Intel® MPI Library Developer Guide for Windows* OS

Contents

Chapter 1: Intel® MPI Library Developer Guide for Windows* OS

Introduction ... 3
- Introducing Intel® MPI Library ... 3
- Conventions and Symbols ... 3
- Related Information ... 3

Installation and Prerequisites ... 4
- Installation ... 4
- Prerequisite Steps ... 4
- User Authorization ... 5

Compiling and Linking ... 7
- Compiling an MPI Program ... 7
- Configuring a Microsoft Visual Studio* Project ... 8

Running Applications ... 8
- Running Intel® MPI Library in Containers ... 8
  - Build a Singularity* Container for an MPI Application ... 9
  - Run the Application with a Container ... 9
- Selecting a Library Configuration ... 11
- Running an MPI Program ... 12
- Running an MPI/OpenMP* Program ... 12
- MPMD Launch Mode ... 13
- Fabrics Control ... 13
  - Selecting Fabrics ... 14
  - Libfabric* Support ... 14
  - OFI* Providers Support ... 15
- Job Schedulers Support ... 17
- Controlling Process Placement ... 18
- Java* MPI Applications Support ... 20

Debugging Applications ... 21
- Debugging ... 21
- Using -gtool for Debugging ... 22

Analysis and Tuning ... 22
- Displaying MPI Debug Information ... 22
- Tracing Applications ... 23
- MPI Tuning ... 23

Troubleshooting ... 24
- Error Message: Bad Termination ... 25
- Error Message: No such file or Directory ... 26
- Error Message: Permission Denied ... 26
- Error Message: Fatal Error ... 26
- Error Message: Bad File Descriptor ... 27
- Error Message: Too Many Open Files ... 28
- Problem: MPI Application Hangs ... 28
- Problem: Password Required ... 29
- Problem: Cannot Execute Binary File ... 29
- Problem: MPI limitation for Docker* ... 30

Notices and Disclaimers ... 30

Intel® MPI Library Developer Guide for Windows* OS

1

Documentation for older versions of the Intel® MPI Library are available for download only. For a list of available Intel® Parallel Studio XE documentation by product version, see Download Documentation for Intel Parallel Studio XE. For previous versions of Intel MPI Library documentation, see the Legacy Documentation page.

Find useful information about using the Intel® MPI Library in the following topics:

- Introducing Intel MPI Library gives a general introduction for the Intel MPI Library and its components.
- Compiling and Linking provides instructions on compiling and linking MPI applications with the Intel MPI Library in the command line and in Visual Studio*.
- Running Applications describes how to run MPI and hybrid OpenMP*/MPI programs in the command line and through job schedulers.
- Analysis and Tuning discusses the methods for MPI analysis using the built-in Intel MPI Library features and other analyzing tools.
- Troubleshooting provides the troubleshooting steps for some common issues with the Intel MPI Library.

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 to start using the Intel MPI Library. For full information, see the Intel® MPI Library Developer Reference.

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.

Conventions and Symbols

The following conventions are used in this document:

This type style	Document names
`This type style`	Commands, arguments, options, file names
THIS_TYPE_STYLE	Environment variables
<this type style>	Variables, or placeholders for actual values
[items]	Optional items
{ item | item }	Selectable items separated by vertical bar(s)

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.

For additional resources, see:

- Intel® MPI Library Product Web Site
- Intel® Software Documentation Library
- Intel® Software Products Support

Installation and Prerequisites

This section describes the installation process and prerequisite steps.

- Installation
- Prerequisite Steps
- User Authorization

Installation

New Installations

The Intel® MPI Library is part of the Intel® oneAPI HPC Toolkit (HPC Kit), which includes a variety of tools to help you build, analyze, and deploy HPC applications. You can download the HPC Kit here, or visit the HPC Kit product page for information on downloading a stand-alone or runtime version of the Intel MPI Library.

Upgrading

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 the Intel MPI Library, double-click on the distribution file `w_mpi_oneapi_p_<version>.<package-num>.exe`.

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

Prerequisite Steps

Before you start using any of the Intel® MPI Library functionality, make sure to establish the proper environment settings:

1. Set up the Intel MPI Library environment by running the `setvars.bat` file, which is found in your installation directory (by default, `C:\Program Files (x86)\Intel\oneAPI\mpi\<version>`).

NOTE You must run `setvars` at the start of each command-line session unless you are using Microsoft Visual Studio for Windows* (Visual Studio runs `setvars` automatically). See the Intel oneAPI HPC Toolkit Get Started Guide for more details, including options for automatic setup of environment settings.

2. To run an MPI application on a cluster, the 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
```

IntelMPI uses PowerShell bootstrap by default. It is available by default and does not require any extra steps. If you need more fine-grain setup information, refer to Microsoft PowerShell Setup Guide (https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_remote_requirements?view=powershell-7.3), Security Guide (https://learn.microsoft.com/en-us/powershell/scripting/learn/remoting/winrmsecurity?view=powershell-7.3), and Troubleshooting Guide (https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_remote_troubleshooting?view=powershell-7.3). You can also consider domain-based authorization as an alternative to the default mechanism. However, this method requires additional configuration. See User Authorization.

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

User Authorization

The Intel® MPI Library supports several authentication methods under the Microsoft Windows* OS:

- Password-based authorization
- Domain-based authorization with the delegation ability
- Limited 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.

Intel® MPI Library additionally supports domain-based authorization with the delegation ability as well as limited domain-based authorization. The domain-based authorization methods use the Security Service Provider Interface (SSPI) provided by Microsoft* in the Windows* OS environment. The SSPI allows a domain to authenticate the user on the remote machine by the domain policies. You do not need to enter and store your account name and password when using such methods.

NOTE Both domain-based authorization methods may increase MPI task launch time in comparison with the password-based authorization. This depends on the domain configuration.

NOTE The limited domain-based authorization restricts your access to the network. You will not be able to open files on remote machines or access mapped network drives.

This feature is supported on clusters under Windows HPC Server 2012 R2. Microsoft's Kerberos Distribution Center* must be enabled on your domain controller (this is the default behavior).

Using the domain-based authorization method with the delegation ability requires specific installation of the domain. You can perform this installation by using the Intel® MPI Library installer if you have domain administrator rights or by following the instructions below.

Active Directory* Setup

To enable the delegation in the Active Directory*, do the following:

1. Log in on the domain controller under the administrator account.
2. Enable the delegation for cluster nodes:
   a. Go to **Administrative Tools**.
   b. In the **Active Directory Users and Computers** administrative utility open the **Computers** list.
   c. Right click on a desired computer object and select **Properties**.
   d. Select the **Delegation** tab and check the **Trust this computer for delegation to any service (Kerberos only)** option.
3. Enable the delegation for users:
   a. In the **Active Directory Users and Computers** administrative utility open the **Users** list.
   b. Right click on a desired user object and select **Properties**.
   c. Select the **Account** tab and disable the **Account is sensitive and cannot be delegated** option.
4. 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
   ```

   NOTE Run this command as an administrator.

5. Register service principal name (SPN) for cluster nodes. Use one of the following methods for registering SPN:
   a. Use the `setspn.exe` utility from Microsoft*. For example, execute the following command on the domain controller:

```
> setspn.exe -A
                impi_hydra/<host>:<port>/impi_hydra
                <host>
```

   where:

   - *<host>* is the cluster node name.
   - *<port>* is the Hydra port. The default value is `8679`. Change this number only if your hydra service uses the non-default port.

   b. Log into each desired node under the administrator account and execute the command:

```
> hydra_service -register_spn
```

   NOTE In case of any issues with the MPI task start, reboot the machine from which the MPI task is started. Alternatively, execute this command: `> klist purge`

To select a user authorization method, use the `I_MPI_AUTH_METHOD` environment variable with the `password`, `delegate`, or `impersonate` argument. For more details, see the *Developer Reference*, section User Authorization.

Compiling and Linking

This section gives instructions on how to compile and link MPI applications using the command line and Visual Studio* :

- Compiling an MPI program
- Configuring a Visual Studio* Project

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 script `setvars.bat` to set the required environment variables.
2. In the same command-line session, run the `setvars.bat` script in the installation directory to set the proper environment variables for the Intel MPI Library (the default installation directory location is `C:\Program Files(x86)\Intel\oneAPI\<version>`
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.exe` in the current directory, which you can start immediately. For instructions of how to launch MPI applications, see Running an MPI Program.

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.

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.

Test MPI Programs

The 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 in the `test` directory in your installation directory.

See Also

Compiler Commands

Configuring a Microsoft Visual Studio* Project

To configure a Microsoft 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, go to **Configuration Properties > Debugging** and set the following parameters:

   Command: `$(I_MPI_ONEAPI_ROOT)\bin\mpiexec.exe`

   Command arguments: `-n` *`<processes-number>`* `"$(TargetPath)"`

   Environment: `PATH=$(I_MPI_ONEAPI_ROOT)\bin\$(ConfigurationName);$(I_MPI_ONEAPI_ROOT)\libfabric\bin;$(PATH)`

   > **NOTE** If you have installed the Microsoft Visual Studio* integration, go to **Project > Properties > Intel Libraries for oneAPI > Use Intel MPI Library** and set it to **Yes**. You can skip the steps below.

3. In **Configuration Properties > C/C++** or **Fortran**, as appropriate, set the following parameter:

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

   Additional Library Directories: `$(I_MPI_ONEAPI_ROOT)\lib\$(ConfigurationName);$(I_MPI_ONEAPI_ROOT)\lib`
5. In **Configuration Properties > Linker > Input**, set the following parameters:

   - For Fortran, set Additional Dependencies: `impi.lib`
   - For C++, set Additional Dependencies: `impi.lib` and `impicxx.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 available options, see Running Applications.

Running Applications

After you have compiled and linked your MPI application, you are ready to run it. This topic provides instructions on how to run various MPI applications in various modes:

- Running an MPI Program
- Running an MPI/OpenMP* Program
- MPMD Launch Mode
- Selecting Fabrics
- Selecting Library Configuration
- libfabric* Support
- Job Schedulers Support
- Controlling Process Placement

Running Intel® MPI Library in Containers

A container is a self-contained execution environment platform that enable flexibility and portability of your MPI appliction. It lets you package an application and its dependencies in a virtual container that can run on an operating system, such as Linux*.

This guide describes the use of the Intel® MPI Library with the Singularity* container type.

Singularity Containers

Singularity* is a lightweight container model aligned with the needs of high performance computing (HPC). Singularity has a built-in support of MPI and allows you to leverage the resources of the host you are on, including HPC interconnects, resource managers, and accelerators.

This chapter provides information on running Intel® MPI Library in a Singularity container built from a recipe file. To run Intel® MPI Library in a Singularity environment, do the following:

1. Make sure you have the following components installed on each machine of a cluster:
   a. Singularity (version not lower than 3.0).
   b. A container including your application.
   c. Intel MPI Library.
2. Create a Singularity recipe file and use it to build a container.
3. Run your MPI application from the Singularity container.

Build a Singularity* Container for an MPI Application

There are several ways to build Singularity* containers described in the Singularity official documentation.

This section demonstrates how to build a container for an MPI application from scratch using recipes. Singularity recipes are files that include software requirements, environment variables, metadata, and other useful details for designing a custom container.

Recipe File Structure

A recipe file consists of the header and sections. The header part defines the core operating system and core packages to be installed. In particular:

- **Bootstrap** - specifies the bootstrap module.
- **OSVersion** - specifies the OS version. Required if only you have specified the %{OSVERSION} variable in **MirrorURL.**
- **MirrorURL** - specifies the URL to use as a mirror to download the OS.
- **Include** - specifies additional packages to be installed into the core OS (optional).

The content of a recipe file is divided into sections that execute commands at different times during the build process. The build process stops if a command fails. The main sections of a recipe are:

- **%help** - provides help information.
- **%setup** - executes commands on the host system outside of the container after the base OS is installed.
- **%post** - executes commands within the container after the base OS has been installed at build time.
- **%environment** - adds environment variables sourced at runtime. If you need environment variables sourced during build time, define them in the %post section.

Build a Container

After the recipe file is created, use it to create a Singularity container. The example below shows how to build a container with default parameters:

```
$ singularity build mpi.img ./Singularity_recipe_mpi
```

Run the Application with a Container

You can choose from three usage models for running your application using a Singularity* container:

1. Everything packed into a single container
2. The Intel MPI Library installed both inside and outside the container
3. The Intel MPI Library outside the container

Usage model 1: Everything packed into a single container

This approach presumes that the Intel® MPI library, target application, and all its dependencies are packed into a container.

Recipe file

```
BootStrap: yum
OSVersion: 8
MirrorURL: http://linux-ftp.jf.intel.com/pub/mirrors/centos/8/BaseOS/$basearch/os/
Include: yum
%environment
source /opt/intel/oneapi/mpi/latest/env/vars.sh
%post
export http_proxy=http://***
yum repolist
yum install -y yum-utils
tee > /tmp/oneAPI.repo << EOF
[oneAPI]
name=Intel(R) oneAPI repository
baseurl=https://yum.repos.intel.com/oneapi
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
EOF
mv /tmp/oneAPI.repo /etc/yum.repos.d
yum install -y intel-oneapi-mpi*
yum install -y sudo wget vi which numactl bzip2 tar gcc hostname util-linux redhat-lsb openssh-
server openssh-clients
```

Launch

When recipe is created, execute the following command:

```
> singularity exec <container-name> mpirun -n <number-of-processes> -ppn <processes-per-node> -
hostlist <hosts> <application>
```

Usage model 2: The Intel MPI Library installed both inside and outside the container

In this approach, additional dependency on hosts (for example, external mpirun) is required. Each rank is a separate Singularity container instance execution.

Recipe file

```
BootStrap: yum
OSVersion: 8
MirrorURL: http://linux-ftp.jf.intel.com/pub/mirrors/centos/8/BaseOS/$basearch/os/
Include: yum
%environment
source /opt/intel/oneapi/mpi/latest/env/vars.sh
%post
export http_proxy=http://***
yum repolist
yum install -y yum-utils
tee > /tmp/oneAPI.repo << EOF
[oneAPI]
name=Intel(R) oneAPI repository
baseurl=https://yum.repos.intel.com/oneapi
enabled=1
gpgcheck=1
repo_gpgcheck=1
```

```
gpgkey=https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
EOF
mv /tmp/oneAPI.repo /etc/yum.repos.d
yum install -y intel-oneapi-mpi*
yum install -y sudo wget vi which numactl bzip2 tar gcc hostname util-linux redhat-lsb
```

Launch

When recipe is created, execute the following command:

```
> mpiexec  -n <number-of-processes> -ppn <processes-per-node> -hostlist <hosts> singularity exec
<container-name> <application>
```

Usage model 3: The Intel MPI Library outside the container

In this approach, additional dependency on hosts (for example, external mpirun) is required. Each host has a single Singularity container instance executed for all ranks.

Recipe file

```
BootStrap: yum
OSVersion: 8
MirrorURL: http://linux-ftp.jf.intel.com/pub/mirrors/centos/8/BaseOS/$basearch/os/
Include: yum
%environment
source /opt/intel/oneapi/mpi/latest/env/vars.sh
%post
export http_proxy=http://***
yum repolist
yum install -y yum-utils
tee > /tmp/oneAPI.repo << EOF
[oneAPI]
name=Intel(R) oneAPI repository
baseurl=https://yum.repos.intel.com/oneapi
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB
EOF
mv /tmp/oneAPI.repo /etc/yum.repos.d
yum install -y intel-oneapi-mpi*
yum install -y sudo wget vi which numactl bzip2 tar gcc hostname util-linux redhat-lsb openssh-
server openssh-clients
```

Launch

When recipe is created, execute the following command:

```
> singularity shell --bind  <path-to-mpi-installation-on-hosts:/mnt> mpiexec -n <number-of-
processes> -ppn <processes-per-node> -hostlist <hosts> <application>
```

See Also

Singularity Official Documentation

Selecting a Library Configuration

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 multi-threading debug or release version with the global lock.

To specify the configuration, run the `vars.bat` file with the `-i_mpi_library_kind` environment variable and the `release` or `debug`. For example:

```
> <install-dir>\env\vars.bat -i_mpi_library_kind=release
```

`-i_mpi_library_kind` environment variable sets the library configuration. See the Intel® MPI Library Developer Reference for details.

You can use the following arguments:

Argument	Definition
release	Set this argument to use multi-threaded optimized library (with the global lock). This is the default value
debug	Set this argument to use multi-threaded debug library (with the global lock)

> **NOTE** You do not need to recompile the application to change the configuration. Run the `vars.bat` file with appropriate arguments before an application launch.

If you want to enable or disable usage of libfabric* from the Intel MPI Library, set the `–i_mpi_ofi_internal` environment variable. See the Intel® MPI Library Developer Reference for details.

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 <number-of-processes> -ppn <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 section mpiexec.hydra in the *Intel® MPI Library Developer Reference*.

See Also

Controlling Process Placement

Job Schedulers Support

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 (release is the default version). To switch to such a configuration, run `vars.bat` with the appropriate argument. See Selecting Library Configuration for details. For example:

```
> vars.bat release
```

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 Intel® compiler `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**.

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.

Fabrics Control

The Intel® MPI Library switched from the Open Fabrics Alliance* (OFA) framework to the Open Fabrics Interfaces* (OFI) framework and currently supports libfabric*.

NOTE The supported fabric environment has changed since Intel® MPI Library 2019. The `dapl` and`tcp` fabrics are now deprecated.

OFI is a framework focused on exporting communication services to applications. OFI is specifically designed to meet the performance and scalability requirements of high-performance computing (HPC) applications running in a tightly coupled network environment. The key components of OFI are application interfaces, provider libraries, kernel services, daemons, and test applications.

Libfabric is a library that defines and exports the user-space API of OFI, and is typically the only software that applications deal with directly. The libfabric's API does not depend on the underlying networking protocols, as well as on the implementation of the particular networking devices, over which it may be implemented. OFI is based on the notion of application centric I/O, meaning that the libfabric library is designed to align fabric services with application needs, providing a tight semantic fit between applications and the underlying fabric hardware. This reduces overall software overhead and improves application efficiency when transmitting or receiving data over a fabric.

For more information, refer to the following topics:

- Selecting Fabrics
- Libfabric* Support
- OFI* Providers Support

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. The following fabrics are available:

Fabric	Network hardware and software used
shm	Shared memory (for intra-node communication only).
ofi	OpenFabrics Interfaces* (OFI)-capable network fabrics, such as Intel® Omni-Path Architecture, InfiniBand* and Ethernet (through OFI API).

Use the `I_MPI_FABRICS` environment variable to specify a fabric. The description is available in the topic Communication Fabrics Control in the *Intel® MPI Library Developer Reference*.

Libfabric* Support

The Intel® MPI Library switched from the Open Fabrics Alliance* (OFA) framework to the Open Fabrics Interfaces* (OFI) framework and currently supports libfabric*.

Enabling Libfabric Support

By default, the script that sets the environmental variables (`vars.bat`) sets the environment to libfabric shipped with the Intel MPI Library. To disable this, use the `I_MPI_OFI_LIBRARY_INTERNAL` environment variable or the `-i_mpi_ofi_internal` option passed to the script:

```
# Do not set the environment to libfabric from the Intel MPI Library.
> call <install-dir>\env\vars.bat -i_mpi_ofi_internal=0

# Set the environment to libfabric from the Intel MPI Library.
> call <install-dir>\env\vars.bat -i_mpi_ofi_internal=1
```

NOTE Set the `I_MPI_DEBUG` environment variable to 1 before running an MPI application to see the libfabric version and provider.

Example

```
> set I_MPI_DEBUG=1
> mpiexec -n 2 IMB-MPI1 pingpong
[0] MPI startup(): libfabric version: 1.5.3-impi
[0] MPI startup(): libfabric provider: sockets

...
```

Supported Providers

- tcp
- netdir

See Also

- Intel® MPI LIbrary 2019 over libfabric
- "OFI-Capable Network Fabrics Control" in the Intel MPI Library Developer Reference

OFI* Providers Support

Intel® MPI Library supports tcp, netdir, psm2, sockets, verbs, and RxM OFI* providers. Each OFI provider is built as a separate dynamic library to ensure that a single libfabric* library can be run on top of different network adapters.

NOTE Use the environment variable `FI_PROVIDER` to select a provider. Set the `FI_PROVIDER_PATH` environment variable to specify the path to provider libraries.

To get a full list of environment variables available for configuring OFI, run the following command:

```
> fi_info  -e
```

tcp

The TCP provider is a general purpose provider for the Intel MPI Library that can be used on any system that supports TCP sockets to implement the libfabric API. The provider lets you run the Intel MPI Library application over regular Ethernet, in a cloud environment that has no specific fast interconnect (e.g., GCP, Ethernet empowered Azure*, and AWS* instances) or using IPoIB.

The following runtime parameters can be used:

Name	Description
`FI_TCP_IFACE`	Specifies a particular network interface.
`FI_TCP_PORT_LOW_RANGE`	Sets the range of ports to be used by the TCP provider for its passive endpoint creation. This is useful when only a range of ports are allowed by the firewall for TCP connections.
`FI_TCP_PORT_HIGH_RANGE`	

netdir

The NETDIR provider runs over NetworkDirect* interface for the compatible hardware. The following runtime parameters can be used:

Name	Description
`FI_NETDIR_MR_CACHE`	Enables memory registration cache (default: 0)
`FI_NETDIR_MR_CACHE_MAX_SIZE`	Sets the maximum memory registration cache size (default: mem/core_count/2)

psm2

The PSM2 provider runs over the PSM 2.x interface supported by the Intel® Omni-Path Fabric. PSM 2.x has all the PSM 1.x features, plus a set of new functions with enhanced capabilities. Since PSM 1.x and PSM 2.x are not application binary interface (ABI) compatible, the PSM2 provider works with PSM 2.x only and does not support Intel® True Scale Fabric.

The following runtime parameters can be used:

Name	Description
FI_PSM2_INJECT_SIZE	Define the maximum message size allowed for fi_inject and fi_tinject calls. The default value is 64.
FI_PSM2_LAZY_CONN	Control the connection mode established between PSM2 endpoints that OFI endpoints are built on top of. When set to 0 (eager connection mode), connections are established when addresses are inserted into the address vector. When set to 1 (lazy connection mode), connections are established when addresses are used the first time in communication. **NOTE** Lazy connection mode may reduce the start-up time on large systems at the expense of higher data path overhead.

sockets

The sockets provider is a general purpose provider that can be used on any system that supports TCP sockets. The provider is not intended to provide performance improvements over regular TCP sockets, but rather to allow developers to write, test, and debug application code even on platforms that do not have high-performance fabric hardware. The sockets provider supports all libfabric provider requirements and interfaces.

The following runtime parameter can be used:

Name	Description
FI_SOCKETS_IFACE	Define the prefix or the name of the network interface. By default, it uses any.

verbs

The verbs provider enables applications using OFI to be run over any verbs hardware (InfiniBand*, iWarp*, and so on). It uses the Linux Verbs API for network transport and provides a translation of OFI calls to appropriate verbs API calls. It uses librdmacm for communication management and libibverbs for other control and data transfer operations.

The verbs provider uses RxM utility provider to emulate FI_EP_RDM endpoint over verbs FI_EP_MSG endpoint by default. The verbs provider with FI_EP_RDM endpoint can be used instead of RxM by setting the FI_PROVIDER=^ofi_rxm runtime parameter.

The following runtime parameters can be used:

Name	Description
FI_VERBS_INLINE_SIZE	Define the maximum message size allowed for fi_inject and fi_tinject calls. The default value is 64.
FI_VERBS_IFACE	Define the prefix or the full name of the network interface associated with the verbs device. The default value is ib.
FI_VERBS_MR_CACHE_ENABLE	Enable Memory Registration caching. The default value is 0. Set this environment variable to enable the memory registration cache.

Name	Description
	NOTE Cache usage substantially improves performance, but may lead to correctness issues.

Dependencies

The verbs provider requires libibverbs (v1.1.8 or newer) and librdmacm (v1.0.16 or newer). If you are compiling libfabric from source and want to enable verbs support, it is essential to have the matching header files for the above two libraries. If the libraries and header files are not in default paths, specify them in the `CFLAGS`, `LDFLAGS`, and `LD_LIBRARY_PATH` environment variables.

RxM

The RxM (RDM over MSG) provider (`ofi_rxm`) is a utility provider that supports `FI_EP_RDM` endpoint emulated over `FI_EP_MSG` endpoint of the core provider.

The RxM provider requires the core provider to support the following features:

- MSG endpoints (`FI_EP_MSG`)
- `FI_MSG` transport (to support data transfers)
- `FI_RMA` transport (to support rendezvous protocol for large messages and RMA transfers)
- `FI_OPT_CM_DATA_SIZE` of at least 24 bytes

The following runtime parameters can be used:

Name	Description
`FI_OFI_RXM_BUFFER_SIZE`	Define the transmit buffer size/inject size. Messages of smaller size are transmitted via an eager protocol and those above would be transmitted via a rendezvous protocol. Transmitted data is copied up to the specified size. By default, the size is 16k.
`FI_OFI_RXM_SAR_LIMIT`	Control the RxM SAR (Segmentation and Reassembly) protocol. Messages of greater size are transmitted via rendezvous protocol.
`FI_OFI_RXM_USE_SRX`	Control the RxM receive path. If the variable is set to 1, the RxM uses Shared Receive Context of the core provider. The default value is 0. **NOTE** Setting this variable to 1 improves memory consumption, but may increase small message latency as a side-effect.

Job Schedulers Support

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

The following job schedulers are supported on Windows* OS:

- Microsoft HPC Pack*
- Altair PBS Pro*

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.

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.

Controlling Per-Host Process Placement

When using a job scheduler, by default the 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
```

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.

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 `mpi exec`. 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.

Using a 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
```

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 node1 -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

Java* MPI Applications Support

The Intel® MPI Library provides an experimental feature to enable support for Java MPI applications. Java bindings are available for a subset of MPI-2 routines. For a full list of supported routines, refer to the Developer Reference, section *Miscellaneous > Java Bindings for MPI-2 Routines*.

Running Java MPI applications

Follow these steps to set up the environment and run your Java MPI application:

1. Source `mpivars.sh` from the Intel® MPI Library package to set up all required environment variables, including `LIBRARY_PATH` and `CLASSPATH`.
2. Build your Java MPI application as usual.
3. Update `CLASSPATH` with the path to the `jar` application or pass it explicitly with the `-cp` option of the `java` command.
4. Run your Java MPI application using the following command:

```
> mpirun <options> java <app>
```

where:

- `<options>` is a list of `mpirun` options
- `<app>` is the main class of your Java application

For example:

```
> mpirun -n 8 -ppn 1 -f ./hostfile java mpi.samples.Allreduce
```

Development Recommendations

You can use the following tips when developing Java* MPI applications:

- To reduce memory footprint, you can use Java direct buffers as buffer parameters of collective operations in addition to using Java arrays. This approach allows you to allocate the memory out of the JVM heap and avoid additional memory copying when passing the pointer to the buffer from JVM to the native layer.
- When you create Java MPI entities such as `Group`, `Comm`, `Datatype`, and similar, memory is allocated on the native layer and is not tracked by the garbage collector. Therefore, this memory must be released explicitly. Pointers to the allocated memory are stored in a special pool and can be deallocated using one of the following methods:

- `entity.free()`: frees the memory backing the `entity` Java object, which can be an instance of `Comm`, `Group`, etc.
- `AllocablePool.remove(entity)`: frees the memory backing the `entity` Java object, which can be an instance of `Comm`, `Group`, etc.
- `AllocablePool.cleanUp()`: explicitly deallocates the memory backing all Java MPI objects created by that moment.
- `MPI.Finalize()`: implicitly deallocates the memory backing all Java MPI objects and that has not been explicitly deallocated by that moment.

Debugging Applications

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

- Debugging
- Using -gtool for Debugging

Debugging

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

GDB*: The GNU Project Debugger

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

```
>  mpiexec -gdb -n 4 testc.exe
```

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 https://sourceware.org/gdb/onlinedocs/gdb/Inferiors-Connections-and-Programs.html#Inferiors-Connections-and-Programs.

You can also attach to a running job with:

```
>  mpiexec -n 4 -gdba <pid>
```

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

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 mpiexec -n <number-of-processes> [<other-mpirun-arguments>] <executable>
```

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

See Also

Using -gtool for Debugging

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:

```
> mpiexec -n 16 -gtool "gdb:3,5,7-9=attach" myprog.exe
```

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

See Also

Intel® MPI Library Developer Reference, topic gtool Options.

Analysis and Tuning

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.

Displaying MPI 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
-genv I_MPI_DEBUG=2 -n 2 testc[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 the 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, topic Other Environment Variables.

Tracing Applications

The 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 these tools, the Intel MPI Library provides compilation and runtime options and environment variables for easier interoperability.

The Intel MPI Library is tightly integrated with the Intel® Trace Analyzer and Collector, which enables you to analyze and debug MPI applications. The Intel MPI Library has several compile-time options to simplify the application analysis.

Intel Trace Analyzer and Collector is available as standalone software and as part of the Intel® oneAPI HPC Toolkit. Before proceeding to the next steps, make sure you have the product installed.

Trace an Application

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.

```
> run <compiler-install-dir>\env\vars.bat
```

```
> run <itac-install-dir>\env\vars.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.

See Also

Intel Trace Collector User and Reference Guide

MPI Tuning

Intel® MPI Library provides the following tuning utilities:

- Autotuner
- mpitune

Autotuner

Autotuner is the recommended utility for the application-specific tuning. If an application is spending significant time in MPI collective operations, autotuning might improve its performance. Autotuner is easy-to-use, and its overhead is close to zero.

The autotuning utility's tuning scope is `I_MPI_ADJUST_<opname>` family of environment variables, which are MPI collective operation algorithms. Autotuner limits tuning to the current cluster configuration (fabric, number of ranks, number of ranks per node). It works while an application is running, so performance could be potentially improved just by enabling the autotuner. It is also possible to generate new tuning file with MPI collective operations adjusted to application needs, and this file can be further passed to the `I_MPI_TUNING_BIN` variable.

mpitune

`mpitune` is useful If the search space of the autotuner is not sufficient for your needs. `mpitune` iteratively launches a benchmarking application with different configurations to measure performance and stores the results of each launch. Based on these results, the tuner generates optimal values for parameters that are being tuned. `mpitune` has an ability to search for optimal values of variables other than I_MPI_ADJUST_<opname>, and it could be used for application-specific and cluster-wide tuning. For example, it could tune parameters (like radix) of collective operations.

Differences between the tuning utilities:

Parameter	Autotuner	mpitune
Low tuning overhead	+	-
Ease of use	+	-
Application tuning	+	+
Microbenchmark tuning	+	+
Tuning beyond collective operations	-	+

Troubleshooting

This section provides the troubleshooting information on typical MPI failures with corresponding output messages and behavior when a failure occurs.

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

1. Check the Intel® MPI Library 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 the Support page, or ask experts on the community forum.

Error Message: Bad Termination

NOTE: The values in the tables below may not reflect the exact node or MPI process where a failure can occur.

Case 1

Error Message

```
===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   RANK 1 PID 27494 RUNNING AT node1
=   KILLED BY SIGNAL: 11 (Segmentation fault)
===================================================================================
```

or:

```
===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   RANK 1 PID 27494 RUNNING AT node1
=   KILLED BY SIGNAL: 8 (Floating point exception)
===================================================================================
```

Cause

One of MPI processes is terminated by a signal (for example, `Segmentation fault` or `Floating point exception`) on the `node01`.

Solution

Find the reason of the MPI process termination. It can be the out-of-memory issue in case of `Segmentation fault` or division by zero in case of `Floating point exception`.

Case 2

Error Message

```
=================================================================================
= BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
= RANK 1 PID 20066 RUNNING AT node01
= KILLED BY SIGNAL: 9 (Killed)
=================================================================================
```

Cause

One of MPI processes is terminated by a signal (for example, `SIGTERM` or `SIGKILL`) on the `node01` due to:

- the host reboot;
- an unexpected signal received;
- out-of-memory manager (OOM) errors;
- killing by the process manager (if another process was terminated before the current process);
- job termination by the Job Scheduler (PBS Pro*, SLURM*) in case of resources limitation (for example, walltime or cputime limitation).

Solution

1. Check the system log files.
2. Try to find the reason of the MPI process termination and fix the issue.

Error Message: No such file or Directory

Error Message

```
[proxy:0:0@node1] HYD_spawn
(../../../../../src/pm/i_hydra/libhydra/spawn/hydra_spawn.c:113): execvp error on file {path to
binary file}/{binary file} (No such file or directory)
```

Cause

Wrong path to the binary file or the binary file does not exist on the `node01`. The name of the binary file is misprinted or the shared space cannot be reached.

Solution

Check the name of the binary file and check if the shared path is available across all the nodes.

Error Message: Permission Denied

Case 1

Error Message

```
[proxy:0:0@node1] HYD_spawn
(../../../../../src/pm/i_hydra/libhydra/spawn/hydra_spawn.c:113): execvp error on file {path to
binary file}/{binary file} (Permission denied)
```

Cause

You do not have permissions to execute the binary file.

Solution

Check your execute permissions for `{binary file}` and for folders in `{path to binary file}`.

Case 2

Error Message

```
[proxy:0:0@node1] HYD_spawn
(../../../../../src/pm/i_hydra/libhydra/spawn/hydra_spawn.c:113): execvp error on file {path to
binary file}/{binary file} (Permission denied)
```

Cause

You exceeded the limitation of 16 groups on Linux* OS.

Solution

Try reducing the number of groups.

Error Message: Fatal Error

Case 1

Error Message

```
Abort(1094543) on node 0 (rank 0 in comm 0): Fatal error in PMPI_Init: Other MPI error, error
stack:
MPIR_Init_thread(653)......:
```

```
MPID_Init(860).............:
MPIDI_NM_mpi_init_hook(698): OFI addrinfo() failed
(ofi_init.h:698:MPIDI_NM_mpi_init_hook:No data available)
```

Cause

The current provider cannot be run on these nodes. The MPI application is run over the `psm2` provider on the non-Intel® Omni-Path card or over the `verbs` provider on the non-InfiniBand*, non-iWARP, or non-RoCE card.

Solution

1. Change the provider or run MPI application on the right nodes. Use `fi_info` to get information about the current provider.
2. Check if services are running on nodes (opafm for Intel® Omni-Path and opensmd for InfiniBand).

Case 2

Error Message

```
Abort(6337423) on node 0 (rank 0 in comm 0): Fatal error in PMPI_Init_thread:
Other MPI error, error stack:
…
MPIDI_OFI_send_handler(704)............: OFI tagged inject failed
(ofi_impl.h:704:MPIDI_OFI_send_handler:Transport endpoint is not connected)
```

Cause

OFI transport uses IP interface without access to remote ranks.

Solution

Set `FI_SOCKET_IFACE` If the `socket` provider is used or `FI_TCP_IFACE` and `FI_VERBS_IFACE` in case of `TCP` and `verbs` providers, respectively. To retrieve the list of configured and active IP interfaces, use the `ifconfig` utility.

Case 3

Error Message

```
Abort(6337423) on node 0 (rank 0 in comm 0): Fatal error in PMPI_Init_thread:
Other MPI error, error stack:
…
MPIDI_OFI_send_handler(704)............: OFI tagged inject failed
(ofi_impl.h:704:MPIDI_OFI_send_handler:Transport endpoint is not connected)
```

Cause

Ethernet is used as an interconnection network.

Solution

Run `FI_PROVIDER = sockets mpirun …` to overcome this problem.

Error Message: Bad File Descriptor

Error Message

```
[mpiexec@node00] HYD_sock_write (../../../../../src/pm/i_hydra/libhydra/sock/
hydra_sock_intel.c:353): write error (Bad file descriptor)
[mpiexec@node00] cmd_bcast_root (../../../../../src/pm/i_hydra/mpiexec/mpiexec.c:147): error
sending cwd cmd to proxy
[mpiexec@node00] stdin_cb (../../../../../src/pm/i_hydra/mpiexec/mpiexec.c:324): unable to send
response downstream
```

```
[mpiexec@node00] HYDI_dmx_poll_wait_for_event (../../../../../src/pm/i_hydra/libhydra/demux/
hydra_demux_poll.c:79): callback returned error status
[mpiexec@node00] main (../../../../../src/pm/i_hydra/mpiexec/mpiexec.c:2064): error waiting for
event
```

or:

```
[mpiexec@host1] wait_proxies_to_terminate (../../../../../src/pm/i_hydra/mpiexec/intel/
i_mpiexec.c:389): downstream from host host2 exited with status 255
```

Cause

The remote hydra_pmi_proxy process is unavailable due to:

- the host reboot;
- an unexpected signal received;
- out-of-memory manager (OOM) errors;
- job termination by the Job Scheduler (PBS Pro*, SLURM*) in case of resources limitation (for example, walltime or cputime limitation).

Solution

1. Check the system log files.
2. Try to find the reason of the hydra_pmi_proxy process termination and fix the issue.

Error Message: Too Many Open Files

Error Message

```
[proxy:0:0@host1] HYD_spawn (../../../../../src/pm/i_hydra/libhydra/spawn/intel/
hydra_spawn.c:57): pipe error (Too many open files)
[proxy:0:0@host1] launch_processes (../../../../../src/pm/i_hydra/proxy/proxy.c:509): error
creating process
[proxy:0:0@host1] main (../../../../../src/pm/i_hydra/proxy/proxy.c:860): error
launching_processes
```

Cause

Too many processes per node are launched on Linux* OS.

Solution

Specify fewer processes per node by the -ppn option or the I_MPI_PERHOST environment variable.

Problem: MPI Application Hangs

Problem

MPI application hangs without any output.

Case 1

Cause

Application does not use MPI in a correct way.

Solution

Run your MPI application with the -check_mpi option to perform correctness checking. The correctness checker is specifically designed to find MPI errors, and provides tight integration with the Intel® MPI Library. In case of a deadlock, the checker will set up a one-minute timeout and show the state of each rank.

For more information, refer to this page.

Case 2

Cause

The remote service (for example, SSH) is not running on all nodes or it is not configured properly.

Solution

Check the state of the remote service on the nodes and connection to all nodes.

Case 3

Cause

The Intel® MPI Library runtime scripts are not available, so the shared space cannot be reached.

Solution

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

Case 4

Cause

Different CPU architectures are used in a single MPI run.

Solution

Set `export I_MPI_PLATFORM=<arch>` , where `<arch>` is the oldest platform you have, for example `skx`. Note that usage of different CPU architectures in a single MPI job negatively affects application performance, so it is recommended not to mix different CPU architecture in a single MPI job.

Problem: Password Required

Problem

Password required.

Cause

The Intel® MPI Library uses SSH mechanism to access remote nodes. SSH requires password and this may cause the MPI application hang.

Solution

1. Check the SSH settings.
2. Make sure that the passwordless authorization by public keys is enabled and configured.

Problem: Cannot Execute Binary File

Problem

Cannot execute a binary file.

Cause

Wrong format or architecture of the binary executable file.

Solution

Check the accuracy of the binary file and command line options.

Problem: MPI limitation for Docker*

Problem

The command fails with the following message:

```
[root@n1 /]# I_MPI_DEBUG=12 mpirun -n 2 -ppn 1 -env I_MPI_PIN_DOMAIN socket IMB-MPI1 bcast
impi_shm_heap_init(): mbind failed (p=0x7f3078b0e000, size=536870912)
impi_shm_heap_init(): mbind failed (p=0x7f9b808bc000, size=536870912)
```

Cause

MPI has a limitation on the `dev/shm` area. It should be not less than 4GB for a node with 2 sockets. By default, the Docker* container set 64MB, which is not enough.

Solution

1. Make sure the problem is the small size of `shm` area:

```
$df -h /dev/shm
root@n1 /]# df -h /dev/shm
Filesystem      Size Used Avail Use% Mounted on
shm             4.0G   0  4.0G   0%  /dev/shm
```

2. If it is true, restart Docker using the following command:

```
docker run --shm-size=4gb …
```

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

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

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

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.