Intel® MPI Library for Windows* OS

Developer Guide

Contents

Legal Information

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

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)* has the tools you need to run programs, including scalable process management system (Hydra*) and supporting utilities, dynamic (`.dll`) 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, debug libraries, program database (`.pdb`) files, 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 Windows* 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 Windows* OS installed, you do not need to uninstall it before installing a newer version.

To install Intel MPI Library, double-click on the distribution file `w_mpi_p_<version>.<package_num>.exe` (complete product), and `w_mpi-rt_p_<version>.<package_num>.exe` (RTO component only).

You will be asked to choose a directory in which the contents of the self-extracting installation file will be placed before the actual installation begins. After installation, the files will still be located in this directory. By default, `C:\Program Files (x86)\Intel\Download` is used on machines with Intel® 64 architecture.

Follow the prompts outlined by the installation wizard to complete the installation.

NOTE

You need the domain administrator rights when you install the Intel® MPI Library on Windows* OS. Otherwise, you cannot proceed with the Active Directory* setup on Windows* OS. See the *Intel® MPI Library Developer Reference* for more Active Directory* setup information.

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 wrappers (`mpiicc`, `mpicc`, and so on), files and modules, 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. Run the `mpivars.bat` script:

   ```
   > <installdir>\intel64\bin\mpivars.bat
   ```

   By default, `<installdir>` is `C:\Program Files (x86)\IntelSWTools\mpi\<version>`.

NOTE

The `mpivars.bat` script sets environment variables required for working with the Intel® MPI Library. To use all the Intel MPI Library functionality, launch its commands in the same command-line session where you run the `mpivars.bat` script.

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:

   ```
   > type hosts
   # this line is ignored
   clusternode1
   clusternode2
   clusternode3
   clusternode4
   ```

3. Make sure the Hydra service is installed and running on the cluster nodes. To check this, enter the command:

   ```
   > hydra_service -status
   ```

   If the service is not running, use the following command to install and run it:

   ```
   > hydra_service -install
   ```

4. Register your Windows* user credentials to enable the process manager to launch MPI jobs. Credentials are encrypted and stored in the registry:

   ```
   > mpiexec -register
   ```

   If you do not do this in advance, you will be prompted to enter credentials when running an MPI job with `mpiexec`.

   You can also use the domain-based authorization, which does not ask for your credentials, but requires some additional configuration. See User Authorization for details.

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

3.1. User Authorization

Intel® MPI Library supports two main methods to allow user authentication across a Windows* OS cluster:

- Password-based authorization
- Domain-based authorization

The password-based authorization is the most common method of providing remote node access through a user's existing account name and password. Intel MPI Library allows you to encrypt your login information and store it in the registry with the `mpiexec -register` command. You need to do this once, during the first application run.

The domain-based authorization uses Microsoft* Security Service Provided Interface* (SSPI*) to allow user authentication on the remote machines. This is done in accordance with the established domain policies, as defined by the cluster administrator. No user account information (user name and password) is stored on the machine.

To enable the domain-based authorization method on a Windows* OS cluster, the administrator needs to setup Active Directory* delegation for the compute nodes and cluster users.

For more details on how to enable the Active Directory setup, see the *Developer Reference*, section *Miscellaneous > User Authorization*.

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/C++ Compiler, enter the command:

   ```
   > icl
   ```

   If the command is not found, add the full path to your compiler into the `PATH`. For Intel® compilers, you can run the `compilervars.bat` script to set the required environment variables.
2. In the same command-line session, run the `mpivars.bat` script to set up the proper environment for Intel MPI Library:

   ```
   > <installdir>\intel64\bin\mpivars.bat
   ```
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
   ```

You will get an executable file `myprog.exe` 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® 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
```

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. 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 `C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_<version>\windows\mpi` by default.

4.2. Configuring a Visual Studio* Project

To configure a Visual Studio* project with Intel® MPI Library, do the following:

1. In Microsoft* Visual Studio*, create a console application project, or open an existing one.
2. Open the project properties and go to **Configuration Properties > Debugging**. Set the following parameters:

   Command: `$(I_MPI_ROOT)\intel64\bin\mpiexec.exe`
   Command arguments: `-n <processes_number> "$(TargetPath)"`
3. In **Configuration Properties > C/C++** or **Fortran**, as appropriate, set the following parameter:

   Additional Include Directories: `$(I_MPI_ROOT)\intel64\include`
4. In **Configuration Properties > Linker**, set the following parameter:

   Additional Library Directories: `$(I_MPI_ROOT)\intel64\lib\<configuration>`

   You can set the following values for `<configuration>`:

   - `debug`: single-threaded debug library
   - `release`: single-threaded optimized library
   - `debug_mt`: multi-threaded debug library
   - `release_mt`: multi-threaded optimized library
5. In **Configuration Properties > Linker > Input**, set the following parameter:

   Additional Dependencies: `impi.lib`

After completing these steps, you can build the solution and run the application. To run the application from Visual Studio*, you can use the Ctrl + F5 key combination (Start Without Debugging). For other options of running MPI applications, see Running Applications.

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 `mpiexec` command. The command line syntax is as follows:

```
> mpiexec -n <# of processes> -ppn <# of processes per node> -f <hostfile>
myprog.exe
```

For example:

```
> mpiexec -n 4 -ppn 2 -f hosts myprog.exe
```

The `mpiexec` command launches the Hydra process manager, which controls the execution of your MPI program on the cluster.

In the command line above:

- `-n` sets the number of MPI processes to launch; if the option is not specified, 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.exe` is the name of your MPI program.

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

See Also

Controlling Process Placement
Job Schedulers 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 a configuration, run `mpivars.bat` with the appropriate argument. For example:

   ```
   > mpivars.bat release_mt
   ```

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

   ```
   > set 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:

```
> mpiexec -n 4 -genv OMP_NUM_THREADS=4 -genv I_MPI_PIN_DOMAIN=omp myprog.exe
```

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 `mpiexec`, one group per line. For example:

```
> type mpmd_config
-n 1 -host node1 io.exe <io_args>
-n 4 -host node2 compute.exe <compute_args_1>
-n 4 -host node3 compute.exe <compute_args_2>
> mpiexec -configfile mpmd_config
```

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

```
> mpiexec -n 1 -host node1 io.exe <io_args> :^
-n 4 -host node2 compute.exe <compute_args_1> :^
-n 4 -host node3 compute.exe <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 Windows* OS:

- Microsoft* HPC Pack*
- Altair* PBS Pro*

5.4.1. Microsoft* HPC Pack*

The Intel® MPI Library job startup command `mpiexec` can be called out of Microsoft* HPC Job Scheduler to execute an MPI application. In this case, the `mpiexec` command automatically inherits the host list, process count, and the working directory allocated to the job.

Use the following command to submit an MPI job:

```
> job submit /numprocessors:4 /stdout:test.out mpiexec -delegate test.exe
```

Make sure the `mpiexec` and dynamic libraries are available in `PATH`.

5.4.2. Altair* PBS Pro*

The Intel® MPI Library job startup command `mpiexec` can be called out of PBS Pro* job scheduler to execute an MPI application. In this case the `mpiexec` command automatically inherits the host list, process count allocated to the job if they were not specified manually by the user. `mpiexec` reads `%PBS_NODEFILE%` environment variable to count a number of processes and uses it as a machine file.

Example of a job script contents:

```
REM PBS -l nodes=4:ppn=2
REM PBS -l walltime=1:00:00
cd %PBS_O_WORKDIR%
mpiexec test.exe
```

Use the following command to submit the job:

```
> qsub -C "REM PBS" job
```

`mpiexec` will run two processes on each of four nodes for this job.

5.4.3. 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:

```
> set 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. On the Linux hosts, start the Hydra persist server in the background mode:

   ```
   $ hydra_persist &
   ```
4. 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 LD_LIBRARY_PATH` – local environment variables for the specified host
- `-bootstrap service` – specifies the remote node access mechanism

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

```
> mpiexec -demux select -bootstrap service -genv I_MPI_FABRICS shm:tcp ^
-env I_MPI_ROOT=<linux_installdir> -env
LD_LIBRARY_PATH=<linux_installdir>/<arch>/lib ^
-hostos linux -host <lin_host> -n 2 <linux_installdir>/<arch>/bin/IMB-MPI1 pingpong
:^
-host <win_host> -n 3 <windows_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, run the `mpivars.bat` script with appropriate arguments. For example:

```
> <installdir>\intel64\bin\mpivars.bat 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. Run the `mpivars.bat` script with appropriate arguments before an application launch.

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:

```
> mpiexec -n 4 -ppn 2 -hosts node1,node2 testc.exe
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 `mpiexec`. For example:

```
> type hosts
#nodes
node1
node2
> mpiexec -n 4 -ppn 2 -f hosts testc.exe
```

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:

```
> mpiexec -n 6 -hosts node1,node2 testc.exe
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:

```
> type machines
node1:2
node2:2
```

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

```
> mpiexec -machine machines testc.exe
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:

```
> mpiexec -genv I_MPI_DEBUG=2 -host node1 -n 2 testc.exe : -host node2 -n 2
testc.exe
```

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:

```
> type config
-genv I_MPI_DEBUG=2
-host node1 -n 2 testc.exe
-host node2 -n 2 testc.exe
```

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

```
> mpiexec -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*).

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.

By default, this list is as follows:

`dapl,tcp`

If the first fabric is unavailable, Intel® MPI Library falls back to the second fabric from the list.

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

```
> mpiexec -n 4 -ppn 2 -hosts <node1>,<node2> -genv I_MPI_DEBUG=2 myprog.exe
...
```

```
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:

```
> set I_MPI_FABRICS_LIST=tcp,dapl
> mpiexec -n 4 -ppn 2 -hosts <node1>,<node2> myprog.exe
```

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

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

```
> set I_MPI_FALLBACK=0
```

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

6.3.3. Selecting Specific Fabric

You can also explicitly specify a 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:

```
> set I_MPI_FABRICS=shm:tcp
> mpiexec -n 4 -ppn 2 -hosts <node1>,<node2> myprog.exe
```

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:

```
> set 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:

```
> set 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. 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.

7.1. 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:

```
> mpiexec -genv I_MPI_DEBUG=2 -n 2 testc.exe
[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:

  ```
  > mpiexec -genv I_MPI_DEBUG=+2 -n 2 testc.exe
  [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:

  ```
  > mpiexec -genv I_MPI_DEBUG=2,time,norank -n 2 testc.exe
  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:

```
> mpiexec -genv I_MPI_DEBUG=2 -genv I_MPI_DEBUG_OUTPUT=debug_output.txt -n 2
testc.exe
```

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*

7.2. 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.

7.2.1. 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:

```
> set I_MPI_STATS=10
```

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

```
> set 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:

```
> set 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:

```
> set I_MPI_STATS=20
> set 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:

```
> set 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:

```
> set I_MPI_STATS_BUCKETS="16@4"
```

7.2.2. 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:

```
> set 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:

```
> set I_MPI_STATS=ipm
> set I_MPI_STATS_ACCURACY=3
```

7.2.3. 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:

```
> set 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:

```
> set I_MPI_STATS=native:2-10,ipm:terse
```

7.2.4. 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*

7.3. 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-time 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 the product installed.

7.3.1. Tracing Applications

To analyze an application, 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 linking with the Intel® Trace Collector profiling library, which intercepts all MPI calls and generates a trace file. Intel MPI Library provides the `-trace` (`-t`) compiler option to simplify this process.

Complete the following steps:

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

   ```
   > <compiler_installdir>\bin\compilervars.bat intel64
   > <mpi_installdir>\intel64\bin\mpivars.bat
   > <itac_installdir>\bin\itacvars.bat
   ```

2. Relink your application with the Intel Trace Collector profiling library and run the application:

   ```
   > mpiicc -trace myprog.c
   > mpiexec -n 4 myprog.exe
   ```

   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.

7.3.2. 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` option during compilation, and run the application. If errors are detected, they are printed in the console output. For example:

```
> mpiicc -check_mpi myprog.c
> mpiexec -n 4 myprog.exe
...
[0] ERROR: GLOBAL:COLLECTIVE:DATATYPE:MISMATCH: error
...
```

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

See Also

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

8. 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>\intel64\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.

The general workflow for both 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.

8.1. 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:

   ```
   > mpiexec -tune -genv I_MPI_FABRICS=<value> -ppn 8 -n 64 myprog.exe
   ```

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

   ```
   > mpiexec -tune <results_directory> -genv I_MPI_FABRICS=<value> -ppn 8 -n 64
   myprog.exe
   ```

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:

```
> mpiexec -tune <path_to_your_config_file>\mpitune_config.conf
```

8.1.1. 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,tcp <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

8.1.2. 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.

8.2. 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 \"mpiexec <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:

```
> mpiexec -tune mpitune_config.conf -ppn 8 -n 64 myprog.exe
```

8.3. 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.

8.3.1. 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
```

8.3.2. 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>
```

8.3.3. 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>
```

8.3.4. 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>
```

8.4. 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*.

9. 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

9.1. General Troubleshooting Procedures

If you encounter errors or failures when using the 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 `mpiexec`. For example:

   ```
   > mpiexec -ppn 1 -n 2 -hosts node01,node02 hostname
   node01
   node02
   ```

   This may help reveal an environmental problem, 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.

9.2. Examples of MPI Failures

This section provides examples of typical MPI errors including error descriptions, error messages, and related recommendations.

9.2.1. Example 1

Symptom/Error Message

```
Error connecting to the Service
[mpiexec@node01] ..\hydra\utils\sock\sock.c (270): unable to connect from "node01"
to "node02" (No error) read from stdin failed, error 9.
[mpiexec@node01] ..\hydra\tools\demux\demux_select.c (78): select error (No such
file or directory)
[mpiexec@node01] ..\hydra\pm\pmiserv\pmiserv_pmci.c (501): error waiting for event
[mpiexec@node01] ..\hydra\ui\mpich\mpiexec.c (1063): process manager error waiting
for completion
```

Cause

The Hydra service is not running on one of the nodes.

Solution

Check the current status of the Hydra service on all the nodes.

9.2.2. Example 2

Symptom/Error Message

```
Error connecting to the Service
[mpiexec@node01] ..\hydra\utils\sock\sock.c (224): unable to get host address for
node02 (11001) read from stdin failed, error 9.
[mpiexec@node01] ..\hydra\tools\demux\demux_select.c (78): select error (No such
file or directory)
[mpiexec@node01] ..\hydra\pm\pmiserv\pmiserv_pmci.c (501): error waiting for event
[mpiexec@node01] ..\hydra\ui\mpich\mpiexec.c (1063): process manager error waiting
for completion
```

Cause

One of the nodes is unreachable.

Solution

Check that all the nodes are operable.

9.2.3. Example 3

Symptom/Error Message

```
pmi_proxy not found on node02. Set Intel MPI environment variables read from stdin
failed, error 9.
[mpiexec@node01] ..\hydra\tools\demux\demux_select.c (78): select error (No such
file or directory)
[mpiexec@node01] ..\hydra\pm\pmiserv\pmiserv_pmci.c (501): error waiting for event
[mpiexec@node01] ..\hydra\ui\mpich\mpiexec.c (1063): process manager error waiting
for completion
```

Cause

The Intel® MPI Library runtime scripts are not available. A possible reason is that the shared space cannot be reached.

Solution

Check MPI availability on all the nodes. Possibly, there are some problems with network shared drive.

9.2.4. Example 4

Symptom/Error Message

```
[-1:5612]ERROR:execvp error on file: <mpi_application_path>,
The filename, directory name, or volume label syntax is incorrect
```

Cause

The MPI application is not available on some nodes.

Solution

Check the availability of the MPI application across the nodes. Possibly there are some problems with network shared drive.

9.2.5. Example 5

Symptom/Error Message

```
[3:node01] unexpected disconnect completion event from [8:node02]
```

Cause

One of MPI processes is terminated. The MPI application was run over the `dapl` fabric.

Solution

Try to find out the reason of the MPI process termination.

9.2.6. Example 6

Symptom/Error Message

```
Attempting to use an MPI routine before initializing MPI
```

Cause

A cross-linked application with dependencies on different Intel® MPI Library configurations.

Solution

1. Check your application dependencies with `dumpbin` Microsoft* Visual Studio* utility:

   ```
   > dumpbin /DEPENDENTS app.exe
   Microsoft (R) COFF/PE Dumper Version 11.00.61030.0
   Copyright (C) Microsoft Corporation. All rights reserved.
   Dump of file app.exe
   File Type: EXECUTABLE IMAGE
    Image has the following dependencies:
    impi.dll
    foo.dll
   ```

2. Check the third-party library dependencies as well:

   ```
   > dumpbin /DEPENDENTS foo.dll
   Microsoft (R) COFF/PE Dumper Version 11.00.61030.0
   Copyright (C) Microsoft Corporation. All rights reserved.
   Dump of file foo.dll
   File Type: DLL
   ```

```
Image has the following dependencies:
impimt.dll
```

3. `Foo.dll` depends on `impimt.dll` and `app.exe` depends on `impi.dll`.
4. Relink your application in backward compatibility mode to link with `impimt.dll`. Use the `-link_mpi` compiler driver option or the `I_MPI_LINK` environment variable:

   ```
   > mpiicc -link_mpi opt_mt_compat app.c -o app.exe foo.lib
   ```

   or

   ```
   > set I_MPI_LINK=opt_mt_compat
   > mpiicc app.c -o app.exe foo.lib
   ```

9.2.7. Example 7

Symptom/Error Message

```
> hydra_service -start
OpenSCManager failed:
Access is denied. (error 5)
```

Cause

You should have administrator privileges to start the Hydra service. Once the Hydra service has started, all users can launch processes with `mpiexec`.

Solution

Run the command line in the command prompt with administrator privileges:

```
> hydra_service -start
Intel(R) MPI Library Hydra Process Manager started.
```