# Intel® MPI Library for Linux* OS

## User's Guide

Document Number: 316404-012

# Contents

# Legal Information

# 1. Introduction

The *Intel® MPI Library for Linux* OS User Guide* explains how to use the Intel® MPI Library in some common usage scenarios. It provides information regarding compiling, linking, running, and debugging MPI applications, as well as information on integration within a cluster environment.

This *User's Guide* contains the following sections

## Document Organization

| **Section** | **Description** |
|---|---|
| Section 1 Introduction | Section 1 introduces this document |
| Section 2 Usage Model | Section 2 presents the usage model for working with the Intel® MPI Library |
| Section 3 Installation and Licensing | Section 3 describes the installation process and provides information about licensing |
| Section 4 Compiling and Linking | Section 4 gives instructions about how to compile and link MPI applications |
| Section 5 Running Applications | Section 5 describes the steps for running an application |
| Section 6 Debugging and Testing | Section 6 explains how to start an application under a debugger |
| Section 7 Process Management | Section 7 gives information about process managers and how to set up password-less ssh connections |
| Section 8 Tuning with mpitune Utility | Section 8 describes how to use the mpitune utility to find optimal settings for the Intel® MPI Library. |
| Section 9 Job Schedulers Support | Section 9 describes integration with job schedulers |
| Section 10 General Cluster Considerations | Section 10 discusses general considerations for clusters related to MPI usage |
| Section 11 Using the Intel® MPI Library with the Intel® Many Integrated Core (Intel® MIC) | Section 11 describes some special considerations when using the Intel® MPI Library with the Intel® MIC architecture |

|  |  |
|---|---|
| Architecture |  |

## 1.1. Introducing Intel® MPI Library

The Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, version 3.0 (MPI-3.0) specification. It provides a standard library across Intel® platforms that:

- Delivers best in class performance for enterprise, divisional, departmental and workgroup high performance computing. The Intel® MPI Library focuses on improving application performance on Intel® architecture based clusters.
- Enables you to adopt MPI-3.0 functions as your needs dictate

## 1.2. Intended Audience

This *User's Guide* helps an experienced user to start using the Intel® MPI Library and contains brief descriptions of the main functionality as a set of how-to instructions and examples. For full information, see *Intel® MPI Library Reference Manual for Linux* OS*.

## 1.3. Conventions and Symbols

The following conventions are used in this document.

**Table 1.3-1 Conventions and Symbols used in this Document**

|  |  |
|---|---|
| *This type style* | Document or product names |
| *This type style* | Hyperlinks |
| This type style | Commands, arguments, options, file names |
| `THIS_TYPE_STYLE` | Environment variables |
| `<this type style>` | Placeholders for actual values |
| `[ items ]` | Optional items |
| `{ item \| item }` | Selectable items separated by vertical bar(s) |
| **(SDK only)** | For Software Development Kit (SDK) users only |

## 1.4. Related Information

To get more information about the Intel® MPI Library, explore the following resources:

- *Intel® MPI Library Release Notes* for updated information on requirements, technical support, and known limitations.
- *Intel® MPI Library Reference Manual* for in-depth knowledge of the product features, commands, options, and environment variables.
- *Intel® MPI Library for Linux* OS Knowledge Base* for additional troubleshooting tips and tricks, compatibility notes, known issues, and technical notes.

For additional resources, see:

*Intel® MPI Library Product Web Site*

*Intel Product Support*

*Intel® Cluster Tools Products Website*

*Intel® Software Development Products Website*

# 2. Usage Model

Using the Intel® MPI Library involves the following steps:

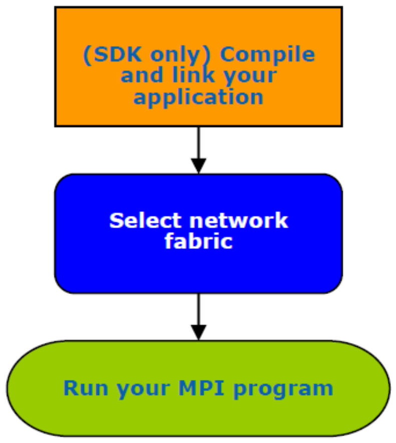

**Figure 1: Flowchart representing the usage model for working with the Intel® MPI Library.**

# 3. Installation and Licensing

This topic describes the installation process and provides information about licensing for the following products:

- Intel® MPI Library
- Intel® MPI Library Runtime Environment (RTO)
- Intel® MPI Library Software Development Kit (SDK)

## 3.1. Installing Intel® MPI Library

If you have a previous version of the Intel® MPI Library for Linux* OS installed, you do not need to uninstall it before installing the latest version.

Extract the `l_mpi[-rt]_p_<version>.<package_num>.tar.gz` package by using following command:

```
tar -xvzf l_mpi[-rt]_p_<version>.<package_num>.tar.gz
```

This command creates the subdirectory `l_mpi[-rt]_p_<version>.<package_num>`.

To start installation, run install.sh. The default installation path for the Intel® MPI Library is `/opt/intel/impi/<version>.<package_num>`.

There are 2 different installations:

- RPM-based installation- this installation requires root password. The product can be installed either on a shared file system or on each node of your cluster.
- Non-RPM installation- this installation does not require root access and it installs all scripts, libraries, and files in the desired directory (usually `$HOME` for the user).

Scripts, include files, and libraries for different architectures are located in different directories. By default, you can find binary files and all needed scripts under `<installdir>/<arch>` directory. For example, for Intel®64 architecture, `<arch>` is bin64.

***NOTE***

To use the full functionality of the library on Intel®64 architecture, set the environment from the `<installdir>/bin64` directory. You can use the provided script files to simplify setting the environment variables. Source the file `<installdir>/bin[64]/mpivars.[c]sh` to set appropriate values for your desired development environment.

For more information on installation, see *Intel® MPI Library for Linux* OS Installation Guide*. You can also find information about how to install the product in silent mode and some useful installer options.

## 3.2. Licensing for Intel® MPI Library Runtime Environment (RTO) and Intel® MPI Library Software Development Kit (SDK)

There are two different licensing options:

- Intel® MPI Library Runtime Environment (RTO) license. This license includes tools you need to run MPI programs including runtime process managers, supporting utilities, shared (.so) libraries and documentation. The license includes everything you need to run Intel® MPI Library-based applications and is free and permanent.

- Intel® MPI Library Software Development Kit (SDK) license. This license includes all of Runtime Environment components as well as the compilation tools: compiler commands (`mpiicc`, `mpicc`, and so on.), files and modules, static (.a) libraries, debug libraries, trace libraries, and test sources. This license is fee-based, with the following options:
    - evaluation
    - named-user
    - floating

For licensing details refer to the EULA or visit http://www.intel.com/go/mpi.

# 4. Compiling and Linking

This topic gives instructions about how to compile and link different kinds of your Intel® MPI Library applications, and details on different debugging and compiler support options.

## 4.1. Compiling a Simple Program

This topic describes the basic steps required to compile and link an MPI program, when using only the Intel® MPI Library Development Kit. To compile and link an MPI program with the Intel® MPI Library:

1. Source the appropriate `mpivars.[c]sh` script to get the proper environment settings (using Bash* shell):

   ```
   $ . <installdir>/bin64/mpivars.sh
   ```

2. Compile your MPI program using the appropriate mpixxx compiler command. For example, to compile a program written in C, use the mpiicc command as follows:

   ```
   $ mpiicc <installdir>/test/test.c -o testc
   ```

You'll get an executable file `testc` in the current directory which can be started immediately. For description of how to launch your application, see Running Applications in this document.

***NOTE:*** By default, the executable file `testc` is linked with the multi-threaded optimized Intel MPI Library. If you need to use other configuration, see Intel® MPI Library Configurations.

Other supported compilers have an equivalent command that uses the prefix mpi on the standard compiler command. For the full list of supported compilers, see the Compiler Commands topic in *Intel® MPI Library Reference Manual for Linux* OS*.

## 4.2. Adding Debug Symbols

If you need to debug your application, add the `-g` option. In this case debug information is added to the binary. You can use any debugger to debug the application.

```
$ mpiicc test.c -o testc -g
```

## 4.3. Other Compilers Support

Intel® MPI Library provides binding libraries to support different operating systems (different glibc* versions) and different compilers. These libraries provide C++, F77, F90 interfaces.

GNU* and Intel® Compilers binding libraries:

- libmpicxx.{a|so} – for g++ >= 3.4;
- libmpifort.{a|so} – for g77/gfortran interface for GNU* and Intel® Compilers;

Your application will be linked against correct GNU* and Intel® Compilers binding libraries if you are using `mpicc`, `mpicxx`, `mpifc`, `mpif77`, `mpif90`, `mpigcc`, `mpigxx`, `mpiicc`, `mpiicpc` or `mpiifort` compiler commands.

For third-party compilers, there is a binding kit which allows you to add support for a certain compiler to the Intel® MPI Library for Linux* OS.

---

***NOTE:***

The Intel® MPI Library supports Intel® compilers as well as GNU* compilers out of the box. See the *Intel® MPI Library Release Notes* for more details.

---

The Intel® MPI Library supports PGI* C, PGI* Fortran 77, Absoft* Fortran 77 compilers out of the box, with the following caveats:

- Your PGI* compiled source files must not transfer long double entities
- Your Absoft* based build procedure must use the `-g77`, `-B108` compiler options
- Install and select the right compilers
- Ensure that the respective compiler runtime is installed on all nodes

You have to build extra Intel® MPI Library binding libraries if you need the support for PGI* C++, PGI* Fortran 95, Absoft* Fortran 95 and GNU* Fortran 95 higher than version 4.0 bindings.

This binding kit provides all the necessary source files, convenience scripts, and instructions you need.

The binding kit and detailed description are located in `binding` directory. To get access to the binding, submit a request to the Intel® MPI Library for Linux* product at the Intel® Premier Support site.

# 5. Running Applications

This topic describes the steps for running applications.

## 5.1. Running an Intel® MPI Program

The easiest way to run an MPI program is using the `mpirun` command:

```
    $ mpirun -n <# of processes> ./myprog
```

This command invokes the `mpiexec.hydra` command which uses the Hydra Process Manager by default. Use the `mpiexec.hydra` options on the `mpirun` command line.

Use the `-n` option to set the number of MPI processes. If the `-n` option is not specified, the process manager pulls the host list from a job scheduler, or uses the number of cores on the machine.

By default, the ssh protocol is used for communication between nodes. If you are using rsh instead, use `-r rsh` option:

```
    $ mpirun -r rsh -n <# of processes> ./myprog
```

For a successful run, configure password-less ssh connections for all nodes. For more details, see the Job Startup Commands topic in *Intel® MPI Library Reference Manual for Linux* OS*.

If you successfully run your application using the Intel® MPI Library, you can move your application from one cluster to another and use different fabrics between the nodes without re-linking. If you encounter problems, see Debugging and Testing for possible solutions.

## 5.2. Intel® MPI Library Configurations

To configure your Intel® MPI Library, source the script `mpivars.[c]sh` with appropriate arguments. For example:

```
$ . <installdir>/bin64/mpivars.sh release
```

You can use the following arguments in this command. The multi-threaded optimized Intel® MPI Library is used by default.

| **Argument** | **Definition** |
|---|---|
| `release` | Set this argument to use single-threaded optimized Intel® MPI Library. |

| Argument | Definition |
|---|---|
| debug | Set this argument to use single-threaded debug Intel® MPI Library. |
| release_mt | Set this argument to use multi-threaded optimized Intel® MPI Library. |
| debug_mt | Set this argument to use multi-threaded debug Intel® MPI Library. |

***NOTE:*** If you want to use different configuration of Intel® MPI Library, run the `mpivars.[c]sh` script with appropriate arguments before an application launch. You do not need to recompile applications.

## 5.3. Multi-threaded Applications

To run OpenMP* application and pin threads inside the domain, make sure the `KMP_AFFINITY` environment variable is set to use the corresponding OpenMP* feature.

Run the application:

```
$ mpirun -genv OMP_NUM_THREADS 4 -n <# of processes> ./myprog
```

For more details see the Interoperability with OpenMP* topic in the *Intel® MPI Library Reference Manual for Linux* OS*.

## 5.4. Selecting Fabrics

By default, the Intel® MPI Library selects a network fabric based on the list of fabrics specified in `I_MPI_FABRICS_LIST`. To select a specific network fabric combination, use the `-genv` option to assign a value to the `I_MPI_FABRICS` variable. You can also assign a value using the export command.

If the specified fabric is not available, Intel® MPI Library will go down the list specified in `I_MPI_FABRICS_LIST` and select the next available fabric.

You can disable this fallback behavior by using the `I_MPI_FALLBACK` variable:

```
$ export I_MPI_FALLBACK=0
```

By default, if `I_MPI_FABRICS` is not set, fallback will be enabled. If `I_MPI_FABRICS` is set, fallback will be disabled.

### Socket connection

Use the following command to run an MPI program over TCP sockets using the available Ethernet connection on the cluster. The program does not use shared memory within a node:

```
    $ mpirun -genv I_MPI_FABRICS tcp -n <# of processes> ./myprog
```

### Shared memory

Use the following command to run an MPI program over the shared-memory fabric (shm) only:

```
    $ mpirun -genv I_MPI_FABRICS shm -n <# of processes> ./myprog
```

### Shared memory and DAPL* connection

To use shared memory for intra-node communication and the Direct Access Programming Library* (DAPL*) layer for inter-node communication, use the following command:

```
$ mpirun -genv I_MPI_FABRICS shm:dapl -n <# of processes> ./myprog
```

Use the `I_MPI_DAPL_UD` environment variable to enable connectionless DAPL User Datagrams* (DAPL UD*):

```
    $ export I_MPI_DAPL_UD=enable

    $ mpirun -genv I_MPI_FABRICS shm:dapl -n <# of processes> ./myprog
```

This is the default method if no fabric options are selected.

### Shared memory and TMI*

To use shared memory for intra-node communication and the Tag Matching Interface* (TMI*) for inter-node communication, use the following command (make sure that you have libtmi.so library in the search path of the ldd command):

```
    $ mpirun -genv I_MPI_FABRICS shm:tmi -n <# of processes> ./myprog
```

This is the recommended method if using Intel® Omni Scale Fabric (formerly Intel® True Scale) or the Myricom* MX interface.

### Shared memory and *OFA

To select shared memory for intra-node communication and OpenFabrics* Enterprise Distribution (OFED) verbs for inter-node communication, use the following command:

```
    $ mpirun -genv I_MPI_FABRICS shm:ofa -n <# of processes> ./myprog
```

This is the recommended method if using the Open Fabrics* Enterprise Distribution (OFED*) software stack.

## Multi-rail capability

If your cluster is equipped with several connection cards or multi-port cards, you can improve bandwidth of communications using the following settings:

```
$ export I_MPI_FABRICS=shm:ofa

$ export I_MPI_OFA_NUM_ADAPTERS=<num>
```

Where `<num>` is the number of connection adapters (1 by default).

If connection cards have several ports, you can specify the number of ports using the following setting:

```
$ export I_MPI_OFA_NUM_PORTS=<num>
```

For more details see the Fabrics Control topic in *Intel® MPI Library Reference Manual for Linux* OS*.

If you successfully run your application using the Intel MPI Library over any of the fabrics described, you can move your application from one cluster to another and use different fabrics between the nodes without re-linking. If you encounter problems, see Debugging and Testing for possible solutions.

Additionally, using `mpirun` is the recommended practice when using a resource manager, such as PBS Pro* or LSF*.

For example, to run the application in the PBS environment, follow these steps:

1. Create a PBS launch script that specifies number of nodes requested and sets your Intel MPI Library environment. For example, create a `pbs_run.sh` file with the following content:

```
#PBS -l nodes=2:ppn=1

#PBS -l walltime=1:30:00

#PBS -q workq

#PBS -V

# Set Intel MPI environment

mpi_dir=<installdir>/<arch>/bin

cd $PBS_O_WORKDIR

source $mpi_dir/mpivars.sh

# Launch application

mpirun -n <# of processes> ./myprog
```

2. Submit the job using the PBS `qsub` command:

   ```
   $ qsub pbs_run.sh
   ```

When using `mpirun` under a job scheduler, you do not need to determine the number of available nodes. Intel MPI Library automatically detects the available nodes through the Hydra process manager.

### 5.4.1. I_MPI_FABRICS

This topic is an excerpt from the *Intel® MPI Library Reference Manual for Linux* OS* which provides further details on the `I_MPI_FABRICS` environment variable.

Select a particular network fabric to be used for communication.

#### Syntax

```
I_MPI_FABRICS=<fabric>|<intra-node fabric>:<inter-nodes fabric>
```

Where

- `<fabric> := {shm, dapl, tcp, tmi, ofa}`
- `<intra-node fabric> := {shm, dapl, tcp, tmi, ofa}`
- `<inter-nodes fabric> := {shm, tcp, tmi, ofa}`

#### Arguments

| **Argument** | **Definition** |
|---|---|
| *`<fabric>`* | Define a network fabric |
| `shm` | Shared-memory |
| `dapl` | DAPL-capable network fabrics, such as InfiniBand*, iWarp*, Dolphin*, and XPMEM* (through DAPL*) |
| `tcp` | TCP/IP-capable network fabrics, such as Ethernet and InfiniBand* (through IPoIB*) |
| `tmi` | Network fabrics with tag matching capabilities through the Tag Matching Interface (TMI), such as Intel® True Scale Fabric and Myrinet* |
| `ofa` | Network fabric, such as InfiniBand* (through OpenFabrics* Enterprise Distribution (OFED*) verbs) provided by the Open Fabrics Alliance* |

|  |  |
|---|---|
|  | (OFA*) |

For example, to select the winOFED* InfiniBand* device, use the following command:

```
$ mpirun -n <# of processes>  \

-env I_MPI_FABRICS shm:dapl <executable>
```

For these devices, if `<provider>` is not specified, the first DAPL* provider in the `/etc/dat.conf` file is used. The `shm` fabric is available for both Intel® and non-Intel microprocessors, but it may perform additional optimizations for Intel microprocessors than it performs for non-Intel microprocessors.

***NOTE***

Ensure the selected fabric is available. For example, use `shm` only if all the processes can communicate with each other through the availability of the `/dev/shm` device. Use `dapl` only when all processes can communicate with each other through a single DAPL provider.

# 6. Debugging and Testing

This topic explains how to debug MPI applications with different debugger tools.

## 6.1. GDB*: The GNU* Project Debugger

Use the following command to launch the GDB* debugger with Intel® MPI Library:

```
$ mpirun -gdb -n 4 ./testc
```

You can work with the GDB debugger as you usually do with a single-process application. For details about how to work with parallel programs, see the GDB documentation at http://www.gnu.org/software/gdb/.

You can also attach to a running job with

```
$ mpirun -n 4 -gdba <pid>
```

Where `<pid>` is the process ID for the running MPI rank.

## 6.2. TotalView* Debugger

Intel® MPI Library supports the use of the TotalView* debugger from Rogue Wave* Software, Inc. To debug an MPI program, add `-tv` to the global `mpirun` arguments, as in

```
$ mpirun -tv -n 4 ./testc
```

***NOTE***

In case of ssh communication, you need to set the `TVDSVRLAUNCHCMD` environment variable to ssh.

You will get a popup window from TotalView asking whether you want to start the job in a stopped state. If so, when the TotalView window appears, you may see assembly code in the source window. Click on `main()` in the stack window (upper left) to see the source of the main function. TotalView shows that the program (all processes) are stopped in the call to `MPI_Init()`. When debugging with TotalView using the above startup sequence, you need to exit TotalView before restarting an Intel MPI Library job.

To debug with TotalView an enable restarting the session, use the following command line:

```
$ totalview python -a 'which mpirun' -tvsu <mpirun_args> <prog>
<prog_args>
```

If you have TotalView 8.1.0 or later, you can use a feature called indirect launch.

1. Invoke TotalView as:

   ```
   $ totalview <prog> -a <prog_args>
   ```

2. Select the **Process/Startup Parameters** command.

3. Choose the **Parallel** tab in the resulting dialog box and choose **MPICH2** as the parallel system.

4. Set the number of tasks using the **Tasks** field.

5. Enter other needed `mpirun` arguments into the **Additional Starter Arguments** field.

If you want to be able to attach to a running MPI job using TotalView, you must use the `-tvsu` option in the `mpirun` command when starting the job. Using this option adds a barrier inside `MPI_Init()` and hence may affect startup performance slightly. After all tasks have returned from `MPI_Init()`, there is no performance degradation incurred from this option.

## 6.3. DDT* Debugger

You can debug MPI applications using DDT* debugger. Intel does not provide support for this debugger. You should obtain the support from Allinea*. According to the DDT User Guide at http://www.allinea.com/products/ddt-support/, you can use the –tv option to run DDT with certain TotalView* variables set beforehand.

```
$ export TOTALVIEW=DDT_INSTALLATION_PATH/bin/ddt-debugger-mps

$ mpirun -np 4 -tv ./your_app
```

If you have problems with the DDT debugger, see DDT documentation for help.

## 6.4. Logging

Sometimes debugging an application is not effective and you prefer to use logging instead. There are several ways to get logging information from running applications.

### 6.4.1. I_MPI_DEBUG

Environment variable `I_MPI_DEBUG` provides a very convenient way to get information from an MPI application at runtime. You can set value of this variable from 0 (the default value) to 1000. The higher the value, the more debug information you get.

```
$ mpirun -genv I_MPI_DEBUG 5 -n 8 ./my_application
```

***NOTE***

High values of `I_MPI_DEBUG` can output a lot of information and significantly reduce performance of an application.  A value of `I_MPI_DEBUG=5` is generally a good starting point, which provides sufficient information to find common errors. See the `I_MPI_DEBUG` description in Intel® MPI Library Reference Manual for Linux* OS for more details.

### 6.4.2. Tracing an Application

Use the `-t` or `-trace` option to link the resulting executable files against the Intel® Trace Collector library. This has the same effect as when `-profile=vt` is used as an argument to `mpiicc` or another compiler script.

```
$ mpiicc -trace test.c -o testc
```

This option requires that you include the installation path of the Intel® Trace Collector in the `VT_ROOT` environment variable. Set `I_MPI_TRACE_PROFILE` to the `<profile_name>` environment variable to specify another profiling library. For example, set `I_MPI_TRACE_PROFILE` to `vtfs` to link against the fail-safe version of the Intel® Trace Collector.

***NOTE***

To use this option, you need to install the Intel® Trace Analyzer and Collector first.  The tool is distributed as part of the Intel® Parallel Studio XE Cluster Edition bundle only.

### 6.4.3. Correctness Checking

Use `-check_mpi` option to link the resulting executable file against the Intel® Trace Collector correctness checking library. This has the same effect as when `-profile=vtmc` is used as an argument to `mpiicc` or another compiler script.

```
$ mpiicc -profile=vtmc test.c -o testc
```

Or

```
$ mpiicc -check_mpi test.c -o testc
```

Thi option requires that you include the installation path of the Intel® Trace Collector in the `VT_ROOT` environment variable. Set `I_MPI_CHECK_PROFILE` to the `<profile_name>` environment variable to specify another checking library.

***NOTE***

To use this option, you need to install the Intel® Trace Analyzer and Collector first. The tool is distributed as part of the Intel® Parallel Studio XE Cluster Edition bundle only.

For more information on the Intel® Trace Analyzer and Collector, see the documentation provided with this product.

### 6.4.4. Statistics Gathering

If you want to collect statistics on MPI functions used in your application, you can set the `I_MPI_STATS` environment variable to a number from 1 to 10. You set this environment variable to control the amount of statistics information collected and the output to the log file. By default, no statistics are gathered.

For details, see Statistics Gathering Mode in *Intel® MPI Library Reference Manual for Linux* OS*.

## 6.5. Testing the Installation

To ensure that the Intel® MPI Library is installed and functioning correctly, complete the general testing below, in addition to compiling and running a test program.

To test the installation (on each node of your cluster):

1. Verify that `<installdir>/<arch>/bin` is in your `PATH`:

   `$ ssh <nodename>` which mpirun
   You should see the correct path for each node you test.

   (SDK only) If you use the Intel® Composer XE packages, verify that the appropriate directories are included in the `PATH` and `LD_LIBRARY_PATH` environment variables
   ```
   $ mpirun -n <# of processes> env | grep PATH
   ```
   You should see the correct directories for these path variables for each node you test. If not, call the appropriate `compilervars.[c]sh` script. For example, for the Intel® Composer XE 2015 use the following source command:
   ```
   $ ./opt/intel/composer_xe_2015/bin/compilervars.sh intel64
   ```

2. In some unusual circumstances, you need to include the `<installdir>/<arch>/lib` directory in your `LD_LIBRARY_PATH`. To verify your `LD_LIBRARY_PATH` settings, use the command:
   ```
   $ mpirun -n <# of processes> env | grep LD_LIBRARY_PATH
   ```

### 6.5.1. Compiling and Running a Test Program

To compile and run a test program, do the following:

1. (SDK only) Compile one of the test programs included with the product release as follows:

   ```
   $ cd <installdir>/test

   $ mpiicc -o myprog test.c
   ```

2. If you are using InfiniBand*, Myrinet*, or other RDMA-capable network hardware and software, verify that everything is functioning correctly using the testing facilities of the respective network.

3. Run the test program with all available configurations on your cluster.

- Test the TCP/IP-capable network fabric using:
  ```
  $ mpirun -n 2 -genv I_MPI_DEBUG 2 -genv I_MPI_FABRICS tcp ./myprog
  ```
  You should see one line of output for each rank, as well as debug output indicating the TCP/IP-capable network fabric is used.

- Test the shared-memory and DAPL-capable network fabrics using:
  ```
  $ mpirun -n 2 -genv I_MPI_DEBUG 2 -genv I_MPI_FABRICS shm:dapl
  ./myprog
  ```
  You should see one line of output for each rank, as well as debug output indicating the shared-memory and DAPL-capable network fabrics are being used.

- Test any other fabric using:
  ```
  $ mpirun -n 2 -genv I_MPI_DEBUG 2 -genv I_MPI_FABRICS <fabric>
  ./myprog
  ```
  where `<fabric>` is a supported fabric. For more information, see Selecting Fabrics.

For each of the `mpirun` commands used, you should see one line of output for each rank, as well as debug output indicating which fabric was used. The fabric(s) should agree with the `I_MPI_FABRICS` setting.

The `<installdir>/test` directory in the Intel® MPI Library Development Kit contains other test programs in addition to `test.c`

# 7. Process Management

This topic describes the process managers included with the Intel® MPI Library.

## 7.1. Selecting a Process Manager

The `mpirun` script uses the process manager specified by the `I_MPI_PROCESS_MANAGER` variable. By default, `mpirun` selects the Hydra Process Manager. Setting `I_MPI_PROCESS_MANAGER` to hydra will explicitly select Hydra, and setting it to `mpd` will explicitly select MPD. The process manager can also be selected by directly calling the appropriate `mpiexec` file: `mpiexec.hydra` for Hydra or `mpiexec` for MPD.

## 7.2. Scalable Process Management System (Hydra)

Hydra is a simplified, scalable process manager. Hydra will check for known resource managers to determine where the processes may be run and to distribute the processes among the targets using proxies on each host. These proxies will be used for process launching, cleanup, I/O forwarding, signal forwarding, and other tasks.

You can start Hydra by using `mpiexec.hydra`. See Scalable Process Management System (Hydra) Commands topic for a detailed list of options in the *Intel® MPI Library Reference Manual*.

## 7.3. Multipurpose Daemon* (MPD*)

MPD stands for Multipurpose Daemon. This is the Intel® MPI Library process management system for starting parallel jobs, which have to run on all nodes. MPDs gather information about the system and hardware, as well as communicate with each other to exchange required information. For example, an MPD ring is required for correct pinning under the MPD process manager.

***NOTE***

Multipurpose daemon* (MPD) has been deprecated starting with Intel® MPI Library 5.0 release. Convert to using the scalable process management system (Hydra) instead to start parallel jobs.

## 7.4. Controlling MPI Process Placement

The `mpirun` command controls how the ranks of the processes are allocated to the nodes of the cluster. By default, the `mpirun` command uses group round-robin assignment, putting consecutive MPI process on all processor ranks of a node. This placement algorithm may not be the best choice for your application, particularly for clusters with symmetric multi-processor (SMP) nodes.

Suppose that the geometry is `<#ranks> = 4` and `<#nodes> = 2`, where adjacent pairs of ranks are assigned to each node (for example, for two-way SMP nodes). To see the cluster nodes, enter the command:

```
cat ~/mpi_hosts
```

The results should look as follows:

```
clusternode1

clusternode2
```

To equally distribute four processes of the application on two-way SMP clusters, enter the following command:

```
mpirun -perhost 2 -n 4 ./myprog.exe
```

The output for the `myprog.exe` executable file may look as follows:

```
Hello world: rank 0 of 4 running on clusternode1

Hello world: rank 1 of 4 running on clusternode1

Hello world: rank 2 of 4 running on clusternode2

Hello world: rank 3 of 4 running on clusternode2
```

Alternatively, you can explicitly set the number of processes to be executed on each host through the use of argument sets. One common use case is when employing the master-worker model in your application. For example, the following command equally distributes the four processes on `clusternode1` and on `clusternode2`:

```
mpirun -n 2 -host clusternode1 ./myprog.exe : -n 2 -host clusternode2
./myprog.exe
```

### See Also

You can get more details in the *Local Options* topic of the *Intel® MPI Library Reference Manual* for Linux* OS.

You can get more information about controlling MPI process placement online at Controlling Process Placement with the Intel® MPI Library.

# 8. Tuning with mpitune Utility

This section describes how to use the `mpitune` utility to find optimal settings for the Intel® MPI Library.

## 8.1. Cluster-Specific Tuning

Intel® MPI Library has more than 100 parameters. The defaults are set for common usage and generally provide good performance for most clusters and most applications. However, if you want to get even higher performance, you can use the `mpitune` utility. This utility uses the Intel® MPI Benchmarks (IMB) as a benchmark program running tests several times with different parameters and searching for the best ones. Start the `mpitune` utility with the following command:

```
$ mpitune
```

Then, start your application with the `-tune` option to enable the tuned settings:

```
$ mpirun -tune -perhost 8 -n 64 ./your_app
```

For best results, run `mpitune` with write access permissions for `<installdir>/<arch>/etc` which is the location for tuned parameters. If you do not have write access, a new configuration file will be saved in your current directory.

By default, `mpitune` uses the Intel® MPI Benchmarks (IMB) as benchmark program. Alternatively, you can substitute with your benchmark of choice by using the following command:

```
$ mpitune -test \”your_benchmark -param1 -param2\”
```

You can then apply the new settings as described in Cluster-Specific Tuning.

The Intel® MPI Benchmarks executable files, which are more optimized for Intel microprocessors than for non-Intel microprocessors, are used by default. This may result in different tuning settings on Intel microprocessors than on non-Intel microprocessors.

## 8.2. Application-Specific Tuning

Use the `mpitune` utility to find optimal settings for your specific application.

```
$ mpitune --application \”your_app\” --output-file yourapp.conf
```

`“your_app”` here means that you need to provide the exact command line you use to start your application. For example:

```
“mpitune --application \"./my_test\" --output-file $PWD/my_test.conf
```

Tuned settings are saved in `yourapp.conf` file. To apply them, call `mpirun` as:

```
$ mpirun -tune $PWD/yourapp.conf -perhost 8 -n 64 ./your_app
```

"`your_app`" is not only executable file but any script which can be started as a separate process.

***NOTE***

That script should not change the `I_MPI_*` variables.

## 8.3. Setting Time Limit

The process of tuning can take a lot of time. Due to the varying factors involved in each cluster and application setup, the tuning time can be unpredictable.

To restrict the tuning time, you can set the `-time-limit` option. For example, to limit the tuning to 8 hours (480 minutes), run the following command:

```
$ mpitune --time-limit 480
```

The time limit value should be set in minutes.

## 8.4. Setting a Fabrics List

To set a limitation for which fabrics should be tested, you can use the `fabrics-list` option.

```
$ mpitune --fabrics-list shm,ofa,dapl
```

Fabrics can be chosen from the following list: `shm:dapl`, `shm:tcp`, `shm`, `dapl`, `shm:ofa`, `shm:tmi`, `ofa`, `tmi`, `tcp`.

## 8.5. Setting a Range for the Number of Processes

If you want to set a limitation on the number of processes running on one node, you can use the `perhost-range min:max` option. For example: the following command restricts running anywhere from 4 to 8 MPI ranks on each node:

```
$ mpitune --perhost-range 4:8
```

## 8.6. Setting a Limitation for Hosts Usage

You can set a limitation on the number of nodes on which tuning will be performed. For example, the following command will restrict running on 8 to 16 nodes only:

```
$ mpitune --host-range 8:16
```

## 8.7. Restoring mpitune from the Last Saved Session

Sometimes an unexpected event can occur and the results of many days tuning can be lost. `mpitune` saves intermediate results into a `mpituner_session_<session-id>.mts` file and you can restart `mpitune` from the last saved session:

```
$ mpitune --session-file ./mpituner_session_<session-id>.mts
```

Where `<session-id>` is the Unix* timestamp of the moment tuner started.

## 8.8. Tuning Applications Manually

There is a family of `I_MPI_ADJUST_*` environment variables that allow you to manually tune the collective operations of the Intel® MPI Library. By setting a range of message sizes and choosing different algorithms, you can improve the performance of your application. For more information, see the `I_MPI_ADJUST` Family topic in *Intel® MPI Library Reference Manual for Linux* OS* for details.

# 9. Job Schedulers Support

The Intel® MPI Library supports the majority of commonly used job schedulers in the HPC field.

The following job schedulers are supported on Linux* OS:

- Altair* PBS Pro*;
- Torque*;
- OpenPBS*;
- IBM* Platform LSF*;
- Parallelnavi* NQS*;
- SLURM*;
- Oracle* Grid Engine*.

On Linux* OS, this support is implemented in the `mpirun` wrapper script. `mpirun` determines the job scheduler under which it was started by checking specific environment variables and then chooses the appropriate method to start an application.

## 9.1. Altair* PBS Pro*, TORQUE*, and OpenPBS*

If you use one of these job schedulers, with the `$PBS_ENVIRONMENT` exists and the value is `PBS_BATCH` or `PBS_INTERACTIVE` , `mpiru` n uses `$PBS_NODEFILE` as a machine file for `mpirun`. You do not need to specify the `-machinefile` option directly.

Example of a batch job script:

```
#PBS -l nodes=4:ppn=4

#PBS -q queue_name

cd $PBS_O_WORKDIR

mpirun -n 16 ./myprog.exe
```

## 9.2. IBM* Platform LSF*

If yo use the IBM* Platform LSF* job scheduler, and the `$LSB_MCPU_HOSTS` is set, it will be parsed to get the list of hosts for the parallel job. `$LSB_MCPU_HOSTS` does not store the main process name; thus the local hostname will be added to the top of the hosts list. Based on this hosts list, a machine file for `mpirun` is generated with a unique name: `/tmp/lsf_${username}.$$`. The machine file is removed when the job is completed.

Example:

```
$ bsub -n 16 mpirun -n 16 ./myprog.exe
```

## 9.3. Parallelnavi NQS*

If you use Parallelnavi NQS* job scheduler and the `$ENVIRONMENT`, `$QSUB_REQID`, `$QSUB_NODEINF` options are set, the `$QSUB_NODEINF` file is used as a machine file for `mpirun`. Also `-r plesh` is used as remote shell by the process manager during startup.

## 9.4. SLURM*

If the `$SLURM_JOBID` is set, the `$SLURM_TASKS_PER_NODE`, `$SLURM_NODELIST` environment variables will be used to generate a machine file for `mpirun`. The name of the machine file is `/tmp/slurm_${username}.$$`. The machine file will be removed when the job is completed.

Example:

```
$ srun -N2 --nodelist=host1,host2 -A

$ mpirun -n 2 myprog.exe
```

## 9.5. Oracle* Grid Engine*

If you use the Oracle* Grid Engine* job scheduler and the `$PE_HOSTFILE` is set, then two files will be generated: `/tmp/sge_hostfile_${username}_$$` and `/tmp/sge_machifile_${username}_$$`. The latter is used as the machine file for `mpirun`. These files are removed when the job is completed.

## 9.6. SIGINT, SIGTERM Signals Intercepting

If resources allocated to a job exceed the limit, most job schedulers terminate the job by sending a signal to all processes.

For example, Torque* sends `SIGTERM` three times to a job and if this job is still alive, `SIGKILL` will be sent to terminate it.

For Oracle Grid Engine*, the default signal to terminate a job is `SIGKILL`. Intel® MPI Library is unable to process or catch that signal causing `mpirun` to kill the entire job.  You can change the value of the termination signal through the following queue configuration:

- Use the following command to see available queues:

  ```
  $ qconf –sql
  ```

- Execute the following command to modify the queue settings:

  ```
  $ qconf -mq <queue_name>
  ```

- Find `"terminate_method"` and change signal to `SIGTERM`

- Save queue configuration

# 10. General Cluster Considerations

This topic discusses general considerations for clusters related to MPI usage:

- Defining which nodes to use
- Password-less ssh connection
- Heterogeneous system and jobs

## 10.1. Defining which Nodes to Use

Intel® MPI Library will look for a file called `mpd.hosts` by default. This file should contain a list of all available nodes on the cluster which can be used for your application. The format of the `mpd.hosts` file is a list of node names, one name per line. Blank lines and the portions of any lines that follow a # character are ignored.

You can specify the full path to this file by the `–f` option.

When running under a supported job scheduler, using the `–f` option is unnecessary as the hosts will be determined from the scheduler.

## 10.2. Password-less ssh Connection

When a process is started remotely, `ssh` is used to launch the processes by default. Without a password-less SSH capability enabled, your password will be requested when launching jobs. A script is provided with the Intel® MPI Library installation package that will automatically generate and distribute SSH keys for a user. The script is named sshconnectivity.exp and is located in the main folder after extracting the tarball. If the script does not work for your system, the keys can be generated and distributed manually by following these steps:

1. Generate a public key

   ```
   local> ssh-keygen -t dsa -f .ssh/id_dsa
   ```

   When you are prompted for a password, leave it blank by pressing the <enter> key

   Two new files `id_dsa` and `id_dsa.pub` are created in the `.ssh` directory. The latter one is the public key.

2. Distribute the public key to remote nodes

Go to the `.ssh` directory. Copy the public key to the remote machines.

```
local> cd .ssh

local> scp id_dsa.pub user@remote:~/.ssh/id_dsa.pub
```

Log into the remote machine and go to the `.ssh` directory on the remote side.

```
local> ssh user@remote

remote> cd .ssh
```

Add the client's public key to the known public keys on the remote server.

```
remote> cat id_dsa.pub >> authorized_keys

remote> chmod 640 authorized_keys

remote> rm id_dsa.pub

remote> exit
```

Next time you log into the remote server, you will not be prompted for a password.

***NOTE***

`ssh` setup depends on the ssh client distribution.

## 10.3. Heterogeneous Systems and Jobs

All clusters are not homogeneous.  All jobs are not homogeneous.  The Intel® MPI Library is able to run multiple sets of commands and arguments in one command line through two different methods.

The easiest option for running multiple commands in two methods is by creating a configuration file and using it by the `-configfile` option. A configuration file contains a set of arguments to `mpirun`, one group per line.

```
-n 1 -host node1 ./io <io_args>

-n 4 -host node2 ./compute <compute_args_1>

-n 4 -host node3 ./compute <compute_args_2>
```

Alternatively, a set of options can be passed on the command line by separating each group with ":".

```
mpirun -n 1 -host node1 ./io <io_args> : -n 4 -host node2 ./compute
<compute_args_1> : -n 4 -host node3 ./compute <compute_args_2>
```

When a process is launched, the working directory will be set to the working directory of the machine where the job was launched. To change this, use the `-wdir <path>.`

Use `-env <var> <value>` to set an environment variable to a value for only one process group. Using `-genv` instead will apply the environment variable to all process groups. By default, all environment variables are propagated from the environment at launch.

# 11. Using the Intel® MPI Library with the Intel® Many Integrated Core (Intel® MIC) Architecture

Using the Intel® MPI Library in combination with an Intel® MIC Architecture card is similar to using another node, but there are a few special considerations. This topic provides the information on these special considerations.

## 11.1. Libraries

The Intel® MIC Architecture uses different binaries and libraries, and these must be present on the card. In order to copy the appropriate files to the card, you can use the following commands:

```
(host)$ scp <installdir>/mic/bin/* host0-mic0:/bin/
```

```
(host)$ scp <installdir>/mic/lib/* host0-mic0:/lib64/
```

This assumes that the hostname of the card is `host0-mic0`. Any additional libraries needed by the application can be copied in a similar manner.

## 11.2. Multiple Cards

To use multiple cards for a single job, the Intel® Manycore Platform Software Stack (Intel® MPSS) needs to be configured for peer-to-peer support (see the Intel® MPSS documentation for details) and the host(s) needs to have IP forwarding enabled.

```
(host)$ sudo sysctl -w net.ipv4.ip_forward=1
```

Each host/card should be able to ping every other host/card and the launching host should be able to connect to every target, as with a classic cluster.

## 11.3. Using Intel® MPI Library on Intel® Xeon Phi™ Coprocessor

Intel® MPI Library for the Intel® Many Integrated Core Architecture (Intel® MIC Architecture) supports only the Intel® Xeon Phi™ coprocessor (previously codenamed: Knights Corner).

### 11.3.1. Building an MPI Application

To build an MPI application for the host node and the Intel® Xeon Phi™ coprocessor, follow these steps:

1. Establish the environment settings for the compiler and for the Intel® MPI Library:

```
$ . <install-dir>/composerxe/bin/compilervars.sh intel64

$ . <install-dir>/impi/intel64/bin/mpivars.sh
```

2. Build your application for Intel® Xeon Phi™ coprocessor:

```
$ mpiicc -mmic myprog.c -o myprog.mic
```

3. Build your application for Intel® 64 architecture:

```
$ mpiicc myprog.c -o myprog
```

### 11.3.2. Running an MPI Application

To run an MPI application on the host node and the Intel® Xeon Phi™ coprocessor, do the following:

1. Ensure that NFS is properly set up between the hosts and the Intel® Xeon Phi™ coprocessor(s). For information on how to set up NFS on the Intel® Xeon Phi™ coprocessor(s), visit the Intel® Xeon Phi™ coprocessor developer community at http://software.intel.com/en-us/mic-developer.

2. Use the `I_MPI_MIC_POSTFIX` environment variable to append the .mic postfix extension when running on the Intel® Xeon Phi™ coprocessor.

```
$ export I_MPI_MIC_POSTFIX=.mic
```

3. Make sure your `~/mpi_hosts` file contains the machine names of your Intel® Xeon® host processors and the Intel® Xeon Phi™ coprocessor(s). For example:

```
$ cat ~/mpd.hosts

clusternode1

clusternode1-mic0
```

4. Launch the executable file from the host.

```
$ export I_MPI_MIC=on
$ mpirun -n 4 -hostfile ~/mpi_hosts ./myprog
```

***NOTE***

You can also use the `-configfile` and `-machinefile` options.

To run the application on Intel® Xeon Phi™ coprocessor only, follow the steps described above and ensure that `mpd.hosts` contains only the Intel® Xeon Phi™ coprocessor name.

### See Also

You can get more details in the Intel® Xeon Phi™ Coprocessor Support topic of the Intel® MPI Library Reference Manual for Linux* OS.

You can get more information about using Intel® MPI Library on Intel® Xeon Phi™ coprocessor online at How to run Intel® Xeon Phi™ Coprocessor.