Intel® MPI Library for Intel® oneAPI on Windows* OS

Developer Guide

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*.

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.

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 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).

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

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 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 for the Intel MPI Library. Follow these steps:

1. Set up the Intel MPI Library environment. For oneAPI, run the `setvars.bat script`:

   ```
   > <install-dir>\env\vars.bat
   For Parallel Studio XE, run the vars.bat script:

   > <install-dir>\intel64\bin\mpivars.bat
   ```

   By default, <*install-dir*>for oneAPI is `C:\Program Files (x86)\inteloneapi\mpi\<version>` and for Parallel Studio XE is C:\Program Files(x86)\IntelSWTools\compilers_and_libraries_<*version*>\windows\mpi.

   NOTE: The `vars.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 `vars.bat script`.

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
```

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.

User Authorization

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.
The domain-based authorization methods use the Security Service Provider Interface (SSPI) provided by Microsoft in a Windows environment. The SSPI allows domain to authenticate the user on the remote machine in accordance with 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. Register service principal name (SPN) for cluster nodes. Use one of the following methods for
   registering SPN:
   a. Use the Microsoft*-provided `setspn.exe` utility. 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 the command:

```
> klist purge
```

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

Compiling and Linking

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 (`vars.bat` for oneAPI, `compilervars.bat` for PSXE) 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
   ```
   For oneAPI:
   ```
   > <installdir>\env\vars.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 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*.

See Also

Intel® MPI Library Developer Reference, section *Command Reference* > *Compiler Commands*

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

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>` for oneAPI is `C:\Program Files(x86)\inteloneapi\mpi\<version` and for PSXE is `C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_<version>\windows\mpi` by default.

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>`:
   - `release`: multi-threaded optimized library
   - `debug`: multi-threaded debug 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.

Running Applications

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

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` for oneAPI or `mpivars.bat` for PSXE with the appropriate argument, see Selecting Library Configuration for details. For example:
   ```
   > vars.bat release - for oneAPI
   ```

```
> mpivars.bat release - for PSXE
```

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.

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® True Scale Fabric, 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 *Developer Reference*, section *Tuning Reference > Fabrics Control*.

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 script for oneAPI or the mpivars.bat script for PSXE with release, or debug,`
release_mt, or debug_mt argument. For example:

```
> <installdir>\env\vars.bat release

For                                                                    PSXE:
> <installdir>\intel64\bin\mpivars.bat release
```

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` script for oneAPI or the `mpivars.bat` script for PSXE with appropriate arguments before an application launch.

Libfabric* Support

Enabling Libfabric support

By default, the script that sets the environmental variables (`vars.bat` for oneAPI and `mpivars.bat` for PSXE) 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 `-ofi_internal` option passed to the script.
For Intel oneAPI:

```
> call <install-dir>\bin\mpivars.bat -ofi_internal=0 # do not set the environment to libfabric from the Intel MPI Library

> call <install-dir>\bin\mpivars.bat # set the environment to libfabric from the Intel MPI Library
```

For Intel Parallel Studio XE:

```
> call <install-dir>\intel64\bin\mpivars.bat -ofi_internal=0 # do not set the environment to libfabric from the Intel MPI Library

> call <install-dir>\intel64\bin\mpivars.bat # set the environment to libfabric from the Intel MPI Library
```

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

- libfabric.dll

See also

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

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*

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 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 `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.

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
```

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

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.

For step-by-step instructions on tuning an MPI application, see this article:

Tuning the Intel® MPI Library: Basic Techniques

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
[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*

Tracing Applications

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.

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.

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.

   For Intel oneAPI:

   ```
   > <compiler_installdir>\env\vars.bat intel64

   > <itac_installdir>\env\vars.bat
   ```

   For Intel Parallel Studio XE:

   ```
   > <compiler_installdir>\bin\compilervars.bat intel64
   > <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.

See Also

Intel® Trace Collector User and Reference Guide

MPI Tuning

Intel® MPI Library includes the `mpitune` tuning utility, which allows you to automatically adjust Intel MPI Library parameters, such as collective operation algorithms, to your cluster configuration or application. The tuner 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 the parameters that are being tuned.

Note

The `mpitune` usage model changed in the 2018 release. Tuning parameters should be specified in configuration files rather than as command-line options.

Configuration File Format

All tuner parameters should be specified in two configuration files, passed to the tuner with the `--config-file` option. A typical configuration file consists of the main section, specifying generic options, and search space sections for specific library parameters (for example, for specific collective operations). Configuration files differ in mode and dump-file fields only. To comment a line, use the hash symbol #.

You can also specify MPI options to simplify `mpitune` usage. MPI options are useful for Intel® MPI Benchmarks that have special templates for `mpitune` located at `<installdir>/etc/tune_cfg`. The templates require no changes in configuration files to be made.

For example, to tune the `Bcast` collective algorithm, use the following option:

```
$ mpitune -np 2 -ppn 2 -hosts HOST1 -m analyze -c /path/to/Bcast.cfg
```

Experienced users can change configuration files to use this option for other applications.

Output Format

The tuner presents results in a JSON tree view (since the 2019 release), where the `comm_id=-1` layer is added automatically for each tree:

```
{
  "coll=Reduce":                                                    {
    "ppn=2":                                                        {
      "comm_size=2":                                                {
        "comm_id=-1":                                               {
          "msg_size=243":                                           {
            "REDUCE=8":                                            {}
          },
          "msg_size=319":                                           {
            "REDUCE=11":                                           {}
          },
          "msg_size=8192":                                          {
```

```
            "REDUCE=8":                                                    {}
          },
          "msg_size=28383":                                                 {
            "REDUCE=9":                                                    {}
          },
          "msg_size=-1":                                                    {
            "REDUCE=1":                                                    {}
          }
        }
      }
    }
  }
}
```

To add the resulting JSON tree to the library, use the `I_MPI_TUNING` environment variable.

Old Output Format

The old output format is only valid for Intel MPI Library 2018 and prior versions:

```
I_MPI_ADJUST_BCAST=2:0-0;1:1-64;2:65-509;1:510-8832;3:8833-0
```

Use the resulting variable value with the application launch to achieve performance gain.

See Also

For details on `mpitune` configuration options, refer to the Developer Reference section *Command Reference > mpitune*.

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 *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 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.

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.

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.

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.