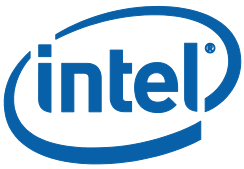

Intel® MPI Library for Linux* OS

Developer Guide

Contents

Legal Information

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

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel technologies features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at Intel.com, or from the OEM or retailer.

Intel, the Intel logo, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.

Optimization Notice

Intel's compilers 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 SSE2, SSE3, and 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. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

* Other names and brands may be claimed as the property of others.

1. Introduction

The *Intel® MPI Library Developer Guide* explains how to use the Intel® MPI Library in some common usage scenarios. It provides information regarding compiling, running, debugging, tuning and analyzing MPI applications, as well as troubleshooting information.

This *Developer Guide* helps a user familiar with the message passing interface start using the Intel® MPI Library. For full information, see the *Intel® MPI Library Developer Reference*.

1.1. Introducing Intel® MPI Library

The Intel® MPI Library is a multi-fabric message-passing library that implements the Message Passing Interface, version 3.1 (MPI-3.1) 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-3.1 functions as your needs dictate

The product comprises the following main components:

- *Runtime Environment (RTO)* includes the tools you need to run programs, including scalable process management system (Hydra*), supporting utilities, shared (`.so`) libraries, and documentation.
- *Software Development Kit (SDK)* includes all of the Runtime Environment components plus compilation tools, including compiler drivers such as `mpiicc`, include files and modules, static (`.a`) libraries, debug libraries, and test codes.

1.2. Conventions and Symbols

The following conventions are used in this document:

This type style	Document or product names.
`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.

1.3. 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 Developer Reference* 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® Software Documentation Library
- Intel® Software Products Support

2. Installation and Licensing

2.1. Installation

If you have a previous version of the Intel® MPI Library for Linux* OS installed, you do not need to uninstall it before installing a newer 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/compilers_and_libraries_<version>.<update>.<package#>/linux/mpi`.

There are two 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`.

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.

2.2. Licensing for Intel® MPI Library Distributions

There are two different licensing options:

- Intel® MPI Library Runtime Environment (RTO) license. The license covers 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 covers all of Runtime Environment components as well as the compilation tools: compiler drivers (`mpiicc`, `mpicc`, and so on), files and modules, static (`.a`) libraries, debug libraries, trace libraries, and test sources. This license is fee-based, with several options described in the product end-user license agreement (EULA).

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

3. Prerequisite Steps

Before you start using any of the Intel® MPI Library functionality, make sure to establish the proper environment for Intel MPI Library. Follow these steps:

1. Set up the Intel MPI Library environment. Source the `mpivars.[c]sh` script:

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

   By default, `<installdir>` is `/opt/intel/compilers_and_libraries_<version>.<update>.<package>/linux/mpi`.

2. To run an MPI application on a cluster, Intel MPI Library needs to know names of all its nodes. Create a text file listing the cluster node names. The format of the file is one name per line, and the lines starting with `#` are ignored. To get the name of a node, use the `hostname` utility.

   A sample host file may look as follows:

   ```
   $ cat ./hosts
   # this line is ignored
   clusternode1
   clusternode2
   clusternode3
   clusternode4
   ```

3. For communication between cluster nodes, Intel MPI Library uses the SSH protocol. You need to establish a passwordless SSH connection to ensure proper communication of MPI processes. Intel MPI Library provides the `sshconnectivity.exp` script that helps you do the job. It automatically generates and distributes SSH authentication keys over the nodes.

   The script is located at `/opt/intel/compilers_and_libraries_<version>.<update>.<package>/linux/mpi`. Run the script and pass the previously created host file as an argument.

   If the script does not work for your system, try generating and distributing authentication keys manually.

After completing these steps, you are ready to use Intel® MPI Library.

4. Compiling and Linking

4.1. Compiling an MPI Program

This topic describes the basic steps required to compile and link an MPI program, using the Intel® MPI Library SDK.

To simplify linking with MPI library files, Intel MPI Library provides a set of compiler wrapper scripts with the `mpi` prefix for all supported compilers. To compile and link an MPI program, do the following:

1. Make sure you have a compiler in your `PATH` environment variable. For example, to check if you have the Intel® C Compiler, enter the command:

   ```
   $ which icc
   ```

   If the command is not found, add the full path to your compiler into the `PATH`. For Intel® compilers, you can source the `compilervars.[c]sh` script to set the required environment variables.

2. Source the `mpivars.[c]sh` script to set up the proper environment for Intel MPI Library. For example, using the Bash* shell:

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

3. Compile your MPI program using the appropriate compiler wrapper script. For example, to compile a C program with the Intel® C Compiler, use the `mpiicc` script as follows:

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

You will get an executable file `myprog` in the current directory, which you can start immediately. For instructions of how to launch MPI applications, see Running Applications.

NOTE

By default, the resulting executable file is linked with the multi-threaded optimized library. If you need to use another library configuration, see Selecting Library Configuration.

For details on the available compiler wrapper scripts, see the *Developer Reference*.

See Also

Intel® Many Integrated Core Architecture Support
Intel® MPI Library Developer Reference, section *Command Reference* > *Compiler Commands*

4.1.1. Compiling an MPI/OpenMP* Program

To compile a hybrid MPI/OpenMP* program using the Intel® compiler, use the `-qopenmp` option. For example:

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

This enables the underlying compiler to generate multi-threaded code based on the OpenMP* pragmas in the source. For details on running such programs, refer to Running an MPI/OpenMP* Program.

4.1.2. Static vs. Dynamic Linking

By default, when compiling an application, the Intel MPI Library gets dynamically linked with it. To link the library statically, use the `-static_mpi` compiler wrapper option. For example:

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

To see the resulting compilation command line without compiling the application, you can use the `-show` option.

4.1.3. Adding Debug Information

If you need to debug your application, add the `-g` option to the compilation command line. For example:

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

This adds debug information to the resulting binary, enabling you to debug your application. Debug information is also used by analysis tools like Intel® Trace Analyzer and Collector for displaying locations of application functions in the source code.

4.1.4. Test MPI Programs

Intel® MPI Library comes with a set of source files for simple MPI programs that enable you to test your installation. Test program sources are available for all supported programming languages and are located at `<installdir>/test`, where `<installdir>` is `/opt/intel/compilers_and_libraries_<version>.x.xxx/linux/mpi` by default.

4.2. Compilers Support

Intel® MPI Library supports the GCC* and Intel® compilers out of the box. It uses binding libraries to provide support for different `glibc` versions and different compilers. These libraries provide C++, Fortran 77, and Fortran 90 interfaces.

The following binding libraries are used for GCC* and Intel® compilers:

- `libmpicxx.{a|so}` – for `g++` version 3.4 or higher
- `libmpifort.{a|so}` – for `g77`/`gfortran` interface for GCC and Intel® compilers

Your application gets linked against the correct GCC* and Intel® compilers binding libraries, if you use one of the following compiler wrappers: `mpicc`, `mpicxx`, `mpifc`, `mpif77`, `mpif90`, `mpigcc`, `mpigxx`, `mpiicc`, `mpiicpc`, or `mpiifort`.

For other compilers, PGI* and Absoft* in particular, there is a binding kit that allows you to add support for a certain compiler to the Intel® MPI Library. This binding kit provides all the necessary source files, convenience scripts, and instructions you need, and is located in the `<install_dir>/binding` directory.

To add support for the PGI* C, PGI* Fortran 77, Absoft* Fortran 77 compilers, you need to manually create the appropriate wrapper script (see instructions in the binding kit *Readme*). When using these compilers, keep in mind the following limitations:

- Your PGI* compiled source files must not transfer `long double` entities
- Your Absoft* based build procedure must use the `-g77`, `-B108` compiler options

To add support for the PGI* C++, PGI* Fortran 95, Absoft* Fortran 95, and GNU* Fortran 95 (4.0 and newer) compilers, you need to build extra binding libraries. Refer to the binding kit *Readme* for detailed instructions.

5. Running Applications

5.1. Running an MPI Program

Before running an MPI program, place it to a shared location and make sure it is accessible from all cluster nodes. Alternatively, you can have a local copy of your program on all the nodes. In this case, make sure the paths to the program match.

Run the MPI program using the `mpirun` command. The command line syntax is as follows:

```
$ mpirun -n <# of processes> -ppn <# of processes per node> -f <hostfile> ./myprog
```

For example:

```
$ mpirun -n 4 -ppn 2 -f hosts ./myprog
```

In the command line above:

- `-n` sets the number of MPI processes to launch; if the option is not specified, the process manager pulls the host list from a job scheduler, or uses the number of cores on the machine.
- `-ppn` sets the number of processes to launch on each node; if the option is not specified, processes are assigned to the physical cores on the first node; if the number of cores is exceeded, the next node is used.
- `-f` specifies the path to the host file listing the cluster nodes; alternatively, you can use the `-hosts` option to specify a comma-separated list of nodes; if hosts are not specified, the local node is used.
- `myprog` is the name of your MPI program.

The `mpirun` command is a wrapper around the `mpiexec.hydra` command, which invokes the Hydra process manager. Consequently, you can use all `mpiexec.hydra` options with the `mpirun` command.

For the list of all available options, run `mpirun` with the `-help` option, or see the *Intel® MPI Library Developer Reference*, section *Command Reference > Hydra Process Manager Command*.

NOTE

The commands `mpirun` and `mpiexec.hydra` are interchangeable. However, you are recommended to use the `mpirun` command for the following reasons:

- You can specify all `mpiexec.hydra` options with the `mpirun` command.
- The `mpirun` command detects if the MPI job is submitted from within a session allocated using a job scheduler like PBS Pro* or LSF*. Thus, you are recommended to use `mpirun` when an MPI program is running under a batch scheduler or job manager.

See Also

Controlling Process Placement
Job Schedulers Support
Intel® Many Integrated Core Architecture Support

5.2. Running an MPI/OpenMP* Program

To run a hybrid MPI/OpenMP* program, follow these steps:

1. Make sure the thread-safe (debug or release, as desired) Intel® MPI Library configuration is enabled (this is the default behavior). To switch to such configuration, source `mpivars.[c]sh` with the appropriate argument. For example:

   ```
   $ source mpivars.sh release_mt
   ```

2. Set the `I_MPI_PIN_DOMAIN` environment variable to specify the desired process pinning scheme. The recommended value is `omp`:

   ```
   $ export I_MPI_PIN_DOMAIN=omp
   ```

   This sets the process pinning domain size to be equal to `OMP_NUM_THREADS`. Therefore, if for example `OMP_NUM_THREADS` is equal to `4`, each MPI process can create up to four threads within the corresponding domain (set of logical processors). If `OMP_NUM_THREADS` is not set, each node is treated as a separate domain, which allows as many threads per MPI process as there are cores.

NOTE

For pinning OpenMP* threads within the domain, use the `KMP_AFFINITY` environment variable. See the Intel® compiler documentation for more details.

---

3. Run your hybrid program as a regular MPI program. You can set the `OMP_NUM_THREADS` and `I_MPI_PIN_DOMAIN` variables directly in the launch command. For example:

   ```
   $ mpirun -n 4 -genv OMP_NUM_THREADS=4 -genv I_MPI_PIN_DOMAIN=omp ./myprog
   ```

See Also

Intel® MPI Library Developer Reference, section *Tuning Reference > Process Pinning > Interoperability with OpenMP**.

5.3. MPMD Launch Mode

Intel® MPI Library supports the multiple programs, multiple data (MPMD) launch mode. There are two ways to do this.

The easiest way is to create a configuration file and pass it to the `-configfile` option. A configuration file should contain a set of arguments for `mpirun`, one group per line. For example:

```
$ cat ./mpmd_config
-n 1 -host node1 ./io <io_args>
-n 4 -host node2 ./compute <compute_args_1>
-n 4 -host node3 ./compute <compute_args_2>
$ mpirun -configfile mpmd_config
```

Alternatively, you can pass a set of options to the command line by separating each group with a colon:

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

The examples above are equivalent. The `io` program is launched as one process on `node1`, and the `compute` program is launched on `node2` and `node3` as four processes on each.

When an MPI job is launched, the working directory is set to the working directory of the machine where the job is launched. To change this, use the `-wdir <path>`.

Use `-env <var> <value>` to set an environment variable for only one argument set. Using `-genv` instead applies the environment variable to all argument sets. By default, all environment variables are propagated from the environment during the launch.

5.4. Job Schedulers Support

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*
- Univa* Grid Engine*

The support is implemented in the `mpirun` wrapper script. `mpirun` detects the job scheduler under which it is started by checking specific environment variables and then chooses the appropriate method to start an application.

5.4.1. Altair* PBS Pro*, TORQUE*, and OpenPBS*

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

An example of a batch job script may look as follows:

```
#PBS -l nodes=4:ppn=4
#PBS -q queue_name
cd $PBS_O_WORKDIR
mpirun -n 16 ./myprog
```

5.4.2. IBM* Platform LSF*

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

For example, to submit a job, run the command:

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

5.4.3. 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, `/usr/bin/plesh` is used as remote shell by the process manager during startup.

5.4.4. 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.

For example, to submit a job, run the command:

```
$ srun -N2 --nodelist=host1,host2 -A
$ mpirun -n 2 ./myprog
```

When running under SLURM on a large scale, the job startup process make take a considerable amount of time. Use the `I_MPI_SLURM_EXT` variable to reduce the job startup time:

```
$ export I_MPI_SLURM_EXT=on
```

5.4.5. Univa* Grid Engine*

If you use the Univa* 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.

5.4.6. 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 Univa* 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:

1. Use the following command to see available queues:

   ```
   $ qconf -sql
   ```

2. Execute the following command to modify the queue settings:

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

3. Find `terminate_method` and change signal to `SIGTERM`.
4. Save queue configuration.

5.4.7. Controlling Per-Host Process Placement

When using a job scheduler, by default Intel MPI Library uses per-host process placement provided by the scheduler. This means that the `-ppn` option has no effect. To change this behavior and control process placement through `-ppn` (and related options and variables), use the `I_MPI_JOB_RESPECT_PROCESS_PLACEMENT` environment variable:

```
$ export I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=off
```

5.5. Cross-OS Launch Mode

Intel® MPI Library provides support for the heterogeneous Windows*-Linux* environment. This means that you can run MPI programs on nodes that operate on Windows and Linux OS as single MPI jobs, using the Hydra process manager.

To run a mixed Linux-Windows MPI job, do the following:

1. Make sure the Intel MPI Library is installed and operable, and the product versions match on all nodes.
2. On the Windows hosts, make sure the Hydra service is running:

   ```
   > hydra_service -status
   ```

   If the service is not running, use the `-start` option to start it.
3. When running a mixed job, set the following options and environment variables:

- `-demux select`
- `-genv I_MPI_FABRICS=shm:tcp`
- `-host <hostname>`
- `-hostos <os>` for nodes operating on the other OS
- `-env I_MPI_ROOT` and `-env PATH` – local environment variables for the specified host
- (optional) `-bootstrap ssh` – specifies the remote node access mechanism. If the option is not specified, the Hydra service is used. If you specify `ssh`, make sure the SSH connectivity is established between the Linux and Windows nodes.

For example, the following command runs the `IMB-MPI1` benchmark under the Windows-Linux heterogeneous environment:

```
$ mpirun -demux select -genv I_MPI_FABRICS=shm:tcp -env
I_MPI_ROOT=<windows_installdir> \
-env PATH=<windows_installdir>\<arch>\bin -hostos windows -host <windows_host> -n 2
IMB-MPI1 pingpong :\
-host <linux_host> -n 3 <linux_installdir>/<arch>/bin/IMB-MPI1 pingpong
```

6. Configuring Program Launch

6.1. Selecting Library Configuration

Before running an application, you can specify a particular configuration of the Intel® MPI Library to be used, depending on your purposes. This can be a library optimized for single- or multi-threading, debug or release version.

To specify the configuration, source the `mpivars.[c]sh` script with appropriate arguments. For example:

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

You can use the following arguments:

Argument	Definition
`release`	Set this argument to use single-threaded optimized library.
`debug`	Set this argument to use single-threaded debug library.
`release_mt`	Set this argument to use multi-threaded optimized library (default).
`debug_mt`	Set this argument to use multi-threaded debug library.

NOTE

You do not need to recompile the application to change the configuration. Source the `mpivars.[c]sh` script with appropriate arguments before an application launch.

Alternatively, if your shell does not support sourcing with arguments, you can use the `I_MPI_LIBRARY_KIND` environment variable to set an argument for `mpivars.[c]sh`. See the *Intel® MPI Library Developer Reference* for details.

6.2. Controlling Process Placement

Placement of MPI processes over the cluster nodes plays a significant role in application performance. Intel® MPI Library provides several options to control process placement.

By default, when you run an MPI program, the process manager launches all MPI processes specified with `-n` on the current node. If you use a job scheduler, processes are assigned according to the information received from the scheduler.

6.2.1. Specifying Hosts

You can explicitly specify the nodes on which you want to run the application using the `-hosts` option. This option takes a comma-separated list of node names as an argument. Use the `-ppn` option to specify the number of processes per node. For example:

```
$ mpirun -n 4 -ppn 2 -hosts node1,node2 ./testc
Hello world: rank 0 of 4 running on node1
Hello world: rank 1 of 4 running on node1
```

```
Hello world: rank 2 of 4 running on node2
Hello world: rank 3 of 4 running on node2
```

To get the name of a node, use the `hostname` utility.

An alternative to using the `-hosts` option is creation of a host file that lists the cluster nodes. The format of the file is one name per line, and the lines starting with `#` are ignored. Use the `-f` option to pass the file to `mpirun`. For example:

```
$ cat ./hosts
#nodes
node1
node2
$ mpirun -n 4 -ppn 2 -f hosts ./testc
```

This program launch produces the same output as the previous example.

If the `-ppn` option is not specified, the process manager assigns as many processes to the first node as there are physical cores on it. Then the next node is used. That is, assuming there are four cores on `node1` and you launch six processes overall, four processes are launched on `node1`, and the remaining two processes are launched on `node2`. For example:

```
$ mpirun -n 6 -hosts node1,node2 ./testc
Hello world: rank 0 of 6 running on node1
Hello world: rank 1 of 6 running on node1
Hello world: rank 2 of 6 running on node1
Hello world: rank 3 of 6 running on node1
Hello world: rank 4 of 6 running on node2
Hello world: rank 5 of 6 running on node2
```

NOTE

If you use a job scheduler, specifying hosts is unnecessary. The processes manager uses the host list provided by the scheduler.

6.2.2. Using Machine File

A machine file is similar to a host file with the only difference that you can assign a specific number of processes to particular nodes directly in the file. Contents of a sample machine file may look as follows:

```
$ cat ./machines
node1:2
node2:2
```

Specify the file with the `-machine` option. Running a simple test program produces the following output:

```
$ mpirun -machine machines ./testc
Hello world: rank 0 of 4 running on node1
Hello world: rank 1 of 4 running on node1
Hello world: rank 2 of 4 running on node2
Hello world: rank 3 of 4 running on node2
```

6.2.3. Using Argument Sets

Argument sets are unique groups of arguments specific to a particular node. Combined together, the argument sets make up a single MPI job. You can provide argument sets on the command line, or in a configuration file. To specify a node, use the `-host` option.

On the command line, argument sets should be separated by a colon ':'. Global options (applied to all argument sets) should appear first, and local options (applied only to the current argument set) should be specified within an argument set. For example:

```
$ mpirun -genv I_MPI_DEBUG=2 -host node1 -n 2 ./testc : -host node2 -n 2 ./testc
```

In the configuration file, each argument set should appear on a new line. Global options should appear on the first line of the file. For example:

```
$ cat ./config
-genv I_MPI_DEBUG=2
-host node1 -n 2 ./testc
-host node2 -n 2 ./testc
```

Specify the configuration file with the `-configfile` option:

```
$ mpirun -configfile config
Hello world: rank 0 of 4 running on node1
Hello world: rank 1 of 4 running on node1
Hello world: rank 2 of 4 running on node2
Hello world: rank 3 of 4 running on node2
```

See Also

Controlling Process Placement with the Intel® MPI Library (online article)
Job Schedulers Support

6.3. Selecting Fabrics

Intel® MPI Library enables you to select a communication fabric at runtime without having to recompile your application. By default, it automatically selects the most appropriate fabric based on your software and hardware configuration. This means that in most cases you do not have to bother about manually selecting a fabric.

However, in certain situations specifying a particular communication fabric can boost performance of your application. You can specify fabrics for communications within the node and between the nodes (intra-node and inter-node communications, respectively). The following fabrics are available:

Fabric	Network hardware and software used
`shm`	Shared memory (for intra-node communication only).
`dapl`	Direct Access Programming Library* (DAPL)-capable network fabrics, such as InfiniBand* and iWarp* (through DAPL).
`tcp`	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand* (through IPoIB*).
`tmi`	Tag Matching Interface (TMI)-capable network fabrics, such as Intel® True Scale Fabric and Myrinet* (through TMI).
`ofa`	OpenFabrics Alliance* (OFA)-capable network fabrics, such as InfiniBand* (through OFED* verbs).
`ofi`	OpenFabrics Interfaces* (OFI)-capable network fabrics, such as Intel® True Scale Fabric and Ethernet (through OFI API).

Use the `I_MPI_FABRICS` environment variable to specify a fabric. Additional environment variables for controlling fabric selection are `I_MPI_FABRICS_LIST` and `I_MPI_FALLBACK`. Their description is available in the *Developer Reference*, section *Tuning Reference > Fabrics Control*.

See below for more information.

6.3.1. Default Behavior

For intra-node communication, Intel® MPI Library uses shared memory by default. For inter-node communication, it uses the first available fabric from the default fabric list. This list is defined automatically for each hardware and software configuration (see `I_MPI_FABRICS_LIST` for details).

For most configurations, this list is as follows:

```
dapl,ofa,tcp,tmi,ofi
```

If the first fabric is unavailable, Intel® MPI Library falls back to the second fabric from the list. In case the second fabric is unavailable, it attempts to use the next one, and so on.

To check which fabric is currently used, you can set the `I_MPI_DEBUG` environment variable to `2`:

```
$ mpirun -n 4 -ppn 2 -hosts <node1>,<node2> -genv I_MPI_DEBUG=2 ./myprog
...
MPI startup(): shm and dapl data transfer modes
```

6.3.2. Defining Fabric List

You can redefine the default fabric list using the `I_MPI_FABRICS_LIST` environment variable. When you redefine the list, other rules are unchanged: shared memory is used for intra-node communication, and the fallback is enabled.

For example, if you run the following commands:

```
$ export I_MPI_FABRICS_LIST=ofi,tcp
$ mpirun -n 4 -ppn 2 -hosts <node1>,<node2> ./myprog
```

and the `ofi` fabric is unavailable, the `tcp` fabric is used instead.

You can disable this fallback behavior by setting the `I_MPI_FALLBACK` environment variable to `0`:

```
$ export I_MPI_FALLBACK=0
```

In this case, if you run the application again, and the `ofi` fabric is still unavailable, the application will be terminated.

To simplify definition of the fabric list, Intel® MPI Library also provides special runtime options that set predefined fabric lists and enable (lowercase options) or disable (UPPERCASE options) the fallback behavior. Description of these options is available in the *Developer Reference*, section *Command Reference > Hydra Process Manager Command > Extended Fabric Control Options*.

6.3.3. Selecting Specific Fabric

You can also select a specific fabric for the intra-node and inter-node communications using the `I_MPI_FABRICS` environment variable.

The variable syntax is as follows:

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

If you specify only `<fabric>`, it is used for both intra-node and inter-node communications.

For example, if you run the following commands:

```
$ export I_MPI_FABRICS=shm:tcp
$ mpirun -n 4 -ppn 2 -hosts <node1>,<node2> ./myprog
```

shared memory is used for communication within a node, and the `tcp` fabric is used for communication between the nodes.

If you set `I_MPI_FABRICS` as follows:

```
$ export I_MPI_FABRICS=tcp
```

the `tcp` fabric used for communication within and between the nodes.

If, however, the specified fabric is unavailable, the application will be terminated. You can avoid this situation by enabling the fallback behavior:

```
$ export I_MPI_FALLBACK=1
```

In this case, if a fabric is unavailable, the next available fabric from the fabric list (default or manually defined) is used.

6.3.4. Advanced Fabric Control

Each of the available fabrics has its advanced parameters, which you can adjust according to your needs. The detailed description of the environment variables controlling these parameters is available in the *Developer Reference*, section *Tuning Reference > Fabrics Control*.

NOTE

These parameters are intended for experienced users, and under normal circumstances, you are not recommended to adjust them manually.

---

7. Debugging

Intel® MPI Library supports the following debuggers for debugging MPI applications: GDB*, TotalView*, and Allinea* DDT. Before using a debugger, make sure you have the application debug symbols available. To generate debug symbols, compile your application with the `-g` option.

This section explains how to debug MPI applications using the listed debugger tools.

7.1. 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 on 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 process.

7.2. TotalView* Debugger

Intel® MPI Library supports the use of the TotalView* debugger from Rogue Wave* Software, Inc. To debug an MPI program, use `-tv` option of `mpirun`. For example:

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

You will get a popup window from TotalView asking whether you want to start the job in a stopped state. Click **Yes**, and the TotalView window will appear with the source code window open. Click on the `main` function in the **Stack Trace** window (upper left) to see the source of the `main` function. TotalView shows that the program (all processes) is stopped in the call to `MPI_Init()`. To start debugging, use buttons on the toolbar. Refer to the TotalView documentation for more details.

7.2.1. Enabling Debug Session Restart

When using the above scenario, you need to exit TotalView to restart a debugging session. To be able to restart the session within TotalView, use the following command line syntax:

```
$ totalview mpiexec.hydra -a -n <# of processes> [<other mpiexec.hydra arguments>]
<executable>
```

In the appeared window, verify your parameters and click **OK**. Start debugging with the **Go** button on the toolbar. TotalView will prompt you to stop the `mpiexec.hydra` job. Click **Yes** to continue.

7.2.2. Attaching to Process

To attach to a running MPI job using TotalView, use the `-tva` option of `mpirun` when starting the job, as follows:

```
$ mpirun -tva <mpiexec.hydra process ID>
```

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.

7.2.3. Displaying Message Queue

TotalView can also display the message queue state of your MPI program. This feature is available only with the debug (single- or multi-threaded) Intel MPI Library configuration. To view the message queue, do the following:

1. Switch to the `debug` or `debug_mt` configuration using the `mpivars.[c]sh` script:
   ```
   $ . mpivars.sh debug_mt
   ```
2. Run your program with the `-tv` option:
   ```
   $ mpirun -tv -n <# of processes> <executable>
   ```
3. Answer **Yes** to the question about stopping the `mpiexec.hydra` job.
4. To view the message queue, go to **Tools > Message Queue**.

For more details on using TotalView with MPI, see the official TotalView documentation.

7.3. DDT* Debugger

You can debug MPI applications using the Allinea* DDT* debugger. Intel does not provide support for this debugger, you should obtain the support from Allinea*. According to the DDT documentation, DDT supports the Express Launch feature for the Intel® MPI Library. You can debug your application as follows:

```
$ ddt mpirun -n <# of processes> [<other mpirun arguments>] <executable>
```

If you have issues with the DDT debugger, refer to the DDT documentation for help.

7.4. Using -gtool for Debugging

The `-gtool` runtime option can help you with debugging, when attaching to several processes at once. Instead of attaching to each process individually, you can specify all the processes in a single command line. For example:

```
$ mpirun -n 16 -gtool "gdb:3,5,7-9=attach" ./myprog
```

The command line above attaches the GNU* Debugger (GDB*) to processes 3, 5, 7, 8 and 9.

The `-gtool` option supports any debuggers including debuggers for the Intel® Xeon Phi™ coprocessors. For the processes residing on the coprocessor, specify the `@<arch>` parameter (see `I_MPI_PLATFORM` for available values). For example:

```
$ mpirun -gtool "gdb-ia:all=attach@generic; /tmp/gdb:all=attach@knc" -host
<hostname> -n 8 <host-app> :\
-host <hostname-mic0> -n 8 <mic-app>
```

In this example, the Intel version of GDB (`gdb-ia`) is launched for all `<hostname>` processes, while the native GNU* Debugger for Intel® Xeon Phi™ coprocessor is applied to all processes residing on the coprocessor. The value `generic` stands for all platforms except for the Intel® MIC Architecture.

See Also

Intel® Many Integrated Core Architecture Support
Intel® MPI Library Developer Reference, section *Miscellaneous > Other Environment Variables > I_MPI_PLATFORM*
Intel® MPI Library Developer Reference, section *Command Reference > Hydra Process Manager Command > Global Options > gtool Options*

8. Statistics and Analysis

Intel® MPI Library provides a variety of options for analyzing MPI applications. Some of these options are available within the Intel MPI Library, while some require additional analysis tools. For such tools, Intel MPI Library provides compilation and runtime options and environment variables for easier interoperability.

8.1. Getting Debug Information

The `I_MPI_DEBUG` environment variable provides a convenient way to get detailed information about an MPI application at runtime. You can set the variable value from 0 (the default value) to 1000. The higher the value, the more debug information you get. For example:

```
$ mpirun -genv I_MPI_DEBUG=2 -n 2 ./testc
[0] MPI startup(): Multi-threaded optimized library
[0] MPI startup(): shm data transfer mode
[1] MPI startup(): shm data transfer mode
[1] MPI startup(): Internal info: pinning initialization was done
[0] MPI startup(): Internal info: pinning initialization was done
...
```

NOTE

High values of `I_MPI_DEBUG` can output a lot of information and significantly reduce performance of your application. A value of `I_MPI_DEBUG=5` is generally a good starting point, which provides sufficient information to find common errors.

By default, each printed line contains the MPI rank number and the message. You can also print additional information in front of each message, like process ID, time, host name and other information, or exclude some information printed by default. You can do this in two ways:

- Add the '+' sign in front of the debug level number. In this case, each line is prefixed by the string `<rank>#<pid>@<hostname>`. For example:

  ```
  $ mpirun -genv I_MPI_DEBUG=+2 -n 2 ./testc
  [0#3520@clusternode1] MPI startup(): Multi-threaded optimized library
  ...
  ```

  To exclude any information printed in front of the message, add the '–' sign in a similar manner.
- Add the appropriate flag after the debug level number to include or exclude some information. For example, to include time but exclude the rank number:

  ```
  $ mpirun -genv I_MPI_DEBUG=2,time,norank -n 2 ./testc
  11:59:59 MPI startup(): Multi-threaded optimized library
  ...
  ```

  For the list of all available flags, see description of `I_MPI_DEBUG` in the *Developer Reference*.

To redirect the debug information output from `stdout` to `stderr` or a text file, use the `I_MPI_DEBUG_OUTPUT` environment variable:

```
$ mpirun -genv I_MPI_DEBUG=2 -genv I_MPI_DEBUG_OUTPUT=/tmp/debug_output.txt -n 2
./testc
```

Note that the output file name should not be longer than 256 symbols.

See Also

Intel® MPI Library Developer Reference, section *Miscellaneous > Other Environment Variables > I_MPI_DEBUG*

8.2. Gathering Statistics

Intel® MPI Library has a built-in statistics gathering facility that collects essential performance data without disturbing the application execution. You do not need to modify the source code or relink your application to collect this information. The statistics information is stored in a text file and has a human-readable format.

There are two types of statistics that the Intel MPI Library can collect:

- Native statistics
- IPM statistics

You can also collect both types of statistics simultaneously. To enable statistics collection, use the `I_MPI_STATS` environment variable for either of the types. There are also additional variables and MPI functions that help you customize the statistics collection. See below for details.

8.2.1. Native Statistics

Native statistics provides detailed information about the application, which includes information about data transfers, point-to-point and collective operations on a rank-by-rank basis.

To enable the native statistics collection, set the `I_MPI_STATS` environment variable to a numeric value to specify the level of detail. The available values are `1`, `2`, `3`, `4`, `10` or `20`. For example:

```
$ export I_MPI_STATS=10
```

You can also specify a range of levels to collect information only on those levels. For example:

```
$ export I_MPI_STATS=4-10
```

After running the application, you will get a `stats.txt` file containing the statistics. To change the output file name, use the `I_MPI_STATS_FILE` variable:

```
$ export I_MPI_STATS_FILE=stats_initial.txt
```

You can limit the amount of output information by collecting information for specific MPI operations or operation types. To do so, use the `I_MPI_STATS_SCOPE` variable. For collective and point-to-point operations, use the `p2p` and `coll` values, respectively. For specific operations, use their names without the `MPI_` prefix. By default, statistics are collected for all MPI operations. For the full list of supported operations, see the *Developer Reference*.

For example, to collect statistics for `MPI_Bcast`, `MPI_Reduce`, and all point-to-point operations:

```
$ export I_MPI_STATS=20
$ export I_MPI_STATS_SCOPE="p2p;coll:bcast,reduce"
```

Use the `I_MPI_STATS_BUCKETS` environment variable to collect statistics for specific ranges of message sizes and communicator sizes. For example, to specify short messages (from 0 to 1000 bytes) and long messages (from 50000 to 100000 bytes), use the following setting:

```
$ export I_MPI_STATS_BUCKETS=0-1000,50000-100000
```

To specify messages that have 16 bytes in size and circulate within four process communicators, use the following setting:

```
$ export I_MPI_STATS_BUCKETS="16@4"
```

8.2.2. IPM Statistics

Intel® MPI Library also provides the integrated performance monitoring (IPM) statistics format, which provides similar information about the application and has two levels of detail. This information, however, is less detailed overall than the native statistics, but it is considered more portable.

To enable the IPM statistics collection, set `I_MPI_STATS` to `ipm:terse` for a brief summary or to `ipm` for complete statistics information. For example:

```
$ export I_MPI_STATS=ipm
```

After running the application, the statistics are saved in the `stats.ipm` file. Use the `I_MPI_STATS_FILE` variable to change the output file name.

In the IPM statistics mode, the `I_MPI_STATS_SCOPE` variable has an extended list of values. For the full list of available values and its descriptions, see the *Developer Reference*.

The IPM statistics mode also provides the `I_MPI_STATS_ACCURACY` environment variable to reduce the statistics output. Set the variable to collect data only on those MPI functions that take the specified portion of the total time spent inside all MPI calls (in percent). For example, to skip all operations that are run less than 3% of all time:

```
$ export I_MPI_STATS=ipm
$ export I_MPI_STATS_ACCURACY=3
```

8.2.3. Native and IPM Statistics

It is also possible to collect both types of statistics simultaneously. To collect statistics in all formats with the maximal level of details, use the `I_MPI_STATS` environment variable as follows:

```
$ export I_MPI_STATS=all
```

NOTE

The `I_MPI_STATS_SCOPE` environment variable is not applicable when both types of statistics are collected.

The value `all` corresponds to `I_MPI_STATS=native:20,ipm`. To control the amount of statistics information, use the ordinary `I_MPI_STATS` values, separated by commas:

```
$ export I_MPI_STATS=native:2-10,ipm:terse
```

8.2.4. Region Control with MPI_Pcontrol

You can mark regions of code to collect statistics specifically for those regions.

To open a region, use the `MPI_Pcontrol(1, <name>)` function call. To close a region, use the `MPI_Pcontrol(-1, <name>)` function call. The `<name>` function argument is a string with the region name. For example:

```
...
/* open "reduce" region for all processes */
MPI_Pcontrol(1, "reduce");
for (i = 0; i < 1000; i++)
    MPI_Reduce(&nsend, &nrecv, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
/* close "reduce" region */
MPI_Pcontrol(-1, "reduce");
...
```

For the native statistics type, statistics for the region is saved in a separate file `stats_<name>.txt`. In the IPM statistics mode, it is marked appropriately in the output file:

```
...
############################################################
# region : reduce [ntasks] = 4
#
...
```

Code regions can be:

- Discontiguous (opened and closed multiple times)
- Intersected
- Covering a subset of MPI processes (in the `MPI_COMM_WORLD` communicator)

Each region contains its own independent statistics information about MPI functions called inside the region. All open regions are closed automatically inside the `MPI_Finalize` function call.

The `MPI_Pcontrol` function cannot be used for the following permanent regions:

- Main region – contains statistics information about all MPI calls from `MPI_Init` to `MPI_Finalize`. The main region gets the "`*`" name in the IPM statistics output. The native statistics output for this region is `stats.txt` by default.
- Complementary region – contains statistics information not included into any named region. The region gets the "`ipm_noregion`" name in the IPM statistics output. The default output file for this region is `stats_noregion.txt` for the native statistics format.

If named regions are not used, the main and complementary regions are identical, and the complementary region is ignored.

See Also

Intel® MPI Library Developer Reference, section *Miscellaneous > Statistics Gathering Mode*

8.3. Tracing and Correctness Checking

Intel® MPI Library provides tight integration with the Intel® Trace Analyzer and Collector, which enables you to analyze MPI applications and find errors in them. Intel® MPI Library has several compile- and runtime options to simplify the application analysis.

Intel® Trace Analyzer and Collector is available as part of the Intel® Parallel Studio XE Cluster Edition. Before proceeding to the next steps, make sure you have this product installed.

8.3.1. High-Level Performance Analysis

For a high-level application analysis, Intel Trace Analyzer and Collector provides a lightweight analysis tool MPI Performance Snapshot, which is also integrated with the Intel MPI Library. The tool provides general information about the application, such as MPI and OpenMP* utilization time and load balance, MPI operations usage, memory and disk usage, and other information. This information enables you to get a general idea about the application performance and identify spots for a more thorough analysis.

Follow these steps to analyze an application with the MPI Performance Snapshot:

1. Set up the environment for the compiler, Intel MPI Library and MPI Performance Snapshot:

   ```
   $ source <installdir>/bin/compilervars.sh intel64
   $ source <installdir>/bin/mpivars.sh
   $ source <installdir>/bin/mpsvars.sh
   ```

2. Run your application with the `-mps` option:

   ```
   $ mpirun -mps -n 4 ./myprog
   ```

   MPI Performance Snapshot generates a directory with the statistics files `stat_<date>-<time>`.

3. Launch the `mps` tool and pass the generated statistics to the tool:

   ```
   $ mps ./stat_<date>-<time>
   ```

   You will see the analysis results printed in the console window. Also, MPI Performance Snapshot will generate an HTML report `mps_report.html` containing the same information.

For more details, refer to the *MPI Performance Snapshot User's Guide*.

8.3.2. Tracing Applications

To analyze an application with the Intel Trace Analyzer and Collector, first you need generate a trace file of your application, and then open this file in Intel® Trace Analyzer to analyze communication patterns, time utilization, etc. Tracing is performed by preloading the Intel® Trace Collector profiling library at runtime, which intercepts all MPI calls and generates a trace file. Intel MPI Library provides the `-trace` (`-t`) option to simplify this process.

Complete the following steps:

1. Set up the environment for the Intel MPI Library and Intel Trace Analyzer and Collector:

   ```
   $ source <installdir>/bin/mpivars.sh
   $ source <installdir>/bin/itacvars.sh
   ```

2. Trace your application with the Intel Trace Collector:

   ```
   $ mpirun -trace -n 4 ./myprog
   ```

   As a result, a trace file `.stf` is generated. For the example above, it is `myprog.stf`.

3. Analyze the application with the Intel Trace Analyzer:

   ```
   $ traceanalyzer ./myprog.stf &
   ```

The workflow above is the most common scenario of tracing with the Intel Trace Collector. For other tracing scenarios, see the Intel Trace Collector documentation.

8.3.3. Checking Correctness

Apart from regular tracing, Intel Trace Analyzer and Collector provides the correctness checking capability, which helps you detect errors with data types, buffers, communicators, point-to-point messages and collective operations, detect deadlocks, or data corruption.

To enable correctness checking, use the `-check_mpi` runtime option. If errors are detected, they are printed in the console output. To map errors to the source code, make sure you have the debug information available (`-g` compiler option). For example:

```
$ mpirun -check_mpi -n 4 ./myprog
...
[0] ERROR: GLOBAL:COLLECTIVE:DATATYPE:MISMATCH: error
...
```

For details on configuring error detection, refer to the Intel Trace Collector documentation.

See Also

MPI Performance Snapshot User's Guide
Intel® Trace Collector User and Reference Guide
Tutorial: Detecting and Resolving Errors with MPI Correctness Checker

8.4. Interoperability with Other Tools

To simplify interoperability with other analysis tools, Intel® MPI Library provides the `-gtool` option and the `I_MPI_GTOOL` environment variable. The option and the variable are equivalent but the option has higher priority. By using the `-gtool` option you can analyze specific MPI processes with such tools as Intel® VTune™ Amplifier XE, Intel® Advisor, Valgrind* and others through the `mpiexec.hydra` or `mpirun` commands.

For example, to analyze processes 3, 5, 6, and 7 with the VTune Amplifier, you can use the following command line:

```
$ mpirun -n 8 -gtool "amplxe-cl -collect hotspots -r result:3,5-7" ./myprog
```

The `-gtool` option also provides several methods for finer process selection. For example, you can easily analyze only one process on each host, using the `exclusive` launch mode:

```
$ mpirun -n 8 -ppn 4 -hosts node1,node2 -gtool "amplxe-cl -collect hotspots -r
result:all=exclusive" ./myprog
```

Besides, you can select processes by the architecture of the processor they are residing on:

```
$ mpirun -n 4 -host node1 -gtool "amplxe-cl -collect hotspots -r result:all@knc"
./myprog :\
-n 4 -host node1-mic0 ./myprog-mic
```

See `I_MPI_PLATFORM` for the available architectures.

For more advanced examples, see the `-gtool` option description in the *Developer Reference*.

See Also

Intel® MPI Library Developer Reference, section *Miscellaneous > Other Environment Variables > I_MPI_PLATFORM*
Intel® MPI Library Developer Reference, section *Command Reference > Hydra Process Manager Command > Global Options > gtool Options*

9. Tuning with mpitune Utility

Besides the standard capabilities for compiling and running MPI applications, Intel® MPI Library provides an automatic utility for optimizing Intel MPI Library parameters for a particular cluster or application. This utility is called `mpitune` and is available in the `<installdir>/bin` directory.

Intel® MPI Library has a wide variety of parameters affecting application performance. The defaults for those parameters are set for common usage and generally provide good performance for most clusters and applications out of the box. However, to achieve the best performance for your particular cluster or application, you can use the `mpitune` utility to adjust the parameters. Note that if performance improvement is insignificant after adjusting a certain parameter, `mpitune` can keep its default value.

The following operation modes are available in `mpitune`:

- **Cluster specific.** In this mode, `mpitune` evaluates a given cluster environment by running a benchmarking program to find the most optimal settings. Intel® MPI Benchmarks are used by default.
- **Application specific.** In this mode, `mpitune` evaluates performance of a given MPI application by running the actual application.
  - **Fast application specific.** A faster application specific mode. Less accurate than the previous, but more suitable in certain cases.
  - **Topology-aware rank placement optimization.** In this mode, `mpitune` evaluates cluster characteristics and rank-to-rank data transfers in an MPI application to identify the optimal rank placement.

The general workflow for all modes is as follows:

1. Generate a configuration file containing the optimal settings for the cluster or application.
2. Apply the settings during the application launch.

See the information below for details. For advanced usage scenarios, see the related online tutorial.

9.1. Cluster Specific Tuning

The cluster specific mode is intended for tuning Intel® MPI Library for a specific cluster. For finding optimal settings, `mpitune` uses a benchmarking program: it runs tests several times with different parameters and searches for the best ones. By default, Intel® MPI Benchmarks are used.

To tune Intel MPI Library in the cluster specific mode, do the following:

1. Start the tuning process on a cluster, run the `mpitune` command. Use the `--hostfile` (`-hf`) option to specify the host file. Otherwise, the local host is used.

   ```
   $ mpitune -hf hosts
   ```

   As a result, a configuration file with the optimized Intel MPI Library settings is created in the `<installdir>/<arch>/etc` directory. If you do not have write permissions for this directory, the file is created in the current location.

   To save the file to an alternative directory, use the `--output-directory-results` (`-odr`) option:

   ```
   $ mpitune -hf hosts -odr <results_directory>
   ```

2. Run your application with the `-tune` option to apply the tuned settings:

   ```
   $ mpirun -tune -genv I_MPI_FABRICS=<value> -ppn 8 -n 64 ./myprog
   ```

   If you generated results in a non-default directory, pass this directory name to the `-tune` option:

```
$ mpirun -tune <results_directory> -genv I_MPI_FABRICS=<value> -ppn 8 -n 64
./myprog
```

NOTE

When you use the `-tune` option, explicitly specify the fabric, the number of processes per node, and the total number of processes, as shown in the examples above. This makes the Intel® MPI Library pick up the appropriate configuration file.

Alternatively, you can pass a specific configuration file to `-tune`, for example:

```
$ mpirun -tune <path_to_your_config_file>/mpitune_config.conf
```

9.1.1. Reducing Tuning Time

A standard tuning procedure may take a considerable amount of time and it may be reasonable to limit its execution. There are a few options that help you reduce cluster tuning time.

To do that, you need to know about the most common MPI workloads on your cluster and perform tuning specifically for those. The following information can help you reduce the amount of adjustments made by `mpitune` and hence the tuning time:

- Fabrics used
- Number of hosts typically used
- Numbers of ranks per host
- Common message sizes

Tuning for Specific Fabrics

Use the `--fabric-list` (`-fl`) option. Use values from `I_MPI_FABRICS` as arguments. Use commas to separate different fabrics:

```
$ mpitune -fl shm:dapl,ofa <other options>
```

`mpitune` creates configuration files for the specified fabrics (fabric combinations). By default, all fabrics from *`<installdir>/<arch>`*`/etc/fabrics.xml` are tested.

Tuning for Specific Host Ranges

Assume the majority of workloads on the cluster use between 4 and 16 nodes. In this case, you may want to create tuned settings only for those numbers. To do so, use the `--host-range` (`-hr`) option:

```
$ mpitune -hr 4:16 <other options>
```

`mpitune` creates configuration files for workloads using 4, 8, and 16 nodes (`mpitune` tests only the values that are powers of two in the specified range). The default minimum is 1, and the default maximum is the number of hosts specified in the host file.

For this option and the options below, you can specify only the minimum or the maximum value as follows: `min:` or `:max`. In this case, the default value is used for the other bound.

Tuning for Specific Ranks per Host

Use the `--perhost-range` (`-pr`) option:

```
$ mpitune -pr 4:16 <other options>
```

Similarly, `mpitune` creates configuration files for workloads running 4, 8, and 16 processes per host. The default minimum is 1, and the default maximum is the number of cores.

Tuning for Specific Message Sizes

Use the `--message-range` (`-mr`) option. You can specify message sizes in the following formats: `1024` (bytes), `2kb`, `4mb`, or `8gb`. For example:

```
$ mpitune -mr 64kb:1mb <other options>
```

`mpitune` creates configuration files for messages within the specified range (for each value that is a power of two in the range). The default range is `0:4mb`.

See Also

Selecting Fabrics

9.1.2. Replacing Default Benchmarks

To replace the default Intel® MPI Benchmarks, use the `--test` (`-t`) option with the benchmark of your choice:

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

You can then perform tuning as described above.

Note that Intel® MPI Benchmarks are more optimized for Intel microprocessors. This may result in different tuning settings on Intel and non-Intel microprocessors.

9.2. Application Specific Tuning

In the application specific mode, you can find optimal Intel® MPI Library settings for your specific application. To enable this mode, use the `--application` (`-a`) option of `mpitune`. The option takes the full program launch command as an argument. The general syntax is as follows:

```
$ mpitune --application \"mpirun <arguments> <executable>\" --output-file
<config_file>.conf
```

After launching the command, `mpitune` performs tuning and saves the optimized settings in the `<config_file>.conf` file.

To apply the optimized settings, run your application and pass the generated `.conf` file to the `-tune` option. For example:

```
$ mpirun -tune mpitune_config.conf -ppn 8 -n 64 ./myprog
```

9.2.1. Fast Application Specific Mode

In addition to the application specific mode, `mpitune` provides its faster alternative, which performs tuning of collective operations. This type of tuning may produce less accurate results but it can speedup applications with non-typical patterns, such as non-typical rank placement per host or communicator, unusual messages sizes, and so on. This mode supports only regular collective operations that have related environment variables in the `I_MPI_ADJUST` family (except for the `KN_RADIX` subset and `I_MPI_ADJUST_REDUCE_SEGMENT`). See the *Intel® MPI Library Developer Reference* for details.

To enable the fast application specific mode, use the `--fast` (`-f`) and `--application` (`-a`) options. Use the `--out` option to specify the output file:

```
$ mpitune --fast --application \"mpirun <arguments> <executable>\" --out
<config_file>.conf
```

This mode uses the internal Intel MPI Library statistics as input. If you have a generated statistics file for your application, you can pass it directly to `mpitune`. Use the `--file` option:

```
$ mpitune --fast --file stats.txt --out <config_file>.conf
```

This can help reduce the tuning time.

TIP

To see other options available in this mode, use the `--fast` and `--help` options together.

To apply the resulting settings, use the `-tune` option as described above.

9.2.2. Topology-Aware Rank Placement Optimization

In this mode, `mpitune` finds the most optimal placement of MPI ranks on the cluster to minimize network traffic. It takes into account the network topology and rank-to-rank data transfers. To enable this mode, use the `--rank-placement` (`-rp`) option.

This mode uses a host file as input. In the host file, a specific cluster node should be specified as many times as many processes are run on it. For example, for two nodes with two processes on each, the host file should look as follows:

```
$ cat ./hostfile.in
node1
node1
node2
node2
```

To pass the host file to `mpitune`, use the `--hostfile-in` (`-hi`) option. For getting tuning results, `mpitune` provides two options: a standard configuration file (`--config-out` option) or an optimized host file (`--hostfile-out` or `-ho` option). You can use either of the options.

Using Configuration File

Do the following:

1. Perform tuning for your application and save results in a configuration file. An example command line may look as follows:

   ```
   $ mpitune --rank-placement --application \"mpirun -n 32 ./myprog"\ --
   hostfile-in hostfile.in --config-out ./myprog.conf
   ```

2. Apply the tuned settings to your application:

   ```
   $ mpirun -tune ./myprog.conf -n 32 ./myprog
   ```

Using Optimized Host File

Do the following:

1. Perform tuning for your application and save results in a host file. An example command line may look as follows:

   ```
   $ mpitune --rank-placement --application \"mpirun -n 32 ./myprog"\ --
   hostfile-in hostfile.in --hostfile-out ./hostfile.out
   ```

2. Use the optimized host file for your application launch. Use the `-machinefile` option:

   ```
   $ mpirun -machinefile ./hostfile.out -n 32 ./myprog
   ```

TIP

To see other options available in this mode, use the `--rank-placement` and `--help` options together.

Reducing Tuning Time

When you perform this type of tuning, `mpitune` generates three files, which it uses as input:

- Intel MPI Library statistics file
- Application communication graph (ACG) file
- Hardware topology graph (HTG) file

The files are saved into a temporary directory. Their exact location and names are printed in the console output when you start tuning:

```
Statistics collecting... done: <statistics file location>
Application communication graph initialization... done: <ACG file location>
Hardware topology initialization... done: <HTG file location>
```

To speedup future tuning sessions, you can skip generation of these files and use the files from the previous tuning session. Use the `-s`, `-acg`, and `-htg` options to pass the statistics file, ACG and HTG files, respectively.

NOTE

Specifying a statistics file or an ACG file requires that you also specify a host file (`--hostfile-in` or `-hi` option) or an HTG file.

Specifying an HTG file requires that you also specify one of the following: an ACG file, an application command line (`--application` or `-a` option), or a statistics file.

For example, to use a previously generated ACG file:

```
$ mpitune -rp -acg acg.txt -hi hosts.in
```

To use a previously generated HTG file:

```
$ mpitune -rp -htg htg.txt -s stats.txt
```

See Also

Intel® MPI Library Developer Reference, section *Tuning Reference* > *Collective Operation Control*
Intel® MPI Library Developer Reference, section *Tuning Reference* > *`mpitune` Utility* > *`mpitune` Environment Variables*

9.3. General mpitune Capabilities

To get an idea about all `mpitune` capabilities, you can review its options, which are available by specifying the `--help` (`-h`) option:

```
$ mpitune --help
```

NOTE

Executing `mpitune` without arguments initiates the tuning process. To print help, you need to use the `--help` (`-h`) option.

The full list of `mpitune` options and related environment variables is also available in the *Intel® MPI Library Developer Reference*.

Some specific scenarios of using `mpitune` options are given below.

9.3.1. Displaying Tasks Involved

Before performing actual tuning, you may want to know which tasks exactly the tuning process involves. For that purpose, you can use the `--scheduler-only` (`-so`) option:

```
$ mpitune -so
```

9.3.2. Enabling Silent Mode

During the tuning process, `mpitune` produces a lot of output in `stdout`. To suppress all diagnostics shown by the tool, use the `--silent` (`-s`) option:

```
$ mpitune -s <other options>
```

9.3.3. 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` (`-tl`) option. This option takes a number of minutes as an argument. For example, to limit tuning to 8 hours (480 minutes), run the following command:

```
$ mpitune --time-limit 480 <other options>
```

9.3.4. Improving Accuracy of Results

In the case of tuning an application that has significant run-to-run performance variation, `mpitune` might not be able to select the best parameters due to a small number of test iterations. By default, this number is `3`. To improve accuracy of resulting settings, increase the number of iterations for each test run, use the `--iterations` (`-i`) option:

```
$ mpitune --iterations 10 <other options>
```

9.4. Tuning Applications Manually

Intel® MPI Library provides a family of `I_MPI_ADJUST_*` environment variables that allow you to manually tune collective MPI operations. By setting a range of message sizes and choosing different algorithms, you can improve performance of your application. For more information, see the *Intel® MPI Library Developer Reference*, section *Tuning Reference* > *Collective Operations Control*.

10. Intel® Many Integrated Core Architecture Support

Intel® MPI Library provides full support for processors based on the Intel® Many Integrated Core (Intel® MIC) Architecture. Using the Intel® MPI Library with an Intel® Xeon Phi™ card is similar to using another node, but there are a few special considerations, which are discussed in this section.

NOTE

Information in this section applies only to the Intel® Xeon Phi™ processors code named Knights Corner. For newer Intel® Xeon Phi™ generations, use the standard usage workflow.

10.1. Prerequisites

Complete the following prerequisite steps to start with the Intel MPI Library on Intel MIC Architecture:

1. Make sure the Intel® Manycore Platform Software Stack (Intel® MPSS) is installed and configured on your heterogeneous system.
2. Make sure the passwordless SSH connection is established between the Intel Xeon host and the Intel Xeon Phi coprocessor.
3. Set up the Intel MPI Library environment on the host:

   ```
   (host)$ source <installdir>/bin/mpivars.sh
   ```
4. The Intel® MIC Architecture uses different binaries and libraries, and they must be present on the card or accessible via an NFS share.

   To copy the appropriate files to the card, you can use the following commands:

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

   This assumes that the host name of the card is `mic0`. You can copy any additional libraries needed by the application in a similar manner.

   Alternatively, set up an NFS share between the host and the coprocessor, so that the Intel MPI Library installation directory is accessible from both. This saves up the RAM space on the coprocessor. For instructions, see the Intel MPSS User's Guide.

After completing these steps, you can proceed to building and running applications.

10.2. Building MPI Applications

(SDK only)

The Intel Xeon host and Intel Xeon Phi coprocessor are based on different architectures. Hence, you need to build different binaries for each. Follow these steps:

1. If you have not done that, set up the environment for the compiler and for the Intel® MPI Library:

   ```
   (host)$ source <installdir>/compilers_and_libraries/linux/bin/compilervars.sh
   intel64
   (host)$ source
   <installdir>/compilers_and_libraries/linux/mpi/intel64/bin/mpivars.sh
   ```

For the GCC* compiler, make sure it is in your `PATH`.

2. Build your application for the Intel® Xeon Phi™ coprocessor. When using the Intel® compilers, specify the `-mmic` option:

```
(host)$ mpiicc -mmic myprog.c -o myprog.mic
```

When using GCC*, you need to specify the GCC cross-compiler:

```
(host)$ mpicc -cc=/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-gcc myprog.c -o
myprog.mic
```

For the Intel Xeon Phi executable, the `.mic` extension is used to differentiate it from the host executable.

3. Build your application for the Intel® 64 architecture:

```
(host)$ mpiicc myprog.c -o myprog
```

10.3. Running MPI Applications

You can use the Intel® MPI Library with the Intel® Xeon Phi™ coprocessors in three major ways:

- **Native model.** In this model, all MPI ranks are run on the Intel Xeon Phi coprocessor card.
- **Symmetric model.** In this model, MPI ranks are run on both the Intel® Xeon® host and the Intel Xeon Phi coprocessor card, simultaneously.
- **Offload model.** In this model, MPI ranks are run on the main Intel Xeon host, and the application utilizes offload directives to run on the Intel Xeon Phi coprocessor card.

This section discusses primarily the first two models. For detailed instructions on using the offload model, refer to the Intel® MPSS User's Guide and the Intel® compiler documentation.

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

1. Copy the executable compiled for the Intel Xeon Phi coprocessor to the card:

```
(host)$ scp myprog.mic mic0:~/
```

Alternatively, make sure it is accessible from the card via an NFS share.

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

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

This allows you to specify the executable without the `.mic` extension during the launch.

You can set a prefix in a similar way, using the `I_MPI_MIC_PREFIX` variable. A prefix may be a folder as well.

3. Enable the Intel Xeon Phi coprocessor recognition by the Intel MPI Library. Set the `I_MPI_MIC` environment variable:

```
$ export I_MPI_MIC=enable
```

4. Launch the application on the Intel Xeon host and/or the Intel Xeon Phi coprocessor:
   - To run natively on the coprocessor only:

```
(host)$ mpirun -n 2 -host mic0 ~/myprog
```

This assumes that the host name of the card is `mic0`.

   - To run symmetrically on the host and the coprocessor:

```
(host)$ mpirun -n 4 -ppn 2 -hosts node0,mic0 ~/myprog
```

This assumes that `node0` is the local host and `mic0` is the coprocessor. In this example, the host and the card are specified with the `-hosts` option. For alternatives, see Controlling Process Placement.

5. When running the application, make sure paths to the executable match on all hosts and coprocessor cards involved.

See Also

Intel® MPI Library Developer Reference, section *Miscellaneous > Intel® Many Integrated Core Architecture Support*
Intel® MIC Architecture Developer Community
Intel® MPSS Web Page

11. Troubleshooting

This section provides the following troubleshooting information:

- General Intel® MPI Library troubleshooting procedures
- Typical MPI failures with corresponding output messages and behavior when a failure occurs
- Recommendations on potential root causes and solutions

11.1. General Troubleshooting Procedures

If you encounter errors or failures when using Intel® MPI Library, take the following general troubleshooting steps:

1. Check the *System Requirements* section and the *Known Issues* section in the *Intel® MPI Library Release Notes*.
2. Check accessibility of the hosts. Run a simple non-MPI application (for example, the `hostname` utility) on the problem hosts using `mpirun`. For example:

   ```
   $ mpirun -ppn 1 -n 2 -hosts node01,node02 hostname
   node01
   node02
   ```

   This may help reveal an environmental problem (such as, the MPI remote access mechanism is not configured properly), or a connectivity problem (such as, unreachable hosts).
3. Run the MPI application with debug information enabled: set the environment variables `I_MPI_DEBUG=6` and/or `I_MPI_HYDRA_DEBUG=on`. Increase the integer value of debug level to get more information. This action helps narrow down to the problematic component.
4. If you have the availability, download and install the latest version of Intel MPI Library from the official product page and check if your problem persists.
5. If the problem still persists, you can submit a ticket via Intel® Premier Support.

11.2. Examples of MPI Failures

This section provides examples of typical MPI failures including descriptions, output messages, and related recommendations. The following problems which may cause MPI failures are discussed in this section:

- Communication problems
- Environmental problems
- Other problems

11.2.1. Communication Problems

Communication problems with the Intel® MPI Library are usually caused by a signal termination (`SIGTERM`, `SIGKILL`, or other signals). Such terminations may be due to a host reboot, receiving an unexpected signal, out-of-memory (OOM) manager errors and others.

To deal with such failures, you need to find out the reason for the MPI process termination (for example, by checking the system log files).

Example 1

Symptom/Error Message

```
[50:node02] unexpected disconnect completion event from [41:node01]
```

and/or

```
===================================================================================
= BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
= PID 20066 RUNNING AT node01
= EXIT CODE: 15
= CLEANING UP REMAINING PROCESSES
= YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
```

The exact node and the MPI process reported in the table may not reflect the one where the initial failure had occurred.

Cause

One of MPI processes is terminated by a signal (for example, `SIGTERM` or `SIGKILL`) on `node01`. The MPI application was run over the `dapl` fabric.

Solution

Try to find out the reason of the MPI process termination. This may be a host reboot, receiving an unexpected signal, OOM manager errors and others. Check the system log files.

Example 2

Symptom/Error Message

```
rank = 26, revents = 25, state = 8
Assertion failed in file ../../src/mpid/ch3/channels/nemesis/netmod/tcp/socksm.c
at line 2969: (it_plfd->revents & POLLERR) == 0
internal ABORT - process 25
Fatal error in PMPI_Alltoall: A process has failed, error stack:
PMPI_Alltoall(1062).......: MPI_Alltoall(sbuf=0x9dd7d0, scount=64, MPI_BYTE,
rbuf=0x9dc7b0,
rcount=64, MPI_BYTE, comm=0x84000000) failed
MPIR_Alltoall_impl(860)...:
MPIR_Alltoall(819)........:
MPIR_Alltoall_intra(360)..:
dequeue_and_set_error(917): Communication error with rank 2rank = 45, revents = 25,
state = 8
Assertion failed in file ../../src/mpid/ch3/channels/nemesis/netmod/tcp/socksm.c
at line 2969: (it_plfd->revents & POLLERR) == 0
internal ABORT - process 84
...
Fatal error in PMPI_Alltoall: A process has failed, error stack:
PMPI_Alltoall(1062).......: MPI_Alltoall(sbuf=MPI_IN_PLACE, scount=-1,
MPI_DATATYPE_NULL,
rbuf=0x2ba2922b4010, rcount=8192, MPI_INT, MPI_COMM_WORLD) failed
MPIR_Alltoall_impl(860)...:
MPIR_Alltoall(819)........:
MPIR Alltoall intra(265)..:
MPIC_Sendrecv_replace(658):
dequeue_and_set_error(917): Communication error with rank 84
...
```

and/or:

```
===================================================================================
= BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
= PID 21686 RUNNING AT node01
= EXIT CODE: 15
= CLEANING UP REMAINING PROCESSES
= YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
```

The exact node and the MPI process reported in the table may not reflect the one where the initial failure had occurred.

Cause

One of MPI processes is terminated by a signal (for example, `SIGTERM` or `SIGKILL`) . The MPI application was run over the `tcp` fabric. In such cases, hang of the MPI application is possible.

Solution

Try to find out the reason of the MPI process termination. This may be a host reboot, receiving an unexpected signal, OOM manager errors and others. Check the system log files.

Example 3

Symptom/Error Message

```
[mpiexec@node00] control_cb (../../pm/pmiserv/pmiserv_cb.c:773): connection to
proxy
1 at host node01 failed
[mpiexec@node00] HYDT_dmxu_poll_wait_for_event (../../tools/demux/demux_poll.c:76):
callback returned error status
[mpiexec@node00] HYD_pmci_wait_for_completion
(../../pm/pmiserv/pmiserv_pmci.c:501):
error waiting for event
[mpiexec@node00] main (../../ui/mpich/mpiexec.c:1063): process manager error
waiting
for completion
```

Cause

The remote `pmi_proxy` process is terminated by the `SIGKILL (9)` signal on `node01`.

Solution

Try to find out the reason of the `pmi_proxy` process termination. This may be a host reboot, receiving an unexpected signal, OOM manager errors and others. Check the system log files.

Example 4

Symptom/Error Message

```
Failed to connect to host node01 port 22: No route to host
```

Cause

One of the MPI compute nodes (`node01`) is not available on the network. In such cases, hang of the MPI application is possible.

Solution

Check the network interfaces on the nodes and make sure the host is accessible.

Example 5

Symptom/Error Message

```
Failed to connect to host node01 port 22: Connection refused
```

Cause

The MPI remote node access mechanism is `SSH`. The `SSH` service is not running on `node01`.

Solution

Check the state of the `SSH` service on the nodes.

11.2.2. Environment Problems

Environmental errors may happen when there are problems with the system environment, such as mandatory system services are not running, shared resources are unavailable and so on.

When you encounter environmental errors, check the environment. For example, verify the current state of important services.

Example 1

Symptom/Error Message

```
librdmacm: Warning: couldn't read ABI version.
librdmacm: Warning: assuming: 4
librdmacm: Fatal: unable to get RDMA device list
```

or:

```
CMA: unable to get RDMA device list
librdmacm: couldn't read ABI version.
librdmacm: assuming: 4
```

Cause

The OFED* stack is not loaded. The application was run over the `dapl` fabric. In such cases, hang of the MPI application is possible.

Solution

See the OFED* documentation for details about OFED* stack usage.

Example 2

Symptom/Error Message

```
[0] MPI startup(): Multi-threaded optimized library
[1] DAPL startup(): trying to open DAPL provider from I_MPI_DAPL_PROVIDER: ofa-v2-
mlx4_0-1
[0] DAPL startup(): trying to open DAPL provider from I_MPI_DAPL_PROVIDER: ofa-v2-
mlx4_0-1
[1] MPI startup(): DAPL provider ofa-v2-mlx4_0-1
[1] MPI startup(): dapl data transfer mode
[0] MPI startup(): DAPL provider ofa-v2-mlx4_0-1
[0] MPI startup(): dapl data transfer mode
```

In such cases, hang of the MPI application is possible.

Cause

The Subnet Manager (`opensmd*`) service is not running. The application was run over the `dapl` fabric. The following output is provided when you set `I_MPI_DEBUG=2`.

Solution

Check the current status of the service. See the OFED* documentation for details on `opensmd*` usage.

Example 3

Symptom/Error Message

```
node01-mic0:MCM:2b66:e56a0b40: 2379 us(2379 us): scif_connect() to port 68, failed
with error Connection refused
node01-mic0:MCM:2b66:e56a0b40: 2494 us(115 us): open hca: SCIF init ERR for mlx4 0
Assertion failed in file
../../src/mpid/ch3/channels/nemesis/netmod/dapl/dapls_module_init.c
at line 761: 0
internal ABORT - process 0
```

Cause

The `mpxyd` daemon (CCL-proxy) is not running. The application was run over the `dapl` fabric. In such cases, hang of the MPI application is possible.

Solution

Check the current status of the service. See the DAPL* documentation for details on `mpxyd` usage.

Example 4

Symptom/Error Message

```
node01-mic0:SCM:2b94:14227b40: 201 us(201 us): open_hca: ibv_get_device_list()
failed
node01-mic0:SCM:2b94:14227b40: 222 us(222 us): open_hca: ibv_get_device_list()
failed
node01-mic0:CMA:2b94:14227b40: 570 us(570 us): open_hca: getaddr_netdev ERROR:No
such device. Is ib0 configured?
...
Fatal error in MPI_Init: Other MPI error, error stack:
MPIR_Init_thread(784).................:
MPID_Init(1326).......................: channel initialization failed
MPIDI_CH3_Init(141)...................:
dapl_rc_setup_all_connections_20(1386): generic failure with errno = 872609295
getConnInfoKVS(849)...................: PMI_KVS_Get failed
```

Cause

The `ofed-mic` service is not running. The application was run over the `dapl` fabric. In such cases, hang of the MPI application is possible.

Solution

Check the current status of the service. See the Intel® MPSS documentation for details on `ofed-mic` usage.

Example 5

Symptom/Error Message

```
pmi_proxy: line 0: exec: pmi_proxy: not found
```

Cause

The Intel® MPI Library runtime scripts are not available. A possible reason is that the shared space cannot be reached. In such cases, hang of the MPI application is possible.

Solution

Check if the shared path is available across all the nodes.

Example 6

Symptom/Error Message

```
[0] DAPL startup: RLIMIT_MEMLOCK too small
[0] MPI startup(): dapl fabric is not available and fallback fabric is not enabled
```

or:

```
node01:SCM:1c66:3f226b40: 6815816 us: DAPL ERR reg_mr Cannot allocate memory
```

Cause

Wrong system limits: the max locked memory is too small. The application was run over the `dapl` fabric.

Solution

Check the system limits and update them if necessary. The following example shows the correct system limits configuration:

```
$ ulimit -a
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
scheduling priority (-e) 0
file size (blocks, -f) unlimited
pending signals (-i) 256273
max locked memory (kbytes, -l) unlimited
max memory size (kbytes, -m) unlimited
open files (-n) 1024
pipe size (512 bytes, -p) 8
POSIX message queues (bytes, -q) 819200
real-time priority (-r) 0
stack size (kbytes, -s) unlimited
cpu time (seconds, -t) unlimited
max user processes (-u) 1024
virtual memory (kbytes, -v) unlimited
file locks (-x) unlimited
```

Example 7

Symptom/Error Message

```
Are you sure you want to continue connecting (yes/no)? The authenticity of host
'node01 (<node01_ip_address>)' can't be established.
```

This message may repeat continuously until manual interruption.

Cause

The MPI remote node access mechanism is SSH. SSH is not configured properly: unexpected messages appear in the standard input (`stdin`).

Solution

Check the SSH connection to the problem node.

Example 8

Symptom/Error Message

```
Password:
```

Cause

The MPI remote node access mechanism is SSH. SSH is not password-less. In such cases, hang of the MPI application is possible.

Solution

Check the SSH settings: password-less authorization by public keys should be enabled and configured.

11.2.3. Other Problems

Example 1

Symptom/Error Message

```
cannot execute binary file
```

Cause

Wrong binary executable file format or architecture.

This error occurs when you run a binary executable file build for `x86_64` architecture on a `k1om` node (for example, on an Intel® Xeon Phi™ coprocessor). In such cases, hang of the MPI application is possible.

Solution

Verify the correctness of the binary file and the command line options.

Example 2

Symptom/Error Message

```
node01.9234ipath_userinit: assign_context command failed: Invalid argument
node01.9234Driver initialization failure on /dev/ipath (err=23)
```

Cause

Intel® True Scale IBA resource exhaustion. The MPI application was run over the `tmi` fabric.

Depending on Intel® True Scale Fabric hardware, PSM* may not support CPU over-subscription of the node. The maximum amount of processes which can be run on the node is limited and depends on a combination of the particular Intel® True Scale Fabric hardware and the amount of CPU cores.

Solution

Limit the number of MPI processes per node.