of relevant header files. It is a space-delimited list.

**--showme:libdirs**
Displays a list of directories that the wrapper script will pass to the underlying linker. These directories indicate the location of relevant libraries. It is a space-delimited list.

**--showme:libs**
Displays a list of library names that the wrapper script will use to link an application. For example:

```
mpi open-rte open-pal util
```

It is a space-delimited list.

**--showme:link**
Displays the linker flags that will be passed to the underlying compiler.

**--showme:version**
Displays the current Open MPI version number.

Refer to the Open MPI web site (www.open-mpi.org) for additional information about compiling applications, such as:

- Compiling programs without using the wrapper compiler scripts
- Overriding the wrapper compiler flags
- Determining the default values of the wrapper compiler flags
- Adding flags to the wrapper compiler scripts.

# Chapter 7. Running applications

With IBM Spectrum MPI, you can run applications using several different tools.

IBM Spectrum MPI provides support for running your applications using:

- The **mpirun** (and **mpiexec**) commands. see "Running programs with mpirun."
- The ssh or rsh command line. See "Running jobs with ssh or rsh" on page 28.
- IBM Platform LSF (LSF). See "Running applications with IBM Platform LSF" on page 28.

For troubleshooting information related to running jobs, refer to the Open MPI web site (www.open-mpi.org).

## Establishing a path to the IBM Spectrum MPI executables and libraries

IBM Spectrum MPI needs to be able to locate its executables and libraries on every node on which applications will run. It can be installed locally, on each node that will be a part of the MPI job, or in a location that is accessible to the network. IBM Spectrum MPI installations are relocatable.

Multiple versions of IBM Spectrum MPI can be installed on a cluster, or made available over a network shared file system.

The full path to the installed IBM Spectrum MPI must be the same on all the nodes that are participating in an MPI job.

To establish a path to your executables and libraries, do the following:

1. Set the **MPI_ROOT** environment variable to the installed root of the version of IBM Spectrum MPI that you want to use.
2. Add `$MPI_ROOT/share/man` to the **MANPATH** environment variable.

No other environmental setup is needed to run jobs with IBM Spectrum MPI.

**Note:** It is not recommended that users add any of the directories under **MPI_ROOT** to the **PATH** or **LD_LIBRARY_PATH** statements. Doing so can interfere with the normal functioning of some IBM Spectrum MPI features.

## Running programs with mpirun

The **mpirun** (as well as **mpiexec** and **orterun**) command can be used with IBM Spectrum MPI to run SPMD or MPMD jobs.

The **mpirun** and **mpiexec** commands are identical in their functionality, and are both symbolic links to **orterun**, which is the job launching command of IBM Spectrum MPI's underlying Open Runtime Environment. Therefore, although this material refers only to the **mpirun** command, all references to it are considered synonymous with the **mpiexec** and **orterun** commands.

## Specifying the hosts on which your application runs

In order to execute your program, IBM Spectrum MPI needs to know the hosts in your network on which it will run.

In general, when using the **mpirun** command, there are two ways that you can do this. You can either:

- Enter the names of the hosts individually on the command line.
- Create a text file containing the names of the hosts, and then specify the list on the command line at runtime. This is called a *host list file*. A host list file is useful when the number of hosts is large, and entering them individually on the command line would be too cumbersome and error-prone.

### Specifying hosts individually

To specify individual hosts on the **mpirun** command line, use the `--host` option. In the following example, the `--host` option is used with **mpirun** to start one instance of prog01 on the h1 node and another instance of prog01 on the h2 node.

```
mpirun -host h1,h2 prog1
```

Note that if you wanted to start two instances of prog01 on the h1 node, and one instance of prog01on the h2 node, you could do the following:

```
mpirun -host h1,h1,h2 prog01
```

See "Running programs with mpirun" on page 21 for additional information and examples of running jobs with **mpirun**.

### Specifying hosts using a host list file

The host list file is a flat text file that contains the names of the hosts on which your applications will run. Each host is included on a separate line. For example, here are the contents of a very simple host list file called *myhosts*:

```
node1.mydomain.com
node2.mydomain.com
node3.mydomain.com
node4.mydomain.com
```

After you have created the host list file, you can specify it on the command line using the `--hostfile` (also known as `--machinefile`) option of the **mpirun** command. For example, using the simple *myhosts* host list file, you could run your application, *prog01*, as follows:

```
mpirun -np 4 --hostfile myhosts prog01
```

For more information about running jobs with the **mpirun** command, see "Running programs with mpirun" on page 21.

For each host, the host list file can also specify:

- The number of slots (the number of available processors on that host). The number of slots can be determined by the number of cores on the node or the number of processor sockets. If no slots are specified for a host, then the number of slots defaults to one. In this example, a host list file called *myhosts* specifies three nodes, and each node has two slots:

  ```
  cat myhosts
  node1 slots=2
  node2 slots=2
  node3 slots=2
  ```

Specifying the following command launches six instances of prog01; two on node1, two on node2, and two on node3:

```
mpirun -hostfile myhosts prog01
```

- The maximum number of slots. Note that the maximum slot count on a host defaults to infinite, thereby allowing IBM Spectrum MPI to oversubscribe to it. To avoid oversubscribing, you can provide a maximum slot value for the host (max-slots=*).

The host list file can also contain comments, which are prefixed by a pound sign (#). Blank lines are ignored.

For example:

```
# This is a single processor node:
node1.mydomain.com

# This is a dual-processor node:
node2.mydomain.com slots=2

# This is a quad-processor node. Oversubscribing
# to it is prevented by setting max-slots=4:
node3.mydomain.com slots=4 max-slots=4
```

For more information about host list files and oversubscribing hosts, see the Open MPI web site (www.open-mpi.org).

## Starting a SPMD (Single Program, Multiple Data) application

In general, for SPMD jobs, the **mpirun** command can be used in the following format:

```
mpirun -np num --hostfile filename program
```

In this command syntax:

- `-np` *num* specifies the number of processes
- `--hostfile` *filename* specifies the name of the host list file
- *program* specifies the name of your application.

In other words, **mpirun** starts *num* instances of *program* on the hosts designated by a host list file called *filename*.

Consider the following example. You have a program called *prog1* and a host list file called *hosts* that contains the following lines:

```
host1.mydomain.com
host2.mydomain.com
host3.mydomain.com
```

You could run *prog1* using the following **mpirun** command syntax:

```
mpirun -np 3 --hostfile hosts prog1
```

## Starting an MPMD (multiple program, multiple data) application

For MPMD applications, the basic syntax of the **mpirun** command is as follows:

```
mpirun -np num1 prog1 : -np num2 prog2
```

In this command syntax:

- `-np` *num1* specifies the number of processes for *prog1*
- `-np` *num2* specifies the number of processes for *prog2*
- *prog1* specifies the name of an application
- *prog2* specifies the name of a second application.

In other words, **mpirun** starts *num1* copies (instances) of *prog1* and also starts *num2* instances of *prog2*.

Consider the following example. You have two programs; one called *prog3* and another called *prog4*. You want to run two instances of *prog3*, and also four instances of *prog4*. In this scenario, you could use the **mpirun** command, as follows:

```
mpirun -np 2 prog3 : -np 4 prog4
```

# mpirun options

**mpirun** supports a large number of command line options. The best way to see a complete list of these options is to issue **mpirun** `--help`. The `--help` option provides usage information and a summary of all of the currently-supported options for **mpirun**.

## mpirun options for general use

Some of the more commonly-used options for starting applications with **mpirun** include:

**-np | -n** *number_of_processes*
Specifies the number of instances of a program to start.

If `-np` *number_of_processes*:

- **Is not specified**, **mpirun** launches the application on the number of slots that it can discover.
- **Is specified**, **mpirun** launches the given number of processes, as long as it will not oversubscribe a node.

**-nooversubscribe | --nooversubscribe**
Indicates that the nodes must not be oversubscribed, even if the system supports such an operation. This is the default.

**-oversubscribe | –-oversubscribe**
Indicates that more processes should be assigned to any node in an allocation than that node has slots for. Nodes can be oversubscribed, even on a managed system. For more information about mapping, binding, and ordering behavior for **mpirun** jobs, see Chapter 9, “Managing processor affinity,” on page 41.

**-display-allocation | --display-allocation**
Displays the *Allocated Nodes* table. This option is useful for verifying that **mpirun** has read in the correct node and slot combinations.

For example:

```
shell$  mpirun -np 2 -host c712f5n07:4,c712f5n08:8 --display-allocation hostname

======================   ALLOCATED NODES   ======================
 c712f5n07: slots=4 max_slots=0 slots_inuse=0 state=UP
```

```
 c712f5n08: slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
c712f5n07
c712f5n07
```

**-do-not-launch | --do-not-launch**
Performs all necessary operations to prepare to launch the application, but without actually launching it. This option is useful for checking the allocation (with `--display-allocation`) without actually launching the daemons and processes.

For example:

```
shell$  mpirun -np 2 -host c712f5n07:4,c712f5n08:8 --display-allocation --do-not-launch hostname

======================   ALLOCATED NODES   ======================
 c712f5n07: slots=4 max_slots=0 slots_inuse=0 state=UP
 c712f5n08: slots=8 max_slots=0 slots_inuse=0 state=UP
=================================================================
```

**-hostfile | --hostfile** *hostfile*, **-machinefile | --machinefile** *machinefile*
Specifies a hostfile for launching the application.

**-H | -host | --host** *hosts*
Specifies a list of hosts on which to invoke processes.

**-rf | --rankfile** *file_names*
Specifies a rankfile file.

**--timeout** *seconds*
Indicates that the job should be terminated after the specified number of seconds.

**--report-state-on-timeout**
Reports all job and process status when a timeout occurs.

**--get-stack-traces**
Gets the stack traces of all application processes when a timeout occurs. The Linux *gstack* tool must be installed on all machines in order to generate the stack trace. If the gstack tool is not available, an error message is displayed instead of the stack trace.

## IBM Spectrum MPI mpirun options

IBM Spectrum MPI includes a number of its own **mpirun** command line options, as follows.

**mpirun options for on-host communication method:**

The IBM Spectrum MPI PAMI component supports on-host shared memory. When running with `-PAMI` (the default), no additional parameters are required for on-host communication.

**-intra=nic**
Specifies that the off-host BTL should also be used for on-host traffic.

**-intra=vader**
Specifies that BTL=vader (shared memory) should be used for on-host traffic. This only applies if the PML (point-to-point messaging layer) is already ob1.

**-intra=shm**
Specifies that BTL=vader (shared memory) should be used for on-host traffic. This only applies if the PML (point-to-point messaging layer) is already ob1.

**-intra=sm**
Specifies that BTL=sm (an older shared memory component) should be used for on-host traffic. This only applies if the PML is already ob1.

**Note:** The `-intra` flag is incompatible with GPU buffers because it does not allow you to specify PAMI.

**mpirun options for display interconnect:**

**-prot** Displays the interconnect type that is used by each host. The first rank on each host connects to all peer hosts in order to establish connections that might otherwise be on-demand.

**-protlazy**
Similar to `-prot`. Displays the interconnect type that is used by each host at MPI_Finalize. Connections to peer hosts are not established, so it is possible that many peers are unconnected.

**-gpu** Enables GPU awareness in PAMI by one MCA option and an `-x` LD_PRELOAD of `libpami_cudahook.so`.

**Note:** Using the `-gpu` option causes additional runtime checking of every buffer that is passed to MPI. `-gpu` is only required for applications that pass pointers to GPU buffers to MPI API calls. Applications that use GPUs, but do not pass pointers that refer to memory that is managed by the GPU, are not required to pass the `-gpu` option.

**mpirun options for standard I/O:**

**-stdio=p**
Specifies that each rank's output should be prefixed with [job,rank].

**-stdio=t**
Specifies that a timestamp should be included with the output.

**-stdio=i[=|all|-|none|***rank***]**
Specifies that stdin should be sent to all ranks (+), no ranks (-), or a single, specific rank (*rank*).

**-stdio=file:***prefix*
Specifies that output should be sent to files that are named *prefix*.*rank*.
Note that *prefix* can be either a file name or a path ending in a file name.

**-stdio=***option*,*option*,**...**
Specifies a comma-separated list of the standard I/O options.

**mpirun options for IP network selection:**

**-netaddr=***spec*,*spec*,**...**
Specifies the networks that should be used for TCP/IP traffic. This option applies to control messages as well as regular MPI rank traffic.

**-netaddr=***type*:*spec*,*spec*,**...**
Specifies the networks that should be used for different types of traffic.

In this syntax, *type* can be one of the following:

**rank** Specifies the network for regular MPI rank-to-rank traffic.

**control | mpirun**
Specifies the network for control messages (for example, launching mpirun).

In this syntax, *spec* can be one of the following:

**interface name**
The interface name. For example, eth0.

**CIDR notation**
The CIDR (Classless Inter-Domain Routing) notation. For example, 10.10.1.0/24.

**mpirun options for affinity:**

**-aff** Enables affinity, with the default option of bandwidth.

**-aff=[***option***,***option***,...]**
Enables affinity, with any of the following options.

**v / vv** Displays output in verbose mode.

**cycle:***unit*
Interleaves the binding over the specified element. The values that can be specified for *unit* are hwthread, core, socket (the default), or numa.

**bandwidth | default**
Interleaves sockets but reorders them.

**latency**
Pack.

**width:***unit*
Binds each rank to an element of the size that is specified by *unit*. The values that can be specified for *unit* are hwthread, core, socket (the default), or numa.

**mpirun options for PMPI layering:**

**-entry lib,...**
Specifies a list of PMPI wrapping libraries. Each library can be specified in one of the following forms:

- libfoo.so
- /path/to/libfoo.so
- foo (which is automatically expanded to libfoo.so for simple strings that contain only characters of a - z, A - Z, or 0 - 9. Expansion is not applicable for the strings fort, fortran, v, and vv.

**-entry fort | fortran**
Specifies the layer into which the base MPI product's Fortran calls (which minimally wrap the C calls) should be installed.

**-entrybase | -baseentry** *lib*
Optionally specifies the libraries from which to get the bottom level MPI calls. The default value is RTLD_NEXT, which is the libmpi to which the executable is linked.

**-entry v | -entry vv**
Displays the layering of the MPI entry points in verbose mode.

Specifying a value of v prints verbose output that shows the layering levels of the MPI entry points.

Specifying a value of vv prints more detailed verbose output than the `-entry v` option. The `-entry vv` option shows the levels that are intended to be used, and confirms the libraries that are being opened. The output from `-entry vv` is less readable, but it allows you to confirm, more visibly, that interception is taking place.

## Running applications with IBM Platform LSF

IBM Spectrum MPI supports IBM Platform LSF version 9.1.3 for launching jobs. When a job is launched, the **mpirun** command searches for the **LSF_ENVDIR** and **LSB_JOBID** environment variables. If they are found, and **mpirun** can successfully use the LSB library, then it determines that it is in an LSF environment.

If **LSB_AFFINITY_HOSTFILE** is set, then the file that is specified by this environment variable determines the mapping, binding, and ordering for the processes that will be launched later. LSF generates **LSB_AFFINITY_HOSTFILE** during the setup of the allocation.

After the list of hosts is known, the PLM framework of **mpirun** launches an Open RTE daemon (orted) on each node in a linear manner.

Previously, a limitation existed regarding the use of both short and long host names with LSF. Short names (for example, nodeA) could not be mixed with long names (for example, nodeA.mycluster.org) by LSF because Open MPI interpreted them as two different nodes, and then failed to launch. However, beginning with IBM Spectrum MPI 10.1.0.2, the MCA parameter setting of **orte_keep_fqdn_hostnames**=false now causes all long host names to be converted to short host names, by default, and MPI interprets them correctly. No other intervention is required by users.

## Running jobs with ssh or rsh

IBM Spectrum MPI supports running jobs under the secure shell (ssh) or the remote shell (rsh).

**mpirun** first looks for allocation information from a resource manager. If none is found, it uses the values provided for the `-hostfile`, `-machinefile`, `-host`, and `-rankfile` options, and then uses ssh or rsh to launch the Open RTE daemons on the remote nodes.

By default, jobs are launched using ssh, however, you can force the use of rsh by using the `-mca` **plm_rsh_force_rsh** parameter. The following list describes `-mca` **plm_rsh_force_rsh**, as well as other MCA parameters that are useful when running jobs under ssh or rsh.

**-mca plm_rsh_agent**
Specifies the agent that will launch executables on remote nodes. The value is a colon-delimited list of agents, in order of precedence.

**Default:** ssh : rsh

**-mca plm_rsh_args**
Specifies arguments that should be added to ssh or rsh.

**Default:** Not set

**-mca plm_rsh_assume_same_shell**
Specifies whether or not to assume that the shell on the remote node is the same as the shell on the local node. Valid values are 0 | f | false | disabled | no or 1 |t | true | enabled | yes.

**Default:** true (assume that the shell on the remote node is the same as the shell on the local node)

**-mca plm_rsh_num_concurrent**
Specifies the number of **plm_rsh_agent** instances to invoke concurrently. You must specify a value that is greater than 0.

**Default:** 128

**-mca plm_rsh_pass_environ_mca_params**
Specifies whether or not to include MCA parameters from the environment on the Open RTE (orted) command line. Valid values are 0 | f | false | disabled | no or 1 | t | true | enabled | yes.

**Default:** true (MCA parameters from the environment will be included on the orted command line)

**-mca plm_rsh_force_rsh**
Specifies whether or not to force the launcher to always use rsh. Valid values are 0 | f | false | disabled | no or 1 | t | true | enabled | yes.

**Default:** false (the launcher will not use rsh)

**-mca plm_rsh_no_tree_spawn**
Specifies whether or not to launch applications using a tree-based topology. Valid values are 0 | f | false | disabled | no or 1 | t | true | enabled | yes.

**Default:** false (applications are launched using a tree-based topology)

**-mca plm_rsh_pass_libpath**
Specifies the library path to prepend to the remote shell's **LD_LIBRARY_PATH**.

**Default:** Not set

**Note:** If you are using ssh to connect to a remote host, in order for **mpirun** to operate properly, it is recommended that you set up a passphrase for passwordless login. For more information, see the Open MPI FAQ (www.open-mpi.org/faq/?category=rsh).

# Managing IBM Spectrum MPI jobs

There are a number of tasks that apply to running IBM Spectrum MPI jobs that you should understand.

## Using LD_PRELOAD with IBM Spectrum MPI

Internally, IBM Spectrum MPI uses LD_PRELOAD to enable GPU-related features. However, this creates a potential conflict when a user also adds a setting to LD_PRELOAD.

To enable users to add settings to **LD_PRELOAD**, IBM Spectrum MPI provides the following environment variables:

**OMPI_LD_PRELOAD_PREPEND**

Inserts the user's setting on the beginning of any existing **LD_PRELOAD** setting before launching the MPI ranks.

**OMPI_LD_PRELOAD_POSTPEND**
Inserts the user's setting on the end of any existing **LD_PRELOAD** setting before launching the MPI ranks.

## Using a pre-launch script

You can insert a program in front of all the ranks of a job by using the **OMPI_LAUNCH** environment variable.

For example:

```
% env OMPI_PRELAUNCH=valgrind mpirun -np 2 ./a.out
```

This is equivalent to the following:

```
% mpirun -np 2 valgrind ./a.out
```

The **OMPI_LAUNCH** environment variable is mainly useful for debugging applications that have complex launching scripts that are otherwise difficult to modify.

Consider another example. Here, a sample pre-launch script is used to attach gdb to a particular rank:

```
#!/bin/sh
if [ $OMPI_COMM_WORLD_RANK -eq 0 ] ; then
  xterm -e gdb --args "$@"
else
  exec "$@"
fi
```

## Using the InfiniBand Dynamically Connected Transport protocol

IBM Spectrum MPI supports the Mellanox InfiniBand Dynamically Connected Transport (DCT) protocol.

To turn DCT on, you can use one of the following environment variables.

**PAMI_IBV_ENABLE_DCT**
Use **PAMI_IBV_ENABLE_DCT** to explicitly turn DCT on and off. A value of 1 specifies that all jobs use DCT, regardless of the value of **PAMI_IBV_RCQP_PERCENT**. A value of 0 specifies that all jobs use RC, regardless of the value of **PAMI_IBV_RCQP_PERCENT**.

**PAMI_IBV_RCQP_PERCENT**
Specifies the threshold, as a percentage, of the number of qpairs that must be available for use in order to automatically enable DCT. By default, the value of **PAMI_IBV_RCQP_PERCENT** is 50. Allowable values are 1 through 100.

By default, the use of DCT is not enabled, and IBM Spectrum MPI uses RC mode. However, PAMI automatically switches to DCT mode when the following condition exists.

*number_of_QPs* >= 50/100 * *total_QPs_available*

In PAMI, the *total_QPs_available* is defined as 64k. The *number_of_QPs* is determined as follows..

(*number_of_ranks*^2)**number_of_nodes*

For example, a 512 rank job on eight nodes uses (64^2)*8=32k QPs. Because 32k is >= 50/100 * 64k, DCT mode is used. In a similar example, a 512 rank job on 16 nodes uses (32^2)*16=16k QPs. Because 16k is not >= 50/100 * 64k, RC mode is used.

Note that if **PAMI_IBV_RCQP_PERCENT** is set to anything other than 50, the new percentage value is used with this formula.

**Note:** Although the PAMI verbs bypass can be used in DCT mode, you will not see the normal performance benefit of the verbs bypass.

# Chapter 8. Debugging and profiling applications

IBM Spectrum MPI supports a number of tools for debugging and profiling parallel applications.

The following debugging and profiling tools can be used with IBM Spectrum MPI.

- TotalView debugger. See "Debugging applications with the TotalView debugger and IBM Spectrum MPI."
- Allinea DDT debugger. See "Debugging applications with the Allinea DDT debugger and IBM Spectrum MPI" on page 34.
- Dynamic MPI standard profiling interface. See "Dynamic MPI profiling interface with layering" on page 36.

## Debugging applications with the TotalView debugger and IBM Spectrum MPI

The RogueWave TotalView debugger can be used with IBM Spectrum MPI for viewing message queues and attaching to running parallel jobs.

In general, if TotalView is the first debugger in your path, you can use the following **mpirun** command to debug an IBM Spectrum MPI application:

```
mpirun --debug mpirun_arguments
```

When it encounters the **mpirun** command, IBM Spectrum MPI calls the correct underlying command to run your application with the TotalView debugger. In this case, the underlying command is:

```
totalview mpirun -a mpirun_arguments
```

If you want to run a two-process job of executable `a.out`, the underlying command would be:

```
totalview mpirun -a -np 2 a.out
```

The **mpirun** command also provides the `-tv` option, which starts a job under the TotalView debugger. The **mpirun** `-tv` option provides the same function as TotalView's `-a` option. The following example shows how the two-process job from the preceding example can be run.

```
mpirun -tv -np 2 a.out
```

By default, TotalView tries to debug the **mpirun** code itself, which, at the least, is probably not useful to you. IBM Spectrum MPI provides instructions for avoiding this problem in a sample TotalView startup file called `etc/openmpi-totalview.tcl`. This file can be used to cause TotalView to ignore the **mpirun** code and instead, debug only the application code. By default, `etc/openmpi-totalview.tcl` is installed to `$prefix/etc/openmpi-totalview.tcl` in the IBM Spectrum MPI installation.

To use the TotalView startup file, you can either copy it into the file that is called `$HOME/.tvdrc` or source it directly from `$HOME/.tvdrc`. For example, you can place the following line in `$HOME/.tvdrc` (replacing */path/to/spectrum_mpi/installation* with the proper directory name), which causes IBM Spectrum MPI to use the TotalView startup file:

```
source /path/to/spectrum_mpi/installation/etc/openmpi-totalview.tcl
```

For more information about using TotalView, see the Open MPI web site (www.open-mpi.org).

## Debugging applications with the Allinea DDT debugger and IBM Spectrum MPI

The Allinea DDT debugger provides built-in support for MPI applications.

In general, if Allinea DDT is the first debugger in your path, you can use the following **mpirun** command to debug an IBM Spectrum MPI application:

```
mpirun --debug mpirun_arguments
```

When it encounters the **mpirun --debug** command, IBM Spectrum MPI calls the correct underlying command to run your application with the Allinea debugger. In this case, the underlying command is:

```
ddt -n {number_of_processes} -start {excutable_name}
```

**Note:** The Allinea DDT debugger does not support passing arbitrary arguments to the **mpirun** command.

With Allinea DDT, you can also attach to processes that are already running, as shown in the following example:

```
ddt -attach {hostname1:pid} [{hostname2:pid} ...] {executable_name}
```

You can also attach to running processes by using the following syntax:

```
ddt -attach-file {filename of newline separated hostname:pid pairs} {executable_name}
```

## Debugging serial applications with IBM Spectrum MPI

It is possible to debug an IBM Spectrum MPI application with a serial debugger such as GDB. The following methods are often used by Open MPI developers:

- Attach to individual MPI processes after they are running
- Use the **mpirun** command to start multiple xterm windows, each running a serial debugger.

For information, see Open MPI web site (www.open-mpi.org).

## Redirecting debugging output from IBM Spectrum MPI

It is sometimes useful to redirect the output from IBM Spectrum MPI to the same stream as the application's stdout and stderr channels. This makes it easier to determine the location in the program from which the particular verbose output was emitted, and to identify, for example, which collective algorithm was called.

The following MCA parameters can be used to redirect the standard output streams that are used by the application and IBM Spectrum MPI.

**-mca iof_base_redirect_app_stderr_to_stdout 1**
For a user's application, specifies that stderr is redirected to stdout at the source.

**Possible Values:** 0 | false, 1 | true

**Default:** 0 | false

**-mca mca_base_verbose stdout**
Specifies where the default error (or verbose) output stream is directed.

**Possible Values:** A comma-delimited list of the following values:

- stderr
- stdout
- syslog
- syslogpri:notice | info | debug
- syslogid:*string* (*string* is the prefix string for all syslog notices.)
- file:*filename* (If a file name is not specified, a default file name is used.)
- fileappend (If fileappend is not specified, the file is opened for truncation.)
- level:*number* (*number* specifies the integer verbose level, 0-9. If *number* is not specified, 0 is implied.)

**Default:** stderr,level:0

**-mca orte_map_stddiag_to_stderr 1**
Specifies that internal IBM Spectrum MPI messages (such as verbose output) are routed over the stderr of the application process rather than an internal *stddiag* channel.

**Note:** This parameter cannot be used with **orte_map_stddiag_to_stdout**. If **orte_map_stddiag_to_stderr** and **orte_map_stddiag_to_stdout** are used together, an error message is issued when the job is started.

**Possible Values:** 0 | false, 1 | true

**Default:** 0 | false

**-mca orte_map_stddiag_to_stdout 1**
Specifies that internal IBM Spectrum MPI messages (such as verbose output) are routed over the stdout of the application process rather than an internal *stddiag* channel. This parameter changes the default value of the **mca_base_verbose** parameter to stdout. The user can still override this parameter.

**Note:** This parameter cannot be used with **orte_map_stddiag_to_stderr**. If **orte_map_stddiag_to_stdout** and **orte_map_stddiag_to_stderr** are used together, an error message is issued when the job is started.

**Possible Values:** 0 | false, 1 | true

**Default:** 0 | false

**Note:** Although the behavior of the **-mca orte_map_stddiag_to_stdout** 1 and **-mca mca_base_verbose stdout** parameters seems identical, there is a subtle difference. **orte_map_stddiag_to_stdout** occurs in the launching daemon as the application process is being forked, whereas **mca_base_verbose** is read in the application process relatively early in MPI_Init. As a result, there is a window of time in which **orte_map_stddiag_to_stdout** takes precedence over **mca_base_verbose**.

The default value of **mca_base_verbose** is stderr,level:0 unless the user specifies **orte_map_stddiag_to_stdout** 1. In this case, IBM Spectrum MPI automatically switches the default value of **mca_base_verbose** to stdout,level:0 (which is closer to what you might expect from the **orte_map_stddiag_to_stdout** option).

So, this option:

```
-mca orte_map_stddiag_to_stdout 1
```

means exactly the same thing as these options, specified together:

```
-mca orte_map_stddiag_to_stdout 1  -mca mca_base_verbose stdout,level:0
```

You can override the switched default value for **mca_base_verbose** by passing both sets of parameters, and changing the value of **mca_base_verbose**, follows:

```
-mca orte_map_stddiag_to_stdout 1  -mca mca_base_verbose stderr,level:0
```

This is useful if you want the internal Open MPI messages that are generated between MPI_Init and MPI_Finalize (inclusive of these functions) to be routed to a different output channel. In this case, you can adjust the **mca_base_verbose** variable separately from the **orte_map_stddiag_to_stdout** variable (which handles internal messages that are generated outside of the region of code between MPI_Init and MPI_Finalize).

When you specify the following:

```
-mca orte_map_stddiag_to_stdout 1
```

all internal Open MPI messages (for example, internal verbose level output) are routed to stdout at all times in the application lifecycle.

### Examples

In the following example, the **orte_map_stddiag_to_stdout** parameter is used to redirect all of the Open MPI error and verbose messages to stdout.

```
mpirun -mca orte_map_stddiag_to_stdout 1 ./myapp
```

In the following example, the **orte_map_stddiag_to_stderr** parameter is used to redirect all of the Open MPI error and verbose messages to stderr.

```
mpirun -mca orte_map_stddiag_to_stderr 1 ./myapp
```

In the following example, the **iof_base_redirect_app_stderr_to_stdout** parameter is used to redirect all of the application stdout and stderr to stdout only.

```
mpirun -mca iof_base_redirect_app_stderr_to_stdout 1 ./myapp
```

In the following examples, the **iof_base_redirect_app_stderr_to_stdout**, **orte_map_stddiag_to_stderr**, and **orte_map_stddiag_to_stdout** parameters can be used to redirect all of the Open MPI error and verbose messages and application stdout and stderr to stdout only. In the first example, stddiag is routed to stderr, which is then routed to stdout. In the second example, which is preferred, stddiag is directly routed to stdout.

```
mpirun -mca iof_base_redirect_app_stderr_to_stdout 1 -mca orte_map_stddiag_to_stderr 1 ./myapp
mpirun -mca iof_base_redirect_app_stderr_to_stdout 1 -mca orte_map_stddiag_to_stdout 1 ./myapp
```

## Dynamic MPI profiling interface with layering

The MPI standard defines a profiling interface (PMPI) that allows you to create profiling libraries by wrapping any of the standard MPI routines. A profiling wrapper library contains a subset of redefined MPI_* entry points, and inside those redefinitions, a combination of both MPI_* and PMPI_* symbols are called.

This means that you can write functions with the MPI_* prefix that call the equivalent PMPI_* function. Functions that are written in this manner behave like the standard MPI function, but can also exhibit any other behavior that you add.

For example:

```
int
MPI_Allgather(void *sbuf, int scount, MPI_Datatype sdt,
    void *rbuf, int rcount, MPI_Datatype rdt, MPI_Comm comm)
{
    int rval;
    double t1, t2, t3;
    t1 = MPI_Wtime();
    MPI_Barrier(comm);
    t2 = MPI_Wtime();
    rval = PMPI_Allgather(sbuf, scount, sdt, rbuf, rcount, rdt, comm);
    t3 = MPI_Wtime();
    // record time waiting vs time spent in allgather..
    return(rval);
}
double MPI_Wtime() {
  // insert hypothetical high-resolution replacement here, for example
}
```

Using two unrelated wrapper libraries is problematic because, in general, it is impossible to link them so that proper layering occurs.

For example, you could have two libraries:

**libJobLog.so**
: In this library, MPI_Init and MPI_Finalize are wrapped, so that a log of every MPI job is generated, which lists hosts, run times, and CPU times.

**libCollPerf.so**
: In this library, MPI_Init, MPI_Finalize and all the MPI collectives are wrapped, in order to gather statistics about how evenly the ranks enter the collectives.

With ordinary linking, each MPI_* call would resolve into one of the wrapper libraries, and from there, the wrapper library's call to PMPI_* would resolve into the bottom level library (`libmpi.so`). As a result, only one of the libraries would have its MPI_Init and MPI_Finalize routines called.

## Defining consistent layering

You can define a consistent approach to layering, with dynamically loaded symbols, for any number of wrapper libraries.

If you have a wrapper library named *libwrap.so*, which redefines an MPI_ symbol, it can either call another MPI_* entry, or it can call a PMPI_* entry. In the case of ordinary single-level wrapping, the calls into MPI_* would resolve into `libwrap.so` first, and then `libmpi.so` if not found. And the calls into PMPI_* would resolve into `libmpi.so`.

If multi-level layering were used, MPI_* would resolve to the current level and PMPI_* would resolve to the next level down in the hierarchy of libraries.

One way to achieve consistent layering is to establish a list of logical levels, where each level consists of MPI_* entry points from a given library. The bottom level would consist of MPI_* entry points from the base MPI library (`libmpi.so`). For example:

```
Level 0: libJobLog.so
Level 1: libCollPerf.so
Level 2: libmpi.so
```

When an application makes an MPI call, a depth counter would start at level 0 and search down the list until it finds a level that defines that MPI call. From there, if that routine calls another MPI or PMPI function, the depth counter would remain the same or be incremented respectively, to control the level from which the next function is called.

## Using the mpirun -entry option to define consistent layering

You can establish this layering scheme by using the **mpirun** command line option `-entry`. With `-entry`, you can specify a library in the form `libfoo.so`, `/path/to/libfoo.so`, or simply `foo` (which will be automatically expanded into `libfoo.so` for simple strings). For example, the following specification:

```
% mpirun -entry JobLog,CollPerf -np 2 ./example.x
```

is automatically expanded to:

```
% mpirun -entry libJobLog.so,libCollPerf.so -np 2 ./example.x
```

Note that the order in which you specify a list of libraries dictates each library's placement in the hierarchy of levels. By default, the base product's MPI library, `libmpi.so`, is placed at the bottom of the list, so it does not need to be specified with `-entry`. However, the `-entrybase` (or `-baseentry`) option enables you to specify a different library from which to get the bottom level MPI calls.

**Note:**

- A profiling wrapper library cannot be specified with the **mpirun** `-entry` unless it is implemented as a shared library.
- In order for the libraries to be found, you must either set **LD_LIBRARY_PATH** or specify full paths to the libraries.

The syntax of the **mpirun** `-entry` option is:

**mpirun -entry** *library*
: Specifies a list of PMPI wrapper libraries.

**mpirun -entry fort**
: Specifies the level at which to install the base MPI product's Fortran calls, which, at a minimum, wrap the C calls. The Fortran calls are placed at the top level, by default.

**mpirun -entrybase** *library*
: Specifies an alternate library from which to get the bottom level calls.

**mpirun -baseentry** *library*
: Synonym for **mpirun** `-baseentry` *library*.

**mpirun -entry v**
: Prints verbose output that shows the layering levels of the MPI entry points. For example:

```
> Entrypoint MPI wrapper levels:
>  1. (fortran from base  product)
>  2. libJobLog.so
>  3. libCollPerf.so
>  4. base  product
> Entrypoint MPI base product:
>  (base MPI product as linked)
```

**mpirun -entry vv**
: Prints more detailed verbose output than the `-entry v` option. The `-entry vv` option shows the levels that are intended to be used, and confirms the

libraries that are being opened. The output from `-entry vv` is less readable, but it allows you to confirm, more visibly, that interception is taking place.

By default, the top layer is always the Fortran calls from the base MPI product. The Fortran calls are wrappers over corresponding C routines. As a result, if a profiling library intercepts the C call MPI_Send, and an application makes the Fortran call mpi_send, the profiling library's MPI_Send gets called, essentially wrapping Fortran for free. If this is not the behavior you want, you can include the **fort** string with the `-entry` option to specify where the base product's Fortran symbols should go. Specifying **fort** last is equivalent to not treating the Fortran symbols as special, and so wrapping C functions is unconnected to wrapping Fortran functions.

## Implementation notes

Layered profiling is implemented by always linking MPI applications against a library called `libmpiprofilesupport.so`. For performance, the default `libmpiprofilesupport.so` library is an empty stub and is, therefore, inactive in ordinary runs. When you specify `-entry` with a list of libraries, LD_LIBRARY_PATH is modified to include an alternate `libmpiprofilesupport.so` that redefines all MPI symbols, thereby allowing the layered profiling scheme.

When `-entry` is not used, there is no performance impact from being linked against the empty stub library. When `-entry` is used, the performance impact varies, depending on the machine. However, `-entry` has been seen to impact ping pong latency by approximately 15 nanoseconds.

## Using the MPE performance visualization tool

IBM Spectrum MPI includes version mpe2-2.4.9b of the MPE logging library from Argonne National Laboratory. MPE uses the PMPI (standard MPI profiling) interface to provide graphical profiles of MPI traffic for performance analysis. The MPE library is packaged with IBM Spectrum MPI as `libmpe.so` and can be accessed dynamically with the **mpirun** `-entry` command without requiring the application to be recompiled or relinked.

For example:

```
% mpirun -np 2 -entry mpe ./program.x
```

The preceding command turns on MPE tracing and produces a logfile as output in the working directory of rank 0 (for example, `program.x.clog2`). The **jumpshot** command can be used to convert this log file to different formats and to view the results.

### Using the MPE Jumpshot viewer

Jumpshot, which includes the **jumpshot** command, is a performance visualization tool that is distributed by Argonne National Laboratory with MPE. The `jumpshot` command is also included with IBM Spectrum MPI (in the `bin\` directory). The **jumpshot** command can be used to view the MPE tracing output file, as follows:

```
% jumpshot program.x.clog2
```

Note that Jumpshot requires Java™. If Java is not in the path, you can set the JVM environment variable to the full path of the Java executable on your system.

The first time you run the **jumpshot** command, it might issue a prompt that asks you if you want to create a setup file with the default settings. Click OK and Yes. After that, for regular runs on a `.clog2` file, Jumpshot issues another prompt that asks if you want to convert to the SLOG2 format. Click Yes, and then, on the next window, click Convert and then OK. The main window is then displayed with the MPE profiling data.

When using Jumpshot to view the MPE timings, several pop-up windows appear. The most important windows are the main window and a window that indicates the MPI calls by color. Time spent in the various MPI calls is displayed in different colors, and messages are shown as arrows. Right-click on the calls and the message arrows for more information.

For more information about MPE, refer to the performance visualization information at Argonne National Laboratory's website (http://www.mcs.anl.gov/research/projects/perfvis/download/index.htm#MPE).

# Chapter 9. Managing processor affinity

IBM Spectrum MPI follows Open MPI's support of processor affinity for improving performance. With processor affinity, MPI processes and their threads are bound to specific hardware resources such as cores, sockets, and so on.

## Understanding MPI process placement and affinity

Open MPI's **mpirun** affinity options are based on the notions of mapping, ranking, and binding as separate steps, as follows:

**Mapping**
Mapping determines the number of processes that are launched, and on which hosts. Mapping can also be used to associate the hardware resources, such as sockets and cores, with each process.

**Ranking**
Ranking determines an MPI rank index for each process in the mapping. If options are not used to specify ranking behavior, a default granularity is chosen. The ranks are interleaved over the chosen granularity element to produce an ordering.

**Binding**
Binding is the final step and can deviate from the hardware associations that were made at the mapping stage. The binding unit can be larger or smaller than specified by the mapper, and is expanded or round-robined to achieve the final binding.

### Mapping options and modifiers

This section explains some of the options that are available for mapping and includes examples. Note that ranking and binding options are sometimes shown in the mapping examples for more complete explanations.

#### --map-by *unit* option

When using the `--map-by unit` option, *unit* can be any of the following values:

- hwthread
- core
- L1cache
- L2cache
- L3cache
- socket
- numa
- board
- node

`--map-by unit` is the most basic of the mapping policies, and makes process assignments by iterating over the specified unit until the process count reaches the number of available slots.

The following example shows the output (in verbose mode) of the `--map-by unit` option, where core is the specified unit.

```
% mpirun -host hostA:4,hostB:2 -map-by core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../BB/../../../../../..][../../../../../../../..]
R2  hostA  [../../BB/../../../../..][../../../../../../../..]
R3  hostA  [../../../BB/../../../..][../../../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../BB/../../../../../..][../../../../../../../..]
```

This is sometimes called a *packed* or *latency* binding because it tends to produce the fastest communication between ranks.

The following example shows the output (in verbose mode) of using the `--map-by unit` option, where slot is the specified unit.

```
% mpirun -host hostA:4,hostB:2 -map-by socket ...
R0  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R1  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R2  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R3  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R4  hostB  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
```

In the preceding examples, `-host` hostA:4,hostB:2 indicates that the cluster has six slots (spaces in which a process can run). Each rank consumes one slot, and processes are assigned hardware elements by iterating over the specified unit until the available slots are consumed.

The ordering of these examples, is implicitly core and socket, respectively, so core and socket are iterated for each rank assignment. The binding is also implicitly core and socket, respectively, so the final binding is to the same element that was chosen by the mapping.

When options, such as the ranking unit and binding unit, are not explicitly specified, the `-display-devel-map` option can be used to display the implicit selections. In the preceding examples, the `-display-devel-map` includes the following, respectively:

```
Mapping policy:
      BYCORE  Ranking policy: CORE Binding policy: CORE:IF-SUPPORTED
```

```
Mapping policy:
      BYSOCKET  Ranking policy: SOCKET Binding policy: SOCKET:IF-SUPPORTED
```

If no binding options are specified, by default, Open MPI assumes `--map-by-socket` for jobs with more than two ranks. This produces the interleaved ordering in the preceding examples.

**Note:** IBM Spectrum MPI enables binding by default when using the orted tree to launch jobs. The default binding for a less than, or fully subscribed node is `--map-by-socket`. In this case, users might see improved latency by using either the `-aff latency` or `--map-by core` option.

A natural hardware ordering can be created by specifying a smaller unit over which to iterate for ranking. For example:

```
% mpirun -host hostA:4,hostB:2 -map-by socket -rank-by core ...
R0  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R1  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
```

```
R3  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R4  hostB  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
```

A common binding pattern involves binding to cores, but spanning those core assignments over all of the available sockets. For example:

```
% mpirun -host hostA:4,hostB:2 -map-by socket -rank-by core -bind-to core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../BB/../../../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/../../../../../../..]
R3  hostA  [../../../../../../../..][../BB/../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/../../../../../../..]
```

In this example, the final binding unit is smaller than the hardware selection that was made in the mapping step. As a result, the cores within the socket are iterated over for the ranks on the same socket. When the mapping unit and the binding unit differ, the `-display-devel-map` output can be used to display the mapping output from which the binding was taken. For example, at rank 0, the `-display-devel-map` output includes:

```
Locale:  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
Binding: [BB/../../../../../../..][../../../../../../../..]
```

A possible purpose for this binding is to use all the available hardware resources such as cache and memory bandwidth. This is sometimes called a *bandwidth* binding, and is a good starting point for overall application performance. The amount of cache and memory bandwidth is maximized, and the ranks are ordered so that close ranks by index are near each other in the hardware as much as possible while still spanning the available sockets.

On the hardware used in these examples, socket and numa are the same. On some hardware it may be desirable to iterate the process placement over the NUMA nodes instead of over the sockets. In this case, `-map-by numa` can be used. For example:

```
% mpirun -host hostA:4,hostB:2 -map-by numa -rank-by core -bind-to core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../BB/../../../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/../../../../../../..]
R3  hostA  [../../../../../../../..][../BB/../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/../../../../../../..]
```

**Note:** In Open MPI's terminology, *numa* refers to a NUMA node within a host, while *node* refers to the whole host.

In the following example, the host (node) is iterated for process assignments. The ranking unit is also implicitly **node**, so the ordering of the ranks alternates between the hosts as well. However, the binding unit defaults to the smaller socket element and, similar to the preceding bandwidth example, iterates over sockets for subsequent ranks that have the same node binding at the mapping step. For example:

```
% mpirun -host hostA:4,hostB:2 -map-by node ...
R0  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R1  hostB  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R3  hostB  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R4  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R5  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
```

## --map-by slot option

Mapping by slot resembles mapping by an actual hardware unit within the hosts, but each slot is associated with the whole host. The slot is essentially an imaginary hardware unit that exists in a certain number on each host.

Because the slot does not represent a specific subset of cores within a host, slots can be useful in separating the assignment of processes to hosts from the assignment of processes to specific sockets or cores within the host.

## --map-by *unit*:PE=*n* and --map-by slot:PE=*n* options

This option is used to bind n cores to each process. This option requires that the specified *unit* contains at least n cores (or that **slot** is used). Otherwise, process assignments are iterated, as in the examples for `--map-by unit` and `--map by slot`, with the caveat that each process assignment also consumes n slots. For example:

```
% mpirun -host hostA:4,hostB:2 -map-by socket:pe=2 ...
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R2  hostB  [BB/BB/../../../../../..][../../../../../../../..]
```

The most immediate point of interest in this example is that the rank count is only three, not six. This is because each process is consuming n=2 slots. In launching modes, where the slot count represents the number of cores, this is probably desirable because it results in bindings that consume the available number of cores. However, if a specific rank count is desired, the `-host` launching method becomes inconvenient. For example:

```
% mpirun -host hostA:8,hostB:4 -map-by socket:pe=2 ...
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R2  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R3  hostA  [../../../../../../../..][../../BB/BB/../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/../../../../../..]
```

This example shows that the sockets are still iterated over and that the binding width becomes two cores.

If alternating sockets are not desired, a similar mapping can be accomplished by using slots. For example:

```
% mpirun -host hostA:8,hostB:4 -map-by slot:pe=2 ...
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/../..][../../../../../../../..]
R3  hostA  [../../../../../../BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
```

The preceding example resembles a packed binding. It also illustrates how iterating over slots for the mapping causes processes to be assigned to the same host, while leaving the assignment to cores within the host to the binding step.

Because the **slot** is an imaginary, largest-possible hardware unit inside the host that maps to the entire host, iterating rank placements over the slots causes processes to be assigned to the same host, until that host is full, and then moved to the next host. At the mapping stage, each process is assigned to the whole host because that is what a slot is. This can be seen in the output of `--display-devel-map`, which shows that the binding is not made more specific until the binding stage:

```
 Locale:  NODE

    Binding: [BB/../../../../../../..][../../../../../../../..]
```

A similar bandwidth style binding can be produced by adding a `-rank-by core` to the socket mapping:

```
% mpirun -host hostA:8,hostB:4 -map-by socket:pe=2 -rank-by core ...
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R3  hostA  [../../../../../../../..][../../BB/BB/../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/../../../../../..]
```

In the preceding examples, the slot counts in `-host` were modified to produce a desired rank count. A host file, with the special **sequential** option for the mapper, can be used to force any mapping of processes to hosts: `--mca rmaps seq -hostfile` *file*.

```
% cat hostfile
hostA
hostA
hostA
hostA
hostB
hostB
hostA
```

```
% mpirun -hostfile hostfile --mca rmaps seq -map-by socket:pe=2  ...
% mpirun -hostfile hostfile --mca rmaps seq -map-by slot:pe=2  ...

R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/../..][../../../../../../../..]
R3  hostA  [../../../../../../BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
R6  hostA  [../../../../../../../..][BB/BB/../../../../../..]
```

The sequential mapper with a host file allows very flexible rank layouts to be made, but a side effect is that the mapping step only outputs host mapping information. Normally the two preceding examples would differ, with the `-map-by socket` alternating between the sockets to produce a more *bandwidth* style result. But the sequential mapper's output is more coarse and the preceding core mappings occur at the binding step.

The tradeoff here is minor, especially if you are launching fully-subscribed jobs, in which case, *latency* and *bandwidth* bindings are identical. Also, the sequential mapper requires that either a `-map-by` or `-bind-to` option be specified, otherwise, it is incomplete and will fail to launch.

## --map-by ppr:*n*:*unit* and --map-by ppr:*n*:*unit*:pe=*n* options

The *ppr* (*processes per resource*) mode is a convenient shortcut for specifying the number of processes to run on each resource (a socket, for example).

The purpose of `ppr:n:socket` option is to launch n ranks on each socket. The purpose of the `ppr:n:socket:pe=m` option is to launch n ranks per socket, with each rank using m cores.

This following restrictions apply to ppr mode:

- It will only launch if the slot count is high enough to satisfy the ppr instruction. For example, if enough processes are being started to put n on each socket.
- The cluster must be fairly homogeneous in order to be able to meaningfully specify a single number as the ranks per socket.

In "--map-by *unit*:PE=*n* and --map-by slot:PE=*n* options" on page 44, special considerations were given to the launching method because the number of slots used was not one-per-process. However with ppr, slots are not taken into account other than the requirement that enough slots exist to satisfy the specified *processes per resource* instruction.

```
% mpirun -host hostA:4,hostB:4 --map-by ppr:2:socket:pe=2 ...

R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R3  hostA  [../../../../../../../..][../../BB/BB/../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
R6  hostB  [../../../../../../../..][BB/BB/../../../../../..]
R7  hostB  [../../../../../../../..][../../BB/BB/../../../..]
```

## --map-by dist:span option (adapter affinity)

This option, along with `--mca rmaps_dist_device` *device name* (for example, ib0) can be used to enable adapter affinity in Open MPI.

With `--mca rmaps_dist_device`, Open MPI must be allowed to choose the rank layout, so an explicit host file should not be used with this mode. For example:

```
% mpirun -host hostA,hostB -np 18 -bind-to core -map-by dist:span
                    -map-by dist:span --mca rmaps_dist_device mthca0 ...
R0  hostA  [../../../../../../../..][BB/../../../../../../..]
R1  hostA  [../../../../../../../..][../BB/../../../../../..]
R2  hostA  [../../../../../../../..][../../BB/../../../../..]
R3  hostA  [../../../../../../../..][../../../BB/../../../..]
R4  hostA  [../../../../../../../..][../../../../BB/../../..]
R5  hostA  [../../../../../../../..][../../../../../BB/../..]
R6  hostA  [../../../../../../../..][../../../../../../BB/..]
R7  hostA  [../../../../../../../..][../../../../../../../BB]
R8  hostA  [BB/../../../../../../..][../../../../../../../..]
R9  hostB  [../../../../../../../..][BB/../../../../../../..]
R10  hostB  [../../../../../../../..][../BB/../../../../../..]
R11  hostB  [../../../../../../../..][../../BB/../../../../..]
R12  hostB  [../../../../../../../..][../../../BB/../../../..]
R13  hostB  [../../../../../../../..][../../../../BB/../../..]
R14  hostB  [../../../../../../../..][../../../../../BB/../..]
R15  hostB  [../../../../../../../..][../../../../../../BB/..]
R16  hostB  [../../../../../../../..][../../../../../../../BB]
R17  hostB  [BB/../../../../../../..][../../../../../../../..]
% mpirun -host hostA,hostB -np 10 -bind-to core
                    -map-by dist:span,pe=2 --mca rmaps_dist_device mthca0 ...
R0  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R1  hostA  [../../../../../../../..][../../BB/BB/../../../..]
R2  hostA  [../../../../../../../..][../../../../BB/BB/../..]
R3  hostA  [../../../../../../../..][../../../../../../BB/BB]
R4  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/../../../../../..]
R6  hostB  [../../../../../../../..][../../BB/BB/../../../..]
R7  hostB  [../../../../../../../..][../../../../BB/BB/../..]
R8  hostB  [../../../../../../../..][../../../../../../BB/BB]
R9  hostB  [BB/BB/../../../../../..][../../../../../../../..]
```

The -map-by dist option without **span** is less useful, as it fills each host before moving to the next:

```
% mpirun -host hostA,hostB -np 17 -bind-to core -map-by dist
                    --mca rmaps_dist_device mthca0 ...
R0  hostA  [../../../../../../../..][BB/../../../../../../..]
R1  hostA  [../../../../../../../..][../BB/../../../../../..]
R2  hostA  [../../../../../../../..][../../BB/../../../../..]
R3  hostA  [../../../../../../../..][../../../BB/../../../..]
R4  hostA  [../../../../../../../..][../../../../BB/../../..]
R5  hostA  [../../../../../../../..][../../../../../BB/../..]
R6  hostA  [../../../../../../../..][../../../../../../BB/..]
R7  hostA  [../../../../../../../..][../../../../../../../BB]
R8  hostA  [BB/../../../../../../..][../../../../../../../..]
R9  hostA  [../BB/../../../../../..][../../../../../../../..]
R10  hostA  [../../BB/../../../../..][../../../../../../../..]
R11  hostA  [../../../BB/../../../..][../../../../../../../..]
R12  hostA  [../../../../BB/../../..][../../../../../../../..]
R13  hostA  [../../../../../BB/../..][../../../../../../../..]
R14  hostA  [../../../../../../BB/..][../../../../../../../..]
R15  hostA  [../../../../../../../BB][../../../../../../../..]
R16  hostB  [../../../../../../../..][BB/../../../../../../..]
```

# Helper options

## -report-bindings option

This option displays the binding for each rank similarly to the preceding examples, but in a slightly more expanded format:

```
% mpirun -host hostA:4,hostB:2 --report-bindings -map-by core ...

[hostA:ppid] MCW rank 0 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../..][../../../../../../../..]

[hostA:ppid] MCW rank 1 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../..][../../../../../../../..]

[hostA:ppid] MCW rank 2 bound to socket 0[core 2[hwt 0-1]]: [../../BB/../../../../..][../../../../../../../..]
[hostA:ppid] MCW rank 3 bound to socket 0[core 3[hwt 0-1]]: [../../../BB/../../../..][../../../../../../../..]

[hostB:ppid] MCW rank 4 bound to socket 0[core 0[hwt 0-1]]: [BB/../../../../../../..][../../../../../../../..]
[hostB:ppid] MCW rank 5 bound to socket 0[core 1[hwt 0-1]]: [../BB/../../../../../..][../../../../../../../..]
```

## -display-devel-map option

Much of the information displayed with this option is internal, but various parts of the output can be helpful in diagnosing why a certain affinity option is behaving the way it is. The output names that the policy used for mapping, ranking, and binding are particularly useful. The -display-devel-map option displays the number of slots that are used. Also, under the *Locale:* output, it shows the hardware associates that were made in the mapping stage.

```
% mpirun -host hostA:4,hostB:2 --display-devel-map -map-by core ...

 Mapper requested: NULL  Last mapper: round_robin  Mapping policy: BYCORE
  Ranking policy: CORE  Binding policy: CORE:IF-SUPPORTED  Cpu set: NULL
  PPR: NULL  Cpus-per-rank: 1
        Num new daemons: 0      New daemon starting vpid INVALID
        Num nodes: 2

 Data for node: hostA   State: 3
        Daemon: [[11988,0],1]   Daemon launched: True
        Num slots: 4    Slots in use: 4 Oversubscribed: FALSE
        Num slots allocated: 4  Max slots: 0
        Num procs: 4    Next node_rank: 4
        Data for proc: [[11988,1],0]
                Pid: 0  Local rank: 0   Node rank: 0    App rank: 0
```

```
                State: INITIALIZED      App_context: 0
                Locale:  [BB/../../../../../7../..][../../../../../../../..]
                Binding: [BB/../../../../../../../..][../../../../../../../..]
        Data for proc: [[11988,1],1]
                Pid: 0  Local rank: 1   Node rank: 1    App rank: 1
                State: INITIALIZED      App_context: 0
                Locale:  [../BB/../../../../7../..][../../../../../../../..]
                Binding: [../BB/../../../../../../..][../../../../../../../..]
        Data for proc: [[11988,1],2]
                Pid: 0  Local rank: 2   Node rank: 2    App rank: 2
                State: INITIALIZED      App_context: 0
                Locale:  [../../BB/../../../7../..][../../../../../../../..]
                Binding: [../../BB/../../../../../..][../../../../../../../..]
        Data for proc: [[11988,1],3]
                Pid: 0  Local rank: 3   Node rank: 3    App rank: 3
                State: INITIALIZED      App_context: 0
                Locale:  [../../../BB/../../../../..][../../../../../../../..]
                Binding: [../../../BB/../../../../..][../../../../../../../..]

 Data for node: hostB   State: 3
        Daemon: [[11988,0],2]   Daemon launched: True
        Num slots: 2    Slots in use: 2 Oversubscribed: FALSE
        Num slots allocated: 2  Max slots: 0
        Num procs: 2    Next node_rank: 2
        Data for proc: [[11988,1],4]
                Pid: 0  Local rank: 0   Node rank: 0    App rank: 4
                State: INITIALIZED      App_context: 0
                Locale:  [BB/../../../../../../../..][../../../../../../../..]
                Binding: [BB/../../../../../../../..][../../../../../../../..]
        Data for proc: [[11988,1],5]
                Pid: 0  Local rank: 1   Node rank: 1    App rank: 5
                State: INITIALIZED      App_context: 0
                Locale:  [../BB/../../../../../../..][../../../../../../../..]
                Binding: [../BB/../../../../../../..][../../../../../../../..]
```

## Managing oversubscription

*Oversubscription* refers to the concept of allowing more ranks to be assigned to a host than the number of slots that are available on that host.

For example, by default, the following command:

```
% mpirun -host hostA:2,hostB:2 -np 5 ...
```

reports the following output:

```
> There are not enough slots available in the system to satisfy the 5 slots
> that were requested by the application:
```

To allow the mapper to put more ranks on the hosts, the `-oversubscribe` modifier can be given to the mapper. For example:

```
% mpirun -host hostA:2,hostB:2 -np 5 -oversubscribe ...
```

In this way, ranks {0,1,2} would be placed on hostA and ranks {3,4} would be placed on hostB. In this example, hostA is considered to be oversubscribed.

For each host that is oversubscribed, the MPI progression is tuned to yield more when handling MPI traffic, in order to use fewer CPU cycles.

The `-oversubscribe` option does not affect CPU affinity.

## Managing overload

*Overload* occurs in the binding stage of affinity, when ranks are assigned sets of cores. There is a small check to see if more ranks are assigned to any hardware element than there are cores within that hardware element. In that case, the MPI job aborts.

For example, on a machine with only 16 cores, the following command:

```
% mpirun -host hostA:17 --bind-to core ...
```

produces an error message similar to the following:

```
> A request was made to bind to that would result in binding more
> processes than cpus on a resource:
>    Bind to:     CORE
>    Node:        hostA
>    #processes:  2
>    #cpus:       1
```

Here, the binding aborts unless overloading is allowed. Overloading can be allowed by using a binding modifier, as follows:

```
% mpirun -host hostA:17 --bind-to core:overload-allowed ...
```

The *overload-allowed* binding modifier would produce the following affinity without aborting, due to 9 ranks that appear on the first socket:

```
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../../../../../../../..][BB/../../../../../../..]
R2  hostA  [../BB/../../../../../..][../../../../../../../..]
R3  hostA  [../../../../../../../..][../BB/../../../../../..]
R4  hostA  [../../BB/../../../../..][../../../../../../../..]
R5  hostA  [../../../../../../../..][../../BB/../../../../..]
R6  hostA  [../../../BB/../../../..][../../../../../../../..]
R7  hostA  [../../../../../../../..][../../../BB/../../../..]
R8  hostA  [../../../../BB/../../..][../../../../../../../..]
R9  hostA  [../../../../../../../..][../../../../BB/../../..]
R10  hostA  [../../../../../BB/../..][../../../../../../../..]
R11  hostA  [../../../../../../../..][../../../../../../BB/../..]
R12  hostA  [../../../../../../BB/..][../../../../../../../..]
R13  hostA  [../../../../../../../..][../../../../../../../BB/..]
R14  hostA  [../../../../../../../BB][../../../../../../../..]
R15  hostA  [../../../../../../../..][../../../../../../../../BB]
R16  hostA  [BB/../../../../../../..][../../../../../../../..]
```

## IBM Spectrum MPI affinity shortcuts

Spectrum MPI provides shortcuts by way of the `-aff` command line option for some of the underlying Open MPI affinity options. The shortcuts are for *bandwidth* bindings, *latency* bindings, and *cyclic* bindings.

Here is an example of a bandwidth binding:

```
% mpirun -host hostA:4,hostB:2 -map-by socket -rank-by core -bind-to core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../BB/../../../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/../../../../../../..]
R3  hostA  [../../../../../../../..][../BB/../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/../../../../../../..]
```

The shortcut for this bandwidth binding example is `-aff bandwidth` (would become `-map-by socket -rank-by core -bind-to core`).

Here is a latency binding:

```
% mpirun -host hostA:4,hostB:2 -map-by core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../BB/../../../../../..][../../../../../../../..]
R2  hostA  [../../BB/../../../../..][../../../../../../../..]
R3  hostA  [../../../BB/../../../..][../../../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../BB/../../../../../..][../../../../../../../..]
```

The shortcut for this latency binding example is `-aff latency` (would become `-map-by core -rank-by core -bind-to core`).

Here is a cyclic binding, which is similar to a bandwidth binding, but without the `-rank-by core` option reordering the output:

```
% mpirun -host hostA:4,hostB:2 -map-by socket -bind-to core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]

R1  hostA  [../../../../../../../..][BB/../../../../../../..]
R2  hostA  [../BB/../../../../../..][../../../../../../../..]
R3  hostA  [../../../../../../../..][../BB/../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/../../../../../../..]
```

The shortcut for this cyclic binding example is `-aff cycle:unit` (would become `-map-by unit -rank-by unit -bind-to core`).

IBM Spectrum MPI provides the following `-aff` shortcuts:

*Table 4. IBM Spectrum MPI -aff shortcuts*

| Shortcut | Description |
|---|---|
| `-aff auto` | Same as `-aff bandwidth` |
| `-aff bandwidth` | Emulates `-map-by socket -rank-by core -bind-to core` |
| `-aff cycle:unit` | Emulates `-map-by unit -rank-by unit -bind-to core` |
| `-aff default` | Same as `-aff bandwidth` |
| `-aff latency` | Emulates `-map-by core -rank-by core -bind-to core` |
| `-aff none` | Same as `-aff off` |
| `-aff off` | Disables affinity (unbind) |
| `-aff on` | Enables affinity with the default option (bandwidth) |
| `-aff option,option,..` | Comma-separated list of options (`--map-by unit`, `--rank-by unit`, `-bind-to unit`) |
| `-aff v / -aff vv` | Specifies verbose output (`--report-bindings`) |
| `-aff width:unit` | Specifies an alternate `-bind-to unit` value. The value specified for *unit* can be hwthread, core, socket, or numa. |

## IBM PE Runtime Edition affinity equivalents

For users who are migrating from IBM Parallel Environment Runtime Edition, the IBM Spectrum MPI affinity options can be used to create nearly the same functionality that is provided by the following **MP_TASK_AFFINITY** and **MP_CPU_BINDLIST** environment variable settings:


- "MP_TASK_AFFINITY=core" on page 51
- "MP_TASK_AFFINITY=core:n" on page 51

## MP_TASK_AFFINITY=core

The options `-map-by core`, `-map-by socket`, `-rank-by core`, and `-bind-to core` offer similar functionality to the **MP_TASK_AFFINITY**=core environment variable setting. For example:

```
% mpirun -host hostA:4,hostB:2 -map-by core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../BB/../../../../../..][../../../../../../../..]
R2  hostA  [../../BB/../../../../..][../../../../../../../..]
R3  hostA  [../../../BB/../../../..][../../../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../BB/../../../../../..][../../../../../../../..]
```

```
% mpirun -host hostA:4,hostB:2 -map-by socket -rank-by core -bind-to core ...
R0  hostA  [BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../BB/../../../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/../../../../../../..]
R3  hostA  [../../../../../../../..][../BB/../../../../../..]
R4  hostB  [BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/../../../../../../..]
```

## MP_TASK_AFFINITY=core:n

The following options offer similar functionality to the **MP_TASK_AFFINITY**=core:*n* environment variable setting:

- `-map-by slot:pe=n`
- `-map-by socket:pe=n`
- `-map-by ppr:ranks-per-socket:slot:pe=n`
- `-map-by ppr:ranks-per-socket:socket:pe=n`

Depending on the launching method, the rank count that is produced by the `-map-by unit:pe=n` options might not be what you expect because each rank uses *n* slots.

For example:

```
% mpirun -host hostA:8,hostB:4 -map-by slot:pe=2 ...
R0  hostA  [BB/BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/../../..][../../../../../../../..]
R3  hostA  [../../../../../../BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../../..][../../../../../../../..]
```

```
% mpirun -host hostA:8,hostB:4 -map-by socket:pe=2 -rank-by core ...
R0  hostA  [BB/BB/../../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R3  hostA  [../../../../../../../..][../../BB/BB/../../../..]
R4  hostB  [BB/BB/../../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/../../../../../..]
```

Using a host file and the `-mca rmaps seq` option allows specific control of host layout, as long as a packed-style binding is acceptable:

```
% mpirun -hostfile hostfile --mca rmaps seq -map-by slot:pe=2  ...
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/../..][../../../../../../../..]
R3  hostA  [../../../../../../BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
R6  hostA  [../../../../../../../..][BB/BB/../../../../../..]
```

For the `-map-by ppr` options, the slot count must be able to satisfy the specified *processes per resource*, and the resulting layout across the hosts is chosen by MPI. For example, the following command is invalid because the two slots that are listed as available on hostB are not enough to satisfy the instruction to put four processes on each host.

```
% mpirun -host hostA:4,hostB:2 -map-by ppr:4:node:pe=2
```

In the next example, the instruction to put four ranks per host (node) is followed. Even though hostA is listed as having six slots, only four processes are placed on it.

```
% mpirun -host hostA:6,hostB:4 -map-by ppr:4:node:pe=2
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/../..][../../../../../../../..]
R3  hostA  [../../../../../../BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
R6  hostB  [../../../../BB/BB/../..][../../../../../../../..]
R7  hostB  [../../../../../../BB/BB][../../../../../../../..]
```

## MP_TASK_AFFINITY=cpu

The following options offer similar functionality to the **MP_TASK_AFFINITY**=cpu environment variable setting:

- `-map-by hwthread`
- `-map-by socket`
- `-rank-by hwthread`
- `-bind-to hwthread`

For example:

```
% mpirun -host hostA:4,hostB:2 -map-by hwthread ...

R0  hostA  [B./../../../../../../..][../../../../../../../..]
R1  hostA  [.B/../../../../../../..][../../../../../../../..]
R2  hostA  [../B./../../../../../..][../../../../../../../..]
R3  hostA  [../.B/../../../../../..][../../../../../../../..]
R4  hostB  [B./../../../../../../..][../../../../../../../..]
R5  hostB  [.B/../../../../../../..][../../../../../../../..]
```

```
% mpirun -host hostA:4,hostB:2 -map-by socket -rank-by hwthread
             -bind-to hwthread ...

R0  hostA  [B./../../../../../../..][../../../../../../../..]
R1  hostA  [.B/../../../../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][B./../../../../../../..]
R3  hostA  [../../../../../../../..][.B/../../../../../../..]
R4  hostB  [B./../../../../../../..][../../../../../../../..]
R5  hostB  [.B/../../../../../../..][../../../../../../../..]
R6  hostB  [../../../../../../../..][B./../../../../../../..]
R7  hostB  [../../../../../../../..][.B/../../../../../../..]
```

## MP_TASK_AFFINITY=cpu:n

The following options offer similar functionality to the **MP_TASK_AFFINITY**=cpu:*n* environment variable setting:

- `-map-by slot:pe=n -use-hwthread-cpus`
- `-map-by socket:pe=n -use-hwthread-cpus`
- `-map-by ppr:ranks-per-host:node:pe=n -use-hwthread-cpus`
- `-map-by ppr:ranks-per-socket:socket:pe=n -use-hwthread-cpus`

The `-use-hwthread-cpus` option causes the `pe=n` option to refer to hyper-threads instead of cores.

For example:

```
% mpirun -host hostA:16,hostB:8 -map-by slot:pe=4 -use-hwthread-cpus ...
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/../..][../../../../../../../..]
R3  hostA  [../../../../../../BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
```

In the preceding example, the slot counts in the `-host` option are again increased to achieve the desired rank counts, because each rank is using four slots.

```
% mpirun -host hostA:16,hostB:8 -map-by socket:pe=4 -use-hwthread-cpus ...

R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R2  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R3  hostA  [../../../../../../../..][../../BB/BB/../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/../../../../../..]
```

The `-map-by ppr` option over hyper-threads works similarly:

```
% mpirun -host hostA:4,hostB:4 -map-by ppr:4:node:pe=4 -use-hwthread-cpus ...

R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/../..][../../../../../../../..]
R3  hostA  [../../../../../../BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
R6  hostB  [../../../../BB/BB/../..][../../../../../../../..]
R7  hostB  [../../../../../../BB/BB][../../../../../../../..]
% mpirun -host hostA:4,hostB:4 -map-by ppr:2:socket:pe=4 -use-hwthread-cpus ...
R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/BB/../../../../../..]
R3  hostA  [../../../../../../../..][../../BB/BB/../../../..]
R4  hostB  [BB/BB/../../../../../..][../../../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../../../../../../..]
R6  hostB  [../../../../../../../..][BB/BB/../../../../../..]
R7  hostB  [../../../../../../../..][../../BB/BB/../../../..]
```

## MP_TASK_AFFINITY=mcm

The functionality of the `-map-by socket` or `-map-by numa` options is similar to the **MP_TASK_AFFINITY**=mcm environment variable setting. Note that in Open MPI terminology, *node* refers to a full host. The *NUMA node level* is referred to as *numa*.

In Open MPI, the levels are:

- hwthread (hyper-thread, or *cpu* in IBM PE Runtime Edition terminology)
- core
- L1cache
- L2cache
- L3cache
- numa (a *NUMA node*)
- socket
- board
- node (the full host)

In Open MPI, the *mcm* level would equate to either **socket** or **numa**. For example:

```
% mpirun -host hostA:4,hostB:4 -map-by numa ...
R0  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R1  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R2  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R3  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R4  hostB  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
```

```
% mpirun -host hostA:4,hostB:4 -map-by socket -rank-by core ...
R0  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R1  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R2  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R3  hostA  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
R4  hostB  [BB/BB/BB/BB/BB/BB/BB/BB][../../../../../../../..]
R5  hostB  [../../../../../../../..][BB/BB/BB/BB/BB/BB/BB/BB]
```

## MP_CPU_BIND_LIST=*list_of_hyper-threads*

In Open MPI, specific bindings on a per-rank basis can be made using a rankfile.

The list of numbers that is specified in the rankfile refers to cores, and uses logical hardware ordering. If s:a-b is given, it refers to a socket and a range of cores on that socket. For example:

```
% cat rankfile

rank 0=hostA slot=0,1
rank 1=hostA slot=2-3
rank 2=hostA slot=1:4-5
rank 3=hostA slot=0:4-7
rank 4=hostB slot=0-1,8,9
rank 5=hostB slot=2-3,7,8,10-11


% mpirun -rankfile rankfile

R0  hostA  [BB/BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/../../../..][../../../../../../../..]
R2  hostA  [../../../../../../../..][../../../../BB/BB/../..]
R3  hostA  [../../../../BB/BB/BB/BB][../../../../../../../..]
R4  hostB  [BB/BB/../../../../../..][BB/BB/../../../../../..]
R5  hostB  [../../BB/BB/../../../BB][BB/../BB/BB/../../../..]
```

When the `-use-hwthread-cpus` option is used, the numbers in the rank file refer to hyper-threads (using logical hardware order):

```
% cat rankfile
rank 0=hostA slot=0-7
rank 1=hostA slot=4,5,6,7,8,9,10,11
rank 2=hostA slot=8-15
rank 3=hostA slot=0-23
```

```
rank 4=hostB slot=0-3,16-19
rank 5=hostB slot=4-7,20-23


% mpirun -rankfile rankfile -use-hwthread-cpus
R0  hostA  [BB/BB/BB/BB/../../../..][../../../../../../../..]
R1  hostA  [../../BB/BB/BB/BB/../..][../../../../../../../..]
R2  hostA  [../../../../BB/BB/BB/BB][../../../../../../../..]
R3  hostA  [BB/BB/BB/BB/BB/BB/BB/BB][BB/BB/BB/BB/../../../..]
R4  hostB  [BB/BB/../../../../../..][BB/BB/../../../../../..]
R5  hostB  [../../BB/BB/../../../..][../../BB/BB/../../../..]
```

If the socket:*core#-core#* syntax is used in a rankfile, those lines are still interpreted as socket:core even though the `-use-hwthread-cpus` option is specified. For example:

```
% cat rankfile
rank 0=hostA slot=2-3
rank 1=hostA slot=1:2-3


% mpirun -rankfile rankfile -use-hwthread-cpus
R0  hostA  [../BB/../../../../../..][../../../../../../../..]
R1  hostA  [../../../../../../../..][../../BB/BB/../../../..]
```

## OpenMP (and similar APIs)

Open MPI only binds at the process level. The number of threads that are created by a rank and the binding of those threads is not directly controlled by Open MPI. However, by default, created threads would inherit the full mask that is given to the rank.

OpenMP should detect the number of hyper-threads in the process' mask to determine how many threads to create. Alternately, the number of threads to create can be set manually using the **OMP_NUM_THREADS** environment variable

In general, OpenMP is also capable of binding the individual threads more specifically than the inherited mask for the whole process. However, the mechanism varies across versions of OpenMP (settings to explore for this option include **GOMP_CPU_AFFINITY**, **OMP_PROC_BIND**, and **KMP_AFFINITY**).

# Chapter 10. Tuning the runtime environment

IBM Spectrum MPI utilizes the parameters of the Modular Component Architecture (MCA) as the primary mechanism for tuning the runtime environment. Each MCA parameter is a simple key=value pair that controls a specific aspect of the IBM Spectrum MPI functionality.

The MCA parameters can be set to meet your particular runtime requirements in several ways. They can be specified on the **mpirun** command line, exported as environment variables, or supplied in a separate text file.

## Frameworks, components, and MCA parameters

In order to understand how to use MCA parameters, you first need to understand their relationship to MCA's frameworks and components.

The MCA frameworks are divided into three basic types. They are:

- OMPI frameworks (in the MPI layer)
- ORTE frameworks (in the runtime layer)
- OPAL frameworks (in the operating system and platform layer)

An MCA framework uses the MCA's services to find and load components (implementations of the framework's interface) at run time.

The frameworks within the OMPI, ORTE, and OPAL types are further divided into subgroups according to function. For example, the OMPI framework contains a subgroup called *btl*, which is used to send and receive data on different kinds of networks. And within the *btl* framework, there are Byte Transfer Layer-related components (for example, components for shared memory, TCP, Infiniband, and so on), which can be used at runtime.

Likewise, there are many MCA parameters that allow you to control the runtime environment, and these parameters apply to the same groups as the frameworks and components. So, considering the example of the *btl* framework, there is a corresponding collection of MCA parameters that can be used for setting conditions for the Byte Transfer Layer.

The frameworks and their components change over time. For the most up-to-date list of the OMPI, ORTE, and OPAL frameworks, refer to the Open MPI readme file.

## Displaying a list of MCA parameters

The **ompi_info** command displays information about the IBM Spectrum MPI installation. It can be used to display the MCA parameters and their values for a specific framework, a specific component, or for the entire installation.

The **ompi_info** command includes many options, including `--param`, which you can use to display MCA parameters. In general, when using the `--param` option, you specify two arguments. The first argument is the component type (framework), and the second argument is the specific component.

**ompi_info** `--param` *type component*

## Displaying the MCA parameters for a framework

To display the parameters for a entire framework, specify `all` for the second argument. This instructs **ompi_info** to display the MCA parameters and their values for all components of the specified type (framework). For example:

```
ompi_info --param pml all
```

## Displaying the MCA parameters for a component

To display the parameters for a particular component, specify the type (framework) as the first argument and the component name as the second argument. For example, to display the MCA parameters for the tcp component of the btl (Byte Transfer Layer) framework (the component that uses TCP for MPI communications), you could specify **ompi_info** as follows:

```
ompi_info --param pml pami
```

## Displaying the MCA parameters for an entire installation

To display the MCA parameters for all frameworks and components in an IBM Spectrum MPI installation, specify **all** for both arguments:

```
ompi_info --param all all
```

## Controlling the level of MCA parameters that are displayed

Although there are many MCA parameters, only a small number are of interest to any given user at any given time. To simplify things when listing the parameters that are available, IBM Spectrum MPI provides the **ompi_info** `–level` option, which allows you to limit the number and type of MCA parameters that are returned. There are nine different levels that can be specified:

1. Basic information that is of interest to end users.
2. Detailed information that is of interest to end users.
3. All remaining information that is of interest to end users.
4. Basic information that is required for application tuners.
5. Detailed information that is required for application tuners.
6. All remaining information that is required for application tuners.
7. Basic information for Open MPI implementers.
8. Detailed information for Open MPI implementers.
9. All remaining information for Open MPI implementers.

By default, **ompi_info** only displays *level 1* MCA parameters (basic information that is of interest to end users). However, you can display the MCA parameters for additional levels (there are nine) by using the **ompi_info** `--level` option. For example:

```
ompi_info --param pml pami --level 9
```

For more information about using the **ompi_info** `--level` command to control MCA parameter levels, refer to the **ompi_info** man page on the Open MPI web site (www.open-mpi.org).

## Setting MCA parameters

In general, there are three ways that you can set an MCA parameter. These include:

- Specifying it with the **mpirun** command
- Specifying it as an environment variable
- Providing it in a text file.

IBM Spectrum MPI gives precedence to parameter values that are set using the **mpirun** command. Therefore, a given parameter's value that was set using **mpirun** will override the same parameter that was previously set as an environment variable or in a text file.

### Setting MCA parameters with the mpirun command

To specify MCA parameters on the **mpirun** command line, use the `--mca` option. The basic syntax is:

```
mpirun --mca param_name value
```

In the following example, the MCA **mpi_show_handle_leaks** parameter is set to a value of 1 and the program `a.out` is run with four processes:

```
mpirun --mca mpi_show_handle_leaks 1 -np 4 a.out
```

Note that if you want to specify a value that includes multiple words, you must surround the value in quotes so that the shell and IBM Spectrum MPI understand that it is a single value. For example:

```
mpirun --mca param "multiple_word_value" ...
```

### Setting MCA parameters as environment variables

The way in which you specify an MCA parameter as an environment variable differs, depending on the shell that you are using.

For ssh style shells, the syntax of this example would be:

```
OMPI_MCA_mpi_show_handle_leaks=1
   export OMPI_MCA_mpi_show_handle_leaks
   mpirun -np 4 a.out
```

For csh style shells, the syntax of this example would be:

```
setenv OMPI_MCA_mpi_show_handle_leaks 1
   mpirun -np 4 a.out
```

Note that if you want to specify a value that includes multiple words, you must surround the value in quotes so that the shell and IBM Spectrum MPI understand that it is a single value.

An ssh style example is:

```
OMPI_MCA_param="multiple_word_value"
```

A csh style example is:

```
setenv OMPI_MCA_param "multiple_word_value"
```

## Setting MCA parameters by way of a text file

MCA parameter values can be provided in a text file, called `mca-params.conf`. At runtime, IBM Spectrum MPI searches for the `mca-params.conf` file in one of the following locations, and in the following order:

- `$HOME/.openmpi/mca-params.conf`: This is the user-supplied set of values, which has the highest precedence.
- `$prefix/etc/openmpi-mca-params.conf`: This is the system-supplied set of values, which has a lower precedence.

The **mca_param_files** parameter specifies a colon-delimited path of files to search for MCA parameters. Files to the left have lower precedence, while files to the right have higher precedence.

The `mca-params.conf` file contains multiple parameter definitions, in which each parameter is specified on a separate line. The following example shows the **mpi_show_handle_leaks** parameter, as it is specified in a file:

```
# This is a comment
   # Set the same MCA parameter as in previous examples
   mpi_show_handle_leaks = 1
```

Note that in MCA parameter files, quotes are not necessary for setting values that contain multiple words. If you include quotes in the MCA parameter file, they will be used as part of the value itself.

# Accessibility features for IBM Spectrum MPI

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

## Accessibility features

IBM Spectrum MPI includes the following major accessibility features:

- Keyboard-only operation
- Operations that use a screen reader

IBM Spectrum MPI uses the latest W3C Standard, WAI-ARIA 1.0 (www.w3.org/TR/wai-aria/), to ensure compliance with US Section 508 (www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards) and Web Content Accessibility Guidelines (WCAG) 2.0 (www.w3.org/TR/WCAG20/). To take advantage of accessibility features, use the latest release of your screen reader and the latest web browser that is supported by IBM Spectrum MPI.

The IBM Spectrum MPI online product documentation in IBM Knowledge Center is enabled for accessibility. The accessibility features of IBM Knowledge Center are described at http://www.ibm.com/support/knowledgecenter/doc/kc_help.html#accessibility.

## Related accessibility information

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service 800-IBM-3383 (800-426-3383) (within North America)

## IBM and accessibility

For more information about the commitment that IBM has to accessibility, see IBM Accessibility (www.ibm.com/able).

# Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing IBM Corporation*
*North Castle Drive, MD-NC119*
*Armonk, NY 10504-1785*
*US*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing*
*Legal and Intellectual Property Law IBM Japan Ltd.*
*19-21, Nihonbashi-Hakozakicho, Chuo-ku*
*Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Director of Licensing IBM Corporation*
*North Castle Drive, MD-NC119*
*Armonk, NY 10504-1785*
*US*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

© (your company name) (year).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. _enter the year or years_.

# Programming interface information

## MPI support statement

The IBM Spectrum MPI product is a complete MPI implementation, based on the Open MPI open source project, designed to comply with all the requirements of the Message Passing Interface standard, MPI: A Message-Passing Interface Standard, Version 3.1, University of Tennessee, Knoxville, Tennessee, June 4, 2015. If you believe that IBM Spectrum MPI does not comply with the MPI-3.1 standard, please contact IBM Service.

## Open MPI license

```
Most files in this release are marked with the copyrights of the
organizations who have edited them.  The copyrights below are in no
particular order and generally reflect members of the Open MPI core
team who have contributed code to this release.  The copyrights for
code used under license from other parties are included in the
corresponding files.

Copyright (c) 2004-2010 The Trustees of Indiana University and Indiana
                        University Research and Technology
                        Corporation.  All rights reserved.
Copyright (c) 2004-2010 The University of Tennessee and The University
                        of Tennessee Research Foundation.  All rights
                        reserved.
Copyright (c) 2004-2010 High Performance Computing Center Stuttgart,
                        University of Stuttgart.  All rights reserved.
Copyright (c) 2004-2008 The Regents of the University of California.
                        All rights reserved.
Copyright (c) 2006-2010 Los Alamos National Security, LLC.  All rights
                        reserved.
Copyright (c) 2006-2010 Cisco Systems, Inc.  All rights reserved.
Copyright (c) 2006-2010 Voltaire, Inc. All rights reserved.
Copyright (c) 2006-2011 Sandia National Laboratories. All rights reserved.
Copyright (c) 2006-2010 Sun Microsystems, Inc.  All rights reserved.
                        Use is subject to license terms.
Copyright (c) 2006-2010 The University of Houston. All rights reserved.
Copyright (c) 2006-2009 Myricom, Inc.  All rights reserved.
Copyright (c) 2007-2008 UT-Battelle, LLC. All rights reserved.
Copyright (c) 2007-2010 IBM Corporation.  All rights reserved.
Copyright (c) 1998-2005 Forschungszentrum Juelich, Juelich Supercomputing
                        Centre, Federal Republic of Germany
Copyright (c) 2005-2008 ZIH, TU Dresden, Federal Republic of Germany
Copyright (c) 2007      Evergrid, Inc. All rights reserved.
Copyright (c) 2008      Chelsio, Inc.  All rights reserved.
Copyright (c) 2008-2009 Institut National de Recherche en
                        Informatique.  All rights reserved.
Copyright (c) 2007      Lawrence Livermore National Security, LLC.
                        All rights reserved.
Copyright (c) 2007-2009 Mellanox Technologies.  All rights reserved.
Copyright (c) 2006-2010 QLogic Corporation.  All rights reserved.
Copyright (c) 2008-2010 Oak Ridge National Labs.  All rights reserved.
Copyright (c) 2006-2010 Oracle and/or its affiliates.  All rights reserved.
Copyright (c) 2009      Bull SAS.  All rights reserved.
Copyright (c) 2010      ARM ltd.  All rights reserved.
Copyright (c) 2010-2011 Alex Brick .  All rights reserved.
Copyright (c) 2012      The University of Wisconsin-La Crosse. All rights
                        reserved.
Copyright (c) 2013-2016 Intel, Inc. All rights reserved.
```

```
Copyright (c) 2011-2014 NVIDIA Corporation.  All rights reserved.

$COPYRIGHT$

Additional copyrights may follow

$HEADER$

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

- Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer listed
  in this license in the documentation and/or other materials
  provided with the distribution.

- Neither the name of the copyright holders nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

The copyright holders provide no reassurances that the source code
provided does not infringe any patent, copyright, or any other
intellectual property rights of third parties.  The copyright holders
disclaim any liability to any recipient for claims brought against
recipient by any third party for infringement of that parties
intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

## Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

### Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

### Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

### Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## IBM Online Privacy Statement

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies", and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

# Index

## Special characters



## Numerics



## A



## C



## D



## E



## F



## G



## H



## I



## J



## L



## M

# O



# P



# R



# S



# T



# U



# W

Product Number: 5725-G83

Printed in USA

GC27-8265-01