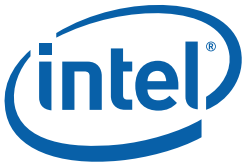

Intel® MPI Library for Linux* OS

Getting Started Guide

The Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, version 2.1 (MPI-2.1) specification.

This *Getting Started Guide* explains how to use the Intel® MPI Library to compile and run a simple MPI program. This guide also includes basic usage examples and troubleshooting tips.

To quickly start using the Intel® MPI Library, print this short guide and walk through the example provided.

Document Number: 315398-008

Revision: 4.0

World Wide Web: http://www.intel.com

Contents

Disclaimer and Legal Notices

1 About this Document

The *Intel® MPI Library for Linux* OS Getting Started Guide* contains information on the following subjects:

- First steps using the Intel® MPI Library
- Troubleshooting outlines first-aid troubleshooting actions

1.1 Intended Audience

This *Getting Started Guide* is intended for first time users.

1.2 Using Doc Type Field

This *Getting Started Guide* contains the following sections:

Document Organization

Section	Description
Section 1 About this Document	Section 1 introduces this document
Section 2 Using the Intel® MPI Library	Section 2 describes how to use the Intel® MPI Library
Section 3 Troubleshooting	Section 3 outlines first-aid troubleshooting actions
Section 4 Next Steps	Section 4 provides links to further resources

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, see the following resources:

Product Web Site

Intel® MPI Library Support

Intel® Cluster Tools Products

Intel® Software Development Products

2 Using the Intel® MPI Library

2.1 Usage Model

Using the Intel® MPI Library involves the following steps:

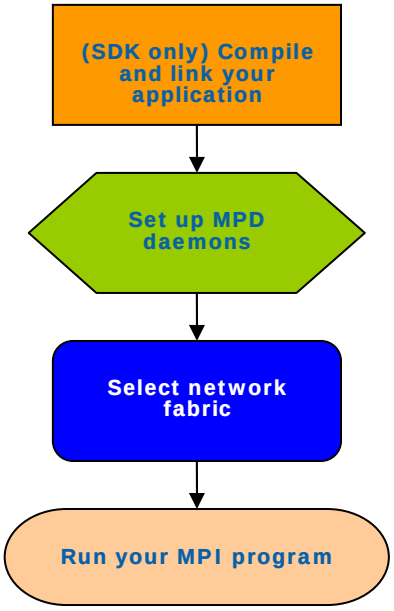

Figure 1: Flowchart representing the usage model for working with the Intel® MPI Library.

2.2 Before You Begin

Before using the Intel® MPI Library, ensure that the library, scripts, and utility applications are installed. See the product *Intel® MPI Library for Linux* OS Installation Guide* for installation instructions.

2.3 Quick Start

1. Source the `mpivars.[c]sh` script to establish the proper environment settings for the Intel® MPI Library. It is located in the `<installdir>/<arch>/bin` directory, where `<installdir>` refers to the Intel MPI Library installation directory (for example, `/opt/intel/impi`) and `<arch>` is one of the following architectures:
   - `ia32` – IA-32 architecture binaries
   - `intel64` – Intel® 64 architecture binaries.
2. Create an `mpd.hosts` text file that lists the nodes in the cluster using one host name per line.
3. **(SDK only)** Make sure you have a compiler in your `PATH`. To find the path to your compiler, run the `which` command on the desired compiler. For example:

```
$ which icc
/opt/intel/cc/11.1/bin/intel64/icc
```

4. **(SDK only)** Compile a test program using the appropriate compiler driver. For example:
   ```
   $ mpiicc -o myprog <installdir>/test/test.c
   ```
5. Execute the test using the `mpirun` command.
   ```
   $ mpirun -r ssh -n <# of processes> ./myprog
   ```
6. See the rest of this document and the *Intel® MPI Library Reference Manual* for more details.

2.4 Compiling and Linking

(SDK only)

To compile and link an MPI program with the Intel® MPI Library:

1. Ensure that the underlying compiler and related software appear in your `PATH`.

   If you are using the Intel® Professional Edition Compilers, ensure that the compiler library directories appear in the `LD_LIBRARY_PATH` environment variable.
   For example, for Intel® C++ Compiler version 11.1 and Intel® Fortran Compiler version 11.1, execute the appropriate setup scripts:

   `/opt/intel/cc/11.1/bin/iccvars.[c]sh`, and

   `/opt/intel/fc/11.1/bin/ifortvars.[c]sh`
2. Compile your MPI program by the appropriate `mpi` compiler script.
   For example, use the `mpicc` command to compile C code using the GNU* C compiler as follows:
   ```
   $ mpicc -o myprog <installdir>/test/test.c
   ```
   where *<installdir>* is the full path to the installed package.

   All supported compilers have equivalent commands that use the prefix `mpi` for the standard compiler command. For example, the Intel MPI Library command for the Intel® Fortran Compiler (`ifort`) is `mpiifort`.

2.5 Setting up MPD Daemons

The Intel® MPI Library uses a Multi-Purpose Daemon (MPD) job startup mechanism. To run programs compiled with the `mpiicc` (or related) commands, set up the MPD daemons first.

Always start and maintain your own set of MPD daemons. This setup enhances system security and gives you flexibility in controlling your execution environment. It is not recommended that the system administrator starts up the MPD daemons once for use by all users on the system.

To set up the MPD daemons:

1. Set up the environment variables with appropriate values and directories. For example, in the `.cshrc` or `.bashrc` files:
   - Ensure that the `PATH` variable includes the *<installdir>*`/<arch>/bin` directory. Use the `mpivars.[c]sh` scripts included with the Intel MPI Library to set up this variable.
   - Ensure that the `PATH` variable includes the directory for Python* version 2.2 or higher.
   - **(SDK only)** If you are using the Intel® Professional Edition Compilers, ensure that the `LD_LIBRARY_PATH` variable contains the directories for the compiler library. Set this variable by using the `{icc, ifort}*vars.[c]sh` scripts included with the compiler.
   - Set any additional environment variables that your application uses.

NOTE: Steps 2 through 4 are optional. They are automatically completed when using the `mpiexec` or `mpirun` commands.

2. *[Optional]* Create a $HOME/.mpd.conf file. To set up your MPD password, enter the following into this file:
   secretword=<mpd_secret_word>

Do not use a Linux* login password. The arbitrary <mpd_secret_word> string only controls access to the MPD daemons by various cluster users.

3. *[Optional]* Set protection on the $HOME/.mpd.conf file using the chmod command so that only you have read and write privileges:

```
$ chmod 600 $HOME/.mpd.conf
```

4. *[Optional]* Verify that you can observe the PATH settings and .mpd.conf contents through ssh on all nodes of the cluster. For example, use the following commands with each <node> on the cluster:

```
$ ssh <node> env
$ ssh <node> cat $HOME/.mpd.conf
```

Make sure that every node, rather than only one of them, can connect to any other node without a password.

5. Create an mpd.hosts text file that lists the nodes in the cluster using one host name per line.

For example:

```
$ cat > mpd.hosts
node1
node2
…
<ctrl>-D
```

6. Shut down any pre-existing MPD daemons using the mpdallexit command:

```
$ mpdallexit
```

7. Use the mpdboot command to start up the MPD daemons:

```
$ mpdboot -n <#nodes>
```

By default, the file $PWD/mpd.hosts is used if present. If there is no host file, the mpdboot command, without any parameters, starts a single MPD daemon on the local machine.

NOTE: If your cluster uses ssh instead of rsh, add the -r ssh or --rsh=ssh option to the mpdboot invocation string.

8. Use the mpdtrace command to determine the status of the MPD daemons:

```
$ mpdtrace
```

The output should be a list of nodes that are currently running MPD daemons and are part of the same MPD ring. This should match the list of nodes in the mpd.hosts file, allowing for name resolution.

2.6 Selecting a Network Fabric

The Intel® MPI Library dynamically selects the most appropriate fabric for communication between MPI processes. To select a specific fabric combination, set the new I_MPI_FABRICS or the old I_MPI_DEVICE environment variable.

I_MPI_FABRICS

(I_MPI_DEVICE)

Select a particular network fabric to be used for communication.

Syntax

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

Where <fabric> := {shm, dapl, tcp tmi, ofa}

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

<inter-nodes fabric> := {dapl, tcp, tmi, ofa}

Deprecated Syntax

I_MPI_DEVICE=<device>[:<provider>]

Arguments

<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 Qlogic* and Myrinet*
ofa	Network fabric, such as InfiniBand* (through OpenFabrics* Enterprise Distribution (OFED*) verbs) provided by the Open Fabrics Alliance* (OFA*)

Correspondence with I_MPI_DEVICE

<device>	Equivalent notation for the I_MPI_FABRICS variable
sock	tcp
shm	shm
ssm	shm:tcp
rdma	dapl
rdssm	shm:dapl is the default value
<provider>	Optional DAPL* provider name (only for the rdma and rdssm devices) I_MPI_DAPL_PROVIDER=<provider> or I_MPI_DAPL_UD_PROVIDER=<provider>

Use the <provider> specification only for the {rdma,rdssm} devices.

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

```
$ mpiexec -n <# of processes> \
          -env I_MPI_DEVICE rdssm:OpenIB-cma <executable>
```

For these devices, if <provider> is not specified, the first DAPL* provider in the /etc/dat.conf file is used.

NOTE: Ensure that the selected fabric is available. For example, use shm only when 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.

2.7 Running an MPI Program

To launch programs linked with the Intel® MPI Library, use the `mpirun` or the `mpiexec` commands.

When using `mpirun`, use the `mpdboot` options on the `mpirun` command line. You do not need to manually start up the MPD daemon by `mpdboot`.

Make sure that you have an MPD ring started before using the `mpiexec` command. Use the `mpdtrace` command to determine the status of the MPD daemons. Use the `mpdboot` command to start up the MPD daemons if necessary.

2.7.1 Running an MPI Program with mpiexec

To launch programs linked with the Intel® MPI Library, use the `mpiexec` command:

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

Use the `-n` option to set the number of MPI processes. This is the only obligatory option for the `mpiexec` command.

If you are using a network fabric different than the default fabric, use the `-genv` option to assign a value to the `I_MPI_FABRICS` variable.

For example, to run an MPI program over the `shm` fabric, use the following command:

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

or

```
$ mpiexec -genv I_MPI_DEVICE shm -n <# of processes> ./myprog
```

For a `dapl`-capable fabric, use the following command:

```
$ mpiexec -genv I_MPI_FABRICS dapl -n <# of processes> ./myprog
```

or

```
$ mpiexec -genv I_MPI_DEVICE rdma -n <# of processes> ./myprog
```

To use shared memory for intra-node communication and the DAPL layer for inter-node communication, use the following command:

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

or

```
$ mpiexec -genv I_MPI_DEVICE rdssm -n <# of processes> ./myprog
```

or simply

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

To use shared memory for intra-node communication and TMI for inter-node communication, use the following command:

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

Make sure that you have `libtmi.so` library in the search path of the `ldd` command. There is no way to select this fabric combination through the deprecated `I_MPI_DEVICE` variable.

To select shared memory for intra-node communication and OFED verbs for inter-node communication, use the following command:

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

There is no way to select this fabric combination through the deprecated I_MPI_DEVICE variable.

Set the I_MPI_OFA_NUM_ADAPTERS or the I_MPI_OFA_NUM_PORT environment variable to utilize the multirail capabilities.

NOTE: The exact settings depend on your cluster configuration. For example, if you have 2 InfiniBand* cards installed on your cluster nodes, use the following command:

```
$ export I_MPI_OFA_NUM_ADAPTERS=2
$ mpiexec -genv I_MPI_FABRICS shm:ofa -n <# of processes> ./myprog
```

Set the I_MPI_DAPL_UD environment variable to enable connectionless DAPL Unreliable Datagrams (DAPL UD).

```
$ export I_MPI_DAPL_UD=enable
$ mpiexec -genv I_MPI_FABRICS shm:dapl -n <# of processes> ./myprog
```

If you successfully ran 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 *Troubleshooting* for possible solutions.

2.7.2 Running an MPI Program with mpirun

An alternative way to run parallel applications using the Intel® MPI Library is to use the following command:

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

This command invokes the mpdboot, mpiexec, and mpdallexit commands. mpirun shuts the MPD daemons down as soon as the program execution is complete. Use this command when you do not need to reuse the MPD daemons after the program run. Using mpirun is also 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, sets your Intel MPI Library environment, etc. 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
   ```

NOTE: When using `mpirun`, use the `mpdboot` options on the `mpirun` command line. You do not need to manually start up the MPD daemon by `mpdboot`. Similarly, `mpirun` will determine the number of available nodes and start the MPD daemons automatically when running under a supported resource manager.

3 Troubleshooting

Use the following sections to troubleshoot problems with installation, setup, and execution of applications using the Intel® MPI Library.

3.1 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 you have Python* version 2.2 or higher in your PATH:
   ```
   $ ssh <nodename> python -V
   ```
   If this command returns an error message or a value lower than 2.2, install Python* version 2.2 or higher, and make sure that you have it in your PATH.
2. Check for the presence of a Python* XML module such as **python-xml** or **libxml2-python**:
   ```
   $ rpm –qa | grep python-xml
   $ rpm –qa | grep libxml2-python
   ```
   Install the missing module if the output does not include the name **python-xml** or **libxml2-python** and a version number.
3. Check for the presence of an XML parser such as **expat** or **pyxml**:
   ```
   $ rpm –qa | grep expat
   $ rpm –qa | grep pyxml
   ```
   Install the missing module if the output does not include the name **expat** or **pyxml** and a version number.
4. Verify that <installdir>/<arch>/bin is in your PATH:
   ```
   $ ssh <nodename> which mpiexec
   ```
   You should see the correct path for each node you test.

   (SDK only) If you use the Intel® Professional Edition Compilers, verify that the appropriate directories are included in the PATH and LD_LIBRARY_PATH environment variables
   ```
   $ mpiexec –n <# of processes> env | grep PATH
   ```
   Start an MPD ring before executing the mpiexec command. You should see the correct directories for these path variables for each node you test. If not, call the appropriate {icc, ifort}*vars.[c]sh script. For example, for the Intel® C++ Compiler version 11.1 use the following source command:
   ```
   $ ./opt/ intel/cc/11.1/bin/iccvars.sh
   ```
5. 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:
   ```
   $ mpiexec –n <# of processes> env | grep LD_LIBRARY_PATH
   ```

3.2 Troubleshooting MPD Setup

Check if it is possible to run the mpd command on the local machine. Do the following:

```
$ mpd &
$ mpdtrace
$ mpdallexit
```

The output of mpdtrace should show the hostname of the machine you are running on. If this is not the case, or if you cannot start up the MPD, check that the installation was correct and the environment was set up properly.

The next troubleshooting steps assume that the MPD daemons are set up and running. To start your diagnosis, verify that the MPD daemons are running on all expected nodes using:

```
$ mpdtrace
```

The output lists all MPD daemons running or indicates an error. If some desired nodes are missing from the output list of mpdtrace, do the following:

1. Try to restart the MPD daemons using the following commands:
   - Kill all running MPD daemons:
     `$ mpdallexit`
   - For each node, ensure all daemons were killed:
     `$ ssh <nodename> ps -ael | grep python`
     `$ ssh <nodename> kill -9 <remaining python processes>`
   - Reboot the MPD daemons. Be sure to use the appropriate configuration options and host file:
     `$ mpdboot [<options>]`
   - Confirm that all expected MPD daemons are now running:
     `$ mpdtrace`
2. If the output of the mpdtrace command is still not indicating that all expected MPD daemons are running, follow the next steps:
   - Kill and restart the MPD daemons as described in step 1, adding the debug and verbose options to the mpdboot command:
     `$ mpdboot –d –v [<options>]`
     Find the ssh commands in the output. For example:
     ```
     launch cmd= ssh  -n <nodename> '<installdir>/<arch>/bin/mpd \
     -h <nodename> -p <port-number> --ncpus=<ncpus> -e -d'
     ```
   - Copy and paste the line of the output from the ssh command up to the end of the line. For example:
     ```
     $ ssh -n <nodename> '<installdir>/<arch>/bin/mpd \
     -h <nodename> -p <port-number> --ncpus=<ncpus> -e -d'
     ```
   - Execute the edited ssh command. Use the result output to diagnose and correct the underlying problem. For example, the most common problems include:
     - Failure of the ssh command to contact <nodename>.
     - Other failure of the ssh command, for example, a system setup problem.
     - The <installdir>/<arch>/bin/mpd command cannot be found or cannot be executed.

3.3 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:
  `$ mpiexec -n 2 -env I_MPI_DEBUG 2 –env 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 being used.

- Test the shared-memory and DAPL–capable network fabrics using:

  ```
  $ mpiexec -n 2 -env I_MPI_DEBUG 2 –env 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:

  ```
  $ mpiexec -n 2 -env I_MPI_DEBUG 2 -env I_MPI_FABRICS <fabric> ./myprog
  ```

  where `<fabric>` can be a supported fabric. For more information, see *Selecting a Network Fabric*.

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

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

4 Next Steps

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

See the *Intel® MPI Library Release Notes* for updated information on requirements, technical support, and known limitations.

The *Intel® MPI Library Reference Manual* for in-depth knowledge of the product features, commands, options, and environment variables.

Visit the Intel® MPI Library for Linux* Knowledge Base for additional troubleshooting tips and tricks, compatibility notes, known issues, and technical notes.

For more information see Websites:

Product Web Site

Intel® MPI Library Support

Intel® Cluster Tools Products

Intel® Software Development Products

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101