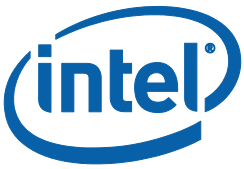


Intel® MPI Library for Linux* OS

Developer Reference

Contents

Legal Information

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

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

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

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

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

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

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

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

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

1. Introduction

This *Developer Reference* provides you with the complete reference for the Intel® MPI Library. It is intended to help an experienced user fully utilize the Intel MPI Library functionality. You can freely redistribute this document in any desired form.

1.1. Introducing Intel® MPI Library

Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, v3.1 (MPI-3.1) specification. It provides a standard library across Intel® platforms that enable adoption of MPI-3.1 functions as their needs dictate.

Intel® MPI Library enables developers to change or to upgrade processors and interconnects as new technology becomes available without changes to the software or to the operating environment.

Intel® MPI Library comprises the following main components:

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

You can get the latest information of Intel® MPI Library at https://software.intel.com/intel-mpi-library.

1.2. What's New

This document reflects the updates for Intel® MPI Library 2018 release for Linux* OS:

The following latest changes in this document were made:

Intel MPI Library 2018

- Removed support of the Intel® Xeon Phi™ coprocessors (formerly code named Knights Corner).
- Changes in environment variables:
  - `I_MPI_DAPL_TRANSLATION_CACHE` is now disabled by default
  - `I_MPI_HARD_FINALIZE` is now enabled by default for the OFI and TMI fabrics
  - `I_MPI_JOB_FAST_STARTUP` is now intended for OFI and TMI fabrics only
  - Default value change for `I_MPI_FABRICS_LIST`
- The `-mps` option has been replaced with `-aps`.
- Added environment variables `I_MPI_{C,CXX,FC,F}FLAGS`, `I_MPI_LDFLAGS` and `I_MPI_FORT_BIND` in Compilation Environment Variables.
- Added environment variables `I_MPI_OFI_ENABLE_LMT` and `I_MPI_OFI_MAX_MSG_SIZE` in OFI-capable Network Fabrics Control.

Intel MPI Library 2017 Update 2

- Added the environment variable `I_MPI_HARD_FINALIZE` in Other Environment Variables.
- Added the environment variable `I_MPI_MEMORY_SWAP_LOCK` in Memory Placement Policy Control.

Intel MPI Library 2017 Update 1

- The environment variable `I_MPI_SLURM_EXT` (Other Environment Variables) is now enabled by default.
- Added a new algorithm for `I_MPI_ADJUST_GATHER` and related environment variable `I_MPI_ADJUST_GATHER_SEGMENT` (I_MPI_ADJUST Family).
- Added the environment variable `I_MPI_PORT_RANGE` in Hydra Environment Variables.

Intel MPI Library 2017

- Document layout changes.
- Updated the topic Memory Placement Policy Control.
- Added the environment variables `I_MPI_OFI_DIRECT_RMA` and `I_MPI_OFI_DSEND` in OFI*-capable Network Fabrics Control.
- Added a new topic Asynchronous Progress Control.
- Added the environment variable `I_MPI_LUSTRE_STRIPE_AWARE` in File System Support.
- Added the environment variable `I_MPI_SLURM_EXT` in Other Environment Variables.
- Updated the Table: Environment Variables, Collective Operations, and Algorithms in I_MPI_ADJUST Family.
- Added the following environment variables in I_MPI_ADJUST Family:
  - `I_MPI_ADJUST_<COLLECTIVE>_SHM_KN_RADIX`
  - `I_MPI_COLL_INTRANODE`
  - `I_MPI_COLL_INTRANODE_SHM_THRESHOLD`

1.3. Notational Conventions

The following conventions are used in this document.

This type style	Document or product names
This type style	Hyperlinks
`This type style`	Commands, arguments, options, file names
`THIS_TYPE_STYLE`	Environment variables
`<this type style>`	Placeholders for actual values
`[ items ]`	Optional items
`{ item | item }`	Selectable items separated by vertical bar(s)
(SDK only)	Functionality available for Software Development Kit (SDK) users only

1.4. Related Information

Description of some of the Intel® MPI Library functionality is available in `man1` pages: `mpiexec.hydra`, `hydra_persist`, `hydra_nameserver`, and compiler wrappers.

The following related documents that might be useful to the user:

- Product Web Site
- Intel® MPI Library Support
- Intel® Cluster Tools Products
- Intel® Software Development Products

2. Command Reference

2.1. Compiler Commands

(SDK only)

The following table lists the available Intel® MPI Library compiler commands with their underlying compilers and programming languages.

Table 2.1-1 Intel® MPI Library Compiler Wrappers

Compiler Command	Default Compiler	Supported Language(s)
Generic Compilers		
`mpicc`	`gcc, cc`	C
`mpicxx`	`g++`	C/C++
`mpifc`	`gfortran`	Fortran77*/Fortran 95*
GNU* Compilers		
`mpigcc`	`gcc`	C
`mpigxx`	`g++`	C/C++
`mpif77`	`g77`	Fortran 77
`mpif90`	`gfortran`	Fortran 95
Intel® Fortran, C++ Compilers		
`mpiicc`	`icc`	C
`mpiicpc`	`icpc`	C++
`mpiifort`	`ifort`	Fortran77/Fortran 95

NOTES:

- Compiler commands are available only in the Intel® MPI Library Software Development Kit (SDK).
- For the supported versions of the listed compilers, refer to the *Release Notes*.
- Compiler wrapper scripts are located in the *<installdir>*`/intel64/bin` directory, where *<installdir>* is the Intel® MPI Library installation directory.
- The environment settings can be established by sourcing the *<installdir>*`/intel64/bin/mpivars.[c]sh` script. If you need to use a specific library configuration, you can pass one of the following arguments to the `mpivars.[c]sh` script to switch to the corresponding configuration: `debug`, `release`, `debug_mt`, or `release_mt`. The multi-threaded

optimized library is chosen by default. Alternatively, you can use the `I_MPI_LIBRARY_KIND` environment variable to specify a configuration and source the script without arguments.

- Ensure that the corresponding underlying compiler is already in your `PATH`. If you use the Intel® Compilers, source the `compilervars.[c]sh` script from the installation directory to set up the compiler environment.
- To display mini-help of a compiler command, execute it without any parameters.

2.1.1. Compiler Command Options

-static_mpi

Use this option to link the Intel® MPI Library statically. This option does not affect the default linkage method for other libraries.

-static

Use this option to link the Intel® MPI Library statically. This option is passed to the compiler.

-nostrip

Use this option to turn off the debug information stripping while linking the Intel® MPI Library statically.

-config=*<name>*

Use this option to source a compiler configuration file. The file should contain the environment settings to be used with the specified compiler.

Use the following naming convention for configuration files:

<installdir>/*<arch>*/etc/mpi*<compiler>*-*<name>*.conf

where:

- *<arch>* = {intel64,mic} for the Intel® 64 architecture and Intel® Many Integrated Core architecture, respectively.
- *<compiler>* = {cc,cxx,f77,f90}, depending on the language compiled.
- *<name>* is the name of the underlying compiler with spaces replaced by hyphens; for example, the *<name>* value for `cc -64` is `cc--64`.

-profile=*<profile_name>*

Use this option to specify an MPI profiling library. *<profile_name>* is the name of the configuration file (profile) that loads the corresponding profiling library. The profiles are taken from *<installdir>*/*<arch>*/etc.

Intel® MPI Library comes with several predefined profiles for the Intel® Trace Collector:

- *<installdir>*/*<arch>*/etc/vt.conf – regular tracing library
- *<installdir>*/*<arch>*/etc/vtfs.conf – fail-safe tracing library
- *<installdir>*/*<arch>*/etc/vtmc.conf – correctness checking library
- *<installdir>*/*<arch>*/etc/vtim.conf – load imbalance tracing library

You can also create your own profile as *<profile_name>*.conf. You can define the following environment variables in a configuration file:

- `PROFILE_PRELIB` – libraries (and paths) to load before the Intel® MPI Library
- `PROFILE_POSTLIB` – libraries to load after the Intel® MPI Library

- PROFILE_INCPATHS – C preprocessor arguments for any include files

For example, create a file myprof.conf with the following lines:

```
PROFILE_PRELIB="-L<path_to_myprof>/lib -lmyprof"
PROFILE_INCPATHS="-I<paths_to_myprof>/include"
```

Use the -profile=myprof option for the relevant compiler wrapper to select this new profile.

-t or -trace

Use the -t or -trace option to link the resulting executable file against the Intel® Trace Collector library. Using this option has the same effect as the -profile=vt option.

You can also use the I_MPI_TRACE_PROFILE environment variable to <profile_name> to specify another profiling library. For example, set I_MPI_TRACE_PROFILE to vtfs to link against the fail-safe version of the Intel® Trace Collector.

To use this option, include the installation path of the Intel® Trace Collector in the VT_ROOT environment variable. Source the itacvars.[c]sh script provided in the Intel® Trace Analyzer and Collector installation folder.

-trace-imbalance

Use the -trace-imbalance option to link the resulting executable file against the load imbalance tracing library of Intel® Trace Collector. Using this option has the same effect as the -profile=vtim option.

To use this option, include the installation path of the Intel® Trace Collector in the VT_ROOT environment variable. Source the itacvars.[c]sh script provided in the Intel® Trace Analyzer and Collector installation folder.

-check_mpi

Use this option to link the resulting executable file against the Intel® Trace Collector correctness checking library. Using this option has the same effect as the -profile=vtmc option.

You can also use the I_MPI_CHECK_PROFILE environment variable to <profile_name> to specify another checking library.

To use this option, include the installation path of the Intel® Trace Collector in the VT_ROOT environment variable. Source the itacvars.[c]sh script provided in the Intel® Trace Analyzer and Collector installation folder.

-ilp64

Use this option to enable partial ILP64 support. All integer arguments of the Intel MPI Library are treated as 64-bit values in this case.

-no_ilp64

Use this option to disable the ILP64 support explicitly. This option must be used in conjunction with -i8 option of Intel® Fortran Compiler.

NOTE

If you specify the -i8 option for the separate compilation with Intel® Fortran Compiler, you still have to use the i8 or ilp64 option for linkage. See ILP64 Support for details.

-dynamic_log

Use this option in combination with the `-t` option to link the Intel® Trace Collector library dynamically. This option does not affect the default linkage method for other libraries.

To run the resulting programs, include `$VT_ROOT/slib` in the `LD_LIBRARY_PATH` environment variable.

-g

Use this option to compile a program in debug mode and link the resulting executable file against the debugging version of the Intel® MPI Library. See `I_MPI_DEBUG` for information on how to use additional debugging features with the `-g` builds.

NOTE

The optimized library is linked with the `-g` option by default.

NOTE

Use `mpivars.{sh|csh} [debug|debug_mt]` at runtime to load a particular `libmpi.so` configuration.

-link_mpi=<*arg*>

Use this option to always link the specified version of the Intel® MPI Library. See the `I_MPI_LINK` environment variable for detailed argument descriptions. This option overrides all other options that select a specific library.

NOTE

Use `mpivars.{sh|csh}[debug|debug_mt]` during runtime to load particular `libmpi.so` configuration.

-O

Use this option to enable compiler optimization.

-fast

Use this option to maximize speed across the entire program. This option forces static linkage method for the Intel® MPI Library.

NOTE

This option is supported only by the `mpiicc`, `mpiicpc`, and `mpiifort` Intel® compiler wrappers.

-echo

Use this option to display everything that the command script does.

-show

Use this option to learn how the underlying compiler is invoked, without actually running it. Use the following command to see the required compiler flags and options:

```
$ mpiicc -show -c test.c
```

Use the following command to see the required link flags, options, and libraries:

```
$ mpiicc -show -o a.out test.o
```

This option is particularly useful for determining the command line for a complex build procedure that directly uses the underlying compilers.

-show_env

Use this option to see the environment settings in effect when the underlying compiler is invoked.

-{cc,cxx,fc,f77,f90}=<*compiler*>

Use this option to select the underlying compiler.

For example, use the following command to select the Intel® C++ Compiler:

```
$ mpicc -cc=icc -c test.c
```

Make sure `icc` is in your `PATH`. Alternatively, you can specify the full path to the compiler.

-nofortbind, -nofortran

Use this option to disable `mpiicc` linking with Fortran bindings. Has the same effect as the `I_MPI_FORT_BIND` variable.

-compchk

Use this option to enable compiler setup checks. In this case, each compiler driver performs checks to ensure that the appropriate underlying compiler is set up correctly.

-v

Use this option to print the compiler wrapper script version and its underlying compiler version.

2.1.2. Compilation Environment Variables

I_MPI_{CC,CXX,FC,F77,F90}_PROFILE (MPI{CC,CXX,FC,F77,F90}_PROFILE)

Specify the default profiling library.

Syntax

`I_MPI_CC_PROFILE=`<*profile_name*>

`I_MPI_CXX_PROFILE=`<*profile_name*>

`I_MPI_FC_PROFILE=`<*profile_name*>

`I_MPI_F77_PROFILE=`<*profile_name*>

`I_MPI_F90_PROFILE=`<*profile_name*>

Deprecated Syntax

`MPICC_PROFILE=`<*profile_name*>

`MPICXX_PROFILE=`<*profile_name*>

`MPIFC_PROFILE=`<*profile_name*>

`MPIF77_PROFILE=`<*profile_name*>

`MPIF90_PROFILE=`<*profile_name*>

Arguments

<profile_name>	Specify a default profiling library.

Description

Set this environment variable to select a specific MPI profiling library to be used by default. This has the same effect as using -profile=*<profile_name>* as an argument for mpiicc or another Intel® MPI Library compiler wrapper.

I_MPI_TRACE_PROFILE

Specify the default profile for the -trace option.

Syntax

I_MPI_TRACE_PROFILE=*<profile_name>*

Arguments

<profile_name>	Specify a tracing profile name. The default value is vt.

Description

Set this environment variable to select a specific MPI profiling library to be used with the -trace option of mpiicc or another Intel® MPI Library compiler wrapper.

The I_MPI_{CC,CXX,F77,F90}_PROFILE environment variable overrides I_MPI_TRACE_PROFILE.

I_MPI_CHECK_PROFILE

Specify the default profile for the -check_mpi option.

Syntax

I_MPI_CHECK_PROFILE=*<profile_name>*

Arguments

<profile_name>	Specify the checking profile name. The default value is vtmc.

Description

Set this environment variable to select a specific MPI checking library to be used with the -check_mpi option to mpiicc or another Intel® MPI Library compiler wrapper.

The I_MPI_{CC,CXX,F77,F90}_PROFILE environment variable overrides I_MPI_CHECK_PROFILE.

I_MPI_CHECK_COMPILER

Turn on/off compiler compatibility check.

Syntax

I_MPI_CHECK_COMPILER=*<arg>*

Arguments

<arg>	Binary indicator.
enable | yes | on | 1	Enable checking the compiler.

`disable | no | off | 0`	Disable checking the compiler. This is the default value.

Description

If `I_MPI_CHECK_COMPILER` is set to `enable`, the Intel MPI Library compiler wrapper checks the underlying compiler for compatibility. Normal compilation requires using a known version of the underlying compiler.

I_MPI_{CC,CXX,FC,F77,F90} (MPICH_{CC,CXX,FC,F77,F90})

Set the path/name of the underlying compiler to be used.

Syntax

`I_MPI_CC=`*`<compiler>`*

`I_MPI_CXX=`*`<compiler>`*

`I_MPI_FC=`*`<compiler>`*

`I_MPI_F77=`*`<compiler>`*

`I_MPI_F90=`*`<compiler>`*

Deprecated Syntax

`MPICH_CC=`*`<compiler>`*

`MPICH_CXX=`*`<compiler>`*

`MPICH_FC=`*`<compiler>`*

`MPICH_F77=`*`<compiler>`*

`MPICH_F90=`*`<compiler>`*

Arguments

`<compiler>`	Specify the full path/name of compiler to be used.

Description

Set this environment variable to select a specific compiler to be used. Specify the full path to the compiler if it is not located in the search path.

NOTE

Some compilers may require additional command line options.

NOTE

The configuration file is sourced if it exists for a specified compiler. See -config for details.

I_MPI_ROOT

Set the Intel® MPI Library installation directory path.

Syntax

`I_MPI_ROOT=`*`<path>`*

Arguments

<path>	Specify the installation directory of the Intel® MPI Library

Description

Set this environment variable to specify the installation directory of the Intel® MPI Library.

VT_ROOT

Set Intel® Trace Collector installation directory path.

Syntax

`VT_ROOT=`*`<path>`*

Arguments

<path>	Specify the installation directory of the Intel® Trace Collector

Description

Set this environment variable to specify the installation directory of the Intel® Trace Collector.

I_MPI_COMPILER_CONFIG_DIR

Set the location of the compiler configuration files.

Syntax

`I_MPI_COMPILER_CONFIG_DIR=`*`<path>`*

Arguments

<path>	Specify the location of the compiler configuration files. The default value is *<installdir>*`/`*<arch>*`/etc`

Description

Set this environment variable to change the default location of the compiler configuration files.

I_MPI_LINK

Select a specific version of the Intel® MPI Library for linking.

Syntax

`I_MPI_LINK=<arg>`

Arguments

<arg>	Version of library
`opt`	The optimized, single threaded version of the Intel® MPI Library
`opt_mt`	The optimized, multithreaded version of the Intel MPI Library
`dbg`	The debugging, single threaded version of the Intel MPI Library
`dbg_mt`	The debugging, multithreaded version of the Intel MPI Library

Description

Set this variable to always link against the specified version of the Intel® MPI Library.

I_MPI_DEBUG_INFO_STRIP

Turn on/off the debug information stripping while linking applications statically.

Syntax

`I_MPI_DEBUG_INFO_STRIP=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on. This is the default value
`disable | no | off | 0`	Turn off

Description

Use this option to turn on/off the debug information stripping while linking the Intel® MPI Library statically. Debug information is stripped by default.

I_MPI_{C,CXX,FC,F}FLAGS

Set special flags needed for compilation.

Syntax

`I_MPI_CFLAGS=<flags>`

`I_MPI_CXXFLAGS=<flags>`

`I_MPI_FCFLAGS=<flags>`

`I_MPI_FFLAGS=<flags>`

Arguments

<flags>	Flag list

Description

Use this environment variable to specify special compilation flags.

I_MPI_LDFLAGS

Set special flags needed for linking.

Syntax

`I_MPI_LDFLAGS=<flags>`

Arguments

<flags>	Flag list

Description

Use this environment variable to specify special linking flags.

I_MPI_FORT_BIND

Disable `mpiicc` linking with Fortran bindings.

Syntax

`I_MPI_FORT_BIND=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable linking. This is the default value
`disable | no | off | 0`	Disable linking

Description

By default, the `mpiicc` also links against the Fortran bindings even if Fortran is not used. Use this environment variable to change this default behavior. Has the same effect as the `-nofortbind` option.

2.2. Simplified Job Startup Command

mpirun

Syntax

`mpirun <options>`

where *<options>:= <mpiexec.hydra options>*

Arguments

<mpiexec.hydra options>	`mpiexec.hydra` options as described in the `mpiexec.hydra` section. This is the default operation mode.

Description

Use this command to launch an MPI job. The `mpirun` command uses Hydra* as the underlying process manager.

The `mpirun` command detects if the MPI job is submitted from within a session allocated using a job scheduler like Torque*, PBS Pro*, LSF*, Parallelnavi* NQS*, SLURM*, Univa* Grid Engine*, or LoadLeveler*. The `mpirun` command extracts the host list from the respective environment and uses these nodes automatically according to the above scheme.

In this case, you do not need to create a host file. Allocate the session using a job scheduler installed on your system, and use the `mpirun` command inside this session to run your MPI job.

Example

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

This command invokes the `mpiexec.hydra` command (Hydra process manager), which launches the `myprog` executable.

2.3. Hydra Process Manager Command

mpiexec.hydra

The `mpiexec.hydra` utility is a scalable MPI process manager for running MPI applications.

Syntax

`mpiexec.hydra <g-options> <l-options> <executable>`

or

`mpiexec.hydra <g-options> <l-options> <executable1> : <l-options> <executable2>`

Arguments

<g-options>	Global options that apply to all MPI processes
<l-options>	Local options that apply to a single argument set
<executable>	`./a.out` or `path/name` of the executable file

Description

Use the `mpiexec.hydra` utility to run MPI applications.

Use the first short command-line syntax to start all MPI processes of the *<executable>* with the single set of arguments. For example, the following command executes `a.out` over the specified processes and hosts:

```
$ mpiexec.hydra -f <hostfile> -n <# of processes> ./a.out
```

where:

- *<# of processes>* specifies the number of processes on which to run the `a.out` executable
- *<hostfile>* specifies a list of hosts on which to run the `a.out` executable

Use the second long command-line syntax to set different argument sets for different MPI program runs. For example, the following command executes two different binaries with different argument sets:

```
$ mpiexec.hydra -f <hostfile> -env <VAR1> <VAL1> -n 2 ./a.out : \
-env <VAR2> <VAL2> -n 2 ./b.out
```

NOTE

You need to distinguish global options from local options. In a command-line syntax, place the local options after the global options.

2.3.1. Global Options

This section describes the global options of the Intel® MPI Library's Hydra process manager. Global options are applied to all arguments sets in the launch command. Argument sets are separated by a colon ':'.

-hostfile *<hostfile>* or -f *<hostfile>*

Use this option to specify host names on which to run the application. If a host name is repeated, this name is used only once.

See also the `I_MPI_HYDRA_HOST_FILE` environment variable for more details.

NOTE

Use the `-perhost`, `-ppn`, `-grr`, and `-rr` options to change the process placement on the cluster nodes.

- Use the `-perhost`, `-ppn`, and `-grr` options to place consecutive MPI processes on every host using the round robin scheduling.
- Use the `-rr` option to place consecutive MPI processes on different hosts using the round robin scheduling.

-machinefile *<machine file>* or -machine *<machine file>*

Use this option to control process placement through a machine file. To define the total number of processes to start, use the `-n` option. To pin processes within a machine, use the option `binding=map` in the machine file. For example:

```
$ cat ./machinefile
node0:2 binding=map=0,3
node1:2 binding=map=[2,8]
node0:1 binding=map=8
```

For details on using the `binding` option, see Binding Option.

-genv *<ENVVAR>* *<value>*

Use this option to set the *<ENVVAR>* environment variable to the specified *<value>* for all MPI processes.

-genvall

Use this option to enable propagation of all environment variables to all MPI processes.

-genvnone

Use this option to suppress propagation of any environment variables to any MPI processes.

-genvexcl *<list of env var names>*

Use this option to suppress propagation of the listed environment variables to any MPI processes.

-genvlist *<list>*

Use this option to pass a list of environment variables with their current values. *<list>* is a comma separated list of environment variables to be sent to all MPI processes.

-pmi-connect *<mode>*

Use this option to choose the caching mode of process management interface (PMI) message. Possible values for *<mode>* are:

<mode>	The caching mode to be used
`nocache`	Do not cache PMI messages.
`cache`	Cache PMI messages on the local `pmi_proxy` management processes to minimize the number of PMI requests. Cached information is automatically propagated to child management processes.
`lazy-cache`	`cache` mode with on-request propagation of the PMI information.

`alltoall`	Information is automatically exchanged between all `pmi_proxy` before any get request can be done. This is the default mode.

See the `I_MPI_HYDRA_PMI_CONNECT` environment variable for more details.

-perhost *<# of processes >*, -ppn *<# of processes >*, or -grr *<# of processes>*

Use this option to place the specified number of consecutive MPI processes on every host in the group using round robin scheduling. See the `I_MPI_PERHOST` environment variable for more details.

NOTE

When running under a job scheduler, these options are ignored by default. To be able to control process placement with these options, disable the `I_MPI_JOB_RESPECT_PROCESS_PLACEMENT` variable.

-rr

Use this option to place consecutive MPI processes on different hosts using the round robin scheduling. This option is equivalent to "`-perhost 1`". See the `I_MPI_PERHOST` environment variable for more details.

-trace [*<profiling_library>*] or -t [*<profiling_library>*]

Use this option to profile your MPI application with Intel® Trace Collector using the indicated *<profiling_library>*. If you do not specify *<profiling_library>*, the default profiling library `libVT.so` is used.

Set the `I_MPI_JOB_TRACE_LIBS` environment variable to override the default profiling library.

-trace-imbalance

Use this option to profile your MPI application with Intel® Trace Collector using the `libVTim.so` library.

-aps

Use this option to collect statistics from your MPI application using Application Performance Snapshot. The collected data includes hardware performance metrics, memory consumption data, internal MPI imbalance and OpenMP* imbalance statistics. When you use this option, a new folder `aps_result_<date>-<time>` with statistics data is generated. You can analyze the collected data with the `aps` utility, for example:

```
$ mpirun -aps -n 2 ./myApp
$ aps aps_result_20171231_235959
```

NOTE

1. To use this option, set up the Application Performance Snapshot environment beforehand. See the tool's *Getting Started Guide* at *<installdir>/performance_snapshot* in Intel® Parallel Studio XE Professional or Cluster Edition.
2. If you use the options `-trace` or `-check_mpi`, the `-aps` option is ignored.

-check_mpi [*<checking_library>*]

Use this option to check your MPI application for correctness using the specified *<checking_library>*. If you do not specify *<checking_library>*, the default checking library `libVTmc.so` is used.

Set the `I_MPI_JOB_CHECK_LIBS` environment variable to override the default checking library.

-trace-pt2pt

Use this option to collect the information about point-to-point operations using Intel® Trace Analyzer and Collector. The option requires that you also use the `-trace` option.

-trace-collectives

Use this option to collect the information about collective operations using Intel® Trace Analyzer and Collector. The option requires that you also use the `-trace` option.

NOTE

Use the `-trace-pt2pt` and `-trace-collectives` to reduce the size of the resulting trace file or the number of message checker reports. These options work with both statically and dynamically linked applications.

-configfile *<filename>*

Use this option to specify the file *<filename>* that contains the command-line options. Blank lines and lines that start with '`#`' as the first character are ignored.

-branch-count *<num>*

Use this option to restrict the number of child management processes launched by the Hydra process manager, or by each `pmi_proxy` management process.

See the `I_MPI_HYDRA_BRANCH_COUNT` environment variable for more details.

-pmi-aggregate or -pmi-noaggregate

Use this option to switch on or off, respectively, the aggregation of the PMI requests. The default value is `-pmi-aggregate`, which means the aggregation is enabled by default.

See the `I_MPI_HYDRA_PMI_AGGREGATE` environment variable for more details.

-tv

Use this option to run *<executable>* under the TotalView* debugger. For example:

```
$ mpiexec.hydra -tv -n <# of processes> <executable>
```

See the `TOTALVIEW` environment variable for information on how to select a different TotalView* executable file.

-tva *<pid>*

Use this option to attach the TotalView* debugger to an existing MPI job. Use the `mpiexec.hydra` process id as *<pid>*. You can use the following command:

```
$ mpiexec.hydra -tva <pid>
```

-gdb

Use this option to run an executable under the GNU* debugger. You can use the following command:

```
$ mpiexec.hydra -gdb -n <# of processes> <executable>
```

-gdba *<pid>*

Use this option to attach the GNU* debugger to the existing MPI job. You can use the following command:

```
$ mpiexec.hydra -gdba <pid>
```

-nolocal

Use this option to avoid running the *<executable>* on the host where `mpiexec.hydra` is launched. You can use this option on clusters that deploy a dedicated master node for starting the MPI jobs and a set of dedicated compute nodes for running the actual MPI processes.

-hosts *<nodelist>*

Use this option to specify a particular *<nodelist>* on which the MPI processes should be run. For example, the following command runs the executable `a.out` on the hosts `host1` and `host2`:

```
$ mpiexec.hydra -n 2 -ppn 1 -hosts host1,host2 ./a.out
```

NOTE

If *<nodelist>* contains only one node, this option is interpreted as a local option. See Local Options for details.

-iface *<interface>*

Use this option to choose the appropriate network interface. For example, if the IP emulation of your InfiniBand* network is configured to `ib0`, you can use the following command.

```
$ mpiexec.hydra -n 2 -iface ib0 ./a.out
```

See the `I_MPI_HYDRA_IFACE` environment variable for more details.

-demux *<mode>*

Use this option to set the polling mode for multiple I/O. The default value is `poll`.

Arguments

<spec>	Define the polling mode for multiple I/O
`poll`	Set `poll` as the polling mode. This is the default value.
`select`	Set `select` as the polling mode.

See the `I_MPI_HYDRA_DEMUX` environment variable for more details.

-enable-x or -disable-x

Use this option to control the Xlib* traffic forwarding. The default value is `-disable-x`, which means the Xlib traffic is not forwarded.

-l, -prepend-rank

Use this option to insert the MPI process rank at the beginning of all lines written to the standard output.

-tune [*<arg >*]

Use this option to optimize the Intel® MPI Library performance by using the data collected by the `mpitune` utility.

NOTE

Use the `mpitune` utility to collect the performance tuning data before using this option.

<arg> is the directory containing tuned settings or a configuration file that applies these settings. If *<arg>* is not specified, the most optimal settings are selected for the given configuration. The default location of the configuration file is *<installdir>*/*<arch>*/etc directory.

-ilp64

Use this option to preload the ILP64 interface. See ILP64 Support for more details.

-s <spec>

Use this option to direct standard input to the specified MPI processes.

Arguments

<spec>	Define MPI process ranks
all	Use all processes.
<l>,<m>,<n>	Specify an exact list and use processes *<l>*, *<m>* and *<n>* only. The default value is zero.
<k>,<l>-<m>,<n>	Specify a range and use processes *<k>*, *<l>* through *<m>*, and *<n>*.

-noconf

Use this option to disable processing of the mpiexec.hydra configuration files.

-ordered-output

Use this option to avoid intermingling of data output from the MPI processes. This option affects both the standard output and the standard error streams.

NOTE

When using this option, end the last output line of each process with the end-of-line '\n' character. Otherwise the application may stop responding.

-path <directory>

Use this option to specify the path to the executable file.

-cleanup

Use this option to create a temporary file containing information about the launched processes. The file name is mpiexec_${username}_$PPID.log, where PPID is a parent process ID. This file is created in the temporary directory selected by the -tmpdir option. This file is used by the mpicleanup utility. If a job terminates successfully, the mpiexec.hydra command automatically removes this file.

See the I_MPI_HYDRA_CLEANUP environment variable for more details.

-tmpdir <dir>

Use this option to set a directory for temporary files. See the I_MPI_TMPDIR environment variable for more details.

-version or -V

Use this option to display the version of the Intel® MPI Library.

-info

Use this option to display build information of the Intel® MPI Library. When this option is used, the other command line arguments are ignored.

-use-app-topology <*value*>

Use this option to optimally place MPI processes on the cluster. The placement is based on results from the `mpitune` utility (in the `--rank-placement` mode). See `mpitune` Utility for details. An alternative to using this option is setting the `I_MPI_HYDRA_USE_APP_TOPOLOGY` variable.

<*value*> is the path to the native Intel MPI Library statistics file level 1 or higher.

-localhost

Use this option to explicitly specify the local host name for the launching node.

-rmk <*RMK*>

Use this option to select a resource management kernel to be used. Intel® MPI Library only supports `pbs`.

See the `I_MPI_HYDRA_RMK` environment variable for more details.

-outfile-pattern <*file*>

Use this option to redirect `stdout` to the specified file.

-errfile-pattern <*file*>

Use this option to redirect `stderr` to the specified file.

-gpath <*path*>

Use this option to specify the path to the executable file.

-gwdir <*dir*>

Use this option to specify the working directory in which the executable file runs.

-gumask <*umask*>

Use this option to perform the "`umask` <*umask*>" command for the remote executable file.

-gdb-ia

Use this option to run processes under Intel® architecture specific GNU* debugger.

-prepend-pattern

Use this option to specify the pattern that is prepended to the process output.

-verbose or -v

Use this option to print debug information from `mpiexec.hydra`, such as:

- Service processes arguments

- Environment variables and arguments passed to start an application
- PMI requests/responses during a job life cycle

See the `I_MPI_HYDRA_DEBUG` environment variable for more details.

-print-rank-map

Use this option to print out the MPI rank mapping.

-print-all-exitcodes

Use this option to print the exit codes of all processes.

Bootstrap Options

-bootstrap *<bootstrap server>*

Use this option to select a built-in bootstrap server to use. A bootstrap server is the basic remote node access mechanism that is provided by the system. Hydra supports multiple runtime bootstrap servers such as `ssh`, `rsh`, `pdsh`, `fork`, `persist`, `slurm`, `ll`, `lsf`, or `sge` to launch MPI processes. The default bootstrap server is `ssh`. By selecting `slurm`, `ll`, `lsf`, or `sge`, you use the corresponding `srun`, `llspawn.stdio`, `blaunch`, or `qrsh` internal job scheduler utility to launch service processes under the respective selected job scheduler (SLURM*, LoadLeveler*, LSF*, and SGE*).

Arguments

<arg>	String parameter
`ssh`	Use secure shell. This is the default value.
`rsh`	Use remote shell.
`pdsh`	Use parallel distributed shell.
`pbsdsh`	Use Torque* and PBS* `pbsdsh` command.
`fork`	Use fork call.
`persist`	Use Hydra persist server. See below for details.
`slurm`	Use SLURM* `srun` command.
`ll`	Use LoadLeveler* `llspawn.stdio` command.
`lsf`	Use LSF `blaunch` command.
`sge`	Use Univa* Grid Engine* `qrsh` command.

See `I_MPI_HYDRA_BOOTSTRAP` for details.

-bootstrap-exec *<bootstrap server>*

Use this option to set the executable to be used as a bootstrap server. The default bootstrap server is `ssh`. For example:

```
$ mpiexec.hydra -bootstrap-exec <bootstrap_server_executable> -f hosts -env <VAR1>
<VAL1> -n 2 ./a.out
```

See I_MPI_HYDRA_BOOTSTRAP for more details.

-bootstrap-exec-args <args>

Use this option to provide the additional parameters to the bootstrap server executable file.

```
$ mpiexec.hydra -bootstrap-exec-args <arguments> -n 2 ./a.out
```

For tight integration with the SLURM* scheduler (including support for suspend/resume), use the method outlined on the SLURM page here: http://www.schedmd.com/slurmdocs/mpi_guide.html#intel_mpi

See I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS for more details.

-bootstrap persist

Use this option to launch MPI processes using Hydra persist server. Before running a job, start these servers on each host:

```
$ hydra_persist &
```

NOTE

Do not start the services under the root account. A server can be shutdown using the Linux* shell `kill` command.

-prefork

Use this option to enable a new method for application processes startup. The new processes start up method allows you to reduce the number of file system operations during the application startup phase, which reduces the process loading time.

```
$ mpiexec.hydra -prefork -n 2 ./a.out
```

Binding Option

-binding

Use this option to pin or bind MPI processes to a particular processor and avoid undesired process migration. In the following syntax, the quotes may be omitted for a one-member list. Each parameter corresponds to a single pinning property.

NOTE

This option is related to the family of I_MPI_PIN environment variables, which have higher priority than the `-binding` option. Hence, if any of these variables are set, the option is ignored.

This option is supported on both Intel® and non-Intel microprocessors, but it may perform additional optimizations for Intel microprocessors than it performs for non-Intel microprocessors.

Syntax

`-binding "<parameter>=<value>[;<parameter>=<value> ...]"`

Parameters

Parameter	
`pin`	Pinning switch

Values	
`enable | yes | on | 1`	Turn on the pinning property. This is the default value
`disable | no | off | 0`	Turn off the pinning property

Parameter	
`cell`	Pinning resolution
Values	
`unit`	Basic processor unit (logical CPU)
`core`	Processor core in multi-core system

Parameter	
`map`	Process mapping
Values	
`spread`	The processes are mapped consecutively to separate processor cells. Thus, the processes do not share the common resources of the adjacent cells.
`scatter`	The processes are mapped to separate processor cells. Adjacent processes are mapped upon the cells that are the most remote in the multi-core topology.
`bunch`	The processes are mapped to separate processor cells by #processes/#sockets processes per socket. Each socket processor portion is a set of the cells that are the closest in the multi-core topology.
`p0,p1,...,pn`	The processes are mapped upon the separate processors according to the processor specification on the `p0,p1,...,pn` list: the i^{th} process is mapped upon the processor `pi`, where `pi` takes one of the following values: • processor number like n • range of processor numbers like n-m • `-1` for no pinning of the corresponding process
`[m0,m1,...,mn]`	The i^{th} process is mapped upon the processor subset defined by m_i hexadecimal mask using the following rule: The j^{th} processor is included into the subset m_i if the j^{th} bit of m_i equals `1`.

Parameter	
`domain`	Processor domain set on a node
Values	

`cell`	Each domain of the set is a single processor cell (unit or core).
`core`	Each domain of the set consists of the processor cells that share a particular core.
`cache1`	Each domain of the set consists of the processor cells that share a particular level 1 cache.
`cache2`	Each domain of the set consists of the processor cells that share a particular level 2 cache.
`cache3`	Each domain of the set consists of the processor cells that share a particular level 3 cache.
`cache`	The set elements of which are the largest domains among cache1, cache2, and cache3
`socket`	Each domain of the set consists of the processor cells that are located on a particular socket.
`node`	All processor cells on a node are arranged into a single domain.
`<size>[:<layout>]`	Each domain of the set consists of *`<size>`* processor cells. *`<size>`* may have the following values: • `auto` - domain size = #cells/#processes • `omp` - domain size = `OMP_NUM_THREADS` environment variable value • positive integer - exact value of the domain size ***NOTE*** Domain size is limited by the number of processor cores on the node. Each member location inside the domain is defined by the optional *`<layout>`* parameter value: • `compact` - as close with others as possible in the multi-core topology • `scatter` - as far away from others as possible in the multi-core topology • `range` - by BIOS numbering of the processors If *`<layout>`* parameter is omitted, `compact` is assumed as the value of *`<layout>`*

Parameter	
`order`	Linear ordering of the domains
Values	
`compact`	Order the domain set so that adjacent domains are the closest in the multi-core topology
`scatter`	Order the domain set so that adjacent domains are the most remote in the multi-core topology

`range`	Order the domain set according to the BIOS processor numbering

Parameter	
`offset`	Domain list offset
Values	
<n>	Integer number of the starting domain among the linear ordered domains. This domain gets number zero. The numbers of other domains will be cyclically shifted.

gtool Options

-gtool

Use this option to launch such tools as Intel® VTune™ Amplifier XE, Intel® Advisor, Valgrind*, and GNU* Debugger (GDB*) for the specified processes through the `mpiexec.hydra` and `mpirun` commands. An alternative to this option is the `I_MPI_GTOOL` environment variable.

Without the `-gtool` option, to analyze an MPI process with the VTune Amplifier, for example, you have to specify the relevant command in the corresponding argument set:

```
$ mpirun -n 3 ./myprog : -n 1 amplxe-cl -c advanced-hotspots -r ah -- ./myprog
```

The `-gtool` option is meant to simplify analysis of particular processes, relieving you of specifying the analysis tool's command line in each argument set (separated by colons ':'). Even though it is allowed to use `-gtool` within an argument set, do not use it in several sets at once and do not mix the two analysis methods (with `-gtool` and argument sets).

NOTE

You cannot use the `-gdb` and `-gtool` options simultaneously, unless you specify GDB with `-gtool`.

Syntax

```
-gtool "<command line for tool 1>:<ranks set 1>[=launch mode 1][@arch 1]; <command line for tool 2>:<ranks set 2>[=exclusive][@arch 2]; … ;<command line for a tool n>:<ranks set n>[=exclusive][@arch n]" <executable>
```

or:

```
$ mpirun -n <# of processes> -gtool "<command line for tool 1>:<ranks set 1>[=launch mode 1][@arch 1]" -gtool "<command line for a tool 2>:<ranks set 2>[=launch mode 2][@arch 2]" … -gtool "<command line for a tool n>:<ranks set n>[=launch mode 3][@arch n]" <executable>
```

In the syntax, the separator ';' and the `-gtool` option are interchangeable.

Arguments

<arg>	Parameters
<rank set>	Specify the range of ranks that are involved in the tool execution. Separate ranks with a comma or use the '-' symbol for a set of contiguous ranks. To run the tool for all ranks, use

	the `all` argument. **NOTE** If you specify incorrect rank index, the corresponding warning is printed and the tool continues working for valid ranks.
`[=launch mode]`	Specify the launch mode (optional). See below for the available values.
`[@arch]`	Specify the architecture on which the tool runs (optional). For a given *<rank set>*, if you specify this argument, the tool is launched only for the processes residing on hosts with the specified architecture. This parameter is optional. For the available values of `[@arch]`, see `I_MPI_PLATFORM`.

NOTE

Rank sets cannot overlap for the same `@arch` parameter. Missing `@arch` parameter is also considered a different architecture. Thus, the following syntax is considered valid:

```
-gtool "gdb:0-3=attach;gdb:0-3=attach@hsw;/usr/bin/gdb:0-3=attach@knl"
```

Also, note that some tools cannot work together or their simultaneous use may lead to incorrect results.

The following table lists the parameter values for `[=launch mode]`:

`[=launch mode]`	Tool launch mode (optional). You can specify several values for each tool, which are separated with a comma ','.
exclusive	Specify this value to prevent the tool from launching for more than one rank per host.
attach	Specify this value to attach the tool from `-gtool` to the executable. If you use debuggers or other tools that can attach to a process in a debugger manner, you need to specify this value. This mode has been tested with debuggers only.
node-wide	Specify this value to apply the tool from `-gtool` to all ranks where the `<rank set>` resides or for all nodes in the case of `all` ranks. That is, the tool is applied to a higher level than the executable (to the `pmi_proxy` daemon).

Examples

The following examples demonstrate different scenarios of using the `-gtool` option.

Example 1

Launch the Intel® VTune™ Amplifier XE and Valgrind* through the `mpirun` command:

```
$ mpirun -n 16 -gtool "amplxe-cl -collect hotspots -analyze-system \
-r result1:5,3,7-9=exclusive@bdw;valgrind -log-file=log_%p:0,1,10-12@hsw" a.out
```

This command launches `amplxe-cl` for the processes that are run on the Intel® microarchitecture codenamed Broadwell. Only one copy of `amplxe-cl` is launched for each host, the process with the minimal index is affected. At the same time, Valgrind* is launched for all specified processes that are run on the Intel® microarchitecture codenamed Haswell. Valgrind's results are saved to the files `log_<process ID>`.

Example 2

Set different environment variables for different rank sets:

```
$ mpirun -n 16 -gtool "env VARIABLE1=value1 VARIABLE2=value2:3,5,7-9; env
VARIABLE3=value3:0,11" a.out
```

Example 3

Apply a tool for a certain process through the `-machinefile` option.

In this example, suppose `m_file` has the following content:

```
$ cat ./m_file
hostname_1:2
hostname_2:3
hostname_3:1
```

The following command line demonstrates how to use the `-machinefile` option to apply a tool:

```
$ mpirun -n 6 -machinefile m_file -gtool "amplxe-cl -collect hotspots -analyze-
system \
-r result1:5,3=exclusive@hsw;valgrind:0,1@bdw" a.out
```

In this example, the use of `-machinefie` option means that processes with indices `0` and `1` are located on the `hostname_1` machine, process `3` is located on the `hostname_2` machine, and process `5` - on the `hostname_3` machine. After that, `amplxe-cl` is applied only ranks `3` and `5` (since these ranks belong to different machines, the `exclusive` option matches both of them) in case if `hostname_2` and `hostname_3` machines have Intel® microarchitecture codenamed Haswell. At the same time, the Valgrind* tool is applied to both ranks allocated on `hostname_1` machine in case if it has Intel® microarchitecture codenamed Broadwell.

Example 4

To show how the ranks are distributed across the cluster nodes, use the "`z show map`" command with `-gtool`:

```
$ mpirun -gtool "gdb-ia:0,2,3,9,10=attach;/tmp/gdb:5,6=attach@knc" -host
<hostname_1> -n 4 <host-app> :\
-host <hostname_1-mic0> -n 4 <mic-app> :\
-host <hostname_2> -n 4 <host-app>
[0,2,3,5,6,9,10] (mpigdb) z show map
[0,2,3]: hostname_1
[5,6]: hostname_1-mic0
[9,10]: hostname_2
[0,2,3,5,6,9,10] (mpigdb) z help
z <positive number(s) up to 11 or all> - Sets ranks for debugging
z show map - Shows ranks distribution across the cluster nodes
z help - Shows help information
[0,2,3,5,6,9,10] (mpigdb)
```

-gtoolfile <gtool_config_file>

Use this option to specify the `-gtool` parameters in a configuration file. All the same rules apply. Additionally, you can separate different command lines with section breaks.

For example, if `gtool_config_file` contains the following settings:

```
env VARIABLE1=value1 VARIABLE2=value2:3,5,7-9; env VARIABLE3=value3:0,11
env VARIABLE4=value4:1,12
```

The following command sets `VARIABLE1` and `VARIABLE2` for processes 3, 5, 7, 8, and 9 and sets `VARIABLE3` for processes 0 and 11, while `VARIABLE4` is set for processes 1 and 12:

```
$ mpirun -n 16 -gtoolfile gtool_config_file a.out
```

NOTE

The options and the environment variable `-gtool`, `-gtoolfile` and `I_MPI_GTOOL` are mutually exclusive. The options `-gtool` and `-gtoolfile` are of the same priority and have higher priority than `I_MPI_GTOOL`.

The first specified option in a command line is effective and the second one is ignored. Therefore, use `I_MPI_GTOOL` if you do not specify `-gtool` or `-gtoolfile`.

2.3.2. Local Options

This section describes the local options of the Intel® MPI Library's Hydra process manager. Local options are applied only to the argument set they are specified in. Argument sets are separated by a colon ':'.

-n *<# of processes>* or -np *<# of processes>*

Use this option to set the number of MPI processes to run with the current argument set.

-env *<ENVVAR> <value>*

Use this option to set the *<ENVVAR>* environment variable to the specified *<value>* for all MPI processes in the current argument set.

-envall

Use this option to propagate all environment variables in the current argument set. See the `I_MPI_HYDRA_ENV` environment variable for more details.

-envnone

Use this option to suppress propagation of any environment variables to the MPI processes in the current argument set.

-envexcl *<list of env var names>*

Use this option to suppress propagation of the listed environment variables to the MPI processes in the current argument set.

-envlist *<list>*

Use this option to pass a list of environment variables with their current values. *<list>* is a comma separated list of environment variables to be sent to the MPI processes.

-host *<nodename>*

Use this option to specify a particular *<nodename>* on which the MPI processes are to be run. For example, the following command executes `a.out` on hosts `host1` and `host2`:

```
$ mpiexec.hydra -n 2 -host host1 ./a.out : -n 2 -host host2 ./a.out
```

-path *<directory>*

Use this option to specify the path to the *<executable>* file to be run in the current argument set.

-wdir *<directory>*

Use this option to specify the working directory in which the *<executable>* file runs in the current argument set.

-umask *<umask>*

Use this option to perform the `umask` *<umask>* command for the remote *<executable>* file.

-hostos <host OS>

Use this option to specify an operating system installed on a particular host. MPI processes are launched on each host in accordance with this option specified. The default value is `linux`.

Arguments

<arg>	String parameter
`linux`	The host with Linux* OS installed. This is the default value.
`windows`	The host with Windows* OS installed.

NOTE

The option is used in conjunction with `-host` option. For example, the following command runs the executable `a.exe` on `host1` and `b.out` on `host2`:

```
$ mpiexec.hydra -n 1 -host host1 -hostos windows a.exe :\
-n 1 -host host2 -hostos linux ./a.out
```

2.3.3. Extended Fabric Control Options

This section lists the options used for quick selection of the communication fabrics. Using these options overrides the related environment variables, described in Fabrics Control.

-rdma

Use this option to select an RDMA-capable network fabric. The application attempts to use the first available RDMA-capable network fabric from the list: `dapl,ofa`. If no such fabric is available, another fabric from the list `tcp,tmi,ofi` is used. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST dapl,ofa,tcp,tmi,ofi -genv I_MPI_FALLBACK 1`.

-RDMA

Use this option to select an RDMA-capable network fabric. The application attempts to use the first available RDMA-capable network fabric from the list: `dapl,ofa`. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST dapl,ofa -genv I_MPI_FALLBACK 1`.

-dapl

Use this option to select a DAPL-capable network fabric. The application attempts to use a DAPL-capable network fabric. If no such fabric is available, another fabric from the list `tcp,tmi,ofa,ofi` is used. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST dapl,tcp,tmi,ofa,ofi -genv I_MPI_FALLBACK 1`.

-DAPL

Use this option to select a DAPL-capable network fabric. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST dapl -genv I_MPI_FALLBACK 0`.

-ib

Use this option to select an OFA-capable network fabric. The application attempts to use an OFA-capable network fabric. If no such fabric is available, another fabric from the list `dapl,tcp,tmi,ofi` is used. This

option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST ofa,dapl,tcp,tmi,ofi -genv I_MPI_FALLBACK 1`.

-IB

Use this option to select an OFA-capable network fabric. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST ofa -genv I_MPI_FALLBACK 0`.

-tmi

Use this option to select a TMI-capable network fabric. The application attempts to use a TMI-capable network fabric. If no such fabric is available, another fabric from the list `dapl,tcp,ofa,ofi` is used. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi,dapl,tcp,ofa,ofi -genv I_MPI_FALLBACK 1`.

-TMI

Use this option to select a TMI-capable network fabric. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi -genv I_MPI_FALLBACK 0`.

-mx

Use this option to select the Myrinet MX* network fabric. The application attempts to use the Myrinet MX* network fabric. If no such fabric is available, another fabric from the list `dapl,tcp,ofa,ofi` is used. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi,dapl,tcp,ofa,ofi -genv I_MPI_TMI_PROVIDER mx -genv I_MPI_DAPL_PROVIDER mx -genv I_MPI_FALLBACK 1`.

-MX

Use this option to select Myrinet MX* network fabric. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi -genv I_MPI_TMI_PROVIDER mx -genv I_MPI_FALLBACK 0`.

-psm

Use this option to select a PSM-capable network fabric: Intel® True Scale Fabric or Intel® Omni-Path Fabric in PSM-compatibility mode. The application attempts to use a PSM-capable network fabric. If no such fabric is available, another fabric from the list `dapl,tcp,ofa,ofi` is used. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi,dapl,tcp,ofa,ofi -genv I_MPI_TMI_PROVIDER psm -genv I_MPI_FALLBACK 1`.

-PSM

Use this option to select a PSM-capable network fabric: Intel® True Scale Fabric or Intel® Omni-Path Fabric in PSM-compatibility mode. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi -genv I_MPI_TMI_PROVIDER psm -genv I_MPI_FALLBACK 0`.

-psm2

Use this option to select the Intel® Omni-Path Fabric. The application attempts to use the Intel® Omni-Path Fabric. If no such fabric is available, another fabric from the list `dapl,tcp,ofa,ofi` is used. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi,dapl,tcp,ofa,ofi -genv I_MPI_TMI_PROVIDER psm2 -genv I_MPI_FALLBACK 1`.

-PSM2

Use this option to select the Intel® Omni-Path Fabric. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST tmi -genv I_MPI_TMI_PROVIDER psm2 -genv I_MPI_FALLBACK 0`.

-ofi

Use this option to select an OpenFabrics Interfaces* (OFI*) capable network fabric. The application attempts to use an OFI-capable network fabric. If no such fabric is available, another fabric from the list `tmi,dapl,tcp,ofa` is used. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST ofi,tmi,dapl,tcp,ofa -genv I_MPI_FALLBACK 1`.

-OFI

Use this option to select an OFI-capable network fabric. The application fails if no such fabric is found. This option is equivalent to the setting: `-genv I_MPI_FABRICS_LIST ofi -genv I_MPI_FALLBACK 0`.

2.3.4. Hydra Environment Variables

I_MPI_HYDRA_HOST_FILE

Set the host file to run the application.

Syntax

`I_MPI_HYDRA_HOST_FILE=<arg>`

Deprecated Syntax

`HYDRA_HOST_FILE=<arg>`

Arguments

<arg>	String parameter
<hostsfile>	The full or relative path to the host file

Description

Set this environment variable to specify the hosts file.

I_MPI_HYDRA_DEBUG

Print out the debug information.

Syntax

`I_MPI_HYDRA_DEBUG=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the debug output
`disable | no | off | 0`	Turn off the debug output. This is the default value

Description

Set this environment variable to enable the debug mode.

I_MPI_HYDRA_ENV

Control the environment propagation.

Syntax

`I_MPI_HYDRA_ENV=<arg>`

Arguments

`<arg>`	String parameter
`all`	Pass all environment to all MPI processes

Description

Set this environment variable to control the environment propagation to the MPI processes. By default, the entire launching node environment is passed to the MPI processes. Setting this variable also overwrites environment variables set by the remote shell.

I_MPI_JOB_TIMEOUT, I_MPI_MPIEXEC_TIMEOUT (MPIEXEC_TIMEOUT)

Set the timeout period for `mpiexec.hydra`.

Syntax

`I_MPI_JOB_TIMEOUT=<timeout>`

`I_MPI_MPIEXEC_TIMEOUT=<timeout>`

Deprecated Syntax

`MPIEXEC_TIMEOUT=<timeout>`

Arguments

`<timeout>`	Define `mpiexec.hydra` timeout period in seconds
`<n> >= 0`	The value of the timeout period. The default timeout value is zero, which means no timeout.

Description

Set this environment variable to make `mpiexec.hydra` terminate the job in `<timeout>` seconds after its launch. The `<timeout>` value should be greater than zero. Otherwise the environment variable setting is ignored.

NOTE

Set this environment variable in the shell environment before executing the `mpiexec.hydra` command. Setting the variable through the `-genv` and `-env` options has no effect.

I_MPI_JOB_TIMEOUT_SIGNAL (MPIEXEC_TIMEOUT_SIGNAL)

Define the signal to be sent when a job is terminated because of a timeout.

Syntax

`I_MPI_JOB_TIMEOUT_SIGNAL=<number>`

Deprecated Syntax

`MPIEXEC_TIMEOUT_SIGNAL=<number>`

Arguments

`<number>`	Define the signal number
`<n> > 0`	The signal number. The default value is `9 (SIGKILL)`

Description

Define a signal number to be sent to stop the MPI job if the timeout period specified by the `I_MPI_JOB_TIMEOUT` environment variable expires. If you set a signal number unsupported by the system, the `mpiexec.hydra` command prints a warning message and continues the task termination using the default signal number `9` (`SIGKILL`).

NOTE

Set this environment variable in the shell environment before executing the `mpiexec.hydra` command. Setting the variable through the `-genv` and `-env` options has no effect.

I_MPI_JOB_ABORT_SIGNAL

Define a signal to be sent to all processes when a job is terminated unexpectedly.

Syntax

`I_MPI_JOB_ABORT_SIGNAL=<number>`

Arguments

`<number>`	Define signal number
`<n> > 0`	The default value is `9 (SIGKILL)`

Description

Set this environment variable to define a signal for task termination. If you set an unsupported signal number, `mpiexec.hydra` prints a warning message and uses the default signal `9` (`SIGKILL`).

NOTE

Set this environment variable in the shell environment before executing the `mpiexec.hydra` command. Setting the variable through the `-genv` and `-env` options has no effect.

I_MPI_JOB_SIGNAL_PROPAGATION (MPIEXEC_SIGNAL_PROPAGATION)

Control signal propagation.

Syntax

`I_MPI_JOB_SIGNAL_PROPAGATION=<arg>`

Deprecated Syntax

`MPIEXEC_SIGNAL_PROPAGATION=<arg>`

Arguments

`<arg>`	Binary indicator
`enable | yes | on | 1`	Turn on propagation
`disable | no | off | 0`	Turn off propagation. This is the default value

Description

Set this environment variable to control propagation of the signals (`SIGINT`, `SIGALRM`, and `SIGTERM`). If you enable signal propagation, the received signal is sent to all processes of the MPI job. If you disable signal propagation, all processes of the MPI job are stopped with the default signal 9 (`SIGKILL`).

NOTE

Set this environment variable in the shell environment before executing the `mpiexec.hydra` command. Setting the variable through the `-genv` and `-env` options has no effect.

I_MPI_HYDRA_BOOTSTRAP

Set the bootstrap server.

Syntax

`I_MPI_HYDRA_BOOTSTRAP=<arg>`

Arguments

`<arg>`	String parameter
`ssh`	Use secure shell. This is the default value
`rsh`	Use remote shell
`pdsh`	Use parallel distributed shell
`pbsdsh`	Use Torque* and PBS* `pbsdsh` command
`fork`	Use fork call
`slurm`	Use SLURM* `srun` command
`ll`	Use LoadLeveler* `llspawn.stdio` command
`lsf`	Use LSF* `blaunch` command
`sge`	Use Univa* Grid Engine* `qrsh` command

Description

Set this environment variable to specify the bootstrap server.

NOTE

Set the `I_MPI_HYDRA_BOOTSTRAP` environment variable in the shell environment before executing the `mpiexec.hydra` command. Do not use the `-env` option to set the *<arg>* value. This option is used for passing environment variables to the MPI process environment.

I_MPI_HYDRA_BOOTSTRAP_EXEC

Set the executable file to be used as a bootstrap server.

Syntax

`I_MPI_HYDRA_BOOTSTRAP_EXEC=`*<arg>*

Arguments

<arg>	String parameter
<executable>	The name of the executable file

Description

Set this environment variable to specify the executable file to be used as a bootstrap server.

I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS

Set additional arguments for the bootstrap server.

Syntax

`I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS=<arg>`

Arguments

<arg>	String parameter
`<args>`	Additional bootstrap server arguments

Description

Set this environment variable to specify additional arguments for the bootstrap server.

I_MPI_HYDRA_BOOTSTRAP_AUTOFORK

Control the usage of `fork` call for local processes.

Syntax

`I_MPI_HYDRA_BOOTSTRAP_AUTOFORK = <arg>`

Arguments

<arg>	String parameter
`enable | yes | on | 1`	Use `fork` for the local processes. This is default value for `ssh`, `rsh`, `ll`, `lsf`, and `pbsdsh` bootstrap servers
`disable | no | off |`	Do not use `fork` for the local processes. This is default value for the `sge`

0	bootstrap server

Description

Set this environment variable to control usage of `fork` call for the local processes.

NOTE

This option is not applicable to `slurm`, `pdsh`, and `persist` bootstrap servers.

I_MPI_HYDRA_RMK

Use the resource management kernel.

Syntax

`I_MPI_HYDRA_RMK=<arg>`

Arguments

<arg>	String parameter
<rmk>	Resource management kernel. The only supported value is `pbs`

Description

Set this environment variable to use the `pbs` resource management kernel. Intel® MPI Library only supports `pbs`.

I_MPI_HYDRA_PMI_CONNECT

Define the processing method for PMI messages.

Syntax

`I_MPI_HYDRA_PMI_CONNECT=<value>`

Arguments

<value>	The algorithm to be used
`nocache`	Do not cache PMI messages
`cache`	Cache `PMI` messages on the local `pmi_proxy` management processes to minimize the number of `PMI` requests. Cached information is automatically propagated to child management processes.
`lazy-cache`	`cache` mode with on-demand propagation.
`alltoall`	Information is automatically exchanged between all `pmi_proxy` before any get request can be done. This is the default value.

Description

Use this environment variable to select the PMI messages processing method.

I_MPI_PMI2

Control the use of PMI-2 protocol.

Syntax

`I_MPI_PMI2=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable PMI-2 protocol
`disable | no | off | 0`	Disable PMI-2 protocol. This is the default value

Description

Set this environment variable to control the use of PMI-2 protocol.

I_MPI_PERHOST

Define the default behavior for the `-perhost` option of the `mpiexec.hydra` command.

Syntax

`I_MPI_PERHOST=<value>`

Arguments

<value>	Define a value used for `-perhost` by default
`integer > 0`	Exact value for the option
`all`	All logical CPUs on the node
`allcores`	All cores (physical CPUs) on the node. This is the default value.

Description

Set this environment variable to define the default behavior for the `-perhost` option. Unless specified explicitly, the `-perhost` option is implied with the value set in `I_MPI_PERHOST`.

NOTE

When running under a job scheduler, this environment variable is ignored by default. To be able to control process placement with `I_MPI_PERHOST`, disable the `I_MPI_JOB_RESPECT_PROCESS_PLACEMENT` variable.

I_MPI_JOB_TRACE_LIBS

Choose the libraries to preload through the `-trace` option.

Syntax

`I_MPI_JOB_TRACE_LIBS=<arg>`

Deprecated Syntax

`MPIEXEC_TRACE_LIBS=<arg>`

Arguments

<arg>	String parameter
<list>	Blank separated list of the libraries to preload. The default value is vt

Description

Set this environment variable to choose an alternative library for preloading through the -trace option.

I_MPI_JOB_CHECK_LIBS

Choose the libraries to preload through the -check_mpi option.

Syntax

I_MPI_JOB_CHECK_LIBS=*<arg>*

Arguments

<arg>	String parameter
<list>	Blank separated list of the libraries to preload. The default value is vtmc

Description

Set this environment variable to choose an alternative library for preloading through the -check_mpi option.

I_MPI_HYDRA_BRANCH_COUNT

Set the hierarchical branch count.

Syntax

I_MPI_HYDRA_BRANCH_COUNT =*<num>*

Arguments

<num>	Number
<n> >= 0	• The default value is -1 if less than 128 nodes are used. This value also means that there is no hierarchical structure • The default value is 32 if more than 127 nodes are used

Description

Set this environment variable to restrict the number of child management processes launched by the mpiexec.hydra operation or by each pmi_proxy management process.

I_MPI_HYDRA_PMI_AGGREGATE

Turn on/off aggregation of the PMI messages.

Syntax

I_MPI_HYDRA_PMI_AGGREGATE=<arg>

Arguments

<arg>	Binary indicator

`enable | yes | on | 1`	Enable PMI message aggregation. This is the default value.
`disable | no | off | 0`	Disable PMI message aggregation.

Description

Set this environment variable to enable/disable aggregation of PMI messages.

I_MPI_HYDRA_GDB_REMOTE_SHELL

Set the remote shell command to run GNU* debugger.

Syntax

`I_MPI_HYDRA_GDB_REMOTE_SHELL=<arg>`

Arguments

<arg>	String parameter
`ssh`	Secure Shell (SSH). This is the default value
`rsh`	Remote shell (RSH)

Description

Set this environment variable to specify the remote shell command to run the GNU* debugger on the remote machines. You can use this environment variable to specify any shell command that has the same syntax as SSH or RSH.

I_MPI_HYDRA_IFACE

Set the network interface.

Syntax

`I_MPI_HYDRA_IFACE=<arg>`

Arguments

<arg>	String parameter
<network interface>	The network interface configured in your system

Description

Set this environment variable to specify the network interface to use. For example, use "`-iface ib0`", if the IP emulation of your InfiniBand* network is configured on `ib0`.

I_MPI_HYDRA_DEMUX

Set the demultiplexer (demux) mode.

Syntax

`I_MPI_HYDRA_DEMUX=<arg>`

Arguments

<arg>	String parameter
`poll`	Set `poll` as the multiple I/O demultiplexer (`demux`) mode engine. This is the default value.
`select`	Set `select` as the multiple I/O demultiplexer (`demux`) mode engine

Description

Set this environment variable to specify the multiple I/O `demux` mode engine. The default value is `poll`.

I_MPI_HYDRA_CLEANUP

Control the creation of the default `mpicleanup` input file.

Syntax

`I_MPI_HYDRA_CLEANUP=`*<value>*

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Enable the `mpicleanup` input file creation
`disable | no | off | 0`	Disable the `mpicleanup` input file creation. This is the default value

Description

Set the `I_MPI_HYDRA_CLEANUP` environment variable to create the input file for the `mpicleanup` utility. For more information about the `mpicleanup` utility, see Cleanup Utility.

I_MPI_TMPDIR (TMPDIR)

Set the temporary directory.

Syntax

`I_MPI_TMPDIR=`*<arg>*

Arguments

<arg>	String parameter
`<path>`	Set the temporary directory. The default value is `/tmp`

Description

Set this environment variable to specify the temporary directory to store the `mpicleanup` input file.

I_MPI_JOB_RESPECT_PROCESS_PLACEMENT

Specify whether to use the process-per-node placement provided by the job scheduler, or set explicitly.

Syntax

`I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=`*<arg>*

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Use the process placement provided by job scheduler. This is the default value
`disable | no | off | 0`	Do not use the process placement provided by job scheduler

Description

If the variable is set, the Hydra process manager uses the process placement provided by job scheduler (default). In this case the `-ppn` option and its equivalents are ignored. If you disable the variable, the Hydra process manager uses the process placement set with `-ppn` or its equivalents.

I_MPI_GTOOL

Specify the tools to be launched for selected ranks. An alternative to this variable is the `-gtool` option.

Syntax

```
I_MPI_GTOOL="<command line for a tool 1>:<ranks set 1>[=exclusive][@arch 1];
<command line for a tool 2>:<ranks set 2>[=exclusive][@arch 2]; … ;<command line
for a tool n>:<ranks set n>[=exclusive][@arch n]"
```

Arguments

<arg>	Parameters
<command line for a tool>	Specify a tool's launch command, including parameters.
<rank set>	Specify the range of ranks that are involved in the tool execution. Separate ranks with a comma or use the '–' symbol for a set of contiguous ranks. To run the tool for all ranks, use the `all` argument. ***NOTE*** If you specify incorrect rank index, the corresponding warning is printed and the tool continues working for valid ranks.
`[=exclusive]`	Specify this parameter to prevent launching a tool for more than one rank per host. This parameter is optional.
`[@arch]`	Specify the architecture on which the tool runs (optional). For a given *<rank set>*, if you specify this argument, the tool is launched only for the processes residing on hosts with the specified architecture. This parameter is optional. For the available values of `[@arch]`, see `I_MPI_PLATFORM`. If you launch the debugger on Intel® Xeon Phi™ coprocessor, setting `[@arch]` is required. See the examples below for details.

Description

Use this option to launch the tools such as Intel® VTune™ Amplifier XE, Valgrind*, and GNU* Debugger for the specified processes.

Examples

The following command line examples demonstrate different scenarios of using the `I_MPI_GTOOL` environment variable.

Launch Intel® VTune™ Amplifier XE and Valgrind* by setting the `I_MPI_GTOOL` environment variable:

```
$ export I_MPI_GTOOL="amplxe-cl -collect hotspots -analyze-system -r result1:5,3,7-9=exclusive@bdw;\
valgrind -log-file=log_%p:0,1,10-12@hsw"
$ mpiexec.hydra -n 16 a.out
```

This command launches `amplxe-cl` for the processes that are run on the Intel® microarchitecture codenamed Broadwell. Only one copy of `amplxe-cl` is launched for each host, the process with the minimal index is affected. At the same time, Valgrind* is launched for all specified processes that are run on the Intel® microarchitecture codenamed Haswell. Valgrind's results are saved to the files `log_<process ID>`.

Launch GNU* Debugger (GDB*) by setting the `I_MPI_GTOOL` environment variable:

```
$ mpiexec.hydra -n 16 -genv I_MPI_GTOOL="gdb:3,5,7-9" a.out
```

Use this command to apply `gdb` to the given rank set.

NOTE

The options and the environment variable `-gtool`, `-gtoolfile` and `I_MPI_GTOOL` are mutually exclusive. The options `-gtool` and `-gtoolfile` are of the same priority and have higher priority than `I_MPI_GTOOL`. The first specified option in a command line is effective and the second one is ignored. Therefore, use `I_MPI_GTOOL` if you do not specify `-gtool` or `-gtoolfile`.

I_MPI_HYDRA_USE_APP_TOPOLOGY

Syntax

`I_MPI_HYDRA_USE_APP_TOPOLOGY=<value>`

Arguments

<value>	Path to the native Intel MPI Library statistics file level 1 or higher.

Description

Set this variable to optimally place MPI ranks on the cluster. The placement is based on results from the `mpitune` utility (in the `--rank-placement` mode). See `mpitune` Utility for details. An alternative to using this option is specifying the `-use-app-topology` option.

I_MPI_HYDRA_TOPOLIB

Set the interface for topology detection.

Syntax

`I_MPI_HYDRA_TOPOLIB=<arg>`

Arguments

<arg>	String parameter
`hwloc`	The `hwloc`* library functions are invoked for topology detection.

Description

Set this environment variable to define the interface for platform detection. The hwloc* interface is utilized if the variable is set explicitly: `I_MPI_HYDRA_TOPOLIB=hwloc`. Otherwise, the native Intel® MPI Library interface is used, which is the default behavior.

I_MPI_HYDRA_PREFORK

Enable a new method for application processes startup.

Syntax

`I_MPI_HYDRA_PREFORK=<arg>`

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Use the new method for application processes startup.
`disable | no | off | 0`	Do not use the new method for application processes startup. This is default value

Description

Set this environment variable to control the usage of the new processes startup method. The new processes start up method allows you to reduce the number of file system operations during the application startup phase, which reduces the process loading time.

I_MPI_PORT_RANGE

Set allowed port range.

Syntax

`I_MPI_PORT_RANGE=<range>`

Arguments

<range>	String parameter
`<min>:<max>`	Allowed port range

Description

Set this environment variable to specify the allowed port range for the Intel® MPI Library.

2.3.5. Cleanup Utility

mpicleanup

Clean up the environment after an abnormally terminated MPI run under the `mpiexec.hydra` process manager.

Syntax

`mpicleanup <arguments>`

Arguments

`-i <input_file> | --input <input_file>`	Specify the input file generated by `mpiexec.hydra`. The default value is `mpiexec_${username}_$PPID.log` located in the temporary directory determined by the values of the `I_MPI_TMPDIR` or `TMPDIR` environment variables, or in the `/tmp` directory.
`-t | --total`	Use the total mode to stop all user processes on the specified machines. This option is not supported for the `root` user.
`-f <hostsfile> | --file <hostsfile>`	Specify the file containing the list of machines to clean up.
`-r <rshcmd> | --rsh <rshcmd>`	Specify the remote shell to use. The default shell is `ssh`.
`-b <branch_count> | --branch <branch_count>`	Define the number of the child processes. The default value is `32`.
`-p | --parallel`	Use the parallel launch mode. This option is only applicable if all hosts are available. Otherwise a part of machines may stay in an undefined state.
`-s | --silent`	Suppress extra output generation.
`-d | --verbose`	Output verbose information.
`-h | --help`	Display a help message.
`-V | --version`	Display Intel® MPI Library version information.

Description

Use this command to clean up the environment after an abnormal MPI job termination.

For example, use the following command to stop processes mentioned in the input file generated by the prior `mpiexec.hydra` invocation:

```
$ mpicleanup
```

or

```
$ mpicleanup --input /path/to/input.file
```

Use the following command to stop all your user processes on the machines specified in the `hostfile` file:

```
$ mpicleanup --file hostfile --total
```

2.3.6. Checkpoint-Restart Support

The checkpoint-restart feature in the Intel® MPI Library enables you to resume the application run from a previously set checkpoint, in case an application fails at some point of its execution.

NOTE

The checkpoint-restart feature requires the OFA* network module. You can choose the OFA network module, for example, with the `I_MPI_FABRICS` environment variable by setting the value to `ofa`, or by using the the `-ib` option.

NOTE

To enable the checkpoint-restart feature, set the following:

- `1` for `I_MPI_OFA_DYNAMIC_QPS` environment variable
- `0` for `I_MPI_OFA_NUM_RDMA_CONNECTIONS` environment variable

NOTE

Install the Berkeley Lab Checkpoint/Restart* (BLCR) Software for the Checkpoint-Restart function.

Checkpoint-Restart Global Options

-ckpoint *<switch>*

Arguments

<switch>	Checkpoint switch
`enable | yes | on | 1`	Enables the check point function for the application started
`disable | no | off | 0`	Disables the check point function for the application started. This is the default value

Use this option to enable/disable checkpoints capability. When this capability is disabled, other checkpoint options are ignored.

-ckpoint-interval *<sec>*

Arguments

<sec>	Interval between consecutive checkpoints in seconds

Use this option to turn on timer driven checkpoints. See also Timer Driven Checkpoint. The checkpoints are taken every *<sec>* seconds. If this option is not specified, signal driven checkpoint function may be used. See Explicit Signal Driven Checkpoint for more details.

-ckpoint-preserve *<N>*

Arguments

<N>	Maximal number of checkpoint images kept. The default value is `1`

Use this option while running the checkpoint function to keep last *<N>* checkpoints to reduce checkpoint image space. By default, only the last checkpoint is kept.

-restart

Use this option to restart an application from one of the stored checkpoints. `-ckpointlib`, `-ckpoint-prefix` and `-ckpoint-num` options are meaningful for restarting. The executable name may be provided to the process manager, but is ignored. Taking checkpoints is allowed for the restarted application, so `-restart` option may be accompanied with `-ckpoint` and other applicable checkpoint options.

-ckpoint-num *<N>*

Arguments

<N>	Identifier of the checkpoint image to restart an application with. Valid values are any number equal or lower than the last checkpoint number. The default is the last checkpoint number.

Use this option while restarting an application. The checkpoint number *<N>* (counting from `0`) is taken as a restart point. To determine the best choice for this value, examine the checkpoint storage directory setting with the `-ckpoint-prefix` option.

NOTE

The number of images determined by the `-ckpoint-preserve` option is kept at maximum.

The application will abort with an error message during startup if this checkpoint does not exist. By default, the last checkpoint is selected.

Checkpoint-Restart Local Options

-ckpointlib *<lib>*

Arguments

<lib>	Checkpoint-Restart system library
`blcr`	Berkeley Lab Checkpoint/Restart* (BLCR) Library. This is the default value

Use this option to select underlying Checkpoint-Restart system library. Only the Berkeley Lab Checkpoint/Restart* (BLCR) Library is supported.

NOTE

You need to provide the same option when using the checkpoint function, or when restarting the application.

-ckpoint-prefix *<dir>*

Arguments

<dir>	The directory to store checkpoints. The default value is `/tmp`

Use this option to specify a directory to store checkpoints. By default, `/tmp` is used. The directory *<dir>* should be writable, otherwise an error will be raised during process launch, and the application will abort with an error message.

NOTE

You need to provide the same option when using the checkpoint function, or when restarting the application.

-ckpoint-tmp-prefix *<dir>*

Arguments

<dir>	The directory to store temporary checkpoints. The default value is `/tmp`

Use this option to indicate the directory to store temporary checkpoints. Checkpoints are migrated from `-ckpoint-tmp-prefix` to the directory specified in `-ckpoint-prefix`. The directory *<dir>* should be writable; otherwise the application will abort during startup with an error message. Temporary storage is not used if the option is not set.

-ckpoint-logfile *<file>*

Use this option for monitoring checkpoint activity. The trace is dumped into `<file>`. You should be able to write in `<file>`; otherwise the application will abort during startup with an error message. This is an optional feature.

Checkpoint-Restart Environment Variables

I_MPI_CKPOINT

Syntax

`I_MPI_CKPOINT=<switch>`

Arguments

<switch>	Checkpoint switch
`enable | yes | on | 1`	Enables the check point function for the application started
`disable | no | off | 0`	Disables the check point function for the application started. This is the default value

Description

Use this variable to turn on taking checkpoints capability. This has the same effect as the `-ckpoint` option. If you have set the `-ckpoint` option, the Hydra process manager sets the `I_MPI_CKPOINT` even if you do not set this environment variable.

I_MPI_CKPOINTLIB

Syntax

`I_MPI_CKPOINTLIB=<lib>`

Arguments

<lib>	Checkpoint-Restart system library
`blcr`	Berkeley Lab Checkpoint/Restart* (BLCR) Library. This is the default value

Description

Use this variable to select underlying Checkpoint-Restart system library. This has the same effect as the `-ckpointlib` option.

I_MPI_CKPOINT_PREFIX

Syntax

`I_MPI_CKPOINT_PREFIX=<dir>`

Arguments

<dir>	The directory to store checkpoints. The default value is `/tmp`

Description

Use this variable to specify a directory to store checkpoints. This has the same effect as the `-ckpoint-prefix` option.

I_MPI_CKPOINT_TMP_PREFIX

Syntax

`I_MPI_CKPOINT_TMP_PREFIX=<dir>`

Arguments

<dir>	The directory to store temporary checkpoints

Description

Use this variable to indicate storage of temporary checkpoints while `-ckpoint-prefix` indicates permanent storage. This has the same effect as the `-ckpoint-tmp-prefix` option.

I_MPI_CKPOINT_INTERVAL

Syntax

`I_MPI_CKPOINT_INTERVAL=<sec>`

Arguments

<sec>	Interval between consecutive checkpoints in seconds

Description

Use this variable to turn on timer driven checkpoints. This has the same effect as the `-ckpoint-interval` option.

I_MPI_CKPOINT_PRESERVE

Syntax

`I_MPI_CKPOINT_PRESERVE=<N>`

Arguments

<N>	Maximal number of checkpoint images kept. The default value is `1`

Description

Use this option while running the checkpoint function to keep last *<N>* checkpoints to reduce checkpoint image space. This has the same effect as the `-ckpoint-preserve` option.

I_MPI_CKPOINT_LOGFILE

Syntax

`I_MPI_CKPOINT_LOGFILE=<file>`

Arguments

<file>	The file keeping the trace for checkpoint activity

Description

Use this option for checkpoint activity monitoring. The trace is dumped into *<file>*. This has the same effect as the `-ckpoint-logfile` option.

I_MPI_CKPOINT_NUM

Syntax

`I_MPI_CKPOINT_NUM=<N>`

Arguments

<N>	Number of checkpoint image to restart an application with

Description

Use this option while restarting application. This has the same effect as the `-ckpoint-num` option.

I_MPI_RESTART

Syntax

`I_MPI_RESTART=<switch>`

Arguments

<switch>	Restart switch
`enable | yes | on | 1`	Enables the restart of the application from one of the stored checkpoints.
`disable | no | off | 0`	Disables the restart of the application. This is the default value.

Description

Use this variable to restart an application from one of the stored checkpoints. Using this variable has the same effect as `-restart` option.

Using Checkpoint-Restart

The checkpoint-restart feature is available with the Hydra process launcher (`mpiexec.hydra`) or its wrapper (`mpirun`). The launcher provides two mutually exclusive methods of taking checkpoints:

- By timers
- By an explicit signal

You can provide directory paths where checkpoints can be stored temporarily and permanently.

Timer Driven Checkpoint

In the following example, a checkpoint is taken every 3600 seconds (=1hour). The checkpoints are stored in a directory called `ckptdir`. Each node generates one checkpoint, which is named by the node number and number of that checkpoint.

```
$ mpirun -ckpoint on -ckpoint-prefix ~/user/ckptdir -ckpoint-interval 3600 -
ckpointlib blcr -n 32 -f hosts ./myprog
```

Explicit Signal Driven Checkpoint

In the following example, an application is started and then an explicit signal (`SIGUSR1`) is passed to the application to take a checkpoint. The checkpoints are stored in the `ckptdir` directory.

```
$ mpirun -ckpoint on -ckpoint-prefix ~/user/ckptdir -ckpointlib blcr -n 32 -f hosts
./myprog
...
$ kill -s SIGUSR1 <PID of mpiexec.hydra>
```

It is necessary and sufficient that you signal the `mpiexec.hydra` process on the local node.

Using Local Storage

In the following example, there are two locations for storing checkpoints.

- Temporary location: indicated in the argument to `-ckpoint-tmp-prefix`
- Permanent location: indicated in the argument to `-ckpoint-prefix`

```
$ mpirun -ckpoint on -ckpoint-tmp-prefix /ssd/user/ckptdir \
-ckpoint-prefix ~/user/ckptdir -ckpointlib blcr -n 32 -f hosts ./myprog
```

Restarting MPI Applications

The following is an example of restarting an application from checkpoint number *<N>*.

```
$ mpirun -restart -ckpoint-prefix ~/user/ckptdir -ckpointlib blcr -ckpoint-num <N>
-n 32 -f hosts
```

When restarting, you need to revise the host file to eliminate any dead or unavailable nodes. Also, providing the executable name is not necessary when restarting because it is already stored in the checkpoint images.

Viewing Checkpoint Activity in Log File

The following is an example of launching an MPI job and specifying a checkpoint log file for being able to watch the checkpoint activity.

```
$ mpirun -ckpoint on -ckpoint-logfile ~/user/ckpt.log \
-ckpoint-tmp-prefix /ssd/user/ckptdir -ckpoint-prefix ~/user/ckptdir -ckpointlib
blcr \
-n 32 -f hosts ./myprog
```

A sample output log may look as follows:

```
Checkpoint log initialized (master mpiexec pid 10687, 48 processes, 6 nodes)
Permanent checkpoint storage: /mnt/lustre/user
Temporary checkpoint storage: /tmp
Started checkpoint number 0 ...
Finished checkpoint number 0.
Moving checkpoint 0 from /tmp to /mnt/lustre/user...
Moved checkpoint 0 from /tmp to /mnt/lustre/user
```

Automatic Cleanup of Previous Checkpoints

Checkpoint images are large. Therefore, the Intel® MPI Library only keeps the last useful checkpoint, by default. The following is an example demonstrating how to keep *<N>* previous checkpoints. The `-ckpoint-preserve` *<N>* option is used to do so. The option default value is `1` (keep only the last checkpoint).

```
$ mpirun -ckpoint on -ckpoint-preserve <N> \
-ckpoint-tmp-prefix /ssd/user/ckptdir -ckpoint-prefix ~/user/ckptdir -ckpointlib
blcr \
-n 32 -f hosts ./myprog
```

2.4. Processor Information Utility

cpuinfo

The `cpuinfo` utility provides processor architecture information.

Syntax

`cpuinfo [[-]<options>]`

Arguments

`<options>`	Sequence of one-letter options. Each option controls a specific part of the output data.
`g`	General information about single cluster node shows: • the processor product name • the number of packages/sockets on the node • core and threads numbers on the node and within each package • SMT mode enabling
`i`	Logical processors identification table identifies threads, cores, and packages of each logical processor accordingly. • *Processor* - logical processor number. • *Thread Id* - unique processor identifier within a core. • *Core Id* - unique core identifier within a package. • *Package Id* - unique package identifier within a node.
`d`	Node decomposition table shows the node contents. Each entry contains the information on packages, cores, and logical processors. • *Package Id* - physical package identifier. • *Cores Id* - list of core identifiers that belong to this package. • *Processors Id* - list of processors that belong to this package. This list order directly corresponds to the core list. A group of processors enclosed in brackets belongs to one core.
`c`	Cache sharing by logical processors shows information of sizes and processors groups, which share particular cache level. • Size - cache size in bytes. • Processors - a list of processor groups enclosed in the parentheses those share this cache or no sharing otherwise.

s	Microprocessor signature hexadecimal fields (Intel platform notation) show signature values: • extended family • extended model • family • model • type • stepping
f	Microprocessor feature flags indicate what features the microprocessor supports. The Intel platform notation is used.
A	Equivalent to gidcsf
gidc	Default sequence
?	Utility usage info

Description

The cpuinfo utility prints out the processor architecture information that can be used to define suitable process pinning settings. The output consists of a number of tables. Each table corresponds to one of the single options listed in the arguments table.

NOTE

The architecture information is available on systems based on the Intel® 64 architecture.

The cpuinfo utility is available for both Intel microprocessors and non-Intel microprocessors, but it may provide only partial information about non-Intel microprocessors.

An example of the cpuinfo output:

```
$ cpuinfo -gdcs
=====  Processor composition  =====
Processor name    : Intel(R) Xeon(R)  X5570
Packages(sockets) : 2
Cores             : 8
Processors(CPUs)  : 8
Cores per package : 4
Threads per core  : 1
=====  Processor identification  =====
Processor       Thread Id.      Core Id.        Package Id.
0               0               0               0
1               0               0               1
2               0               1               0
3               0               1               1
4               0               2               0
5               0               2               1
6               0               3               0
7               0               3               1
=====  Placement on packages  =====
Package Id.     Core Id.        Processors
0               0,1,2,3         0,2,4,6
```

```
1               0,1,2,3         1,3,5,7
=====  Cache sharing  =====
Cache   Size            Processors
L1      32  KB          no sharing
L2      256 KB          no sharing
L3      8   MB          (0,2,4,6)(1,3,5,7)
=====  Processor Signature  =====
 _________ ________ ______ ________ _______ ___________
| xFamily | xModel | Type | Family | Model | Stepping |
|_________|________|______|________|_______|___________|
| 00      | 1      | 0    | 6      | a     | 5        |
|_________|________|______|________|_______|___________|
```

3. Tuning Reference

3.1. mpitune Utility

mpitune

Use the `mpitune` utility to find optimal settings for the Intel® MPI Library relevant to your cluster configuration or your application.

Syntax

```
mpitune [options]
```

Options

`-a \"<app_cmd_line>\"` `--application \"<app_cmd_line>\"`	Enable the application-specific mode. Quote the full command line as shown, including the backslashes.
`-f {on|off}` `--fast {on|off}`	Enable or disable the fast application tuning mode. See `I_MPI_TUNE_FAST` for details.
`-rp {on|off}` `--rank-placement {on|off}`	Enable or disable the topology aware tuning mode. See for more details: `-use-app-topology`, `I_MPI_HYDRA_USE_APP_TOPOLOGY` and `I_MPI_TUNE_RANK_PLACEMENT`.
`-of <file-name>` `--output-file <file-name>`	Specify the name of the application configuration file to be generated in the application-specific mode. By default, use the file name `$PWD/app.conf`.
`-t \"<test_cmd_line>\"` `--test \"<test_cmd_line>\"`	Replace the default Intel® MPI Benchmarks by the indicated benchmarking program in the cluster-specific mode. Quote the full command line as shown including the backslashes.
`-cm {exclusive|full}` `--cluster-mode {exclusive|full}`	Set the cluster usage mode: • `full` - maximum number of tasks are executed. This is the default mode. • `exclusive` - only one task is executed on the cluster at a time.
`-d | --debug`	Print out the debug information.
`-D | --distinct`	Tune all options separately from each other. This argument is applicable only for the cluster-specific mode.
`-dl [d1[,d2...[,dN]]]` `--device-list [d1[,d2,…[,dN]]]`	Select the device(s) you want to tune. Any previously set fabrics are ignored. By default, use all devices listed in the `<installdir>/<arch>/etc/devices.xml` file.
`-fl [f1[,f2...[,fN]]]`	Select the fabric(s) you want to tune. Any previously set devices are ignored.

`--fabric-list` *`[f1[,f2…[,fN]]]`*	By default, use all fabrics listed in the *`<installdir>/<arch>`*`/etc/fabrics.xml` file.
`-hf` *`<hostsfile>`* `--host-file` *`<hostsfile>`*	Specify an alternative host file name. By default, use the `$PWD/mpd.hosts`.
`-h | --help`	Display the help message.
`-hr` *`{min:max|min:|:max}`* `--host-range` *`{min:max|min:|:max}`*	Set the range of hosts used for testing. The default minimum value is `1`. The default maximum value is the number of hosts defined by the `mpd.hosts`. The `min:` or `:max` format uses the default values as appropriate.
`-i` *`<count>`* `--iterations` *`<count>`*	Define how many times to run each tuning step. Higher iteration counts increase the tuning time, but may also increase the accuracy of the results. The default value is `3`.
`-mr` *`{min:max|min:|:max}`* `--message-range` *`{min:max|min:|:max}`*	Set the message size range. The default minimum value is `0`. The default maximum value is `4194304` (`4mb`). By default, the values are given in bytes. They can also be given in the following format: `16kb`, `8mb` or `2gb`. The `min:` or `:max` format uses the default values as appropriate.
`-od` *`<outputdir>`* `--output-directory` *`<outputdir>`*	Specify the directory name for all output files: log-files, session-files, local host-files and report-files. By default, use the current directory. This directory should be accessible from all hosts.
`-odr` *`<outputdir>`* `--output-directory-results` *`<outputdir>`*	Specify the directory name for the resulting configuration files. By default, use the current directory in the application-specific mode and the `<installdir>/<arch>/etc` in the cluster-specific mode. If `<installdir>/<arch>/etc` is unavailable, `$PWD` is used as the default value in the cluster-specific mode.
`-r` *`<rshcmd>`* `--rsh` *`<rshcmd>`*	Specify the remote shell used to start daemons (as applicable) and jobs. The default value is `ssh`.
`-pr` *`{min:max|min:|:max}`* `--ppn-range` *`{min:max|min:|:max}`* `--perhost-range` *`{min:max|min:|:max}`*	Set the maximum number of processes per host. The default minimum value is `1`. The default maximum value is the number of cores of the processor. The `min:` or `:max` format uses the default values as appropriate.
`-sf` *`[file-path]`* `--session-file` *`[file-path]`*	Continue the tuning process starting from the state saved in the `file-path` session file.
`-ss | --show-session`	Show information about the session file and exit. This option works only jointly with the `-sf` option.
`-s | --silent`	Suppress all diagnostics.

`-td <dir-path>` `--temp-directory <dir-path>`	Specify a directory name for the temporary data. Use `$PWD/mpitunertemp` by default. This directory should be accessible from all hosts.
`-tl <minutes>` `--time-limit <minutes>`	Set `mpitune` execution time limit in minutes. The default value is `0`, which means no limitations.
`-mh | --master-host`	Dedicate a single host to run the `mpitune`.
`-os <opt1,...,optN>` `--options-set <opt1,...,optN>`	Use `mpitune` to tune the only required options you have set in the option values
`-oe <opt1,...,optN>` `--options-exclude <opt1,...,optN>`	Exclude the settings of the indicated Intel® MPI Library options from the tuning process.
`-V | --version`	Print out the version information.

`-vi <percent>`
`--valuable-improvement <percent>`
`-vix < X factor>`
`--valuable-improvement-x <X factor>`
	Control the threshold for performance improvement. The default threshold is `3%`.

`-zb | --zero-based`	Set zero as the base for all options before tuning. This argument is applicable only for the cluster-specific mode.
`-t | --trace`	Print out error information such as error codes and tuner trace back.
`-so | --scheduler-only`	Create the list of tasks to be executed, display the tasks, and terminate execution.
`-ar \"reg-expr\"` `--application-regexp \"reg-expr\"`	Use `reg-expr` to determine the performance expectations of the application. This option is applicable only for the application-specific mode. The `reg-expr` setting should contain only one group of numeric values which is used by `mpitune` for analysis. Use backslash for symbols when setting the value of this argument in accordance with the operating system requirements.
`-trf <appoutfile>` `--test-regexp-file <appoutfile>`	Use a test output file to check the correctness of the regular expression. This argument is applicable only for the cluster-specific mode when you use the `–ar` option.
`-m {base|optimized}` `--model {base|optimized}`	Specify the search model: • Set `base` to use the old model. • Set `optimized` to use the new faster search model. This is the default

	value.
`-avd {min|max}` `--application-value-direction {min|max}`	Specify the direction of the value optimization : • Set `min` to specify that lower is better. For example, use this value when optimizing the wall time. • Set `max` to specify that higher is better. For example, use this value when optimizing the solver ratio.
`-co | --collectives-only`	Tune collective operations only.
`-sd | --save-defaults`	Use `mpitune` to save the default values of the Intel® MPI Library options.
`-soc` `--skip-options-check`	Specify whether to check the command line options.

Deprecated Options

Deprecated Option	New Option
`--outdir`	`-od | --output-directory`
`--verbose`	`-d | --debug`
`--file`	`-hf | --host-file`
`--logs`	`-lf | --log-file`
`--app`	`-a | --application`
`--session-file (-sf)`	None
`--show-session (-ss)`	None

Description

Use the `mpitune` utility to create a set of Intel® MPI Library configuration files that contain optimal settings for a particular cluster or application. You can reuse these configuration files in the `mpirun` job launcher by using the `-tune` option. If configuration files from previous `mpitune` sessions exist, `mpitune` creates a copy of the existing files before starting execution.

The MPI tuner utility operates in two modes:

- Cluster-specific, evaluating a given cluster environment using either the Intel® MPI Benchmarks or a user-provided benchmarking program to find the most suitable configuration of the Intel® MPI Library. This mode is used by default.
- Application-specific, evaluating the performance of a given MPI application to find the best configuration for the Intel® MPI Library for the particular application. Application tuning is enabled by the `--application` command line option.

3.1.1. mpitune Environment Variables

I_MPI_TUNE_FAST

Syntax

```
I_MPI_TUNE_FAST=<value>
```

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Enable the fast application tuning. This value is only applicable for the application tuning mode.
`disable | no | off | 0`	Disable the fast application tuning. This is the default value.

Description

If you set `I_MPI_TUNE_FAST` to `enable`, the `mpitune` utility executes the alternative fast application tuning procedure. The fast application tuning uses the same resulting configuration file used in the standard application tuning. You can also use the `--fast` option to enable or disable this mode. This option overrides the `I_MPI_TUNE_FAST` value.

Examples

To enable the fast application tuning mode, run the command:

```
$ mpitune --application \"mpirun …\" --fast
```

or

```
$ export I_MPI_TUNE_FAST=enable
$ mpitune --application \"mpirun …\"
```

To disable the fast application tuning mode, run the command:

```
$ mpitune --application \"mpirun …\" --fast off
```

or

```
$ export I_MPI_TUNE_FAST=disable
$ mpitune --application \"mpirun …\"
```

NOTE

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

I_MPI_TUNE_RANK_PLACEMENT

Syntax

```
I_MPI_TUNE_RANK_PLACEMENT=<value>
```

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Enable rank placement optimization.

`disable | no | off | 0`	Disable rank placement optimization. This is the default value.

Description

If you set `I_MPI_TUNE_RANK_PLACEMENT` to `enable`, `mpitune` performs the rank placement optimization tuning. You can also use the `--rank-placement` option to enable or disable this mode. This option overrides the `I_MPI_TUNE_RANK_PLACEMENT` value.

Examples

Using the environment variable:

```
$ export I_MPI_TUNE_RANK_PLACEMENT=enable
$ mpitune --application \"mpirun … \"
```

Using the option:

```
$ mpitune --application \"mpirun … \" --rank-placement -hi <input host file>
$ mpitune --application \"mpirun … \" --rank-placement enable -hi <input host file>
```

The result is an optimized host file and a configuration file that applies the host file.

NOTE

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

I_MPI_TUNE_APPLICATION_STATISTICS

Syntax

`--statistics <value>` or `-s <value>`

or

`I_MPI_TUNE_APPLICATION_STATICSTICS=<value>`

Arguments

<value>	Path to the native Intel MPI Library statistics file level 1 or higher.

Description

Use this option or variable to pass the Intel MPI Library statistics file to `mpitune`. You can generate a statistics file manually using the `I_MPI_STATS` environment variable. A statistics file is also generated when you perform rank placement optimization tuning, its name and location is printed in the console output:

```
Statistics collecting... done: <file location>
```

Specifying the statistics file helps reduce the tuning time. This setting is applicable only with `--rank-placement` or `I_MPI_TUNE_RANK_PLACEMENT` enabled.

NOTE

When specifying a statistics file, you also need to specify the host file (`--hostfile-in` or `-hi` option) or a hardware topology graph file (`-htg` option, see below).

Example:

```
$ mpitune -rp -s stats.txt -hi <hostfile>
```

I_MPI_TUNE_APPLICATION_COMMUNICATION_GRAPH

Syntax

`--application-communication-graph <value>` or `-acg <value>`

or

`I_MPI_TUNE_APPLICATION_COMMUNICATION_GRAPH=<value>`

Arguments

<value>	Path to the application communication graph (ACG) file.

Description

Use this option or variable to pass the previously generated ACG file to `mpitune`. The ACG file is generated when you perform rank placement optimization tuning, its name and location is printed in the console output:

```
Application communication graph initialization... done: <file location>
```

Specifying the ACG file helps reduce the tuning time. This setting is applicable only with `--rank-placement` or `I_MPI_TUNE_RANK_PLACEMENT` enabled.

NOTE

When specifying an ACG file, you also need to specify the host file (`--hostfile-in` or `-hi` option) or a hardware topology graph file (`-htg` option, see below).

Example:

```
$ mpitune -rp -acg <path to ACG file> -hi <hostfile>
```

I_MPI_TUNE_HARDWARE_TOPOLOGY_GRAPH

Syntax

`--hardware-topology-graph <value>` or `-htg <value>`

or

`I_MPI_TUNE_HARDWARE_TOPOLOGY_GRAPH=<value>`

Arguments

<value>	Path to the hardware topology graph (HTG) file.

Description

Use this option or variable to pass the HTG file to `mpitune`. The HTG file is generated when you perform rank placement optimization tuning, its name and location is printed in the console output:

```
Hardware topology initialization... done: <file location>
```

Specifying the HTG file helps reduce the tuning time. This setting is applicable only with `--rank-placement` or `I_MPI_TUNE_RANK_PLACEMENT` enabled.

NOTE

When specifying an HTG file, you also need to specify one of the following: an ACG file (`-acg` option, see above), an application command line (`--application` or `-a` option), or a statistics file (`--statistics` or `-s` option, see above).

Example:

```
$ mpitune -rp -htg <path to HTG file> -acg <path to ACG file>
```

3.2. Process Pinning

Use this feature to pin a particular MPI process to a corresponding CPU within a node and avoid undesired process migration. This feature is available on operating systems that provide the necessary kernel interfaces.

3.2.1. Processor Identification

The following schemes are used to identify logical processors in a system:

- System-defined logical enumeration
- Topological enumeration based on three-level hierarchical identification through triplets (package/socket, core, thread)

The number of a logical CPU is defined as the corresponding position of this CPU bit in the kernel affinity bit-mask. Use the `cpuinfo` utility, provided with your Intel MPI Library installation or the `cat /proc/cpuinfo` command to find out the logical CPU numbers.

The three-level hierarchical identification uses triplets that provide information about processor location and their order. The triplets are hierarchically ordered (package, core, and thread).

See the example for one possible processor numbering where there are two sockets, four cores (two cores per socket), and eight logical processors (two processors per core).

NOTE

Logical and topological enumerations are not the same.

Table 3.2-1 Logical Enumeration

0	4	1	5	2	6	3	7

Table 3.2-2 Hierarchical Levels

Socket	0	0	0	0	1	1	1	1
Core	0	0	1	1	0	0	1	1
Thread	0	1	0	1	0	1	0	1

Table 3.2-3 Topological Enumeration

0	1	2	3	4	5	6	7

Use the `cpuinfo` utility to identify the correspondence between the logical and topological enumerations. See Processor Information Utility for more details.

3.2.2. Default Settings

If you do not specify values for any process pinning environment variables, the default settings below are used. For details about these settings, see Environment Variables and Interoperability with OpenMP API.

- I_MPI_PIN=on
- I_MPI_PIN_MODE=pm
- I_MPI_PIN_RESPECT_CPUSET=on
- I_MPI_PIN_RESPECT_HCA=on
- I_MPI_PIN_CELL=unit
- I_MPI_PIN_DOMAIN=auto:compact
- I_MPI_PIN_ORDER=compact

3.2.3. Environment Variables for Process Pinning

I_MPI_PIN

Turn on/off process pinning.

Syntax

I_MPI_PIN=*<arg>*

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Enable process pinning. This is the default value
disable | no | off | 0	Disable processes pinning

Description

Set this environment variable to control the process pinning feature of the Intel® MPI Library.

I_MPI_PIN_MODE

Choose the pinning method.

Syntax

I_MPI_PIN_MODE=*<pinmode>*

Arguments

<pinmode>	Choose the CPU pinning mode
pm	Pin processes inside the process manager. This is the default value
lib	Pin processes inside the Intel MPI Library

Description

Set the I_MPI_PIN_MODE environment variable to choose the pinning method. This environment variable is valid only if the I_MPI_PIN environment variable is enabled.

If you set the I_MPI_PIN_MODE environment variable to pm, the Hydra process launcher pins processes through system specific means, if they are available. The pinning is done before the MPI process launch. Therefore, it is possible to co-locate the process CPU and memory. This pinning method has an advantage

over a system with Non-Uniform Memory Architecture (NUMA) like SGI* Altix*. Under NUMA, a processor can access its own local memory faster than non-local memory.

If you set the `I_MPI_PIN_MODE` environment variable to `lib`, the Intel® MPI Library pins the processes. This mode does not offer the capability to co-locate the CPU and memory for a process.

I_MPI_PIN_PROCESSOR_LIST (I_MPI_PIN_PROCS)

Define a processor subset and the mapping rules for MPI processes within this subset.

Syntax

`I_MPI_PIN_PROCESSOR_LIST=<value>`

The environment variable value has the following syntax forms:

1. `<proclist>`
2. `[<procset>][:[grain=<grain>][,shift=<shift>][,preoffset=<preoffset>][,postoffset=<postoffset>]`
3. `[<procset>][:map=<map>]`

The following paragraphs provide detail descriptions for the values of these syntax forms.

Deprecated Syntax

`I_MPI_PIN_PROCS=<proclist>`

NOTE

The `postoffset` keyword has `offset` alias.

NOTE

The second form of the pinning procedure has three steps:

1. Cyclic shift of the source processor list on `preoffset*grain` value.
2. Round robin shift of the list derived on the first step on `shift*grain` value.
3. Cyclic shift of the list derived on the second step on the `postoffset*grain` value.

NOTE

The `grain`, `shift`, `preoffset`, and `postoffset` parameters have a unified definition style.

This environment variable is available for both Intel® and non-Intel microprocessors, but it may perform additional optimizations for Intel microprocessors than it performs for non-Intel microprocessors.

Syntax

`I_MPI_PIN_PROCESSOR_LIST=<proclist>`

Arguments

`<proclist>`	A comma-separated list of logical processor numbers and/or ranges of processors. The process with the i-th rank is pinned to the i-th processor in the list. The number should not exceed the amount of processors on a node.

<l>	Processor with logical number *<l>*.
<l>-<m>	Range of processors with logical numbers from *<l>* to *<m>*.
<k>,<l>-<m>	Processors *<k>*, as well as *<l>* through *<m>*.

Syntax

`I_MPI_PIN_PROCESSOR_LIST=[<procset>][:[grain=<grain>][,shift=<shift>][,preoffset=<preoffset>][,postoffset=<postoffset>]`

Arguments

<procset>	Specify a processor subset based on the topological numeration. The default value is `allcores`.
`all`	All logical processors. Specify this subset to define the number of CPUs on a node.
`allcores`	All cores (physical CPUs). Specify this subset to define the number of cores on a node. This is the default value. If Intel® Hyper-Threading Technology is disabled, `allcores` equals to `all`.
`allsocks`	All packages/sockets. Specify this subset to define the number of sockets on a node.

<grain>	Specify the pinning granularity cell for a defined *<procset>*. The minimal *<grain>* value is a single element of the *<procset>*. The maximal *<grain>* value is the number of *<procset>* elements in a socket. The *<grain>* value must be a multiple of the *<procset>* value. Otherwise, the minimal *<grain>* value is assumed. The default value is the minimal *<grain>* value.
<shift>	Specify the granularity of the round robin scheduling shift of the cells for the *<procset>*. *<shift>* is measured in the defined *<grain>* units. The *<shift>* value must be positive integer. Otherwise, no shift is performed. The default value is no shift, which is equal to `1` normal increment
<preoffset>	Specify the cyclic shift of the processor subset *<procset>* defined before the round robin shifting on the *<preoffset>* value. The value is measured in the defined *<grain>* units. The *<preoffset>* value must be non-negative integer. Otherwise, no shift is performed. The default value is no shift.
<postoffset>	Specify the cyclic shift of the processor subset *<procset>* derived after round robin shifting on the *<postoffset>* value. The value is measured in the defined *<grain>* units. The *<postoffset>* value must be non-negative integer. Otherwise no shift is performed. The default value is no shift.

The following table displays the values for <grain>, <shift>, <preoffset>, and <postoffset> options:

<n>	Specify an explicit value of the corresponding parameters. *<n>* is non-negative integer.
`fine`	Specify the minimal value of the corresponding parameter.

core	Specify the parameter value equal to the amount of the corresponding parameter units contained in one core.
cache1	Specify the parameter value equal to the amount of the corresponding parameter units that share an L1 cache.
cache2	Specify the parameter value equal to the amount of the corresponding parameter units that share an L2 cache.
cache3	Specify the parameter value equal to the amount of the corresponding parameter units that share an L3 cache.
cache	The largest value among cache1, cache2, and cache3.
socket | sock	Specify the parameter value equal to the amount of the corresponding parameter units contained in one physical package/socket.
half | mid	Specify the parameter value equal to socket/2.
third	Specify the parameter value equal to socket/3.
quarter	Specify the parameter value equal to socket/4.
octavo	Specify the parameter value equal to socket/8.

Syntax

I_MPI_PIN_PROCESSOR_LIST=[*<procset>*][:map=*<map>*]

Arguments

<map>	The mapping pattern used for process placement.
bunch	The processes are mapped as close as possible on the sockets.
scatter	The processes are mapped as remotely as possible so as not to share common resources: FSB, caches, and core.
spread	The processes are mapped consecutively with the possibility not to share common resources.

Description

Set the I_MPI_PIN_PROCESSOR_LIST environment variable to define the processor placement. To avoid conflicts with different shell versions, the environment variable value may need to be enclosed in quotes.

NOTE

This environment variable is valid only if I_MPI_PIN is enabled.

The I_MPI_PIN_PROCESSOR_LIST environment variable has the following different syntax variants:

- Explicit processor list. This comma-separated list is defined in terms of logical processor numbers. The relative node rank of a process is an index to the processor list such that the i-th process is pinned on i-th list member. This permits the definition of any process placement on the CPUs.

  For example, process mapping for `I_MPI_PIN_PROCESSOR_LIST=p0,p1,p2,...,pn` is as follows:

Rank on a node	0	1	2	...	n-1	N
Logical CPU	p0	p1	p2	...	pn-1	Pn

- `grain/shift/offset` mapping. This method provides cyclic shift of a defined `grain` along the processor list with steps equal to `shift*grain` and a single shift on `offset*grain` at the end. This shifting action is repeated `shift` times.

  For example: grain = 2 logical processors, shift = 3 grains, offset = 0.

Legend:

gray - MPI process grains

A) red - processor grains chosen on the 1st pass

B) cyan - processor grains chosen on the 2nd pass

C) green - processor grains chosen on the final 3rd pass

D) Final map table ordered by MPI ranks

A)

0 1			2 3			...	2n-2 2n-1		
0 1	2 3	4 5	6 7	8 9	10 11	...	6n-6 6n-5	6n-4 6n-3	6n-2 6n-1

B)

0 1	2n 2n+1		2 3	2n+2 2n+3		...	2n-2 2n-1	4n-2 4n-1	
0 1	2 3	4 5	6 7	8 9	10 11	...	6n-6 6n-5	6n-4 6n-3	6n-2 6n-1

C)

0 1	2n 2n+1	4n 4n+1	2 3	2n+2 2n+3	4n+2 4n+3	...	2n-2 2n-1	4n-2 4n-1	6n-2 6n-1
0 1	2 3	4 5	6 7	8 9	10 11	...	6n-6 6n-5	6n-4 6n-3	6n-2 6n-1

D)

0 1	2 3	...	2n-2 2n-1	2n 2n+1	2n+2 2n+3	...	4n-2 4n-1	4n 4n+1	4n+2 4n+3	...	6n-2 6n-1
0 1	6 7	...	6n-6 6n-5	2 3	8 9	...	6n-4 6n-3	4 5	10 11	...	6n-2 6n-1

- Predefined mapping scenario. In this case popular process pinning schemes are defined as keywords selectable at runtime. There are two such scenarios: `bunch` and `scatter`.

In the `bunch` scenario the processes are mapped proportionally to sockets as closely as possible. This mapping makes sense for partial processor loading. In this case the number of processes is less than the number of processors.

In the `scatter` scenario the processes are mapped as remotely as possible so as not to share common resources: FSB, caches, and cores.

In the example, there are two sockets, four cores per socket, one logical CPU per core, and two cores per shared cache.

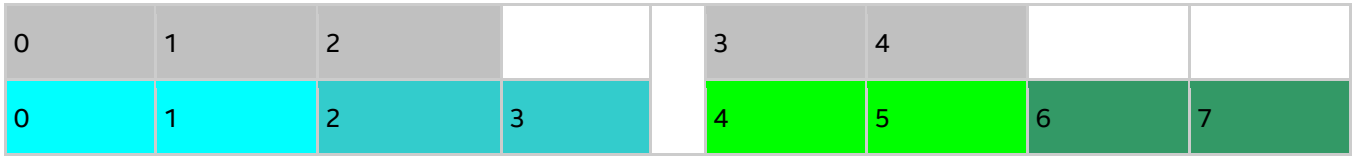
Legend:

gray - MPI processes

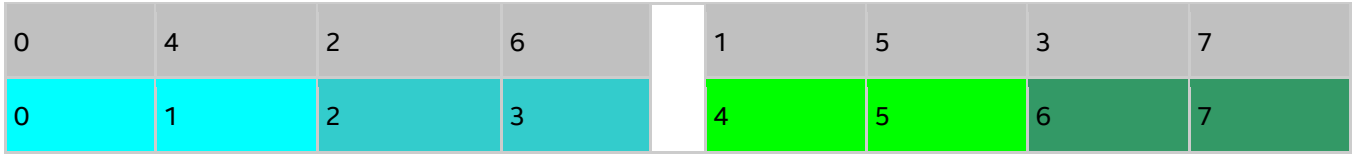
cyan - 1st socket processors

green - 2nd socket processors

Same color defines a processor pair sharing a cache



`bunch` scenario for 5 processes



`scatter` scenario for full loading

Examples

To pin the processes to CPU0 and CPU3 on each node globally, use the following command:

```
$ mpirun -genv I_MPI_PIN_PROCESSOR_LIST=0,3 -n <# of processes> <executable>
```

To pin the processes to different CPUs on each node individually (CPU0 and CPU3 on host1 and CPU0, CPU1 and CPU3 on host2), use the following command:

```
$ mpirun -host host1 -env I_MPI_PIN_PROCESSOR_LIST=0,3 -n <# of processes>
<executable> : \
-host host2 -env I_MPI_PIN_PROCESSOR_LIST=1,2,3 -n <# of processes> <executable>
```

To print extra debug information about process pinning, use the following command:

```
$ mpirun -genv I_MPI_DEBUG=4 -m -host host1 \
-env I_MPI_PIN_PROCESSOR_LIST=0,3 -n <# of processes> <executable> :\
-host host2 -env I_MPI_PIN_PROCESSOR_LIST=1,2,3 -n <# of processes> <executable>
```

NOTE

If the number of processes is greater than the number of CPUs used for pinning, the process list is wrapped around to the start of the processor list.

I_MPI_PIN_PROCESSOR_EXCLUDE_LIST

Define a subset of logical processors to be excluded for the pinning capability on the intended hosts.

Syntax

I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=<*proclist*>

Arguments

<*proclist*>	A comma-separated list of logical processor numbers and/or ranges of processors.
<*l*>	Processor with logical number <*l*>.
<*l*>-<*m*>	Range of processors with logical numbers from <*l*> to <*m*>.
<*k*>,<*l*>-<*m*>	Processors <*k*>, as well as <*l*> through <*m*>.

Description

Set this environment variable to define the logical processors that Intel® MPI Library does not use for pinning capability on the intended hosts. Logical processors are numbered as in `/proc/cpuinfo`.

I_MPI_PIN_CELL

Set this environment variable to define the pinning resolution granularity. `I_MPI_PIN_CELL` specifies the minimal processor cell allocated when an MPI process is running.

Syntax

I_MPI_PIN_CELL=<*cell*>

Arguments

<*cell*>	Specify the resolution granularity
`unit`	Basic processor unit (logical CPU)
`core`	Physical processor core

Description

Set this environment variable to define the processor subset used when a process is running. You can choose from two scenarios:

- all possible CPUs in a node (`unit` value)
- all cores in a node (`core` value)

The environment variable has effect on both pinning types:

- one-to-one pinning through the `I_MPI_PIN_PROCESSOR_LIST` environment variable
- one-to-many pinning through the `I_MPI_PIN_DOMAIN` environment variable

The default value rules are:

- If you use `I_MPI_PIN_DOMAIN`, then the cell granularity is `unit`.
- If you use `I_MPI_PIN_PROCESSOR_LIST`, then the following rules apply:
  - When the number of processes is greater than the number of cores, the cell granularity is `unit`.
  - When the number of processes is equal to or less than the number of cores, the cell granularity is `core`.

NOTE

The `core` value is not affected by the enabling/disabling of Intel® Hyper-Threading Technology in a system.

I_MPI_PIN_RESPECT_CPUSET

Respect the process affinity mask.

Syntax

`I_MPI_PIN_RESPECT_CPUSET=<value>`

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Respect the process affinity mask. This is the default value
`disable | no | off | 0`	Do not respect the process affinity mask

Description

If you set `I_MPI_PIN_RESPECT_CPUSET=enable`, the Hydra process launcher uses its process affinity mask on each intended host to determine logical processors for applying Intel MPI Library pinning capability.

If you set `I_MPI_PIN_RESPECT_CPUSET=disable`, the Hydra process launcher does not use its process affinity mask to determine logical processors for applying Intel MPI Library pinning capability.

I_MPI_PIN_RESPECT_HCA

In the presence of Infiniband architecture* host channel adapter (IBA* HCA), adjust the pinning according to the location of IBA HCA.

Syntax

`I_MPI_PIN_RESPECT_HCA=<value>`

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Use the location of IBA HCA if available. This is the default value
`disable | no | off | 0`	Do not use the location of IBA HCA

Description

If you set `I_MPI_PIN_RESPECT_HCA=enable`, the Hydra process launcher uses the location of IBA HCA on each intended host for applying Intel MPI Library pinning capability.

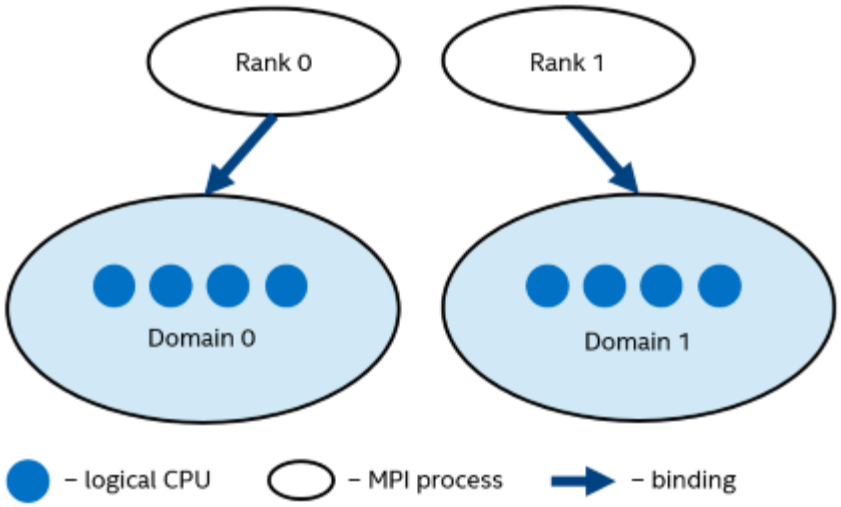
If you set `I_MPI_PIN_RESPECT_HCA=disable`, the Hydra process launcher does not use the location of IBA HCA on each intended host for applying Intel MPI Library pinning capability.

3.2.4. Interoperability with OpenMP* API

I_MPI_PIN_DOMAIN

Intel® MPI Library provides an additional environment variable to control process pinning for hybrid MPI/OpenMP* applications. This environment variable is used to define a number of non-overlapping subsets (domains) of logical processors on a node, and a set of rules on how MPI processes are bound to these domains by the following formula: *one MPI process per one domain*. See the picture below.

Figure 3.2-1 Domain Example



Each MPI process can create a number of children threads for running within the corresponding domain. The process threads can freely migrate from one logical processor to another within the particular domain.

If the `I_MPI_PIN_DOMAIN` environment variable is defined, then the `I_MPI_PIN_PROCESSOR_LIST` environment variable setting is ignored.

If the `I_MPI_PIN_DOMAIN` environment variable is not defined, then MPI processes are pinned according to the current value of the `I_MPI_PIN_PROCESSOR_LIST` environment variable.

The `I_MPI_PIN_DOMAIN` environment variable has the following syntax forms:

- Domain description through multi-core terms *<mc-shape>*
- Domain description through domain size and domain member layout *<size>[:<layout>]*
- Explicit domain description through bit mask *<masklist>*

The following tables describe these syntax forms.

Multi-core Shape

`I_MPI_PIN_DOMAIN=`*<mc-shape>*

<mc-shape>	Define domains through multi-core terms.
`core`	Each domain consists of the logical processors that share a particular core. The number of domains on a node is equal to the number of cores on the node.
`socket | sock`	Each domain consists of the logical processors that share a particular socket. The number of domains on a node is equal to the number of sockets on the node. This is the recommended value.

`numa`	Each domain consists of the logical processors that share a particular NUMA node. The number of domains on a machine is equal to the number of NUMA nodes on the machine.
`node`	All logical processors on a node are arranged into a single domain.
`cache1`	Logical processors that share a particular level 1 cache are arranged into a single domain.
`cache2`	Logical processors that share a particular level 2 cache are arranged into a single domain.
`cache3`	Logical processors that share a particular level 3 cache are arranged into a single domain.
`cache`	The largest domain among `cache1`, `cache2`, and `cache3` is selected.

NOTE

If `Cluster on Die` is disabled on a machine, the number of NUMA nodes equals to the number of sockets. In this case, pinning for `I_MPI_PIN_DOMAIN = numa` is equivalent to pinning for `I_MPI_PIN_DOMAIN = socket`.

Explicit Shape

`I_MPI_PIN_DOMAIN=`*`<size>[:<layout>]`*

`<size>`	Define a number of logical processors in each domain (domain size)
`omp`	The domain size is equal to the `OMP_NUM_THREADS` environment variable value. If the `OMP_NUM_THREADS` environment variable is not set, each node is treated as a separate domain.
`auto`	The domain size is defined by the formula `size=#cpu/#proc`, where `#cpu` is the number of logical processors on a node, and `#proc` is the number of the MPI processes started on a node
`<n>`	The domain size is defined by a positive decimal number *`<n>`*

`<layout>`	Ordering of domain members. The default value is `compact`
`platform`	Domain members are ordered according to their BIOS numbering (platform-depended numbering)
`compact`	Domain members are located as close to each other as possible in terms of common resources (cores, caches, sockets, and so on). This is the default value
`scatter`	Domain members are located as far away from each other as possible in terms of common resources (cores, caches, sockets, and so on)

Explicit Domain Mask

`I_MPI_PIN_DOMAIN=`*`<masklist>`*

`<masklist>`	Define domains through the comma separated list of hexadecimal numbers (domain masks)

[m_1,...,m_n]	For <masklist>, each m_i is a hexadecimail bit mask defining an individual domain. The following rule is used: the i^{th} logical processor is included into the domain if the corresponding mi value is set to 1. All remaining processors are put into a separate domain. BIOS numbering is used. **NOTE** To ensure that your configuration in <masklist> is parsed correctly, use square brackets to enclose the domains specified by the <masklist>. For example: I_MPI_PIN_DOMAIN=[0x55,0xaa]

NOTE

These options are available for both Intel® and non-Intel microprocessors, but they may perform additional optimizations for Intel microprocessors than they perform for non-Intel microprocessors.

NOTE

To pin OpenMP* processes or threads inside the domain, the corresponding OpenMP feature (for example, the KMP_AFFINITY environment variable for Intel® compilers) should be used.

NOTE

The following configurations are effectively the same as if pinning is not applied:

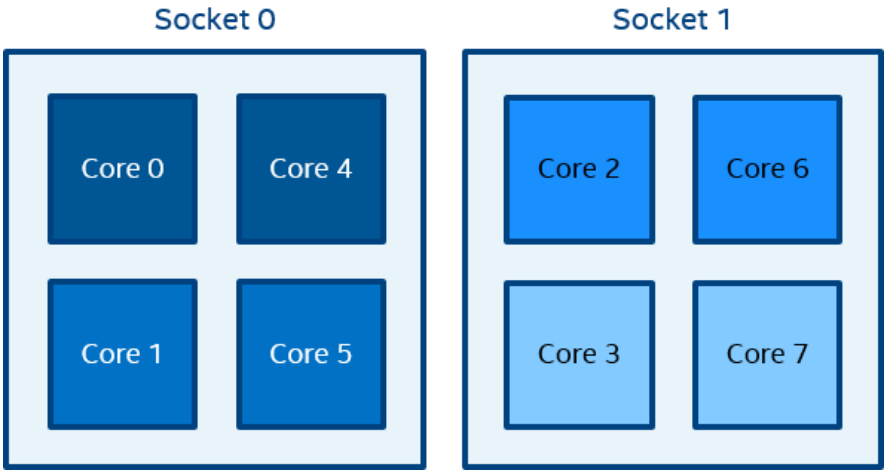
- If you set I_MPI_PIN_DOMAIN=auto and a single process is running on a node (for example, due to I_MPI_PERHOST=1)
- I_MPI_PIN_DOMAIN=node

If you do not want the process to be migrated between sockets on a multi-socket platform, specify the domain size as I_MPI_PIN_DOMAIN=socket or smaller.

You can also use I_MPI_PIN_PROCESSOR_LIST, which produces a single-cpu process affinity mask for each rank (the affinity mask is supposed to be automatically adjusted in presence of IBA* HCA).

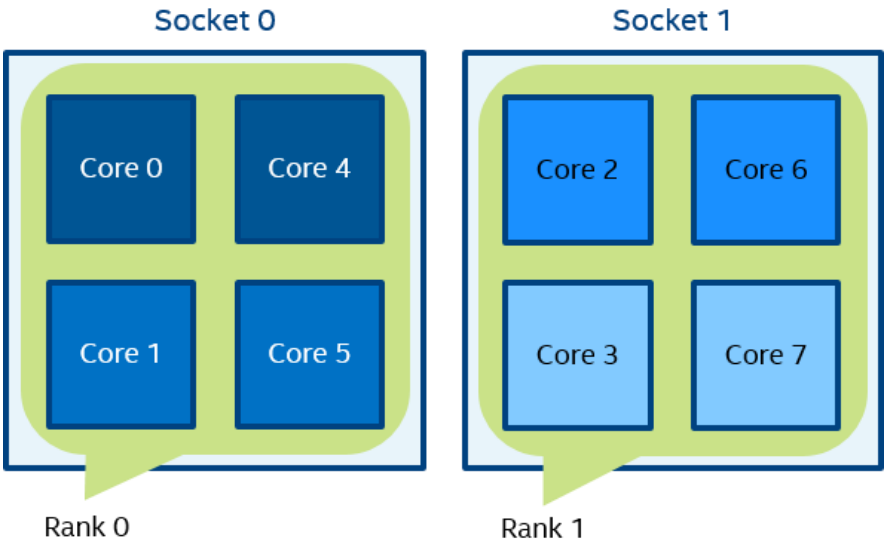
See the following model of a symmetric multiprocessing (SMP) node in the examples:

Figure 3.2-2 Model of a Node



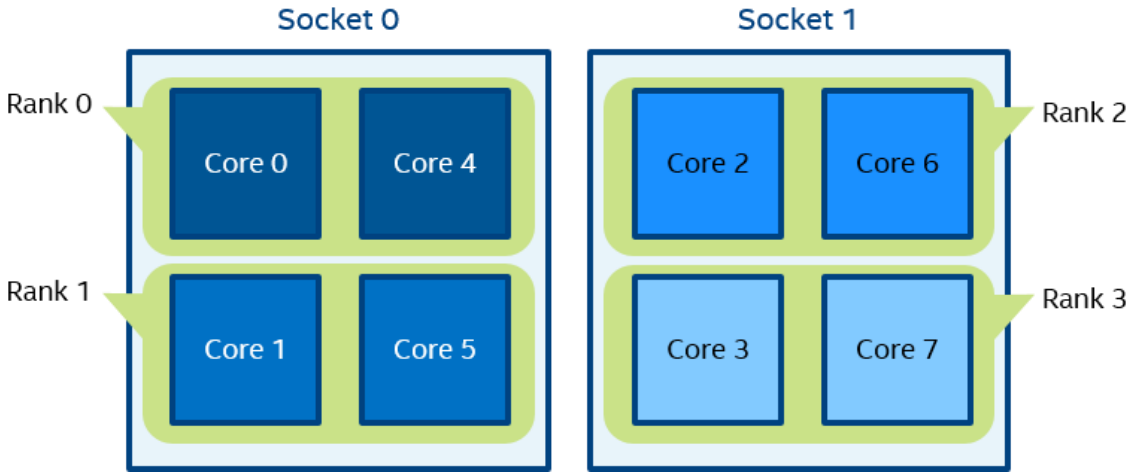
The figure above represents the SMP node model with a total of 8 cores on 2 sockets. Intel® Hyper-Threading Technology is disabled. Core pairs of the same color share the L2 cache.

Figure 3.2-3 `mpirun -n 2 -env I_MPI_PIN_DOMAIN socket ./a.out`



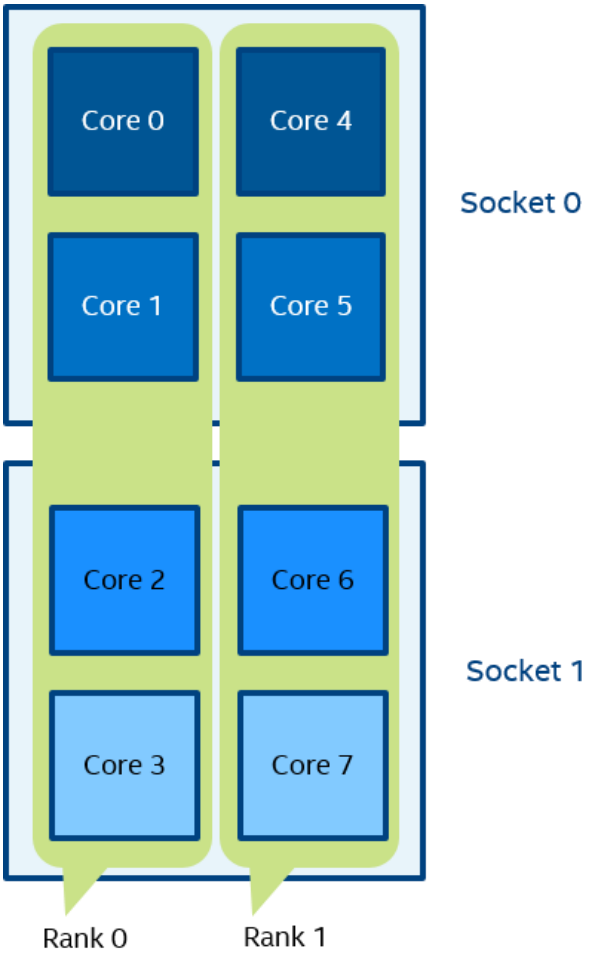
In Figure 3.2-3, two domains are defined according to the number of sockets. Process rank 0 can migrate on all cores on the 0-th socket. Process rank 1 can migrate on all cores on the first socket.

Figure 3.2-4 `mpirun -n 4 -env I_MPI_PIN_DOMAIN cache2 ./a.out`



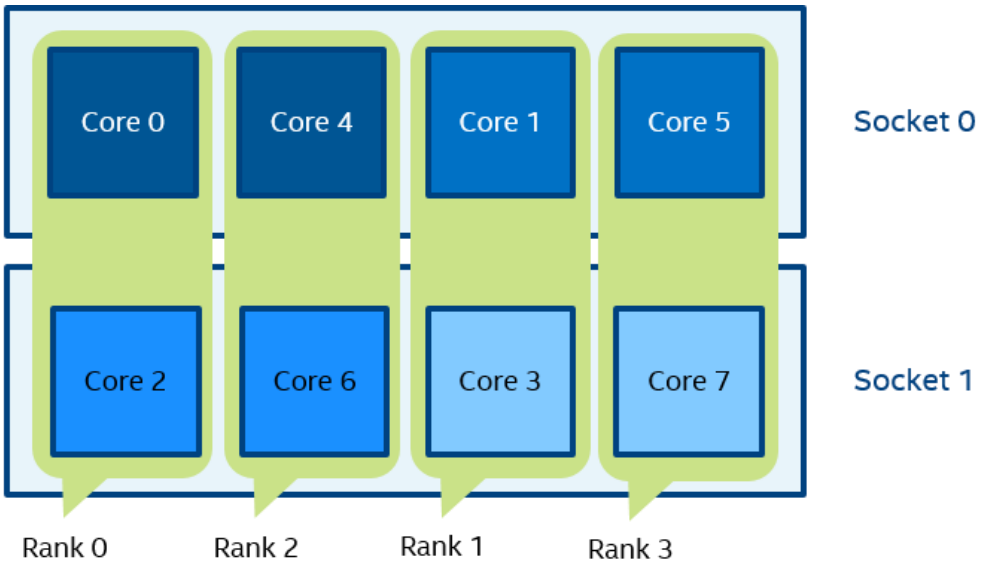
In Figure 3.2-4, four domains are defined according to the amount of common L2 caches. Process rank 0 runs on cores {0,4} that share an L2 cache. Process rank 1 runs on cores {1,5} that share an L2 cache as well, and so on.

Figure 3.2-5 `mpirun -n 2 -env I_MPI_PIN_DOMAIN 4:platform ./a.out`



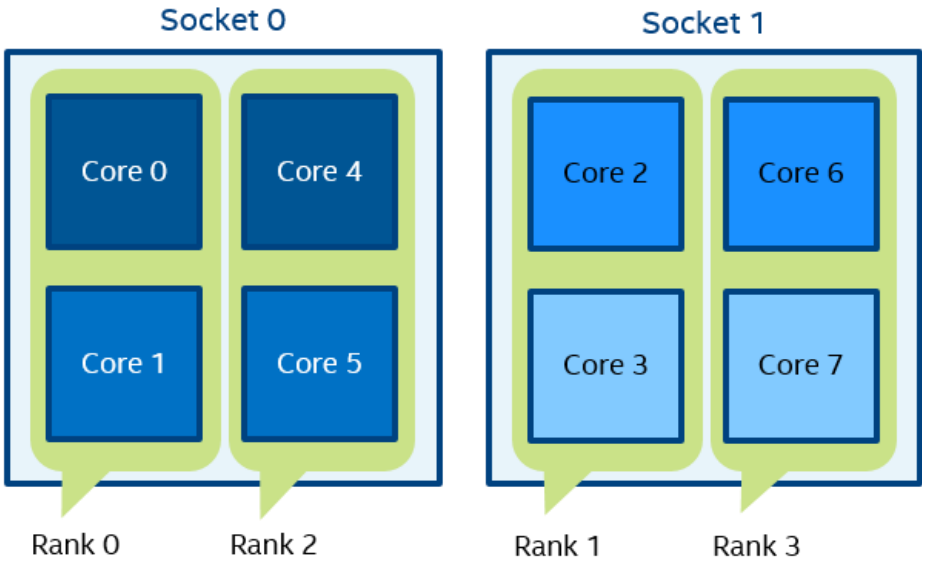
In Figure 3.2-5, two domains with size=4 are defined. The first domain contains cores {0,1,2,3}, and the second domain contains cores {4,5,6,7}. Domain members (cores) have consecutive numbering as defined by the `platform` option.

Figure 3.2-6 `mpirun -n 4 -env I_MPI_PIN_DOMAIN auto:scatter ./a.out`



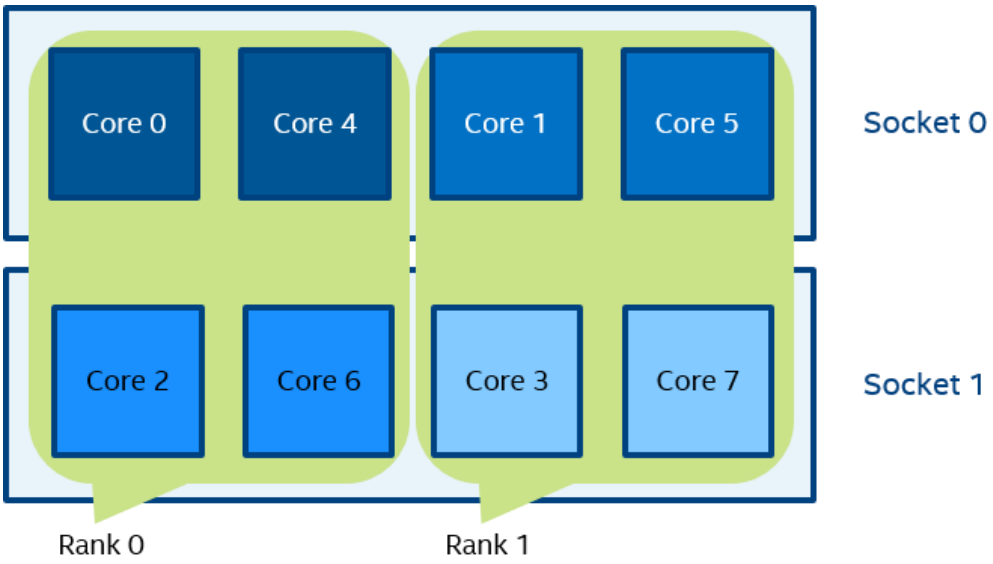
In Figure 3.2-6, domain size=2 (defined by the number of CPUs=8 / number of processes=4), `scatter` layout. Four domains {0,2}, {1,3}, {4,6}, {5,7} are defined. Domain members do not share any common resources.

Figure 3.2-7 `setenv OMP_NUM_THREADS=2`
`mpirun -n 4 -env I_MPI_PIN_DOMAIN omp:platform ./a.out`



In Figure 3.2-7, domain size=2 (defined by `OMP_NUM_THREADS=2`), `platform` layout. Four domains {0,1}, {2,3}, {4,5}, {6,7} are defined. Domain members (cores) have consecutive numbering.

Figure 3.2-8 `mpirun -n 2 -env I_MPI_PIN_DOMAIN [0x55,0xaa] ./a.out`



In Figure 3.2-8 (the example for `I_MPI_PIN_DOMAIN=<masklist>`), the first domain is defined by the 0x55 mask. It contains all cores with even numbers {0,2,4,6}. The second domain is defined by the 0xAA mask. It contains all cores with odd numbers {1,3,5,7}.

I_MPI_PIN_ORDER

Set this environment variable to define the mapping order for MPI processes to domains as specified by the `I_MPI_PIN_DOMAIN` environment variable.

Syntax

`I_MPI_PIN_ORDER=<order>`

Arguments

<order>	Specify the ranking order
`range`	The domains are ordered according to the processor's BIOS numbering. This is a platform-dependent numbering
`scatter`	The domains are ordered so that adjacent domains have minimal sharing of common resources
`compact`	The domains are ordered so that adjacent domains share common resources as much as possible. This is the default value
`spread`	The domains are ordered consecutively with the possibility not to share common resources
`bunch`	The processes are mapped proportionally to sockets and the domains are ordered as close as possible on the sockets

Description

The optimal setting for this environment variable is application-specific. If adjacent MPI processes prefer to share common resources, such as cores, caches, sockets, FSB, use the `compact` or `bunch` values. Otherwise,

use the `scatter` or `spread` values. Use the `range` value as needed. For detail information and examples about these values, see the Arguments table and the Example section of `I_MPI_PIN_ORDER` in this topic.

The options `scatter`, `compact`, `spread` and `bunch` are available for both Intel® and non-Intel microprocessors, but they may perform additional optimizations for Intel microprocessors than they perform for non-Intel microprocessors.

Examples

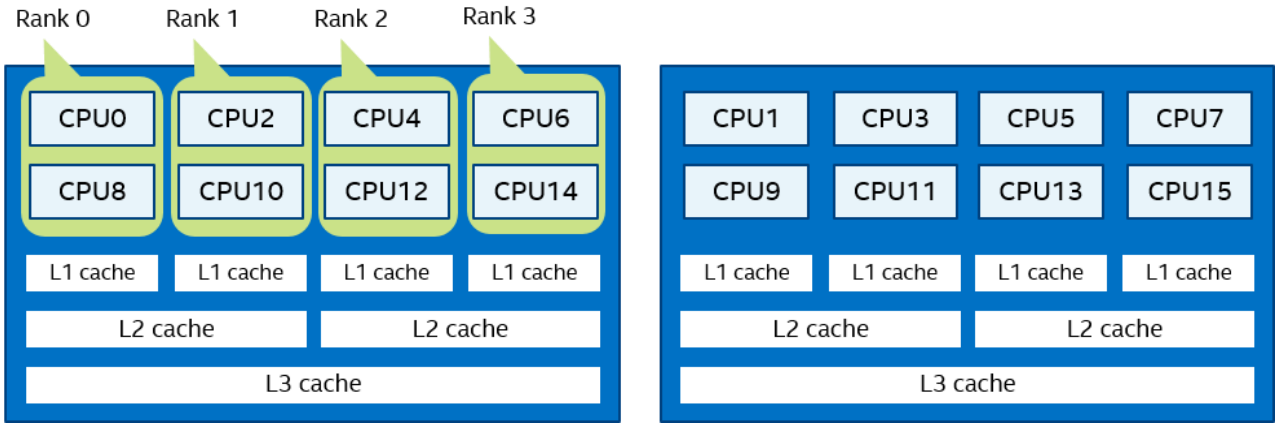
For the following configuration:

- Two socket nodes with four cores and a shared L2 cache for corresponding core pairs.
- 4 MPI processes you want to run on the node using the settings below.

Compact order:

```
I_MPI_PIN_DOMAIN=2
I_MPI_PIN_ORDER=compact
```

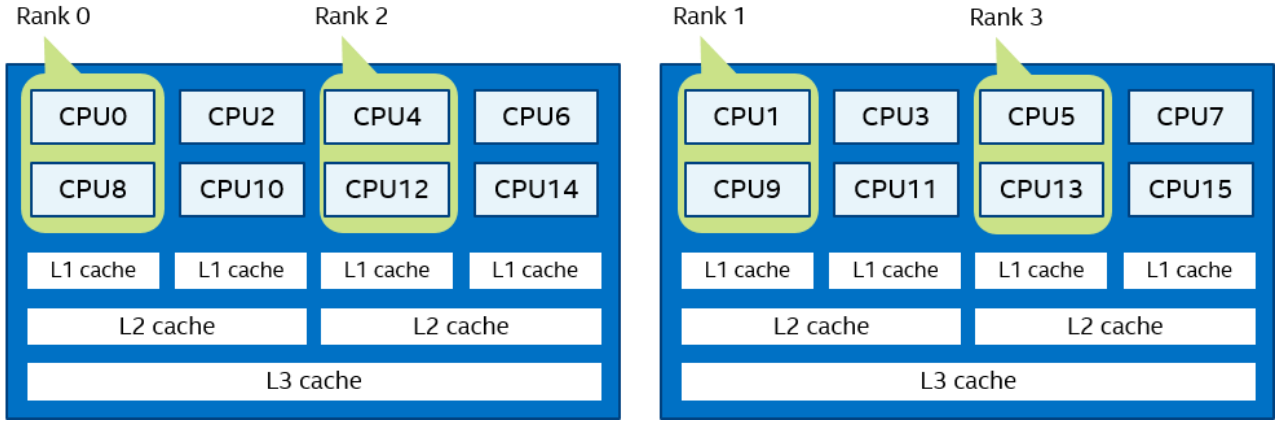
Figure 3.2-9 Compact Order Example



Scatter order:

```
I_MPI_PIN_DOMAIN=2
I_MPI_PIN_ORDER=scatter
```

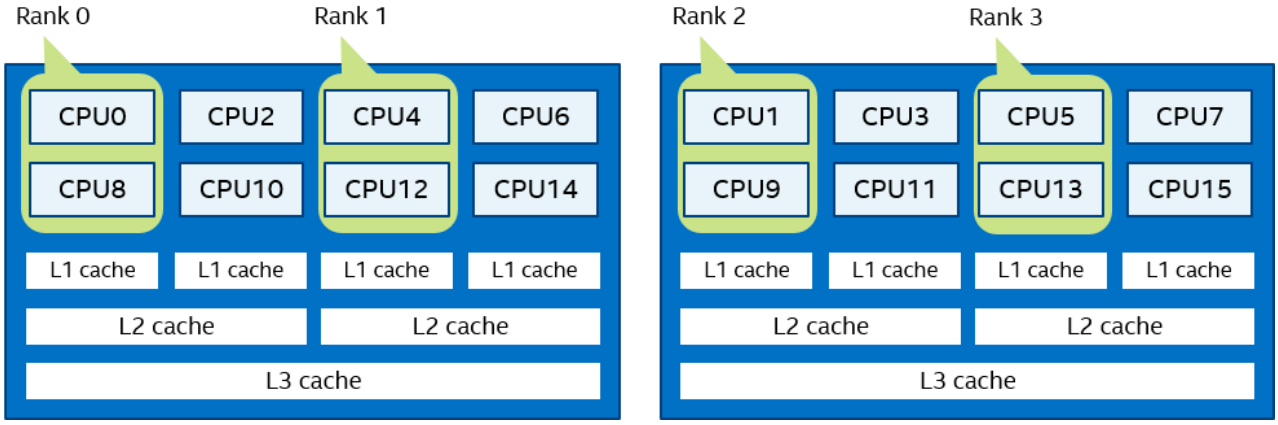
Figure 3.2-10 Scatter Order Example



Spread order:

```
I_MPI_PIN_DOMAIN=2
I_MPI_PIN_ORDER=spread
```

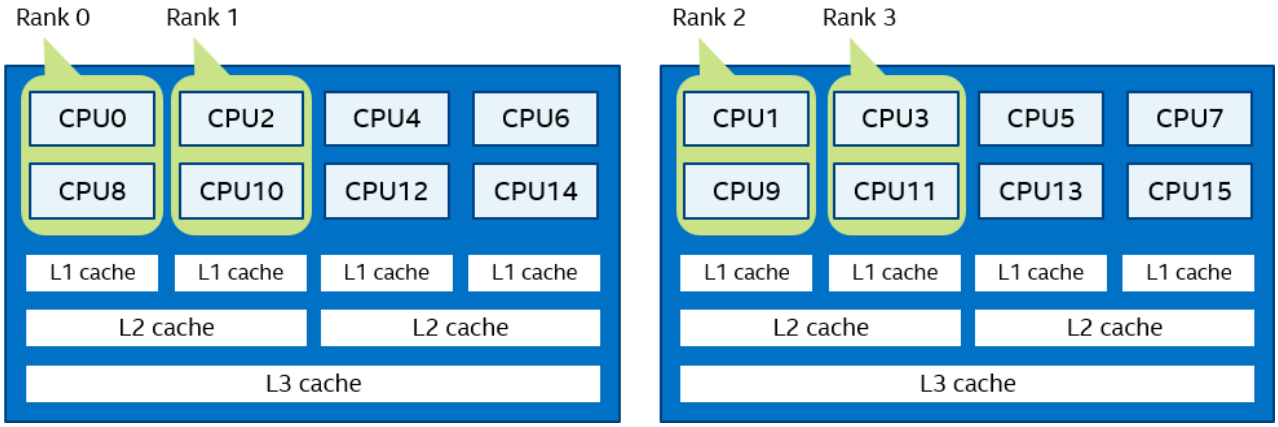
Figure 3.2-11 Spread Order Example



Bunch order:

```
I_MPI_PIN_DOMAIN=2
I_MPI_PIN_ORDER=bunch
```

Figure 3.2-12 Bunch Order Example



3.3. Memory Placement Policy Control

Intel® MPI Library supports non-uniform memory access (NUMA) nodes with high-bandwidth (HBW) memory (MCDRAM) on Intel® Xeon Phi™ processors (codenamed Knights Landing). Intel® MPI Library can attach memory of MPI processes to the memory of specific NUMA nodes. This section describes the environment variables for such memory placement control.

I_MPI_HBW_POLICY

Set the policy for MPI process memory placement for using HBW memory.

Syntax

```
I_MPI_HBW_POLICY=<user memory policy>[,<mpi memory policy>][,<win_allocate policy>]
```

In the syntax:

- `<user memory policy>` - memory policy used to allocate the memory for user applications (required)
- `<mpi memory policy>` - memory policy used to allocate the internal MPI memory (optional)

- `<win_allocate policy>` - memory policy used to allocate memory for window segments for RMA operations (optional)

Each of the listed policies may have the values below:

Arguments

<value>	The memory allocation policy used.
`hbw_preferred`	Allocate the local HBW memory for each process. If the HBW memory is not available, allocate the local dynamic random access memory.
`hbw_bind`	Allocate only the local HBW memory for each process.
`hbw_interleave`	Allocate the HBW memory and dynamic random access memory on the local node in the round-robin manner.

Description

Use this environment variable to specify the policy for MPI process memory placement on a machine with HBW memory.

By default, Intel MPI Library allocates memory for a process in local DDR. The use of HBW memory becomes available only when you specify the `I_MPI_HBW_POLICY` variable.

Examples

The following examples demonstrate different configurations of memory placement:

- `I_MPI_HBW_POLICY=hbw_bind,hbw_preferred,hbw_bind`

  Only use the local HBW memory allocated in user applications and window segments for RMA operations. Use the local HBW memory internally allocated in Intel® MPI Library first. If the HBW memory is not available, use the local DDR internally allocated in Intel MPI Library.
- `I_MPI_HBW_POLICY=hbw_bind,,hbw_bind`

  Only use the local HBW memory allocated in user applications and window segments for RMA operations. Use the local DDR internally allocated in Intel MPI Library.
- `I_MPI_HBW_POLICY=hbw_bind,hbw_preferred`

  Only use the local HBW memory allocated in user applications. Use the local HBW memory internally allocated in Intel MPI Library first. If the HBW memory is not available, use the local DDR internally allocated in Intel MPI Library. Use the local DDR allocated in window segments for RMA operations.

I_MPI_BIND_NUMA

Set the NUMA nodes for memory allocation.

Syntax

`I_MPI_BIND_NUMA=`*<value>*

Arguments

<value>	Specify the NUMA nodes for memory allocation.
`localalloc`	Allocate memory on the local node. This is the default value.

`Node_1,…,Node_k`	Allocate memory according to `I_MPI_BIND_ORDER` on the specified NUMA nodes.

Description

Set this environment variable to specify the NUMA node set that is involved in the memory allocation procedure.

I_MPI_BIND_ORDER

Set this environment variable to define the memory allocation manner.

Syntax

`I_MPI_BIND_ORDER=<value>`

Arguments

`<value>`	Specify the allocation manner.
`compact`	Allocate memory for processes as close as possible (in terms of NUMA nodes), among the NUMA nodes specified in `I_MPI_BIND_NUMA`. This is the default value.
`scatter`	Allocate memory among the NUMA nodes specified in `I_MPI_BIND_NUMA` using the round-robin manner.

Description

Set this environment variable to define the memory allocation manner among the NUMA nodes specified in `I_MPI_BIND_NUMA`. The variable has no effect without `I_MPI_BIND_NUMA` set.

I_MPI_BIND_WIN_ALLOCATE

Set this environment variable to control memory allocation for window segments.

Syntax

`I_MPI_BIND_WIN_ALLOCATE=<value>`

Arguments

`<value>`	Specify the memory allocation behavior for window segments.
`localalloc`	Allocate memory on the local node. This is the default value.
`hbw_preferred`	Allocate the local HBW memory for each process. If the HBW memory is not available, allocate the local dynamic random access memory.
`hbw_bind`	Allocate only the local HBW memory for each process.
`hbw_interleave`	Allocate the HBW memory and dynamic random access memory on a local node in the round-robin manner.
`<NUMA node id>`	Allocate memory on the given NUMA node.

Description

Set this environment variable to create window segments allocated in HBW memory with the help of the `MPI_Win_allocate_shared` or `MPI_Win_allocate` functions.

MPI_Info

You can control memory allocation for window segments with the help of an `MPI_Info` object, which is passed as a parameter to the `MPI_Win_allocate` or `MPI_Win_allocate_shared` function. In an application, if you specify such an object with the `numa_bind_policy` key, window segments are allocated in accordance with the value for `numa_bind_policy`. Possible values are the same as for `I_MPI_BIND_WIN_ALLOCATE`.

A code fragment demonstrating the use of `MPI_Info`:

```
MPI_Info info;
...
MPI_Info_create( &info );
MPI_Info_set( info, "numa_bind_policy", "hbw_preferred" );
...
MPI_Win_allocate_shared( size, disp_unit, info, comm, &baseptr, &win );
```

NOTE

When you specify the memory placement policy for window segments, Intel MPI Library recognizes the configurations according to the following priority:

1. Setting of `MPI_Info`.
2. Setting of `I_MPI_HBW_POLICY`, if you specified `<win_allocate policy>`.
3. Setting of `I_MPI_BIND_WIN_ALLOCATE`.

I_MPI_MEMORY_SWAP_LOCK

Set this environment variable to disable memory swapping to the hard drive.

Syntax

`I_MPI_MEMORY_SWAP_LOCK=<value>`

Arguments

<arg>	Binary indicator
`enable | on | 1`	Prevent memory swapping to the hard drive.
`disable | off | 0`	Allow memory swapping to the hard drive. This is the default value.

Description

Set this environment variable to prevent process memory from being swapped to the hard drive, which is done by default.

3.4. Fabrics Control

3.4.1. Communication Fabrics Control

I_MPI_FABRICS

Select the particular network fabrics to be used.

Syntax

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

where `<fabric> := {shm, dapl, tcp, tmi, ofa, ofi}`

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

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

Arguments

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

Description

Set this environment variable to select a specific fabric combination. If the requested fabric(s) is not available, Intel® MPI Library can fall back to other fabric(s). See `I_MPI_FALLBACK` for details. If the `I_MPI_FABRICS` environment variable is not defined, Intel® MPI Library selects the most appropriate fabric combination automatically.

The exact combination of fabrics depends on the number of processes started per node.

- If all processes start on one node, the library uses `shm` for intra-node communication.
- If the number of started processes is less than or equal to the number of available nodes, the library uses the first available fabric from the fabrics list for inter-node communication.
- For other cases, the library uses `shm` for intra-node communication, and the first available fabric from the fabrics list for inter-node communication. See `I_MPI_FABRICS_LIST` for details.

The `shm` fabric is available for both Intel® and non-Intel microprocessors, but it may perform additional optimizations for Intel microprocessors than it performs for non-Intel microprocessors.

NOTE

The combination of selected fabrics ensures that the job runs, but this combination may not provide the highest possible performance for the given cluster configuration.

For example, to select shared memory and DAPL-capable network fabric as the chosen fabric combination, use the following command:

```
$ mpirun -n <# of processes> -genv I_MPI_FABRICS=shm:dapl <executable>
```

To enable Intel® MPI Library to select most appropriate fabric combination automatically, run the application as usual, without setting the `I_MPI_FABRICS` variable:

```
$ mpirun -n <# of processes> <executable>
```

Set the level of debug information to `2` or higher to check which fabrics have been initialized. See `I_MPI_DEBUG` for details. For example:

```
[0] MPI startup(): shm and dapl data transfer modes
```

I_MPI_FABRICS_LIST

Define a fabric list.

Syntax

`I_MPI_FABRICS_LIST=<fabrics list>`

where `<fabrics list> := <fabric>,...,<fabric>`

`<fabric> := {dapl, tcp, tmi, ofa, ofi}`

Arguments

<fabrics list>	Specify a list of fabrics
`dapl,tmi,ofa,tcp,ofi`	This is the default value.
`dapl,tcp,ofa,tmi,ofi`	If you specify `I_MPI_WAIT_MODE=enable`, this is the default value.
`ofi,tmi,dapl,tcp,ofa`	This is the default value for nodes that have Intel® Omni-Path Fabric or Intel® True Scale Fabric available and do not have any other type of interconnect cards. In case host has several types of host channel adapters (HCAs), this does not apply.

Description

Use this environment variable to define a list of inter-node fabrics. Intel® MPI Library uses the fabric list to choose the most appropriate fabrics combination automatically. For more information on fabric combination, see `I_MPI_FABRICS`.

For example, if `I_MPI_FABRICS_LIST=dapl,tcp`, and `I_MPI_FABRICS` is not defined, and the initialization of a DAPL-capable network fabrics fails, Intel® MPI Library falls back to the TCP-capable network fabric. For more information on fallback, see `I_MPI_FALLBACK`.

I_MPI_FALLBACK

Set this environment variable to enable fallback to the first available fabric.

Syntax

`I_MPI_FALLBACK=<arg>`

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Fall back to the first available fabric. This is the default value unless you set the I_MPI_FABRICS environment variable.
disable | no | off | 0	Terminate the job if MPI cannot initialize the currently set fabric. This is the default value if you set the I_MPI_FABRICS environment variable.

Description

Set this environment variable to control fallback to the first available fabric.

If you set I_MPI_FALLBACK to enable and an attempt to initialize a specified fabric fails, the library uses the first available fabric from the list of fabrics. See I_MPI_FABRICS_LIST for details.

If you set I_MPI_FALLBACK to disable and an attempt to initialize a specified fabric fails, the library terminates the MPI job.

NOTE

If you set I_MPI_FABRICS and I_MPI_FALLBACK=enable, the library falls back to the next fabric in the fabrics list. For example, if I_MPI_FABRICS=dapl, I_MPI_FABRICS_LIST=ofa,tmi,dapl,tcp, I_MPI_FALLBACK=enable and the initialization of DAPL-capable network fabrics fails, the library falls back to TCP-capable network fabric.

I_MPI_LARGE_SCALE_THRESHOLD

Change the threshold for enabling scalable optimizations.

Syntax

I_MPI_LARGE_SCALE_THRESHOLD=*<arg>*

Arguments

<nprocs>	Define the scale threshold
> 0	The default value is 4096

Description

This variable defines the number of processes when the DAPL UD IB extension is turned on automatically.

I_MPI_EAGER_THRESHOLD

Change the eager/rendezvous message size threshold for all devices.

Syntax

I_MPI_EAGER_THRESHOLD=*<nbytes>*

Arguments

<nbytes>	Set the eager/rendezvous message size threshold
> 0	The default *<nbytes>* value is equal to 262144 bytes

Description

Set this environment variable to control the protocol used for point-to-point communication:

- Messages shorter than or equal in size to *<nbytes>* are sent using the eager protocol.
- Messages larger than *<nbytes>* are sent using the rendezvous protocol. The rendezvous protocol uses memory more efficiently.

I_MPI_INTRANODE_EAGER_THRESHOLD

Change the eager/rendezvous message size threshold for intra-node communication mode.

Syntax

`I_MPI_INTRANODE_EAGER_THRESHOLD=<nbytes>`

Arguments

<nbytes>	Set the eager/rendezvous message size threshold for intra-node communication
> 0	The default *<nbytes>* value is equal to `262144` bytes for all fabrics except `shm`. For `shm`, cutover point is equal to the value of `I_MPI_SHM_CELL_SIZE` environment variable

Description

Set this environment variable to change the protocol used for communication within the node:

- Messages shorter than or equal in size to *<nbytes>* are sent using the eager protocol.
- Messages larger than *<nbytes>* are sent using the rendezvous protocol. The rendezvous protocol uses the memory more efficiently.

If you do not set `I_MPI_INTRANODE_EAGER_THRESHOLD`, the value of `I_MPI_EAGER_THRESHOLD` is used.

I_MPI_SPIN_COUNT

Control the spin count value.

Syntax

`I_MPI_SPIN_COUNT=<scount>`

Arguments

<scount>	Define the loop spin count when polling fabric(s)
> 0	The default *<scount>* value is equal to `1` when more than one process runs per processor/core. Otherwise the value equals `250`. The maximum value is equal to `2147483647`

Description

Set the spin count limit. The loop for polling the fabric(s) spins *<scount>* times before the library releases the processes if no incoming messages are received for processing. Within every spin loop, the `shm` fabric (if enabled) is polled an extra `I_MPI_SHM_SPIN_COUNT` times. Smaller values for *<scount>* cause the Intel® MPI Library to release the processor more frequently.

Use the `I_MPI_SPIN_COUNT` environment variable for tuning application performance. The best value for *<scount>* can be chosen on an experimental basis. It depends on the particular computational environment and the application.

I_MPI_SCALABLE_OPTIMIZATION

Turn on/off scalable optimization of the network fabric communication.

Syntax

`I_MPI_SCALABLE_OPTIMIZATION=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on scalable optimization of the network fabric communication. This is the default for 16 or more processes
`disable | no | off | 0`	Turn off scalable optimization of the network fabric communication. This is the default value for less than 16 processes

Description

Set this environment variable to enable scalable optimization of the network fabric communication. In most cases, using optimization decreases latency and increases bandwidth for a large number of processes.

I_MPI_WAIT_MODE

Turn on/off wait mode.

Syntax

`I_MPI_WAIT_MODE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the wait mode
`disable | no | off | 0`	Turn off the wait mode. This is the default

Description

Set this environment variable to control the wait mode. If you enable this mode, the processes wait for receiving messages without polling the fabric(s). This mode can save CPU time for other tasks.

Use the Native POSIX Thread Library* with the wait mode for `shm` communications.

NOTE

To check which version of the thread library is installed, use the following command:

```
$ getconf GNU_LIBPTHREAD_VERSION
```

I_MPI_DYNAMIC_CONNECTION (I_MPI_USE_DYNAMIC_CONNECTIONS)

Control the dynamic connection establishment.

Syntax

`I_MPI_DYNAMIC_CONNECTION=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the dynamic connection establishment. This is the default for 64 or more processes
`disable | no | off | 0`	Turn off the dynamic connection establishment. This is the default for less than 64 processes

Description

Set this environment variable to control dynamic connection establishment.

- If this mode is enabled, all connections are established at the time of the first communication between each pair of processes.
- If this mode is disabled, all connections are established upfront.

The default value depends on the number of processes in the MPI job. The dynamic connection establishment is off if the total number of processes is less than 64.

3.4.2. Shared Memory Control

I_MPI_SHM_CACHE_BYPASS

Control the message transfer algorithm for the shared memory.

Syntax

`I_MPI_SHM_CACHE_BYPASS=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable message transfer bypass cache. This is the default value
`disable| no | off | 0`	Disable message transfer bypass cache

Description

Set this environment variable to enable/disable message transfer bypass cache for the shared memory. When you enable this feature, the MPI sends the messages greater than or equal in size to the value specified by the `I_MPI_SHM_CACHE_BYPASS_THRESHOLD` environment variable through the bypass cache. This feature is enabled by default.

I_MPI_SHM_CACHE_BYPASS_THRESHOLDS

Set the message copying algorithm threshold.

Syntax

`I_MPI_SHM_CACHE_BYPASS_THRESHOLDS=<nb_send>,<nb_recv>[,<nb_send_pk>,<nb_recv_pk>]`

Arguments

<nb_send>	Set the threshold for sent messages in the following situations: • Processes are pinned on cores that are not located in the same physical processor package • Processes are not pinned
<nb_recv>	Set the threshold for received messages in the following situations: • Processes are pinned on cores that are not located in the same physical processor package • Processes are not pinned
<nb_send_pk>	Set the threshold for sent messages when processes are pinned on cores located in the same physical processor package
<nb_recv_pk>	Set the threshold for received messages when processes are pinned on cores located in the same physical processor package

Description

Set this environment variable to control the thresholds for the message copying algorithm. Intel® MPI Library uses different message copying implementations which are optimized to operate with different memory hierarchy levels. Intel® MPI Library copies messages greater than or equal in size to the defined threshold value using copying algorithm optimized for far memory access. The value of `-1` disables using of those algorithms. The default values depend on the architecture and may vary among the Intel® MPI Library versions. This environment variable is valid only when `I_MPI_SHM_CACHE_BYPASS` is enabled.

This environment variable is available for both Intel and non-Intel microprocessors, but it may perform additional optimizations for Intel microprocessors than it performs for non-Intel microprocessors.

I_MPI_SHM_FBOX

Control the usage of the shared memory fast-boxes.

Syntax

`I_MPI_SHM_FBOX=`*<arg>*

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on fast box usage. This is the default value.
`disable | no | off | 0`	Turn off fast box usage.

Description

Set this environment variable to control the usage of fast-boxes. Each pair of MPI processes on the same computing node has two shared memory fast-boxes, for sending and receiving eager messages.

Turn off the usage of fast-boxes to avoid the overhead of message synchronization when the application uses mass transfer of short non-blocking messages.

I_MPI_SHM_FBOX_SIZE

Set the size of the shared memory fast-boxes.

Syntax

`I_MPI_SHM_FBOX_SIZE=<nbytes>`

Arguments

`<nbytes>`	The size of shared memory fast-boxes in bytes
`> 0`	The default `<nbytes>` value depends on the specific platform you use. The value range is from 8K to 64K typically.

Description

Set this environment variable to define the size of shared memory fast-boxes.

I_MPI_SHM_CELL_NUM

Change the number of cells in the shared memory receiving queue.

Syntax

`I_MPI_SHM_CELL_NUM=<num>`

Arguments

`<num>`	The number of shared memory cells
`> 0`	The default value is `128`

Description

Set this environment variable to define the number of cells in the shared memory receive queue. Each MPI process has own shared memory receive queue, where other processes put eager messages. The queue is used when shared memory fast-boxes are blocked by another MPI request.

I_MPI_SHM_CELL_SIZE

Change the size of a shared memory cell.

Syntax

`I_MPI_SHM_CELL_SIZE=<nbytes>`

Arguments

`<nbytes>`	The size of a shared memory cell in bytes
`> 0`	The default `<nbytes>` value depends on the specific platform you use. The value range is from 8K to 64K typically.

Description

Set this environment variable to define the size of shared memory cells.

If you set this environment variable, `I_MPI_INTRANODE_EAGER_THRESHOLD` is also changed and becomes equal to the given value.

I_MPI_SHM_LMT

Control the usage of large message transfer (LMT) mechanism for the shared memory.

Syntax

`I_MPI_SHM_LMT=<arg>`

Arguments

<arg>	Binary indicator
`shm`	Turn on the shared memory copy LMT mechanism.
`direct`	Turn on the direct copy LMT mechanism. This is the default value
`disable | no | off | 0`	Turn off LMT mechanism

Description

Set this environment variable to control the usage of the large message transfer (LMT) mechanism. To transfer rendezvous messages, you can use the LMT mechanism by employing either of the following implementations:

- Use intermediate shared memory queues to send messages.
- Use direct copy mechanism that transfers messages without intermediate buffer if the Linux* kernel is higher than version 3.2 which supports the cross memory attach (CMA) feature. If you set the `I_MPI_SHM_LMT` environment variable to `direct`, but the operating system does not support CMA, then the `shm` LTM mechanism runs.

I_MPI_SHM_LMT_BUFFER_NUM

Change the number of shared memory buffers for the large message transfer (LMT) mechanism.

Syntax

`I_MPI_SHM_LMT_BUFFER_NUM=<num>`

Arguments

<num>	The number of shared memory buffers for each process pair
`> 0`	The default value is `8`

Description

Set this environment variable to define the number of shared memory buffers between each process pair.

I_MPI_SHM_LMT_BUFFER_SIZE

Change the size of shared memory buffers for the LMT mechanism.

Syntax

`I_MPI_SHM_LMT_BUFFER_SIZE=<nbytes>`

Arguments

<nbytes>	The size of shared memory buffers in bytes
> 0	The default *<nbytes>* value is equal to 32768 bytes

Description

Set this environment variable to define the size of shared memory buffers for each pair of processes.

I_MPI_SHM_BYPASS

Turn on/off the intra-node communication mode through network fabric along with shm.

Syntax

I_MPI_SHM_BYPASS=*<arg>*

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Turn on the intra-node communication through network fabric
disable | no | off | 0	Turn off the intra-node communication through network fabric. This is the default

Description

Set this environment variable to specify the communication mode within the node. If the intra-node communication mode through network fabric is enabled, data transfer algorithms are selected according to the following scheme:

- Messages shorter than or equal in size to the threshold value of the I_MPI_INTRANODE_EAGER_THRESHOLD environment variable are transferred using shared memory.
- Messages larger than the threshold value of the I_MPI_INTRANODE_EAGER_THRESHOLD environment variable are transferred through the network fabric layer.

NOTE

This environment variable is applicable only when you turn on shared memory and a network fabric either by default or by setting the I_MPI_FABRICS environment variable to shm:*<fabric>*. This mode is available only for dapl and tcp fabrics.

I_MPI_SHM_SPIN_COUNT

Control the spin count value for the shared memory fabric.

Syntax

I_MPI_SHM_SPIN_COUNT=*<shm_scount>*

Arguments

<scount>	Define the spin count of the loop when polling the shm fabric
> 0	When internode communication uses the tcp fabric, the default *<shm_scount>* value is equal to 100 spins

	When internode communication uses the `ofa, tmi,ofi` or `dapl` fabric, the default *<shm_scount>* value is equal to `10` spins. The maximum value is equal to `2147483647`

Description

Set the spin count limit of the shared memory fabric to increase the frequency of polling. This configuration allows polling of the `shm` fabric *<shm_scount>* times before the control is passed to the overall network fabric polling mechanism.

To tune application performance, use the `I_MPI_SHM_SPIN_COUNT` environment variable. The best value for *<shm_scount>* can be chosen on an experimental basis. It depends largely on the application and the particular computation environment. An increase in the *<shm_scount>* value benefits multi-core platforms when the application uses topological algorithms for message passing.

3.4.3. DAPL-capable Network Fabrics Control

I_MPI_DAPL_PROVIDER

Define the DAPL provider to load.

Syntax

`I_MPI_DAPL_PROVIDER=`*<name>*

Arguments

<name>	Define the name of DAPL provider to load

Description

Set this environment variable to define the name of DAPL provider to load. This name is also defined in the `dat.conf` configuration file.

I_MPI_DAT_LIBRARY

Select the DAT library to be used for DAPL* provider.

Syntax

`I_MPI_DAT_LIBRARY=`*<library>*

Arguments

<library>	Specify the DAT library for DAPL provider to be used. Default values are `libdat.so` or `libdat.so.1` for DAPL* 1.2 providers and `libdat2.so or libdat2.so.2` for DAPL* 2.0 providers

Description

Set this environment variable to select a specific DAT library to be used for DAPL provider. If the library is not located in the dynamic loader search path, specify the full path to the DAT library. This environment variable affects only on DAPL and DAPL UD capable fabrics.

I_MPI_DAPL_TRANSLATION_CACHE

Turn on/off the memory registration cache in the DAPL path.

Syntax

`I_MPI_DAPL_TRANSLATION_CACHE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the memory registration cache
`disable | no | off | 0`	Turn off the memory registration cache. This is the default value

Description

Set this environment variable to turn on/off the memory registration cache in the DAPL path.

The cache substantially increases performance, but may lead to correctness issues in certain situations. See product *Release Notes* for further details.

I_MPI_DAPL_TRANSLATION_CACHE_AVL_TREE

Enable/disable the AVL tree* based implementation of the RDMA translation cache in the DAPL path.

Syntax

`I_MPI_DAPL_TRANSLATION_CACHE_AVL_TREE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the AVL tree based RDMA translation cache
`disable | no | off | 0`	Turn off the AVL tree based RDMA translation cache. This is the default value

Description

Set this environment variable to enable the AVL tree based implementation of RDMA translation cache in the DAPL path. When the search in RDMA translation cache handles over 10,000 elements, the AVL tree based RDMA translation cache is faster than the default implementation.

I_MPI_DAPL_DIRECT_COPY_THRESHOLD

Change the threshold of the DAPL direct-copy protocol.

Syntax

`I_MPI_DAPL_DIRECT_COPY_THRESHOLD=<nbytes>`

Arguments

<nbytes>	Define the DAPL direct-copy protocol threshold
`> 0`	The default *<nbytes>* value depends on the platform

Description

Set this environment variable to control the DAPL direct-copy protocol threshold. Data transfer algorithms for the DAPL-capable network fabrics are selected based on the following scheme:

- Messages shorter than or equal to <nbytes> are sent using the eager protocol through the internal pre-registered buffers. This approach is faster for short messages.
- Messages larger than <nbytes> are sent using the direct-copy protocol. It does not use any buffering but involves registration of memory on sender and receiver sides. This approach is faster for large messages.

This environment variable is available for both Intel® and non-Intel microprocessors, but it may perform additional optimizations for Intel microprocessors than it performs for non-Intel microprocessors.

NOTE

The equivalent of this variable for Intel® Xeon Phi™ Coprocessor is I_MIC_MPI_DAPL_DIRECT_COPY_THRESHOLD

I_MPI_DAPL_EAGER_MESSAGE_AGGREGATION

Control the use of concatenation for adjourned MPI send requests. Adjourned MPI send requests are those that cannot be sent immediately.

Syntax

I_MPI_DAPL_EAGER_MESSAGE_AGGREGATION=<arg>

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Enable the concatenation for adjourned MPI send requests
disable | no | off | 0	Disable the concatenation for adjourned MPI send requests. This is the default value

Set this environment variable to control the use of concatenation for adjourned MPI send requests intended for the same MPI rank. In some cases, this mode can improve the performance of applications, especially when MPI_Isend() is used with short message sizes and the same destination rank, such as:

```
for( i = 0; i < NMSG; i++)
{
    ret = MPI_Isend( sbuf[i], MSG_SIZE, datatype, dest, tag, comm, &req_send[i]);
}
```

I_MPI_DAPL_DYNAMIC_CONNECTION_MODE

Choose the algorithm for establishing the DAPL* connections.

Syntax

I_MPI_DAPL_DYNAMIC_CONNECTION_MODE=<arg>

Arguments

<arg>	Mode selector
reject	Deny one of the two simultaneous connection requests. This is the default
disconnect	Deny one of the two simultaneous connection requests after both connections have been

	established

Description

Set this environment variable to choose the algorithm for handling dynamically established connections for DAPL-capable fabrics according to the following scheme:

- In the `reject` mode, if two processes initiate the connection simultaneously, one of the requests is rejected.
- In the `disconnect` mode, both connections are established, but then one is disconnected. The `disconnect` mode is provided to avoid a bug in certain DAPL* providers.

I_MPI_DAPL_SCALABLE_PROGRESS

Turn on/off scalable algorithm for DAPL read progress.

Syntax

`I_MPI_DAPL_SCALABLE_PROGRESS=`*`<arg>`*

Arguments

`<arg>`	Binary indicator
`enable | yes | on | 1`	Turn on scalable algorithm. When the number of processes is larger than `128`, this is the default value
`disable | no | off | 0`	Turn off scalable algorithm. When the number of processes is less than or equal to `128`, this is the default value

Description

Set this environment variable to enable scalable algorithm for the DAPL read progress. In some cases, this provides advantages for systems with many processes.

I_MPI_DAPL_BUFFER_NUM

Change the number of internal pre-registered buffers for each process pair in the DAPL path.

Syntax

`I_MPI_DAPL_BUFFER_NUM=`*`<nbuf>`*

Arguments

`<nbuf>`	Define the number of buffers for each pair in a process group
`> 0`	The default value depends on the platform

Description

Set this environment variable to change the number of the internal pre-registered buffers for each process pair in the DAPL path.

NOTE

The more pre-registered buffers are available, the more memory is used for every established connection.

I_MPI_DAPL_BUFFER_SIZE

Change the size of internal pre-registered buffers for each process pair in the DAPL path.

Syntax

`I_MPI_DAPL_BUFFER_SIZE=<nbytes>`

Arguments

<nbytes>	Define the size of pre-registered buffers
> 0	The default value depends on the platform

Description

Set this environment variable to define the size of the internal pre-registered buffer for each process pair in the DAPL path. The actual size is calculated by adjusting the *<nbytes>* to align the buffer to an optimal value.

I_MPI_DAPL_RNDV_BUFFER_ALIGNMENT

Define the alignment of the sending buffer for the DAPL direct-copy transfers.

Syntax

`I_MPI_DAPL_RNDV_BUFFER_ALIGNMENT=<arg>`

Arguments

<arg>	Define the alignment for the sending buffer
> 0 and a power of 2	The default value is 64

Set this environment variable to define the alignment of the sending buffer for DAPL direct-copy transfers. When a buffer specified in a DAPL operation is aligned to an optimal value, the data transfer bandwidth may be increased.

I_MPI_DAPL_RDMA_RNDV_WRITE

Turn on/off the RDMA Write-based rendezvous direct-copy protocol in the DAPL path.

Syntax

`I_MPI_DAPL_RDMA_RNDV_WRITE=<arg>`

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Turn on the RDMA Write rendezvous direct-copy protocol
disable | no | off | 0	Turn off the RDMA Write rendezvous direct-copy protocol

Description

Set this environment variable to select the RDMA Write-based rendezvous direct-copy protocol in the DAPL path. Certain DAPL* providers have a slow RDMA Read implementation on certain platforms. Switching on the rendezvous direct-copy protocol based on the RDMA Write operation can increase performance in these cases. The default value depends on the DAPL provider attributes.

I_MPI_DAPL_CHECK_MAX_RDMA_SIZE

Check the value of the DAPL attribute, `max_rdma_size`.

Syntax

`I_MPI_DAPL_CHECK_MAX_RDMA_SIZE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Check the value of the DAPL* attribute `max_rdma_size`
`disable | no | off | 0`	Do not check the value of the DAPL* attribute `max_rdma_size`. This is the default value

Description

Set this environment variable to control message fragmentation according to the following scheme:

- If this mode is enabled, the Intel® MPI Library fragmentizes the messages bigger than the value of the DAPL attribute `max_rdma_size`
- If this mode is disabled, the Intel® MPI Library does not take into account the value of the DAPL attribute `max_rdma_size` for message fragmentation

I_MPI_DAPL_MAX_MSG_SIZE

Control message fragmentation threshold.

Syntax

`I_MPI_DAPL_MAX_MSG_SIZE=<nbytes>`

Arguments

<nbytes>	Define the maximum message size that can be sent through DAPL without fragmentation
`> 0`	If the `I_MPI_DAPL_CHECK_MAX_RDMA_SIZE` environment variable is enabled, the default *<nbytes>* value is equal to the `max_rdma_size` DAPL attribute value. Otherwise the default value is `MAX_INT`

Description

Set this environment variable to control message fragmentation size according to the following scheme:

- If the `I_MPI_DAPL_CHECK_MAX_RDMA_SIZE` environment variable is set to `disable`, the Intel® MPI Library fragmentizes the messages whose sizes are greater than *<nbytes>*.
- If the `I_MPI_DAPL_CHECK_MAX_RDMA_SIZE` environment variable is set to `enable`, the Intel® MPI Library fragmentizes the messages whose sizes are greater than the minimum of *<nbytes>* and the `max_rdma_size` DAPL* attribute value.

I_MPI_DAPL_CONN_EVD_SIZE

Define the event queue size of the DAPL event dispatcher for connections.

Syntax

`I_MPI_DAPL_CONN_EVD_SIZE=<size>`

Arguments

<size>	Define the length of the event queue
> 0	The default value is 2*number of processes + 32 in the MPI job

Description

Set this environment variable to define the event queue size of the DAPL event dispatcher that handles connection related events. If this environment variable is set, the minimum value between *<size>* and the value obtained from the provider is used as the size of the event queue. The provider is required to supply a queue size that equal or larger than the calculated value.

I_MPI_DAPL_SR_THRESHOLD

Change the threshold of switching send/recv to rdma path for DAPL wait mode.

Syntax

I_MPI_DAPL_SR_THRESHOLD=*<arg>*

Arguments

<nbytes>	Define the message size threshold of switching send/recv to rdma
>= 0	The default *<nbytes>* value is 256 bytes

Description

Set this environment variable to control the protocol used for point-to-point communication in DAPL wait mode:

- Messages shorter than or equal in size to *<nbytes>* are sent using DAPL send/recv data transfer operations.
- Messages greater in size than *<nbytes>* are sent using DAPL RDMA WRITE or RDMA WRITE immediate data transfer operations.

I_MPI_DAPL_SR_BUF_NUM

Change the number of internal pre-registered buffers for each process pair used in DAPL wait mode for send/recv path.

Syntax

I_MPI_DAPL_SR_BUF_NUM=*<nbuf>*

Arguments

<nbuf>	Define the number of send/recv buffers for each pair in a process group
> 0	The default value is 32

Description

Set this environment variable to change the number of the internal send/recv pre-registered buffers for each process pair.

I_MPI_DAPL_RDMA_WRITE_IMM

Enable/disable RDMA Write with immediate data InfiniBand (IB) extension in DAPL wait mode.

Syntax

`I_MPI_DAPL_RDMA_WRITE_IMM=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on RDMA Write with immediate data IB extension
`disable | no | off | 0`	Turn off RDMA Write with immediate data IB extension

Description

Set this environment variable to utilize RDMA Write with immediate data IB extension. The algorithm is enabled if this environment variable is set and a certain DAPL provider attribute indicates that RDMA Write with immediate data IB extension is supported.

I_MPI_DAPL_DESIRED_STATIC_CONNECTIONS_NUM

Define the number of processes that establish DAPL static connections at the same time.

Syntax

`I_MPI_DAPL_DESIRED_STATIC_CONNECTIONS_NUM=<num_processess>`

Arguments

<num_processess>	Define the number of processes that establish DAPL static connections at the same time
`> 0`	The default *<num_processess>* value is equal to 256

Description

Set this environment variable to control the algorithm of DAPL static connection establishment.

If the number of processes in the MPI job is less than or equal to *<num_processess>*, all MPI processes establish the static connections simultaneously. Otherwise, the processes are distributed into several groups. The number of processes in each group is calculated to be close to *<num_processess>*. Then static connections are established in several iterations, including intergroup connection setup.

I_MPI_CHECK_DAPL_PROVIDER_COMPATIBILITY

Enable/disable the check that the same DAPL provider is selected by all ranks.

Syntax

`I_MPI_CHECK_DAPL_PROVIDER_COMPATIBILITY=<arg>`

Arguments

<arg>	Binary indicator

`enable | yes | on | 1`	Turn on the check that the DAPL provider is the same on all ranks. This is default value
`disable | no | off | 0`	Turn off the check that the DAPL provider is the same on all ranks

Description

Set this variable to make a check if the DAPL provider is selected by all MPI ranks. If this check is enabled, Intel® MPI Library checks the name of DAPL provider and the version of DAPL. If these parameters are not the same on all ranks, Intel MPI Library does not select the RDMA path and may fall to sockets. Turning off the check reduces the execution time of `MPI_Init()`. It may be significant for MPI jobs with a large number of processes.

3.4.4. DAPL UD-capable Network Fabrics Control

I_MPI_DAPL_UD

Enable/disable using DAPL UD path.

Syntax

`I_MPI_DAPL_UD=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on using DAPL UD IB extension
`disable | no | off | 0`	Turn off using DAPL UD IB extension. This is the default value

Description

Set this environment variable to enable DAPL UD path for transferring data. The algorithm is enabled if you set this environment variable and a certain DAPL provider attribute indicates that UD IB extension is supported.

I_MPI_DAPL_UD_PROVIDER

Define the DAPL provider to work with IB UD transport.

Syntax

`I_MPI_DAPL_UD_PROVIDER=<name>`

Arguments

<name>	Define the name of DAPL provider to load

Description

Set this environment variable to define the name of DAPL provider to load. This name is also defined in the `dat.conf` configuration file. Make sure that specified DAPL provider supports UD IB extension.

I_MPI_DAPL_UD_DIRECT_COPY_THRESHOLD

Change the message size threshold of the DAPL UD direct-copy protocol.

Syntax

`I_MPI_DAPL_UD_DIRECT_COPY_THRESHOLD=<nbytes>`

Arguments

<nbytes>	Define the DAPL UD direct-copy protocol threshold
`> 0`	The default *<nbytes>* value is equal to `16456` bytes

Description

Set this environment variable to control the DAPL UD direct-copy protocol threshold. Data transfer algorithms for the DAPL-capable network fabrics are selected based on the following scheme:

- Messages shorter than or equal to *<nbytes>* are sent using the eager protocol through the internal pre-registered buffers. This approach is faster for short messages.
- Messages larger than *<nbytes>* are sent using the direct-copy protocol. It does not use any buffering but involves registration of memory on sender and receiver sides. This approach is faster for large messages.

This environment variable is available for both Intel® and non-Intel microprocessors, but it may perform additional optimizations for Intel microprocessors than it performs for non-Intel microprocessors.

I_MPI_DAPL_UD_RECV_BUFFER_NUM

Change the number of the internal pre-registered UD buffers for receiving messages.

Syntax

`I_MPI_DAPL_UD_RECV_BUFFER_NUM=<nbuf>`

Arguments

<nbuf>	Define the number of buffers for receiving messages
`> 0`	The default value is `16 + n*4` where `n` is a total number of process in MPI job

Description

Set this environment variable to change the number of the internal pre-registered buffers for receiving messages. These buffers are common for all connections or process pairs.

NOTE

The pre-registered buffers use up memory, however they help avoid the loss of packets.

I_MPI_DAPL_UD_SEND_BUFFER_NUM

Change the number of internal pre-registered UD buffers for sending messages.

Syntax

`I_MPI_DAPL_UD_SEND_BUFFER_NUM=<nbuf>`

Arguments

<nbuf>	Define the number of buffers for sending messages

> 0	The default value is 16 + n*4 where n is a total number of process in MPI job

Description

Set this environment variable to change the number of the internal pre-registered buffers for sending messages. These buffers are common for all connections or process pairs.

NOTE

The pre-registered buffers use up memory, however they help avoid the loss of packets.

I_MPI_DAPL_UD_ACK_SEND_POOL_SIZE

Change the number of ACK UD buffers for sending messages.

Syntax

I_MPI_DAPL_UD_ACK_SEND_POOL_SIZE=*<nbuf>*

Arguments

<nbuf>	Define the number of ACK UD buffers for sending messages
> 0	The default value is 256

Description

Set this environment variable to change the number of the internal pre-registered ACK buffers for sending service messages. These buffers are common for all connections or process pairs.

I_MPI_DAPL_UD_ACK_RECV_POOL_SIZE

Change the number of ACK UD buffers for receiving messages.

Syntax

I_MPI_DAPL_UD_ACK_RECV_POOL_SIZE=*<nbuf>*

Arguments

<nbuf>	Define the number of ACK UD buffers for receiving messages
> 0	The default value is 512+n*4, where n is total number of process in MPI job

Description

Set this environment variable to change the number of the internal pre-registered ACK buffers for receiving service messages. These buffers are common for all connections or process pairs.

I_MPI_DAPL_UD_TRANSLATION_CACHE

Turn on/off the memory registration cache in the DAPL UD path.

Syntax

I_MPI_DAPL_UD_TRANSLATION_CACHE=*<arg>*

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Turn on the memory registration cache. This is the default
disable | no | off | 0	Turn off the memory registration cache

Description

Set this environment variable to turn off the memory registration cache in the DAPL UD path.

Using the cache substantially improves performance. See product *Release Notes* for further details.

I_MPI_DAPL_UD_TRANSLATION_CACHE_AVL_TREE

Enable/disable the AVL* tree based implementation of RDMA translation cache in the DAPL UD path.

Syntax

I_MPI_DAPL_UD_TRANSLATION_CACHE_AVL_TREE=*<arg>*

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Turn on the AVL tree based RDMA translation cache
disable | no | off | 0	Turn off the AVL tree based RDMA translation cache. This is the default value

Description

Set this environment variable to enable the AVL tree based implementation of RDMA translation cache in the DAPL UD path. When the search in RDMA translation cache handles over 10,000 elements, the AVL tree based RDMA translation cache is faster than the default implementation.

I_MPI_DAPL_UD_REQ_EVD_SIZE

Define the event queue size of the DAPL UD event dispatcher for sending data transfer operations.

Syntax

I_MPI_DAPL_UD_REQ_EVD_SIZE=*<size>*

Arguments

<size>	Define the length of the event queue
> 0	The default value is 2,000

Description

Set this environment variable to define the event queue size of the DAPL event dispatcher that handles completions of sending DAPL UD data transfer operations (DTO). If this environment variable is set, the minimum value between *<size>* and the value obtained from the provider is used as the size of the event queue. The provider is required to supply a queue size that is at least equal to the calculated value, but it can also provide a larger queue size.

I_MPI_DAPL_UD_CONN_EVD_SIZE

Define the event queue size of the DAPL UD event dispatcher for connections.

Syntax

`I_MPI_DAPL_UD_CONN_EVD_SIZE=<size>`

Arguments

<size>	Define the length of the event queue
> 0	The default value is `2*number of processes + 32`

Description

Set this environment variable to define the event queue size of the DAPL event dispatcher that handles connection related events. If this environment variable is set, the minimum value between *<size>* and the value obtained from the provider is used as the size of the event queue. The provider is required to supply a queue size that is at least equal to the calculated value, but it can also provide a larger queue size.

I_MPI_DAPL_UD_RECV_EVD_SIZE

Define the event queue size of the DAPL UD event dispatcher for receiving data transfer operations.

Syntax

`I_MPI_DAPL_UD_RECV_EVD_SIZE=<size>`

Arguments

<size>	Define the length of the event queue
> 0	The default value depends on the number UD and ACK buffers

Description

Set this environment variable to define the event queue size of the DAPL event dispatcher that handles completions of receiving DAPL UD data transfer operations (DTO). If this environment variable is set, the minimum value between *<size>* and the value obtained from the provider is used as the size of the event queue. The provider is required to supply a queue size that is at least equal to the calculated value, but it can also provide a larger queue size.

I_MPI_DAPL_UD_RNDV_MAX_BLOCK_LEN

Define maximum size of block that is passed at one iteration of DAPL UD direct-copy protocol.

Syntax

`I_MPI_DAPL_UD_RNDV_MAX_BLOCK_LEN=<nbytes>`

Arguments

<arg>	Define maximum size of block that is passed at one iteration of DAPL UD direct-copy protocol
> 0	The default value is `1,048,576`

Set this environment variable to define the maximum size of memory block that is passed at one iteration of DAPL UD direct-copy protocol. If the size of message in direct-copy protocol is greater than given value, the message will be divided in several blocks and passed in several operations.

I_MPI_DAPL_UD_RNDV_BUFFER_ALIGNMENT

Define the alignment of the sending buffer for the DAPL UD direct-copy transfers.

Syntax

`I_MPI_DAPL_UD_RNDV_BUFFER_ALIGNMENT=<arg>`

Arguments

<arg>	Define the alignment of the sending buffer
`> 0 and a power of 2`	The default value is `16`

Set this environment variable to define the alignment of the sending buffer for DAPL direct-copy transfers. When a buffer specified in a DAPL operation is aligned to an optimal value, this may increase data transfer bandwidth.

I_MPI_DAPL_UD_RNDV_COPY_ALIGNMENT_THRESHOLD

Define threshold where alignment is applied to send buffer for the DAPL UD direct-copy transfers.

Syntax

`I_MPI_DAPL_UD_RNDV_COPY_ALIGNMENT_THRESHOLD=<nbytes>`

Arguments

<nbytes>	Define send buffer alignment threshold
`> 0 and a power of 2`	The default value is `32,768`

Set this environment variable to define the threshold where the alignment of the sending buffer is applied in DAPL direct-copy transfers. When a buffer specified in a DAPL operation is aligned to an optimal value, this may increase data transfer bandwidth.

I_MPI_DAPL_UD_RNDV_DYNAMIC_CONNECTION

Control the algorithm of dynamic connection establishment for DAPL UD endpoints used in the direct copy protocol.

Syntax

`I_MPI_DAPL_UD_RNDV_DYNAMIC_CONNECTION=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turns on the dynamic connection mode. This is the default value
`disable | no | off | 0`	Turns off the dynamic connections mode

Set this variable to control the dynamic connection establishment of DAPL UD endpoints used in the direct copy protocol.

If you disable the dynamic connection mode, all possible connections are established during the MPI startup phase.

If you enable the mode, the connection is established when an application calls the MPI function to pass the data from one process to another and invokes the communication between the two processes.

NOTE

For the RNDV dynamic connection mode, the size of the messages passed in the data is larger than the value you set in the `I_MPI_DAPL_UD_DIRECT_COPY_THRESHOLD` environment variable.

I_MPI_DAPL_UD_EAGER_DYNAMIC_CONNECTION

Control the algorithm of the dynamic connection establishment for DAPL UD endpoints used in eager protocol.

Syntax

`I_MPI_DAPL_UD_EAGER_DYNAMIC_CONNECTION=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the dynamic connection mode. If the number of processes is over `64`, this is the default value
`disable | no | off | 0`	Turn off the dynamic connections mode

Set this variable to control the dynamic connection establishment of DAPL UD endpoints involved in eager protocol. Eager protocol is used to transfer messages through internal pre-registered buffers.

If you disable this mode, all possible connections are established during MPI startup phase.

If you enable this mode, the connection is established when an application calls the MPI function to pass the data from one process to another and invokes the communication between the two processes.

NOTE

For the eager dynamic connection mode, the size of the messages passed in the data is shorter than or equal to the value you set in the `I_MPI_DAPL_UD_DIRECT_COPY_THRESHOLD` environment variable.

I_MPI_DAPL_UD_DESIRED_STATIC_CONNECTIONS_NUM

Define the number of processes that establish DAPL static connections at the same time.

Syntax

`I_MPI_DAPL_UD_DESIRED_STATIC_CONNECTIONS_NUM=<num_processess>`

Arguments

<num_processess>	Define the number of processes that establish DAPL UD static connections at the same time

> 0	The default value is equal to 200

Description

Set this environment variable to control the algorithm of DAPL UD static connections establishment.

If the number of processes in an MPI job is less than or equal to <num_procesess>, all MPI processes establish the static connections simultaneously. Otherwise, the processes are distributed into several groups. The number of processes in each group is calculated to be close to <num_procesess>. Then static connections are established in several iterations, including intergroup connection setup.

I_MPI_DAPL_UD_RDMA_MIXED

Control the use of the DAPL UD/RDMA mixed communication.

Syntax

I_MPI_DAPL_UD_RDMA_MIXED =<arg>

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Turn on the use of DAPL UD/RDMA mixed communication
disable | no | off | 0	Turn off the use of DAPL UD/RDMA mixed communication. This is the default value

Description

Set this environment variable to enable the DAPL UD/RDMA mixed mode for transferring data. In the DAPL UD/RDMA mixed mode, small messages are passed through the UD transport and large messages are passed through the RDMA transport. If you set the I_MPI_DAPL_UD_RDMA_MIXED environment variable and a certain DAPL provider attribute indicates that UD IB extension is supported, the DAPL UD/RDMA mixed mode is enabled.

The following set of I_MPI_DAPL_UD* environment variables also controls the DAPL UD/RDMA mixed mode:

- I_MPI_DAPL_UD_PROVIDER
- I_MPI_DAPL_UD_EAGER_DYNAMIC_CONNECTION
- I_MPI_DAPL_UD_RNDV_DYNAMIC_CONNECTION
- I_MPI_DAPL_UD_DIRECT_COPY_THRESHOLD
- I_MPI_DAPL_UD_RECV_BUFFER_NUM
- I_MPI_DAPL_UD_SEND_BUFFER_NUM
- I_MPI_DAPL_UD_NUMBER_CREDIT_UPDATE
- I_MPI_DAPL_UD_ACK_SEND_POOL_SIZE
- I_MPI_DAPL_UD_ACK_RECV_POOL_SIZE
- I_MPI_DAPL_UD_RESENT_TIMEOUT
- I_MPI_DAPL_UD_MAX_MSG_SIZE
- I_MPI_DAPL_UD_SEND_BUFFER_SIZE
- I_MPI_DAPL_UD_REQ_EVD_SIZE
- I_MPI_DAPL_UD_REQUEST_QUEUE_SIZE

- I_MPI_DAPL_UD_MULTIPLE_EAGER_SEND
- I_MPI_DAPL_UD_NA_SBUF_LIMIT
- I_MPI_DAPL_UD_RECV_EVD_SIZE
- I_MPI_DAPL_UD_CONNECTION_TIMEOUT
- I_MPI_DAPL_UD_PORT
- I_MPI_DAPL_UD_CREATE_CONN_QUAL,
- I_MPI_DAPL_UD_FINALIZE_RETRY_COUNT
- I_MPI_DAPL_UD_FINALIZE_TIMEOUT
- I_MPI_DAPL_UD_TRANSLATION_CACHE
- I_MPI_DAPL_UD_TRANSLATION_CACHE_AVL_TREE
- I_MPI_DAPL_UD_TRANSLATION_CACHE_MAX_ENTRY_NUM
- I_MPI_DAPL_UD_TRANSLATION_CACHE_MAX_MEMORY_SIZE
- I_MPI_DAPL_UD_PKT_LOSS_OPTIMIZATION
- I_MPI_DAPL_UD_DFACTOR
- I_MPI_DAPL_UD_DESIRED_STATIC_CONNECTIONS_NUM
- I_MPI_DAPL_UD_CONN_EVD_SIZE
- I_MPI_DAPL_UD_RNDV_BUFFER_ALIGNMENT
- I_MPI_DAPL_UD_RNDV_COPY_ALIGNMENT_THRESHOLD

The following set of environment variables is specific for DAPL UD/RDMA mixed mode:

- I_MPI_DAPL_UD_MAX_RDMA_SIZE
- I_MPI_DAPL_UD_MAX_RDMA_DTOS

I_MPI_DAPL_UD_MAX_RDMA_SIZE

Control the maximum message size that can be sent though the RDMA for DAPL UD/RDMA mixed mode.

Syntax

`I_MPI_DAPL_UD_MAX_RDMA_SIZE =<nbytes>`

Arguments

<nbytes>	Define the maximum message size that can be sent through RDMA without fragmentation
> 0	The default *<nbytes>* value is 4 MB

Description

Set this environment variable to define the maximum message size that can be sent though RDMA protocol for the DAPL UD/RDMA mixed mode. If the message size is greater than this value, this message is divided into several fragments and is sent by several RDMA operations.

I_MPI_DAPL_UD_MAX_RDMA_DTOS

Control the maximum number of uncompleted RDMA operations per connection for the DAPL UD/RDMA mixed mode.

Syntax

`I_MPI_DAPL_UD_MAX_RDMA_DTOS=<arg>`

Arguments

<arg>	Define the maximum number of RDMA operations per connection
> 0	The default *<arg>* value is 8

Description

Set this environment variable to define the maximum number of RDMA operations per connection for the DAPL UD/RDMA mixed mode.

3.4.5. TCP-capable Network Fabrics Control

I_MPI_TCP_NETMASK

Choose the network interface for MPI communication over TCP-capable network fabrics.

Syntax

I_MPI_TCP_NETMASK=*<arg>*

Arguments

<arg>	Define the network interface (string parameter)
<interface_mnemonic>	Mnemonic of the network interface: ib or eth
ib	Use IPoIB* network interface
eth	Use Ethernet network interface. This is the default value
<interface_name>	Name of the network interface Usually the UNIX* driver name followed by the unit number
<network_address>	Network address. Trailing zero bits imply a netmask
<network_address/ <netmask>	Network address. The *<netmask>* value specifies the netmask length
<list of interfaces>	A colon separated list of network addresses and interface names

Description

Set this environment variable to choose the network interface for MPI communication over TCP-capable network fabrics. If you specify a list of interfaces, the first available interface on the node is used for communication.

Examples

- Use the following setting to select the IP over InfiniBand* (IPoIB) fabric:

  I_MPI_TCP_NETMASK=ib
- Use the following setting to select the specified network interface for socket communications:

  I_MPI_TCP_NETMASK=ib0

- Use the following setting to select the specified network for socket communications. This setting implies the `255.255.0.0` netmask:
  `I_MPI_TCP_NETMASK=192.169.0.0`
- Use the following setting to select the specified network for socket communications with netmask set explicitly:
  `I_MPI_TCP_NETMASK=192.169.0.0/24`
- Use the following setting to select the specified network interfaces for socket communications:
  `I_MPI_TCP_NETMASK=192.169.0.5/24:ib0:192.169.0.0`

I_MPI_TCP_BUFFER_SIZE

Change the size of the TCP socket buffers.

Syntax

`I_MPI_TCP_BUFFER_SIZE=<nbytes>`

Arguments

<nbytes>	Define the size of the TCP socket buffers
> 0	The default *<nbytes>* value is equal to default value of the TCP socket buffer size on your Linux system.

Description

Set this environment variable to manually define the size of the TCP socket buffers. The TCP socket buffer size is restricted by the existing TCP settings on your Linux system.

Use the `I_MPI_TCP_BUFFER_SIZE` environment variable for tuning your application performance for a given number of processes.

NOTE

TCP socket buffers of a large size can require more memory for an application with large number of processes. Alternatively, TCP socket buffers of a small size can considerably decrease the bandwidth of each socket connection especially for 10 Gigabit Ethernet and IPoIB (see *I_MPI_TCP_NETMASK* for details).

3.4.6. TMI-capable Network Fabrics Control

I_MPI_TMI_LIBRARY

Select the TMI library to be used.

Syntax

`I_MPI_TMI_LIBRARY=<library>`

Arguments

<library>	Specify a TMI library to be used instead of the default `libtmi.so`

Description

Set this environment variable to select a specific TMI library. Specify the full path to the TMI library if the library does not locate in the dynamic loader search path.

I_MPI_TMI_PROVIDER

Define the name of the TMI provider to load.

Syntax

`I_MPI_TMI_PROVIDER=<name>`

Arguments

<name>	The name of the TMI provider to load

Description

Set this environment variable to define the name of the TMI provider to load. The name must also be defined in the `tmi.conf` configuration file.

I_MPI_TMI_NBITS_RANK

Defines the number of the bits that can be reserved for the storage of MPI rank values at the TMI level.

Syntax

`I_MPI_TMI_NBITS_RANK=<num_bits>`

Arguments

<num_bits>	The number of the bits reserved for the MPI rank storage
`<=30 and > 0`	The default value is `24`

Description

The value of `I_MPI_TMI_NBITS_RANK` specifies how many MPI ranks can be referenced and distinguished at TMI level. Thus, if you specify the default value for this environment variable, `I_MPI_TMI_NBITS_RANK=24`, the number of ranks allowed for running a job is 2^{24}=`16M` ranks.

NOTE

The `I_MPI_TMI_NBITS_RANK` variable takes effect only for TMI version 1.2 or lower.

NOTE

The value of `I_MPI_TMI_NBITS_RANK` is related to `MPI_TAG_UB`. The larger the value you specify for `I_MPI_TMI_NBITS_RANK`, the less tag value `MPI_TAG_UB` is supported. The less value you specify for `I_MPI_TMI_NBITS_RANK`, the larger tag value `MPI_TAG_UB` is supported. The correct MPI application should always query `MPI_TAG_UB` for the largest supported tag value.

I_MPI_TMI_DSEND

Control the capability of the direct send in the TMI fabric.

Syntax

`I_MPI_TMI_DSEND=<arg>`

Arguments

<arg>	Binary indicator

`enable | yes | on | 1`	Enable the direct send. This is default value
`disable | no | off | 0`	Disable the direct send

Description

Use the direct send capability to block `MPI_Send` calls only. Before using the direct send capability, ensure that you use it for single-threaded MPI applications and check if you have selected TMI as the network fabrics (setting `I_MPI_FABRICS=tmi`).

NOTE

The direct send capability is only supported in the TMI version 1.1 or higher. If you use a lower TMI version, the specified value of `I_MPI_TMI_DSEND` is ignored.

I_MPI_TMI_DRECV

Control the capability of the direct receive in the TMI fabric.

Syntax

`I_MPI_TMI_DRECV=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable the direct receive. This is default value
`disable | no | off | 0`	Disable the direct receive

Description

Use the direct receive capability to block `MPI_Recv` calls only. Before using the direct receive capability, ensure that you use it for single-threaded MPI applications and check if you have selected TMI as the network fabrics (setting `I_MPI_FABRICS=tmi`).

3.4.7. OFA-capable Network Fabrics Control

I_MPI_OFA_NUM_ADAPTERS

Set the number of connection adapters.

Syntax

`I_MPI_OFA_NUM_ADAPTERS=<arg>`

Arguments

<arg>	Define the maximum number of connection adapters used
`>0`	Use the specified number of adapters. The default value is `1`

Description

Set the number of the adapters that are used. If the number is greater than the available number of adapters, all the available adaptors are used.

I_MPI_OFA_ADAPTER_NAME

Set the name of adapter that is used.

Syntax

`I_MPI_OFA_ADAPTER_NAME=<arg>`

Arguments

<arg>	Define the name of adapter
`Name`	Use the specified adapter. By default, any adapter can be used

Description

Set the name of adaptor to be used. If the adapter with specified name does not exist, the library returns an error. This has effect only if `I_MPI_OFA_NUM_ADAPTERS=1`.

I_MPI_OFA_NUM_PORTS

Set the number of used ports on each adapter.

Syntax

`I_MPI_OFA_NUM_PORTS=<arg>`

Arguments

<arg>	Define the number of ports that are used on each adapter
`> 0`	Use the specified number of ports. The default value is `1`

Description

Set the number of used ports on each adaptor. If the number is greater than the available number of ports, all the available ports are used.

I_MPI_OFA_NUM_RDMA_CONNECTIONS

Set the maximum number of connections that can use the `rdma` exchange protocol.

Syntax

`I_MPI_OFA_NUM_RDMA_CONNECTIONS=<num_conn>`

Arguments

<num_conn>	Define the maximum number of connections that can use the `rdma` exchange protocol
`>= 0`	Create the specified number of connections that use the `rdma` exchange protocol. All other processes use the send/ receive exchange protocol
`-1`	Create $\log_2$(`number of processes`) `rdma` connections

>= number of processes	Create rdma connections for all processes. This is the default value

Description

There are two exchange protocols between two processes: send/receive and rdma. This environment variable specifies the maximum amount of connections that use rdma protocol.

RDMA protocol is faster but requires more resources. For a large application, you can limit the number of connections that use the rdma protocol so that only processes that actively exchange data use the rdma protocol.

I_MPI_OFA_SWITCHING_TO_RDMA

Set the number of messages that a process should receive before switching this connection to RDMA exchange protocol.

Syntax

I_MPI_OFA_SWITCHING_TO_RDMA=<*number*>

Arguments

<*number*>	Define the number of messages that the process receives before switching to use the rdma protocol
>= 0	If this process receives <*number*> of messages, start using the rdma protocol

Description

Count the number of messages received from the specific process. The connection achieved the specified number tries to switch to rdma protocol for exchanging with that process. The connection will not switch to rdma protocol if the maximum number of connections that use the rdma exchange protocol defined in I_MPI_OFA_NUM_RDMA_CONNECTIONS has been reached.

I_MPI_OFA_RAIL_SCHEDULER

Set the method of choosing rails for short messages.

Syntax

I_MPI_OFA_RAIL_SCHEDULER=<*arg*>

Arguments

<*arg*>	Mode selector
ROUND_ROBIN	Next time use next rail
FIRST_RAIL	Always use the first rail for short messages
PROCESS_BIND	Always use the rail specific for process

Description

Set the method of choosing rails for short messages. The algorithms are selected according to the following scheme:

- In the `ROUND_ROBIN` mode, the first message is sent using the first rail; the next message is sent using the second rail, and so on.
- In the `FIRST_RAIL` mode, the first rail is always used for short messages.
- In the `PROCESS_BIND` mode, the process with the smallest rank uses the first rail, and the next uses the second rail.

I_MPI_OFA_TRANSLATION_CACHE

Turn on/off the memory registration cache.

Syntax

`I_MPI_OFA_TRANSLATION_CACHE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the memory registration cache. This is the default
`disable | no | off | 0`	Turn off the memory registration cache

Description

Set this environment variable to turn on/off the memory registration cache.

The cache substantially increases performance, but may lead to correctness issues in certain situations. See product *Release Notes* for further details.

I_MPI_OFA_TRANSLATION_CACHE_AVL_TREE

Enable/disable the AVL tree* based implementation of the RDMA translation cache.

Syntax

`I_MPI_OFA_TRANSLATION_CACHE_AVL_TREE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the AVL tree based RDMA translation cache
`disable | no | off | 0`	Turn off the AVL tree based RDMA translation cache. This is the default value

Description

Set this environment variable to enable the AVL tree based implementation of RDMA translation cache in the OFA path. When the search in RDMA translation cache handles over 10,000 elements, the AVL tree based RDMA translation cache is faster than the default implementation.

I_MPI_OFA_DYNAMIC_QPS

Control the library to create queue pairs (QPs) dynamically.

Syntax

`I_MPI_OFA_DYNAMIC_QPS=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Create QPs dynamically. This is the default value if the number of processes is larger than or equal `2,000`
`disable | no | off | 0`	Create all QPs during the initial stage. This is the default value if the number of processes is less than `2,000`

Description

Set this environment variable to turn on dynamic creation of QPs.

I_MPI_OFA_PACKET_SIZE

Set the size of the packet used for sending.

Syntax

`I_MPI_OFA_PACKET_SIZE=`*<arg>*

Arguments

<arg>	Define the size of packet in bytes
`> 0`	Use the specified packet size. The default value is `8192`

Description

Set the packet size in bytes. If the number is negative, the size is set to `8`.

I_MPI_OFA_LIBRARY

Set the name of the used OFA library.

Syntax

`I_MPI_OFA_LIBRARY=`*<arg>*

Arguments

<arg>	Define the name of the OFA library
`Name`	Use the specified library. By default, the name is `libibverbs.so`

Description

Set the name of the InfiniBand* (IB*) library. If the library with the specified name does not exist, an error is returned.

I_MPI_OFA_NONSWITCH_CONF

Define the nonstandard template for port connections.

Syntax

`I_MPI_OFA_NONSWITCH_CONF=`*<arg>*

Arguments

<arg>	Define the template for port connections
`Name`	Use the specified template

Description

The nodes in clusters are normally connected so that $port_i$ of a node can access $port_i$ of all other nodes. Use this environment variable if ports are not connected in this way. The following example is the template format:

```
host1@port11#port12#...#host2@port21#port22....
```

$Port_i^j$ defines the port used to send from $host_i$ to $host_j$

For example:

```
node1@1#1#2#node2@2#1#1#node3@1#2#1#
```

This sample specifies the following configuration:

- Port1 of node1 connected to port2 of node2
- Port2 of node1 connected to port1 of node3
- Port1 of node2 connected to port2 of node3
- Port2 of node2 connected to port1 of node2
- Port1 of node3 connected to port2 of node1
- Port2 of node3 connected to port1 of node2

Port1 is always used to communicate with itself (loopback).

Failover Support in the OFA* Device

Intel® MPI Library recognizes the following events to detect hardware issues:

- `IBV_EVENT_QP_FATAL` Error occurred on a QP and it transitioned to error state
- `IBV_EVENT_QP_REQ_ERR` Invalid request local work queue error
- `IBV_EVENT_QP_ACCESS_ERR` Local access violation error
- `IBV_EVENT_PATH_MIG_ERR` A connection failed to migrate to the alternate path
- `IBV_EVENT_CQ_ERR` CQ is in error (CQ overrun)
- `IBV_EVENT_SRQ_ERR` Error occurred on an SRQ
- `IBV_EVENT_PORT_ERR` Link became unavailable on a port
- `IBV_EVENT_DEVICE_FATAL` CA is in `FATAL` state

Intel® MPI Library stops using a port or the whole adapter for communications if one of these issues is detected. The communications continue through an available port or adapter, if the application is running in multi-rail mode. The application is aborted if no healthy ports/adapters are available.

Intel® MPI Library also recognizes the following event

- `IBV_EVENT_PORT_ACTIVE` Link became active on a port

The event indicates that the port can be used again and is enabled for communications.

3.4.8. OFI*-capable Network Fabrics Control

I_MPI_OFI_LIBRARY

Select the OpenFabrics Interfaces* (OFI*) library to be used.

Syntax

`I_MPI_OFI_LIBRARY=<library>`

Arguments

<library>	Specify an OFI library to be used instead of the default `libfabric.so`

Description

Set this environment variable to select a specific OFI library. Specify the full path to the OFI library if the library is not located in the dynamic loader search path.

I_MPI_OFI_PROVIDER

Define the name of the OFI provider to load.

Syntax

`I_MPI_OFI_PROVIDER=<name>`

Arguments

<name>	The name of the OFI provider to load

Description

Set this environment variable to define the name of the OFI provider to load. If you do not specify this variable, the OFI library chooses the provider automatically. You can check all available providers by using the `I_MPI_OFI_PROVIDER_DUMP` environment variable.

I_MPI_OFI_PROVIDER_DUMP

Control the capability of printing information about all OFI providers and their attributes from an OFI library.

Syntax

`I_MPI_OFI_PROVIDER_DUMP=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Print the list of all OFI providers and their attributes from an OFI library
`disable | no | off | 0`	No action. This is the default value

Description

Set this environment variable to control the capability of printing information about all OFI providers and their attributes from an OFI library.

I_MPI_OFI_DRECV

Control the capability of the direct receive in the OFI fabric.

Syntax

`I_MPI_OFI_DRECV=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable direct receive. This is the default value
`disable | no | off | 0`	Disable direct receive

Description

Use the direct receive capability to block `MPI_Recv` calls only. Before using the direct receive capability, ensure that you use it for single-threaded MPI applications and check if you have selected OFI as the network fabric by setting `I_MPI_FABRICS=ofi`.

I_MPI_OFI_DIRECT_RMA

Control the capability of the direct remote memory access in the OFI fabric.

Syntax

`I_MPI_OFI_DIRECT_RMA=`*<arg>*

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable direct remote memory access
`disable | no | off | 0`	Disable direct remote memory access. This is the default value

Description

Set this environment variable to enable the capabilities of hardware direct remote memory access for `MPI_Get` and `MPI_Put` calls. This capability may significantly improve the performance of RMA get and put operations. Before using the capability of the direct remote memory access, check if you have selected OFI as the network fabric by setting `I_MPI_FABRICS=ofi`.

NOTE

This is an experimental feature and may cause errors in your programs. To use this feature, meet the following requirements:

- Use libfabric* 1.3 or higher version
- Use OFED* 3.18-2 or higher version

The feature has been tested with verbs OFI provider.

I_MPI_OFI_DSEND

Control the capability of the direct send in the OFI fabric.

Syntax

`I_MPI_OFI_DSEND=`*<arg>*

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable the direct send. This is default value for `MPI_THREAD_SINGLE` level of thread support.
`disable | no | off | 0`	Disable the direct send. This is default value for all levels of thread support except for `MPI_THREAD_SINGLE`.

Description

The direct send capability uses an optimized buffered path for `MPI_Send`, `MPI_Isend`, `MPI_Rsend`, and `MPI_Irsend` calls only. The direct send usage is recommended for single-threaded MPI applications.

I_MPI_OFI_ENABLE_LMT

Control the capability of processing large messages (> 2Gb).

Syntax

`I_MPI_OFI_ENABLE_LMT=`*<arg>*

Arguments

<arg>	Binary indicator
`enable | yes | on| 1`	Enable large message processing.
`disable | no | off | 0`	Disable large message processing. This is default value.

Description

Use this environment variable to enable or disable processing of messages larger than 2Gb with the OFI fabric.

NOTE

Enabling large message processing may negatively impact performance of the data transfer. The variable is disabled by default for this reason.

I_MPI_OFI_MAX_MSG_SIZE

Set the maximum size of message processed as single operation.

Syntax

`I_MPI_OFI_MAX_MSG_SIZE=`*<nbytes>*

Arguments

<nbytes>	Define the maximum size of the message, in bytes
`> 0`	The default *<nbytes>* value depends on the OFI provider and hardware used.

Description

Set this environment variable to manually define the maximum size of the message to be processed in a single operation. All messages above this size will be split into smaller messages. Use the

`I_MPI_OFI_MAX_MSG_SIZE` environment variable for tuning your application performance. The recommended value range for this variable is 1Gb-2Gb (`1073741824-2147483648`).

3.5. Collective Operations Control

Each collective operation in the Intel® MPI Library supports a number of communication algorithms. In addition to highly optimized default settings, the library provides a way to control the algorithm selection explicitly. You can do this by using the `I_MPI_ADJUST` environment variable family, which is described in the following section.

These environment variables are available for both Intel® and non-Intel microprocessors, but they may perform additional optimizations for Intel microprocessors than they perform for non-Intel microprocessors.

3.5.1. I_MPI_ADJUST Family

I_MPI_ADJUST_*<opname>*

Control collective operation algorithm selection.

Syntax

`I_MPI_ADJUST_`*`<opname>`*`="`*`<algid>`*`[:`*`<conditions>`*`][;`*`<algid>`*`:`*`<conditions>`*`[...]]"`

Arguments

`<algid>`	Algorithm identifier
`>= 0`	The default value of zero selects the reasonable settings

`<conditions>`	A comma separated list of conditions. An empty list selects all message sizes and process combinations
`<l>`	Messages of size *`<l>`*
`<l>-<m>`	Messages of size from *`<l>`* to *`<m>`*, inclusive
`<l>@<p>`	Messages of size *`<l>`* and number of processes *`<p>`*
`<l>-<m>@<p>-<q>`	Messages of size from *`<l>`* to *`<m>`* and number of processes from *`<p>`* to *`<q>`*, inclusive

Description

Set this environment variable to select the desired algorithm(s) for the collective operation *`<opname>`* under particular conditions. Each collective operation has its own environment variable and algorithms.

Table 3.5-1 Environment Variables, Collective Operations, and Algorithms

Environment Variable	Collective Operation	Algorithms
`I_MPI_ADJUST_ALLGATHER`	`MPI_Allgather`	1. Recursive doubling 2. Bruck's

		3. Ring 4. Topology aware Gatherv + Bcast 5. Knomial
I_MPI_ADJUST_ALLGATHERV	MPI_Allgatherv	1. Recursive doubling 2. Bruck's 3. Ring 4. Topology aware Gatherv + Bcast
I_MPI_ADJUST_ALLREDUCE	MPI_Allreduce	1. Recursive doubling 2. Rabenseifner's 3. Reduce + Bcast 4. Topology aware Reduce + Bcast 5. Binomial gather + scatter 6. Topology aware binominal gather + scatter 7. Shumilin's ring 8. Ring 9. Knomial 10. Topology aware SHM-based flat 11. Topology aware SHM-based Knomial 12. Topology aware SHM-based Knary
I_MPI_ADJUST_ALLTOALL	MPI_Alltoall	1. Bruck's 2. Isend/Irecv + waitall 3. Pair wise exchange 4. Plum's
I_MPI_ADJUST_ALLTOALLV	MPI_Alltoallv	1. Isend/Irecv + waitall 2. Plum's
I_MPI_ADJUST_ALLTOALLW	MPI_Alltoallw	Isend/Irecv + waitall
I_MPI_ADJUST_BARRIER	MPI_Barrier	1. Dissemination 2. Recursive doubling 3. Topology aware dissemination 4. Topology aware recursive doubling 5. Binominal gather + scatter

		6. Topology aware binominal gather + scatter 7. Topology aware SHM-based flat 8. Topology aware SHM-based Knomial 9. Topology aware SHM-based Knary
I_MPI_ADJUST_BCAST	MPI_Bcast	1. Binomial 2. Recursive doubling 3. Ring 4. Topology aware binomial 5. Topology aware recursive doubling 6. Topology aware ring 7. Shumilin's 8. Knomial 9. Topology aware SHM-based flat 10. Topology aware SHM-based Knomial 11. Topology aware SHM-based Knary
I_MPI_ADJUST_EXSCAN	MPI_Exscan	1. Partial results gathering 2. Partial results gathering regarding layout of processes
I_MPI_ADJUST_GATHER	MPI_Gather	1. Binomial 2. Topology aware binomial 3. Shumilin's 4. Binomial with segmentation
I_MPI_ADJUST_GATHERV	MPI_Gatherv	1. Linear 2. Topology aware linear 3. Knomial
I_MPI_ADJUST_REDUCE_SCATTER	MPI_Reduce_scatter	1. Recursive halving 2. Pair wise exchange 3. Recursive doubling 4. Reduce + Scatterv 5. Topology aware Reduce + Scatterv

I_MPI_ADJUST_REDUCE	MPI_Reduce	1. Shumilin's 2. Binomial 3. Topology aware Shumilin's 4. Topology aware binomial 5. Rabenseifner's 6. Topology aware Rabenseifner's 7. Knomial 8. Topology aware SHM-based flat 9. Topology aware SHM-based Knomial 10. Topology aware SHM-based Knary 11. Topology aware SHM-based binomial
I_MPI_ADJUST_SCAN	MPI_Scan	1. Partial results gathering 2. Topology aware partial results gathering
I_MPI_ADJUST_SCATTER	MPI_Scatter	1. Binomial 2. Topology aware binomial 3. Shumilin's
I_MPI_ADJUST_SCATTERV	MPI_Scatterv	1. Linear 2. Topology aware linear
I_MPI_ADJUST_IALLGATHER	MPI_Iallgather	1. Recursive doubling 2. Bruck’s 3. Ring
I_MPI_ADJUST_IALLGATHERV	MPI_Iallgatherv	1. Recursive doubling 2. Bruck’s 3. Ring
I_MPI_ADJUST_IALLREDUCE	MPI_Iallreduce	1. Recursive doubling 2. Rabenseifner’s 3. Reduce + Bcast 4. Ring (patarasuk) 5. Knomial 6. Binomial

I_MPI_ADJUST_IALLTOALL	MPI_Ialltoall	1. Bruck's 2. Isend/Irecv + Waitall 3. Pairwise exchange
I_MPI_ADJUST_IALLTOALLV	MPI_Ialltoallv	Isend/Irecv + Waitall
I_MPI_ADJUST_IALLTOALLW	MPI_Ialltoallw	Isend/Irecv + Waitall
I_MPI_ADJUST_IBARRIER	MPI_Ibarrier	Dissemination
I_MPI_ADJUST_IBCAST	MPI_Ibcast	1. Binomial 2. Recursive doubling 3. Ring 4. Knomial
I_MPI_ADJUST_IEXSCAN	MPI_Iexscan	Recursive doubling
I_MPI_ADJUST_IGATHER	MPI_Igather	1. Binomial 2. Knomial
I_MPI_ADJUST_IGATHERV	MPI_Igatherv	Linear
I_MPI_ADJUST_IREDUCE_SCATTER	MPI_Ireduce_scatter	1. Recursive halving 2. Pairwise 3. Recursive doubling
I_MPI_ADJUST_IREDUCE	MPI_Ireduce	1. Rabenseifner's 2. Binomial 3. Knomial
I_MPI_ADJUST_ISCAN	MPI_Iscan	Recursive Doubling
I_MPI_ADJUST_ISCATTER	MPI_Iscatter	1. Binomial 2. Knomial
I_MPI_ADJUST_ISCATTERV	MPI_Iscatterv	Linear

The message size calculation rules for the collective operations are described in the table. In the following table, "n/a" means that the corresponding interval *<l>-<m>* should be omitted.

Table 3.5-2 Message Collective Functions

Collective Function	Message Size Formula

MPI_Allgather	recv_count*recv_type_size
MPI_Allgatherv	total_recv_count*recv_type_size
MPI_Allreduce	count*type_size
MPI_Alltoall	send_count*send_type_size
MPI_Alltoallv	n/a
MPI_Alltoallw	n/a
MPI_Barrier	n/a
MPI_Bcast	count*type_size
MPI_Exscan	count*type_size
MPI_Gather	recv_count*recv_type_size if MPI_IN_PLACE is used, otherwise send_count*send_type_size
MPI_Gatherv	n/a
MPI_Reduce_scatter	total_recv_count*type_size
MPI_Reduce	count*type_size
MPI_Scan	count*type_size
MPI_Scatter	send_count*send_type_size if MPI_IN_PLACE is used, otherwise recv_count*recv_type_size
MPI_Scatterv	n/a

Examples

Use the following settings to select the second algorithm for `MPI_Reduce` operation:
`I_MPI_ADJUST_REDUCE=2`

Use the following settings to define the algorithms for `MPI_Reduce_scatter` operation:
`I_MPI_ADJUST_REDUCE_SCATTER="4:0-100,5001-10000;1:101-3200,2:3201-5000;3"`

In this case. algorithm 4 is used for the message sizes between 0 and 100 bytes and from 5001 and 10000 bytes, algorithm 1 is used for the message sizes between 101 and 3200 bytes, algorithm 2 is used for the message sizes between 3201 and 5000 bytes, and algorithm 3 is used for all other messages.

I_MPI_ADJUST_REDUCE_SEGMENT

Syntax

`I_MPI_ADJUST_REDUCE_SEGMENT=<block_size>|<algid>:<block_size>[,<algid>:<block_size>[...]]`

Arguments

<algid>	Algorithm identifier
1	Shumilin's algorithm
3	Topology aware Shumilin's algorithm
<block_size>	Size of a message segment in bytes
> 0	The default value is 14000

Description

Set an internal block size to control MPI_Reduce message segmentation for the specified algorithm. If the <algid> value is not set, the <block_size> value is applied for all the algorithms, where it is relevant.

NOTE

This environment variable is relevant for Shumilin's and topology aware Shumilin's algorithms only (algorithm N1 and algorithm N3 correspondingly).

I_MPI_ADJUST_BCAST_SEGMENT

Syntax

I_MPI_ADJUST_BCAST_SEGMENT=<block_size>|<algid>:<block_size>[,<algid>:<block_size>[...]]

Arguments

<algid>	Algorithm identifier
1	Binomial
4	Topology aware binomial
7	Shumilin's
8	Knomial
<block_size>	Size of a message segment in bytes
> 0	The default value is 12288

Description

Set an internal block size to control MPI_Bcast message segmentation for the specified algorithm. If the <algid> value is not set, the <block_size> value is applied for all the algorithms, where it is relevant.

NOTE

This environment variable is relevant only for Binomial, Topology-aware binomial, Shumilin's and Knomial algorithms.

I_MPI_ADJUST_ALLGATHER_KN_RADIX

Syntax

`I_MPI_ADJUST_ALLGATHER_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial `MPI_Allgather` algorithm to build a knomial communication tree
`> 1`	The default value is `2`

Description

Set this environment variable together with `I_MPI_ADJUST_ALLGATHER=5` to select the knomial tree radix for the corresponding `MPI_Allgather` algorithm.

I_MPI_ADJUST_BCAST_KN_RADIX

Syntax

`I_MPI_ADJUST_BCAST_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial `MPI_Bcast` algorithm to build a knomial communication tree
`> 1`	The default value is `4`

Description

Set this environment variable together with `I_MPI_ADJUST_BCAST=8` to select the knomial tree radix for the corresponding `MPI_Bcast` algorithm.

I_MPI_ADJUST_ALLREDUCE_KN_RADIX

Syntax

`I_MPI_ADJUST_ALLREDUCE_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial `MPI_Allreduce` algorithm to build a knomial communication tree
`> 1`	The default value is `4`

Description

Set this environment variable together with `I_MPI_ADJUST_ALLREDUCE=9` to select the knomial tree radix for the corresponding `MPI_Allreduce` algorithm.

I_MPI_ADJUST_REDUCE_KN_RADIX

Syntax

`I_MPI_ADJUST_REDUCE_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial `MPI_Reduce` algorithm to build a knomial communication tree
`> 1`	The default value is `4`

Description

Set this environment variable together with `I_MPI_ADJUST_REDUCE=7` to select the knomial tree radix for the corresponding `MPI_Reduce` algorithm.

I_MPI_ADJUST_GATHERV_KN_RADIX

Syntax

`I_MPI_ADJUST_GATHERV_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial `MPI_Gatherv` algorithm to build a knomial communication tree
`> 1`	The default value is `2`

Description

Set this environment variable together with `I_MPI_ADJUST_GATHERV=3` to select the knomial tree radix for the corresponding `MPI_Gatherv` algorithm.

I_MPI_ADJUST_IALLREDUCE_KN_RADIX

Syntax

`I_MPI_ADJUST_IALLREDUCE_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial `MPI_Iallreduce` algorithm to build a knomial communication tree
`> 1`	The default value is 4

Description

Set this environment variable together with `I_MPI_ADJUST_IALLREDUCE=5` to select the knomial tree radix for the corresponding `MPI_Iallreduce` algorithm.

I_MPI_ADJUST_IBCAST_KN_RADIX

Syntax

`I_MPI_ADJUST_IBCAST_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial `MPI_Ibcast` algorithm to build a knomial communication tree

> 1	The default value is 4

Description

Set this environment variable together with `I_MPI_ADJUST_IBCAST=4` to select the knomial tree radix for the corresponding `MPI_Ibcast` algorithm.

I_MPI_ADJUST_IREDUCE_KN_RADIX

Syntax

`I_MPI_ADJUST_IREDUCE_KN_RADIX=<radix>`

Arguments

<radix>	An integer that specifies a radix used by the Knomial `MPI_Ireduce` algorithm to build a knomial communication tree
> 1	The default value is 4

Description

Set this environment variable together with `I_MPI_ADJUST_IREDUCE=3` to select the knomial tree radix for the corresponding `MPI_Ireduce` algorithm.

I_MPI_ADJUST_IGATHER_KN_RADIX

Syntax

`I_MPI_ADJUST_IGATHER_KN_RADIX=<radix>`

Arguments

<radix>	An integer that specifies a radix used by the Knomial `MPI_Igather` algorithm to build a knomial communication tree
> 1	The default value is 4

Description

Set this environment variable together with `I_MPI_ADJUST_IGATHER=2` to select the knomial tree radix for the corresponding `MPI_Igather` algorithm.

I_MPI_ADJUST_ISCATTER_KN_RADIX

Syntax

`I_MPI_ADJUST_ISCATTER_KN_RADIX=<radix>`

Arguments

<radix>	An integer that specifies a radix used by the Knomial `MPI_Iscatter` algorithm to build a knomial communication tree
> 1	The default value is 4

Description

Set this environment variable together with `I_MPI_ADJUST_ISCATTER=2` to select the knomial tree radix for the corresponding `MPI_Iscatter` algorithm.

I_MPI_ADJUST_<*COLLECTIVE*>_SHM_KN_RADIX

Syntax

`I_MPI_ADJUST_<COLLECTIVE>_SHM_KN_RADIX=<radix>`

Arguments

`<radix>`	An integer that specifies a radix used by the Knomial or Knary SHM-based algorithm to build a knomial or knary communication tree
`> 0`	• If you specify the environment variables `I_MPI_ADJUST_BCAST_SHM_KN_RADIX` and `I_MPI_ADJUST_BARRIER_SHM_KN_RADIX`, the default value is 3 • If you specify the environment variables `I_MPI_ADJUST_REDUCE_SHM_KN_RADIX` and `I_MPI_ADJUST_ALLREDUCE_SHM_KN_RADIX`, the default value is 4

Description

This environment variable includes the following variables:

- `I_MPI_ADJUST_BCAST_SHM_KN_RADIX`
- `I_MPI_ADJUST_BARRIER_SHM_KN_RADIX`
- `I_MPI_ADJUST_REDUCE_SHM_KN_RADIX`
- `I_MPI_ADJUST_ALLREDUCE_SHM_KN_RADIX`

Set this environment variable to select the knomial or knary tree radix for the corresponding tree SHM-based algorithms. When you build a knomial communication tree, the specified value is used as the power for 2 to generate resulting radix (2^`<radix>`). When you build a knary communication tree, the specified value is used for the radix.

I_MPI_COLL_INTRANODE

Syntax

`I_MPI_COLL_INTRANODE=<mode>`

Arguments

`<mode>`	Intranode collectives type
`pt2pt`	Use only point-to-point communication-based collectives
`shm`	Enables shared memory collectives. This is the default value

Description

Set this environment variable to switch intranode communication type for collective operations. If there is large set of communicators, you can switch off the SHM-collectives to avoid memory overconsumption.

I_MPI_COLL_INTRANODE_SHM_THRESHOLD

Syntax

`I_MPI_COLL_INTRANODE_SHM_THRESHOLD=<nbytes>`

Arguments

`<nbytes>`	Define the maximal data block size processed by shared memory collectives.
`> 0`	Use the specified size. The default value is 16384 bytes.

Description

Set this environment variable to define the size of shared memory area available for each rank for data placement. Messages greater than this value will *not* be processed by SHM-based collective operation, but will be processed by point-to-point based collective operation. The value must be a multiple of 4096.

I_MPI_ADJUST_GATHER_SEGMENT

Syntax

`I_MPI_ADJUST_GATHER_SEGMENT=<block_size>`

Arguments

`<block_size>`	Size of a message segment in bytes.
`> 0`	Use the specified size. The default value is 16384 bytes.

Description

Set an internal block size to control the `MPI_Gather` message segmentation for the binomial algorithm with segmentation.

3.6. Asynchronous Progress Control

Intel® MPI Library supports asynchronous progress model and can provide separate dedicated cores for asynchronous progress threads. This section describes the environment variables for the asynchronous progress control.

I_MPI_ASYNC_PROGRESS

Enable asynchronous progress. It is recommended to use `I_MPI_ASYNC_PROGRESS` instead of `MPICH_ASYNC_PROGRESS`.

Syntax

`I_MPI_ASYNC_PROGRESS=<arg>`

Arguments

`<arg>`	Binary indicator
`disable | no | off | 0`	Turn off asynchronous progress thread for each rank. This is the default value
`enable | yes | on | 1`	Enables asynchronous progress thread

Description

Set this environment variable to control the usage of separate progress threads.

I_MPI_ASYNC_PROGRESS_PIN

Control the helper threads pinning, such as asynchronous progress threads pinning. If you have not set `I_MPI_ASYNC_PROGRESS`, the configuration of `I_MPI_ASYNC_PROGRESS_PIN` is ignored.

Syntax

`I_MPI_ASYNC_PROGRESS_PIN=<arg>`

Arguments

<arg>	Binary indicator
`disable | no | off`	Turn off helper threads pinning. Helper threads inherit affinity mask from parent thread, which means the provided set of cores are used as affinity mask for all helper threads. This is the default value
`yes | auto`	Allow Hydra process manager to pin helper threads to the dedicated cores automatically. If you specify this value, Hydra process manager excludes one CPU from mask dedicated for the rank. This separate CPU is attached to the helper thread; meanwhile all other CPUs from initial affinity mask are used by MPI-rank
`<CPU list>`	Pin all existing progress threads to the listed CPUs

Description

Set this environment variable to control the usage of the helpers thread pinning.

4. Miscellaneous

4.1. Timer Control

I_MPI_TIMER_KIND

Select the timer used by the `MPI_Wtime` and `MPI_Wtick` calls.

Syntax

`I_MPI_TIMER_KIND=<timername>`

Arguments

<timername>	Define the timer type
`gettimeofday`	If this setting is chosen, the `MPI_Wtime` and `MPI_Wtick` functions will work through the function `gettimeofday(2)`. This is the default value
`rdtsc`	If this setting is chosen, the `MPI_Wtime` and `MPI_Wtick` functions will use the high resolution RDTSC timer

Description

Set this environment variable to select either the ordinary or RDTSC timer.

The resolution of the default `gettimeofday(2)` timer may be insufficient on certain platforms.

4.2. Compatibility Control

I_MPI_COMPATIBILITY

Select the runtime compatibility mode.

Syntax

`I_MPI_COMPATIBILITY=<value>`

Arguments

<value>	Define compatibility mode
`not defined`	The MPI-3.1 standard compatibility. This is the default mode
`3`	The Intel® MPI Library 3.x compatible mode
`4`	The Intel® MPI Library 4.0.x compatible mode

Description

Set this environment variable to choose the Intel® MPI Library runtime compatible mode. By default, the library complies with the MPI-3.1 standard. If your application depends on the MPI-2.1 behavior, set the value of the

environment variable `I_MPI_COMPATIBILITY` to `4`. If your application depends on the pre-MPI-2.1 behavior, set the value of the environment variable `I_MPI_COMPATIBILITY` to `3`.

4.3. Dynamic Process Support

Intel® MPI Library provides support for the MPI-2 process model that allows creation and cooperative termination of processes after an MPI application has started. It provides the following:

- A mechanism to establish communication between the newly created processes and the existing MPI application
- A process attachment mechanism to establish communication between two existing MPI applications even when one of them does not spawn the other

The default placement of the spawned processes uses round robin scheduling. The first spawned process is placed after the last process of the parent group. A specific communication fabric combination is selected using the usual fabrics selection algorithm (see `I_MPI_FABRICS` and `I_MPI_FABRICS_LIST` for details).

For example, to run a dynamic application, use the following command:

```
$ mpirun -n 1 -gwdir <path_to_executable> -f hosts -genv I_MPI_FABRICS=shm:tcp
<spawn_app>
```

In the example, `<spawn_app>` spawns 4 dynamic processes. If the `hosts` file contains the following information:

```
host1
host2
host3
host4
```

the original spawning process is placed on `host1`, while the dynamic processes are distributed as follows: 1 - on `host2`, 2 - on `host3`, 3 - on `host4`, and 4 - again on `host1`.

To run a client-server application, use the following commands on the intended server host:

```
$ mpirun -n 1 -genv I_MPI_FABRICS=shm:dapl <server_app> > <port_name>
```

and use the following commands on the intended client hosts:

```
$ mpirun -n 1 -genv I_MPI_FABRICS=shm:dapl <client_app> < <port_name>
```

To run a simple `MPI_COMM_JOIN` based application, use the following commands on the intended server host:

```
$ mpirun -n 1 -genv I_MPI_FABRICS=shm:ofa <join_server_app> < <port_number>
$ mpirun -n 1 -genv I_MPI_FABRICS=shm:ofa <join_client_app> < <port_number>
```

4.4. Intel® Many Integrated Core Architecture Support

This section describes the Intel® MPI Library environment variables related to the support of the Intel® Xeon Phi™ coprocessor (code named Knights Landing) based on the Intel® Many Integrated Core Architecture (Intel® MIC Architecture).

I_MPI_MIC

Syntax

`I_MPI_MIC=<value>`

Arguments

<value>	Binary indicator

`enable | yes | on | 1`	Enable the Intel Xeon Phi coprocessor recognition
`disable | no | off | 0`	Disables the Intel Xeon Phi coprocessor recognition. This is the default value

Description

Set this environment variable to control whether the Intel Xeon processor of the Intel® MPI Library tries to detect and work with the Intel® MIC Architecture components.

NOTE

This is a provisional variable and is only temporarily introduced, until the architecture detection and other related matters are clarified.

I_MPI_ENV_PREFIX_LIST

Define the prefixes of environment variables for the intended platforms.

Syntax

`I_MPI_ENV_PREFIX_LIST=<platform>:<prefix>[,<platform>:<prefix>][,...]`

Argument

`<platform>`	The intended platform (string). See `I_MPI_PLATFORM` for available values.
`<prefix>`	A prefix (string) for a name of an environment variable to be used for the intended platform

Description

Set this environment variable to define the prefix of environment variables to be used for the intended platform.

If you specify a prefix in `I_MPI_ENV_PREFIX_LIST` for a platform, the prefixed environment variable is used on the specified platform. The non-prefixed variable is used on all other platforms.

If you do not specify `I_MPI_ENV_PREFIX_LIST`, environment variables are applied to all platforms.

NOTE

Use the lower case when you specify the platform names.

Example

```
$ export I_MPI_ENV_PREFIX_LIST=knl:KNLONLY
$ export OMP_NUM_THREADS=2
$ export KNLONLY_OMP_NUM_THREADS=4
```

In the example above, the number of OpenMP* threads is set to two on all platforms. While for the Intel Xeon Phi processor code named Knights Landing the number of threads is set to four.

4.5. Fault Tolerance Support

Intel® MPI Library provides extra functionality to enable fault tolerance support in MPI applications. The MPI standard does not define behavior of MPI implementation if one or several processes of an MPI application are abnormally aborted. By default, Intel® MPI Library aborts the whole application if any process stops.

I_MPI_FAULT_CONTINUE

Turn on/off support for fault tolerant applications.

Syntax

`I_MPI_FAULT_CONTINUE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on support for fault tolerant applications
`disable | no | off | 0`	Turn off support for fault tolerant applications. This is default value

Description

Set this environment variable to enable an application to continue executing after one or several processes are stopped.

An application can continue working in the case of MPI processes an issue, if the issue meets the following requirements:

- The application sets error handler `MPI_ERRORS_RETURN` to communicator `MPI_COMM_WORLD` (all new communicators inherit error handler from it).
- The application uses master-slave model. In this case, the application is stopped only if the master is finished or does not respond.
- The application uses only point-to-point communication between a master and a number of slaves. It does not use inter-slave communication or MPI collective operations.
- Any communication operation can be used on a subset communicator system. If an error occurs in a collective operation, any communication inside this communicator will be prohibited.
- The fault tolerance functionality is not available for spawned processes.
- A certain MPI error code is used for a point-to-point operation with a failed slave rank. This makes the application avoid further communication with this rank. The slave rank can use blocking/non-blocking send, receive, probe and test.

For example, an application sets `MPI_ERRORS_RETURN` error handler and checks the return code after each communication call. If a communication call does not return `MPI_SUCCESS`, the destination process should be marked unreachable and exclude communication with it:

```
if(live_ranks[rank]) {
    mpi_err = MPI_Send(buf, count, dtype, rank, tag, MPI_COMM_WORLD);
    if(mpi_err != MPI_SUCCESS) {
        live_ranks[rank] = 0;
    }
}
```

In the case of non-blocking communications, errors can appear during wait/test operations.

4.6. Statistics Gathering Mode

Intel® MPI Library provides the built-in statistics gathering facility that provides essential information about MPI program execution. You can use the native or IPM statistics formats or both at once. See description of the environment variables controlling statistics collection below.

4.6.1. Native Statistics

To enable the native statistics collection, set `I_MPI_STATS` to `native` and specify the level of detail.

I_MPI_STATS

Control statistics collection.

Syntax

`I_MPI_STATS=[native:][n-]m`

Arguments

`n, m`	Possible statistics levels of the output information
`1`	Output the amount of data sent by each process
`2`	Output the number of calls and amount of transferred data
`3`	Output statistics combined according to the actual arguments
`4`	Output statistics defined by a buckets list
`10`	Output collective operation statistics for all communication contexts
`20`	Output additional time information for all MPI functions

Description

Set this environment variable to control the amount of statistics information collected and the output to the log file. No statistics are produced by default.

`n, m` are positive integer numbers and define the range of output information. The statistics from level `n` to level `m` inclusive are printed. If `n` is not provided, the default lower bound is `1`.

I_MPI_STATS_SCOPE

Select the subsystem(s) for which statistics should be collected.

Syntax

`I_MPI_STATS_SCOPE="<subsystem>[:<ops>][;<subsystem>[:<ops>][...]]"`

Arguments

`<subsystem>`	Define the target subsystem(s)
`all`	Collect statistics data for all operations. This is the default value
`coll`	Collect statistics data for all collective operations
`p2p`	Collect statistics data for all point-to-point operations

<ops>	Define the target operations as a comma separated list
Allgather	MPI_Allgather
Iallgather	MPI_Iallgather
Allgatherv	MPI_Allgatherv
Iallgatherv	MPI_Iallgatherv
Allreduce	MPI_Allreduce
Iallreduce	MPI_Iallreduce
Alltoall	MPI_Alltoall
Ialltoall	MPI_Ialltoall
Alltoallv	MPI_Alltoallv
Ialltoallv	MPI_Ialltoallv
Alltoallw	MPI_Alltoallw
Ialltoallw	MPI_Ialltoallw
Barrier	MPI_Barrier
Ibarrier	MPI_Ibarrier
Bcast	MPI_Bcast
Ibcast	MPI_Ibcast
Exscan	MPI_Exscan
Iexscan	MPI_Iexscan
Gather	MPI_Gather
Igather	MPI_Igather
Gatherv	MPI_Gatherv
Igatherv	MPI_Igatherv
Reduce_scatter	MPI_Reduce_scatter
Ireduce_scatter	MPI_Ireduce_scatter
Reduce	MPI_Reduce
Ireduce	MPI_Ireduce

`Scan`	`MPI_Scan`
`Iscan`	`MPI_Iscan`
`Scatter`	`MPI_Scatter`
`Iscatter`	`MPI_Iscatter`
`Scatterv`	`MPI_Scatterv`
`Iscatterv`	`MPI_Iscatterv`
`Send`	Standard transfers (`MPI_Send`, `MPI_Isend`, `MPI_Send_init`)
`Sendrecv`	Send-receive transfers (`MPI_Sendrecv`, `MPI_Sendrecv_replace`)
`Bsend`	Buffered transfers (`MPI_Bsend`, `MPI_Ibsend`, `MPI_Bsend_init`)
`Csend`	Point-to-point operations inside the collectives. This internal operation serves all collectives
`Csendrecv`	Point-to-point send-receive operations inside the collectives. This internal operation serves all collectives
`Rsend`	Ready transfers (`MPI_Rsend`, `MPI_Irsend`, `MPI_Rsend_init`)
`Ssend`	Synchronous transfers (`MPI_Ssend`, `MPI_Issend`, `MPI_Ssend_init`)

Description

Set this environment variable to select the target subsystem in which to collect statistics. All collective and point-to-point operations, including the point-to-point operations performed inside the collectives, are covered by default.

Examples

The default settings are equivalent to:
`I_MPI_STATS_SCOPE="coll;p2p"`

Use the following settings to collect statistics for `MPI_Bcast`, `MPI_Reduce`, and all point-to-point operations:
`I_MPI_STATS_SCOPE="p2p;coll:bcast,reduce"`

Use the following settings to collect statistics for the point-to-point operations inside the collectives:
`I_MPI_STATS_SCOPE=p2p:csend`

I_MPI_STATS_BUCKETS

Set the list of ranges for message sizes and communicator sizes that are used for collecting statistics.

Syntax

`I_MPI_STATS_BUCKETS=<msg>[@<proc>][,<msg>[@<proc>]]...`

Arguments

`<msg>`	Specify range of message sizes in bytes

<l>	Single value of message size
<l>-*<m>*	Range from *<l>* to *<m>*

<proc>	Specify range of processes (ranks) for collective operations
<p>	Single value of communicator size
<p>-*<q>*	Range from *<p>* to *<q>*

Description

Set the I_MPI_STATS_BUCKETS environment variable to define a set of ranges for message sizes and communicator sizes.

Level 4 of the statistics provides profile information for these ranges.

If I_MPI_STATS_BUCKETS environment variable is not used, then level 4 statistics is not gathered.

If a range is not specified, the maximum possible range is assumed.

Examples

To specify short messages (from 0 to 1000 bytes) and long messages (from 50000 to 100000 bytes), use the following setting:

```
-env I_MPI_STATS_BUCKETS 0-1000,50000-100000
```

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

```
-env I_MPI_STATS_BUCKETS "16@4"
```

NOTE

When the '@' symbol is present, the environment variable value must be enclosed in quotes.

I_MPI_STATS_FILE

Define the statistics output file name.

Syntax

I_MPI_STATS_FILE=*<name>*

Arguments

<name>	Define the statistics output file name

Description

Set this environment variable to define the statistics output file. By default, the stats.txt file is created in the current directory.

If this variable is not set and the statistics output file already exists, an index is appended to its name. For example, if stats.txt exists, the created statistics output file is named as stats(2).txt; if stats(2).txt exists, the created file is named as stats(3).txt, and so on.

Statistics Format

The statistics data is grouped and ordered according to the process ranks in the `MPI_COMM_WORLD` communicator. The timing data is presented in microseconds. For example, with the following settings:

```
$ export I MPI STATS=4
$ export I_MPI_STATS_SCOPE="p2p;coll:allreduce"
```

the statistics output for a simple program that performs only one `MPI_Allreduce` operation may look as follows:

```
____ MPI Communication Statistics ____
Stats level: 4
P2P scope:< FULL >
Collectives scope:< Allreduce >
~~~~ Process 0 of 2 on node svlmpihead01 lifetime = 414.13
Data Transfers
Src Dst Amount(MB) Transfers
---------------------------------------
000 --> 000 0.000000e+00 0
000 --> 001 7.629395e-06 2
=======================================
Totals 7.629395e-06 2
Communication Activity
Operation Volume(MB) Calls
---------------------------------------
P2P
Csend 7.629395e-06 2
Csendrecv 0.000000e+00 0
Send 0.000000e+00 0
Sendrecv 0.000000e+00 0
Bsend 0.000000e+00 0
Rsend 0.000000e+00 0
Ssend 0.000000e+00 0
Collectives
Allreduce 7.629395e-06 2
=======================================
Communication Activity by actual args
P2P
Operation Dst Message size Calls
-----------------------------------------
Csend
1 1 4 2
Collectives
Operation Context Algo Comm size Message size Calls Cost(%)
-----------------------------------------------------------------------------------
--
Allreduce
1 0 1 2 4 2 44.96
===================================================================================
~~~~ Process 1 of 2 on node svlmpihead01 lifetime = 306.13
Data Transfers
Src Dst Amount(MB) Transfers
---------------------------------------
001 --> 000 7.629395e-06 2
001 --> 001 0.000000e+00 0
=======================================
Totals 7.629395e-06 2
Communication Activity
Operation Volume(MB) Calls
```

```
----------------------------------------
P2P
Csend 7.629395e-06 2
Csendrecv 0.000000e+00 0
Send 0.000000e+00 0
Sendrecv 0.000000e+00 0
Bsend 0.000000e+00 0
Rsend 0.000000e+00 0
Ssend 0.000000e+00 0
Collectives
Allreduce 7.629395e-06 2
========================================
Communication Activity by actual args
P2P
Operation Dst Message size Calls
-----------------------------------------
Csend
1 0 4 2
Collectives
Operation Context Comm size Message size Calls Cost(%)
-------------------------------------------------------------------------
Allreduce
1 0 2 4 2 37.93
=========================================================================
____ End of stats.txt file ____
```

In the example above:

- All times are measured in microseconds.
- The message sizes are counted in bytes. **MB** means megabyte equal to 2^{20} or 1 048 576 bytes.
- The process life time is calculated as a stretch of time between `MPI_Init` and `MPI_Finalize`.
- The **Algo** field indicates the number of algorithm used by this operation with listed arguments.
- The **Cost** field represents a particular collective operation execution time as a percentage of the process life time.

4.6.2. IPM Statistics

To enable the integrated performance monitoring (IPM) statistics collection, set `I_MPI_STATS` to `ipm` or `ipm:terse`.

The `I_MPI_STATS_BUCKETS` environment variable is not applicable for the IPM format. The `I_MPI_STATS_ACCURACY` environment variable is available to control extra functionality.

I_MPI_STATS

Control the statistics data output format.

Syntax

`I_MPI_STATS=<level>`

Arguments

<level>	Level of statistics data
`ipm`	Summary data throughout all regions

`ipm:terse`	Basic summary data

Description

Set this environment variable to `ipm` to get the statistics output that contains region summary. Set this environment variable to `ipm:terse` argument to get the brief statistics output.

I_MPI_STATS_FILE

Define the output file name.

Syntax

`I_MPI_STATS_FILE=<name>`

Argument

<name>	File name for statistics data gathering

Description

Set this environment variable to change the statistics output file name from the default name of `stats.ipm`.

If this variable is not set and the statistics output file already exists, an index is appended to its name. For example, if `stats.ipm` exists, the created statistics output file is named as `stats(2).ipm`; if `stats(2).ipm` exists, the created file is named as `stats(3).ipm`, and so on.

I_MPI_STATS_SCOPE

Define a semicolon separated list of subsets of MPI functions for statistics gathering.

Syntax

`I_MPI_STATS_SCOPE="<subset>[;<subset>[;…]]"`

Argument

<subset>	Target subset
`all2all`	Collect statistics data for all-to-all functions types
`all2one`	Collect statistics data for all-to-one functions types
`attr`	Collect statistics data for attribute control functions
`comm`	Collect statistics data for communicator control functions
`err`	Collect statistics data for error handling functions
`group`	Collect statistics data for group support functions
`init`	Collect statistics data for initialize/finalize functions
`io`	Collect statistics data for input/output support function

`one2all`	Collect statistics data for one-to-all functions types
`recv`	Collect statistics data for receive functions
`req`	Collect statistics data for request support functions
`rma`	Collect statistics data for one sided communication functions
`scan`	Collect statistics data for scan collective functions
`send`	Collect statistics data for send functions
`sendrecv`	Collect statistics data for send/receive functions
`serv`	Collect statistics data for additional service functions
`spawn`	Collect statistics data for dynamic process functions
`status`	Collect statistics data for status control function
`sync`	Collect statistics data for barrier synchronization
`time`	Collect statistics data for timing support functions
`topo`	Collect statistics data for topology support functions
`type`	Collect statistics data for data type support functions

Description

Use this environment variable to define a subset or subsets of MPI functions for statistics gathering specified by the following table. A union of all subsets is used by default.

Table 4.6-1 Stats Subsets of MPI Functions

all2all	**recv**
MPI_Allgather	MPI_Recv
MPI_Allgatherv	MPI_Irecv
MPI_Allreduce	MPI_Recv_init
MPI_Alltoll	MPI_Probe
MPI_Alltoallv	MPI_Iprobe
MPI_Alltoallw	**req**
MPI_Reduce_scatter	MPI_Start
MPI_Iallgather	MPI_Startall
MPI_Iallgatherv	MPI_Wait
MPI_Iallreduce	MPI_Waitall

MPI_Ialltoll	MPI_Waitany
MPI_Ialltoallv	MPI_Waitsome
MPI_Ialltoallw	MPI_Test
MPI_Ireduce_scatter	MPI_Testall
MPI_Ireduce_scatter_block	MPI_Testany
all2one	MPI_Testsome
MPI_Gather	MPI_Cancel
MPI_Gatherv	MPI_Grequest_start
MPI_Reduce	MPI_Grequest_complete
MPI_Igather	MPI_Request_get_status
MPI_Igatherv	MPI_Request_free
MPI_Ireduce	**rma**
attr	MPI_Accumulate
MPI_Comm_create_keyval	MPI_Get
MPI_Comm_delete_attr	MPI_Put
MPI_Comm_free_keyval	MPI_Win_complete
MPI_Comm_get_attr	MPI_Win_create
MPI_Comm_set_attr	MPI_Win_fence
MPI_Comm_get_name	MPI_Win_free
MPI_Comm_set_name	MPI_Win_get_group
MPI_Type_create_keyval	MPI_Win_lock
MPI_Type_delete_attr	MPI_Win_post
MPI_Type_free_keyval	MPI_Win_start
MPI_Type_get_attr	MPI_Win_test
MPI_Type_get_name	MPI_Win_unlock
MPI_Type_set_attr	MPI_Win_wait
MPI_Type_set_name	MPI_Win_allocate
MPI_Win_create_keyval	MPI_Win_allocate_shared
MPI_Win_delete_attr	MPI_Win_create_dynamic
MPI_Win_free_keyval	MPI_Win_shared_query
MPI_Win_get_attr	MPI_Win_attach
MPI_Win_get_name	MPI_Win_detach
MPI_Win_set_attr	MPI_Win_set_info
MPI_Win_set_name	MPI_Win_get_info
MPI_Get_processor_name	MPI_Win_get_accumulate
comm	MPI_Win_fetch_and_op
MPI_Comm_compare	MPI_Win_compare_and_swap
MPI_Comm_create	MPI_Rput

MPI_Comm_dup	MPI_Rget
MPI_Comm_free	MPI_Raccumulate
MPI_Comm_get_name	MPI_Rget_accumulate
MPI_Comm_group	MPI_Win_lock_all
MPI_Comm_rank	MPI_Win_unlock_all
MPI_Comm_remote_group	MPI_Win_flush
MPI_Comm_remote_size	MPI_Win_flush_all
MPI_Comm_set_name	MPI_Win_flush_local
MPI_Comm_size	MPI_Win_flush_local_all
MPI_Comm_split	MPI_Win_sync
MPI_Comm_test_inter	**scan**
MPI_Intercomm_create	MPI_Exscan
MPI_Intercomm_merge	MPI_Scan
err	MPI_Iexscan
MPI_Add_error_class	MPI_Iscan
MPI_Add_error_code	**send**
MPI_Add_error_string	MPI_Send
MPI_Comm_call_errhandler	MPI_Bsend
MPI_Comm_create_errhandler	MPI_Rsend
MPI_Comm_get_errhandler	MPI_Ssend
MPI_Comm_set_errhandler	MPI_Isend
MPI_Errhandler_free	MPI_Ibsend
MPI_Error_class	MPI_Irsend
MPI_Error_string	MPI_Issend
MPI_File_call_errhandler	MPI_Send_init
MPI_File_create_errhandler	MPI_Bsend_init
MPI_File_get_errhandler	MPI_Rsend_init
MPI_File_set_errhandler	MPI_Ssend_init
MPI_Win_call_errhandler	**sendrecv**
MPI_Win_create_errhandler	MPI_Sendrecv
MPI_Win_get_errhandler	MPI_Sendrecv_replace
MPI_Win_set_errhandler	**serv**
group	MPI_Alloc_mem
MPI_Group_compare	MPI_Free_mem
MPI_Group_difference	MPI_Buffer_attach
MPI_Group_excl	MPI_Buffer_detach
MPI_Group_free	MPI_Op_create
MPI_Group_incl	MPI_Op_free

MPI_Group_intersection	**spawn**
MPI_Group_range_excl	MPI_Close_port
MPI_Group_range_incl	MPI_Comm_accept
MPI_Group_rank	MPI_Comm_connect
MPI_Group_size	MPI_Comm_disconnect
MPI_Group_translate_ranks	MPI_Comm_get_parent
MPI_Group_union	MPI_Comm_join
init	MPI_Comm_spawn
MPI_Init	MPI_Comm_spawn_multiple
MPI_Init_thread	MPI_Lookup_name
MPI_Finalize	MPI_Open_port
io	MPI_Publish_name
MPI_File_close	MPI_Unpublish_name
MPI_File_delete	**status**
MPI_File_get_amode	MPI_Get_count
MPI_File_get_atomicity	MPI_Status_set_elements
MPI_File_get_byte_offset	MPI_Status_set_cancelled
MPI_File_get_group	MPI_Test_cancelled
MPI_File_get_info	**sync**
MPI_File_get_position	MPI_Barrier
MPI_File_get_position_shared	MPI_Ibarrier
MPI_File_get_size	**time**
MPI_File_get_type_extent	MPI_Wtick
MPI_File_get_view	MPI_Wtime
MPI_File_iread_at	**topo**
MPI_File_iread	MPI_Cart_coords
MPI_File_iread_shared	MPI_Cart_create
MPI_File_iwrite_at	MPI_Cart_get
MPI_File_iwrite	MPI_Cart_map
MPI_File_iwrite_shared	MPI_Cart_rank
MPI_File_open	MPI_Cart_shift
MPI_File_preallocate	MPI_Cart_sub
MPI_File_read_all_begin	MPI_Cartdim_get
MPI_File_read_all_end	MPI_Dims_create
MPI_File_read_all	MPI_Graph_create
MPI_File_read_at_all_begin	MPI_Graph_get
MPI_File_read_at_all_end	MPI_Graph_map
MPI_File_read_at_all	MPI_Graph_neighbors

MPI_File_read_at	MPI_Graphdims_get
MPI_File_read	MPI_Graph_neighbors_count
MPI_File_read_ordered_begin	MPI_Topo_test
MPI_File_read_ordered_end	**type**
MPI_File_read_ordered	MPI_Get_address
MPI_File_read_shared	MPI_Get_elements
MPI_File_seek	MPI_Pack
MPI_File_seek_shared	MPI_Pack_external
MPI_File_set_atomicity	MPI_Pack_external_size
MPI_File_set_info	MPI_Pack_size
MPI_File_set_size	MPI_Type_commit
MPI_File_set_view	MPI_Type_contiguous
MPI_File_sync	MPI_Type_create_darray
MPI_File_write_all_begin	MPI_Type_create_hindexed
MPI_File_write_all_end	MPI_Type_create_hvector
MPI_File_write_all	MPI_Type_create_indexed_block
MPI_File_write_at_all_begin	MPI_Type_create_resized
MPI_File_write_at_all_end	MPI_Type_create_struct
MPI_File_write_at_all	MPI_Type_create_subarray
MPI_File_write_at	MPI_Type_dup
MPI_File_write	MPI_Type_free
MPI_File_write_ordered_begin	MPI_Type_get_contents
MPI_File_write_ordered_end	MPI_Type_get_envelope
MPI_File_write_ordered	MPI_Type_get_extent
MPI_File_write_shared	MPI_Type_get_true_extent
MPI_Register_datarep	MPI_Type_indexed
one2all	MPI_Type_size
MPI_Bcast	MPI_Type_vector
MPI_Scatter	MPI_Unpack_external
MPI_Scatterv	MPI_Unpack
MPI_Ibcast	
MPI_Iscatter	
MPI_Iscatterv	

I_MPI_STATS_ACCURACY

Use the `I_MPI_STATS_ACCURACY` environment variable to reduce statistics output.

Syntax

`I_MPI_STATS_ACCURACY=<percentage>`

Argument

<percentage>	Float threshold value

Description

Set this environment variable to collect data only on those MPI functions that take the specified portion of the total time spent inside all MPI calls (in percent).

Examples

The following code example represents a simple application with IPM statistics collection enabled:

```
int main (int argc, char *argv[])
{
    int i, rank, size, nsend, nrecv;
    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    nsend = rank;
    MPI_Wtime();
    for (i = 0; i < 200; i++)
    {
        MPI_Barrier(MPI_COMM_WORLD);
    }
    /* open "reduce" region for all processes */
    MPI_Pcontrol(1, "reduce");
    for (i = 0; i < 1000; i++)
        MPI_Reduce(&nsend, &nrecv, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
    /* close "reduce" region */
    MPI_Pcontrol(-1, "reduce");
    if (rank == 0)
    {
        /* "send" region for the 0th process only */
        MPI_Pcontrol(1, "send");
        MPI_Send(&nsend, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
        MPI_Pcontrol(-1, "send");
    }
    if (rank == 1)
    {
        MPI_Recv(&nrecv, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    /* reopen "reduce" region */
    MPI_Pcontrol(1, "reduce");
    for (i = 0; i < 1000; i++)
        MPI_Reduce(&nsend, &nrecv, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
    MPI_Wtime();
    MPI_Finalize();
    return 0;
}
```

Command:

```
$ mpirun -n 4 -env I_MPI_STATS=ipm:terse ./a.out
```

Stats output:

```
################################################################################
#
# command : ./a.out (completed)
# host : node01/x86_64_Linux mpi_tasks : 4 on 1 nodes
# start : 05/25/11/05:44:13 wallclock : 0.092012 sec
# stop : 05/25/11/05:44:13 %comm : 98.94
# gbytes : 0.00000e+00 total gflop/sec : NA
```

```
#
################################################################################
```

Command:

```
$ mpirun -n 4 -env I_MPI_STATS=ipm ./a.out
```

Stats output:

```
################################################################################
#
# command : ./a.out (completed)
# host : node01/x86_64_Linux mpi_tasks : 4 on 1 nodes
# start : 05/25/11/05:44:13 wallclock : 0.092012 sec
# stop : 05/25/11/05:44:13 %comm : 98.94
# gbytes : 0.00000e+00 total gflop/sec : NA
#
################################################################################
# region : * [ntasks] = 4
#
# [total] <avg> min max
# entries 4 1 1 1
# wallclock 0.332877 0.0832192 0.0732641 0.0920119
# user 0.047992 0.011998 0.006999 0.019996
# system 0.013997 0.00349925 0.002999 0.004
# mpi 0.329348 0.082337 0.0723064 0.0912335
# %comm 98.9398 98.6928 99.154
# gflop/sec NA NA NA NA
# gbytes 0 0 0 0
#
#
# [time] [calls] <%mpi> <%wall>
# MPI_Init 0.236192 4 71.71 70.95
# MPI_Reduce 0.0608737 8000 18.48 18.29
# MPI_Barrier 0.027415 800 8.32 8.24
# MPI_Recv 0.00483489 1 1.47 1.45
# MPI_Send 1.50204e-05 1 0.00 0.00
# MPI_Wtime 1.21593e-05 8 0.00 0.00
# MPI_Finalize 3.33786e-06 4 0.00 0.00
# MPI_Comm_rank 1.90735e-06 4 0.00 0.00
# MPI_TOTAL 0.329348 8822 100.00 98.94
################################################################################
# region : reduce [ntasks] = 4
#
# [total] <avg> min max
# entries 8 2 2 2
# wallclock 0.0638561 0.015964 0.00714302 0.0238571
# user 0.034994 0.0087485 0.003999 0.015997
# system 0.003999 0.00099975 0 0.002999
# mpi 0.0608799 0.01522 0.00633883 0.0231845
# %comm 95.3392 88.7417 97.1808
# gflop/sec NA NA NA NA
# gbytes 0 0 0 0
#
#
# [time] [calls] <%mpi> <%wall>
# MPI_Reduce 0.0608737 8000 99.99 95.33
# MPI_Finalize 3.33786e-06 4 0.01 0.01
# MPI_Wtime 2.86102e-06 4 0.00 0.00
# MPI_TOTAL 0.0608799 8008 100.00 95.34
################################################################################
```

```
# region : send [ntasks] = 4
#
# [total] <avg> min max
# entries 1 0 0 1
# wallclock 2.89876e-05 7.24691e-06 1e-06 2.59876e-05
# user 0 0 0 0
# system 0 0 0 0
# mpi 1.50204e-05 3.75509e-06 0 1.50204e-05
# %comm 51.8165 0 57.7982
# gflop/sec NA NA NA NA
# gbytes 0 0 0 0
#
#
# [time] [calls] <%mpi> <%wall>
# MPI_Send 1.50204e-05 1 100.00 51.82
##############################################################################
# region : ipm_noregion [ntasks] = 4
#
# [total] <avg> min max
# entries 13 3 3 4
# wallclock 0.26898 0.0672451 0.0661182 0.068152
# user 0.012998 0.0032495 0.001 0.004999
# system 0.009998 0.0024995 0 0.004
# mpi 0.268453 0.0671132 0.0659676 0.068049
# %comm 99.8039 99.7721 99.8489
# gflop/sec NA NA NA NA
# gbytes 0 0 0 0
#
#
# [time] [calls] <%mpi> <%wall>
# MPI_Init 0.236192 4 87.98 87.81
# MPI_Barrier 0.027415 800 10.21 10.19
# MPI_Recv 0.00483489 1 1.80 1.80
# MPI_Wtime 9.29832e-06 4 0.00 0.00
# MPI_Comm_rank 1.90735e-06 4 0.00 0.00
# MPI_TOTAL 0.268453 813 100.00 99.80
##############################################################################
```

4.6.3. Native and IPM Statistics

The statistics in each supported format can be collected separately. To collect statistics in all formats with the maximal level of details, use the `I_MPI_STATS` environment variable as follows:

```
I_MPI_STATS=all
```

NOTE

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

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

```
I_MPI_STATS=[native:][n-]m,ipm[:terse]
```

4.7. ILP64 Support

The term ILP64 means that integer, long, and pointer data entities all occupy 8 bytes. This differs from the more conventional LP64 model in which only long and pointer data entities occupy 8 bytes while integer entities occupy 4 bytes. More information on the historical background and the programming model philosophy can be found, for example, in http://www.unix.org/version2/whatsnew/lp64_wp.html

Intel® MPI Library provides support for the ILP64 model for Fortran applications. To enable the ILP64 mode, do the following:

- Use the Fortran compiler wrapper option `-i8` for separate compilation and the `-ilp64` option for separate linkage. For example:

```
$ mpiifort -i8 -c test.f
$ mpiifort -ilp64 -o test test.o
```

- For simple programs, use the Fortran compiler wrapper option `-i8` for compilation and linkage. Specifying `-i8` will automatically assume the ILP64 library. For example:

```
$ mpiifort -i8 test.f
```

- When running the application, use the `-ilp64` option to preload the ILP64 interface. For example:

```
$ mpirun -ilp64 -n 2 ./myprog
```

4.7.1. Known Issues and Limitations

- Data type counts and other arguments with values larger than 2^{31} - 1 are not supported.
- Special MPI types `MPI_FLOAT_INT`, `MPI_DOUBLE_INT`, `MPI_LONG_INT`, `MPI_SHORT_INT`, `MPI_2INT`, `MPI_LONG_DOUBLE_INT`, `MPI_2INTEGER` are not changed and still use a 4-byte integer field.
- Predefined communicator attributes `MPI_APPNUM`, `MPI_HOST`, `MPI_IO`, `MPI_LASTUSEDCODE`, `MPI_TAG_UB`, `MPI_UNIVERSE_SIZE`, and `MPI_WTIME_IS_GLOBAL` are returned by the functions `MPI_GET_ATTR` and `MPI_COMM_GET_ATTR` as 4-byte integers. The same holds for the predefined attributes that may be attached to the window and file objects.
- Do not use the `-i8` option to compile MPI callback functions, such as error handling functions, or user-defined reduction operations.
- Do not use the `-i8` option with the deprecated functions that store or retrieve the 4-byte integer attribute (for example, `MPI_ATTR_GET`, `MPI_ATTR_PUT`, etc.). Use their recommended alternatives instead (`MPI_COMM_GET_ATTR`, `MPI_COMM_SET_ATTR`, etc).
- If you want to use the Intel® Trace Collector with the Intel MPI Library ILP64 executable files, you must use a special Intel Trace Collector library. If necessary, the `mpiifort` compiler wrapper will select the correct Intel Trace Collector library automatically.
- There is currently no support for C and C++ applications.

4.8. Unified Memory Management

Intel® MPI Library provides a way to replace the memory management subsystem by a user-defined package. You may optionally set the following function pointers:

- `i_malloc`
- `i_calloc`
- `i_realloc`

- `i_free`

These pointers also affect the C++ `new` and `delete` operators. The respective standard C library functions are used by default.

The following contrived source code snippet illustrates the usage of the unified memory subsystem:

```
#include <i_malloc.h>
#include <my_malloc.h>
int main( int argc, int argv )
{
    // override normal pointers
    i_malloc = my_malloc;
    i_calloc = my_calloc;
    i_realloc = my_realloc;
    i_free = my_free;
    #ifdef  WIN32
    // also override pointers used by DLLs
    i_malloc_dll = my_malloc;
    i_calloc_dll = my_calloc;
    i_realloc_dll = my_realloc;
    i_free_dll = my_free;
    #endif
   // now start using Intel(R) libraries
}
```

4.9. File System Support

Intel® MPI Library provides loadable shared modules to provide native support for the following file systems:

- Panasas* ActiveScale* File System (PanFS)
- Lustre* File System
- IBM* General Parallel File System* (GPFS*)

Set the `I_MPI_EXTRA_FILESYSTEM` environment variable to `on` to enable parallel file system support. Set the `I_MPI_EXTRA_FILESYSTEM_LIST` environment variable to request native support for the specific file system. For example, to request native support for Panasas* ActiveScale* File System, do the following:

```
$ mpirun -env I_MPI_EXTRA_FILESYSTEM=on -env I_MPI_EXTRA_FILESYSTEM_LIST=panfs -n 2
./test
```

I_MPI_EXTRA_FILESYSTEM

Turn on/off native parallel file systems support.

Syntax

`I_MPI_EXTRA_FILESYSTEM=`*`<arg>`*

Arguments

`<arg>`	Binary indicator
`enable | yes | on | 1`	Enable native support for parallel file systems.
`disable | no | off | 0`	Disable native support for parallel file systems. This is the default value.

Description

Set this environment variable to enable parallel file system support. The `I_MPI_EXTRA_FILESYSTEM_LIST` environment variable must be set to request native support for the specified file system.

I_MPI_EXTRA_FILESYSTEM_LIST

Select specific file systems support.

Syntax

`I_MPI_EXTRA_FILESYSTEM_LIST=<fs>[,<fs>,...,<fs>]`

Arguments

<fs>	Define a target file system
`panfs`	Panasas* ActiveScale* File System
`lustre`	Lustre* File System
`gpfs`	IBM* General Parallel File System* (GPFS*)

Description

Set this environment variable to request support for the specified parallel file system. This environment variable is handled only if `I_MPI_EXTRA_FYLESYSTEM` is enabled. Intel® MPI Library will try to load shared modules to support the file systems specified by `I_MPI_EXTRA_FILESYSTEM_LIST`.

I_MPI_LUSTRE_STRIPE_AWARE

Enable/disable an alternative algorithm for MPI-IO collective read operation on the Lustre* file system.

Syntax

`I_MPI_LUSTRE_STRIPE_AWARE=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Enable the stripe-aware algorithm.
`disable | no | off | 0`	Use the default algorithm. This is the default value.

Description

By default, when ROMIO* collective buffering is enabled, Intel® MPI Library uses the following algorithm for MPI-IO collective reading on the Lustre* file system: a single rank is selected on each computation node for reading data from the file system and then redistributing the data among its local peers. However, this may lead to I/O contention, if several I/O processes access the same object storage target (OST) at the same time. The new algorithm limits the number of I/O processes to the number of OSTs that contain the data, when this information is available. The algorithm also ensures that the single I/O rank is communicating with one OST at most.

NOTE

The default algorithm may still be more efficient in some cases, for example when a single I/O process cannot saturate the bandwidth of a single OST.

Using this algorithm requires that you also do the following:

- Set the ROMIO settings `striping_unit` and `striping_factor` according to the layout of the file read.
- Set the environment variables: `I_MPI_EXTRA_FILESYSTEM=on`, `I_MPI_EXTRA_FILESYSTEM_LIST=lustre`.

4.10. Multi-threaded memcpy Support

This topic provides information on how to use a multi-threaded version of `memcpy` implemented in the Intel® MPI Library for Intel® Xeon Phi™ Coprocessors. You can use this experimental feature to reach higher memory bandwidth between the ranks communicated through shared memory for some applications.

I_MPI_MT_MEMCPY

Controls usage of the multi-threaded `memcpy`.

Syntax

```
I_MPI_MT_MEMCPY=<value>
```

Arguments

<value>	Controls the usage of the multi-threaded `memcpy`
`enable | yes | on | 1`	Enable the multi-threaded `memcpy` in the single threaded version of the Intel® MPI Library (`MPI_THREAD_SINGLE`). This configuration is ignored for the thread safe version of Intel® MPI Library
`disable | no | off | 0`	Disable the usage of the multi-threaded `memcpy`. This is the default value

Description

Set this environment variable to control whether to use multi-threaded version of `memcpy` for intra-node communication.

I_MPI_MT_MEMCPY_NUM_THREADS

Change the number of threads involved in performing multi-threaded `memcpy`.

Syntax

```
I_MPI_MT_MEMCPY_NUM_THREADS=<num>
```

Arguments

<num>	The number of threads involved in performing multi-threaded `memcpy`
`>0`	The default value is the lesser of 8 and the number of physical cores within the MPI process pinning domain

Description

Use this environment variable to set the number of threads which perform `memcpy` operations per each MPI rank. The value `1` is equivalent to the setting `I_MPI_MT_MEMCPY=disable`.

I_MPI_MT_MEMCPY_THRESHOLD

Change the threshold for using multi-threaded memcpy.

Syntax

`I_MPI_MT_MEMCPY_THRESHOLD=<nbytes>`

Arguments

<nbytes>	Define the multi-threaded `memcpy` threshold in bytes
`>0`	The default value is `32768`

Description

Set this environment variable to control the threshold for using multi-threaded `memcpy`. If the threshold is larger than the shared memory buffer size (for example, see `I_MPI_SHM_LMT_BUFFER_SIZE`), multi-threaded `memcpy` will never be used. The usage of multi-threaded `memcpy` is selected according to the following scheme:

- Buffers shorter than or equal to `<nbytes>` are sent using the serial version of `memcpy`. This approach is faster for short and medium buffers.
- Buffers larger than `<nbytes>` are sent using the multi-threaded `memcpy`. This approach is faster for large buffers.

I_MPI_MT_MEMCPY_SPIN_COUNT

Control the spin count value.

Syntax

`I_MPI_MT_MEMCPY_SPIN_COUNT=<scount>`

Arguments

<scount>	Define the loop spin count when a thread waits for data to copy before sleeping
`>0`	The default value is equal to `100000`. The maximum value is equal to `2147483647`

Description

Set the spin count limit for the loop for waiting for data to be copied by the thread. When the limit is exceeded and there is no data to copy, the thread goes to sleep.

Use the `I_MPI_MT_MEMCPY_SPIN_COUNT` environment variable for tuning application performance. The best value for `<scount>` can be chosen on an experimental basis. It depends on the particular computational environment and application.

4.11. Java* Bindings for MPI-2 Routines

Intel® MPI Library supports the experimental feature that provides Java bindings for a subset of MPI-2 routines. You can see the available routines listed in the table below. All the classes below belong to the `mpi` package.

NOTE

- For static methods, parameters fully correspond to the ones of C routines.

- For non-static methods, the object that calls the method corresponds to the `OUT` parameter of the original C routine.

Table 4.11-1 Java* Bindings for MPI-2 Routines

Java Class	Public Fields and Methods	Original C Routine
MPI	`static int Init(String[] args)`	`MPI_Init`
	`static void Finalize()`	`MPI_Finalize`
	`static double wTime()`	`MPI_Wtime`
	`static void abort(Comm comm, int errorCode)`	`MPI_Abort`
	`String getProcessorName()`	`MPI_Get_processor_name`
Aint	`static void getExtent(Datatype dt, Aint lb, Aint extent)`	`MPI_Type_get_extent`
	`static void getTrueExtent(Datatype dt, Aint true_lb, Aint true_extent)`	`MPI_Type_get_true_extent`
	`static void getAddress(long location, Aint address)`	`MPI_Get_address`
	`static void getContents(Datatype dt, int maxIntegers, int maxAddresses, int maxDatatypes, int[] integers, Aint[] addresses, Datatype[] datatypes)`	`MPI_Type_get_contents`
Collective	`static void allToAll(Object sendbuf, int sendcount, Datatype sendtype, Object recvbuf, int recvcount, Datatype recvtype, Comm comm)`	`MPI_Alltoall`
	`static void reduce(Object sendbuf, Object recvbuf, int count, Datatype type, Op op, int root, Comm comm)`	`MPI_Reduce`
	`static void bcast(Object buffer, int count, Datatype type, int root, Comm comm)`	`MPI_Bcast`
	`static void gather(Object sendBuffer, int sendCount, Datatype sendType, Object recvBuffer, int recvCount, Datatype recvType, int root, Comm comm)`	`MPI_Gather`
	`static void gatherv(Object sendBuffer, int sendCount, Datatype sendType, Object recvBuffer, Object recvCount, Object displs, Datatype recvType, int root, Comm`	`MPI_Gatherv`

	comm)	
	static void allGather(Object sendBuffer, int sendCount, Datatype sendType, Object recvBuffer, int recvCount, Datatype recvType, Comm comm)	MPI_Allgather
	static void allGatherv(Object sendBuffer, int sendCount, Datatype sendType, Object recvBuffer, Object recvCount, Object displs, Datatype recvType, Comm comm)	MPI_Allgatherv
	static void allReduce(Object sendbuf, Object recvbuf, int count, Datatype type, Op op, Comm comm)	MPI_Allreduce
	static void allToAllv(Object sendbuf, Object sendCount, Object sdispls, Datatype sendType, Object recvbuf, Object recvCount, Object rdispls, Datatype recvType, Comm comm)	MPI_Alltoallv
	static void reduceScatter(Object sendbuf, Object recvbuf, Object recvcounts, Datatype type, Op op, Comm comm)	MPI_Reduce_scatter
	static void scatter(Object sendBuffer, int sendCount, Datatype sendType, Object recvBuffer, int recvCount, Datatype recvType, int root, Comm comm)	MPI_Scatter
	static void scatterv(Object sendBuffer, Object sendCount, Object displs, Datatype sendType, Object recvBuffer, int recvCount, Datatype recvType, int root, Comm comm)	MPI_Scatterv
	static void barrier(Comm comm)	MPI_Barrier
Comm	Static field: Comm WORLD	MPI_COMM_WORLD
	Static field: Comm SELF	MPI_COMM_SELF
	int getSize()	MPI_Comm_size
	int getRank()	MPI_Comm_rank
	Comm create(Group group)	MPI_Comm_create
	static Comm create(Comm comm, Group group)	MPI_Comm_create
	Comm dup()	MPI_Comm_dup

	`Comm split(int color, int key)`	`MPI_Comm_split`
`Group`	Static field: `int MPI_PROC_NULL`	`MPI_PROC_NULL`
	Static field: `int MPI_IDENT`	`MPI_IDENT`
	Static field: `int MPI_CONGRUENT`	`MPI_CONGRUENT`
	Static field: `int MPI_SIMILAR`	`MPI_SIMILAR`
	Static field: `int MPI_UNEQUAL`	`MPI_UNEQUAL`
	Static field: `Group WORLD`	`MPI_GROUP_WORLD`
	`void group(Comm comm)`	`MPI_Comm_group`
	`int getSize()`	`MPI_Group_size`
	`int getRank()`	`MPI_Group_rank`
	`int MPI_Group_translate_ranks(int[] ranks1, Group group2, int[] ranks2)`	`MPI_Group_translate_ranks`
	`static int MPI_Group_translate_ranks(Group group1, int[] ranks1, Group group2, int[] ranks2)`	`MPI_Group_translate_ranks`
	`int MPI_Group_compare(Group group2)`	`MPI_Group_compare`
	`int MPI_Group_union(Group group1, Group group2)`	`MPI_Group_union`
	`int MPI_Group_intersection(Group group1, Group group2)`	`MPI_Group_intersection`
	`int MPI_Group_difference(Group group1, Group group2)`	`MPI_Group_difference`
	`int MPI_Group_incl(Group group, int n, int[] ranks)`	`MPI_Group_incl`
	`int MPI_Group_excl(Group group, int n, int[] ranks)`	`MPI_Group_excl`
`Datatype`	Static field: `Datatype NULL`	`MPI_DATATYPE_NULL`
	Static field: `Datatype BYTE`	`MPI_UINT8_T`
	Static field: `Datatype CHAR`	`MPI_CHAR`
	Static field: `Datatype SHORT`	`MPI_INT16_T`
	Static field: `Datatype BOOLEAN`	`MPI_UINT8_T`

	Static field: `Datatype INT`	`MPI_INT32_T`
	Static field: `Datatype LONG`	`MPI_INT64_T`
	Static field: `Datatype FLOAT`	`MPI_FLOAT`
	Static field: `Datatype DOUBLE`	`MPI_DOUBLE`
	Static field: `Datatype PACKED`	`MPI_PACKED`
	Static field: `Datatype INT2`	`MPI_2INT`
	Static field: `Datatype SHORT_INT`	`MPI_SHORT_INT`
	Static field: `Datatype LONG_INT`	`MPI_LONG_INT`
	Static field: `Datatype FLOAT_INT`	`MPI_FLOAT_INT`
	Static field: `Datatype DOUBLE_INT`	`MPI_DOUBLE_INT`
	Static field: `Datatype FLOAT_COMPLEX`	`MPI_C_FLOAT_COMPLEX`
	Static field: `Datatype DOUBLE_COMPLEX`	`MPI_C_DOUBLE_COMPLEX`
	`void contiguous(int count, Datatype type)`	`MPI_Type_contiguous`
	`void commit()`	`MPI_Type_commit`
	`int getTypeSize()`	`MPI_Type_size`
	`void free()`	`MPI_Type_free`
	`void vector(int count, int blockLength, int stride, Datatype baseType)`	`MPI_Type_vector`
	`void hvector(int count, int blockLength, int stride, Datatype oldType)`	`MPI_Type_create_hvector`
	`void indexed(int count, int[] blockLength, int[] displacement, Datatype oldType)`	`MPI_Type_indexed`
	`void hindexed(int count, int[] blockLength, Aint[] displacement, Datatype oldType)`	`MPI_Type_create_hindexed`
	`void struct(int count, int[] blockLength, Aint[] displacement, Datatype[] oldTypes)`	`MPI_Type_struct`
`Op`	Static field: `Op MPI_OP_NULL`	`MPI_OP_NULL`
	Static field: `Op MPI_MAX`	`MPI_MAX`
	Static field: `Op MPI_MIN`	`MPI_MIN`

	Static field: `Op MPI_SUM`	`MPI_SUM`
	Static field: `Op MPI_PROD`	`MPI_PROD`
	Static field: `Op MPI_LAND`	`MPI_LAND`
	Static field: `Op MPI_BAND`	`MPI_BAND`
	Static field: `Op MPI_LOR`	`MPI_LOR`
	Static field: `Op MPI_BOR`	`MPI_BOR`
	Static field: `Op MPI_LXOR`	`MPI_LXOR`
	Static field: `Op MPI_BXOR`	`MPI_BXOR`
	Static field: `Op MPI_MINLOC`	`MPI_MINLOC`
	Static field: `Op MPI_MAXLOC`	`MPI_MAXLOC`
	`Op(UserFunction uf)`	-
	`void setUserFunction(UserFunction userFunction)`	-
	`void createOP(boolean commute)`	`MPI_Op_Create`
`UserFunction` (abstract)	`UserFunction(Datatype type, int length)`	-
	`void setInoutvec(ByteBuffer inoutvec)`	-
	`void setInvec(ByteBuffer invec)`	-
	`abstract void call(int type, int length)`	-
`PTP`	`static void send(Buffer buffer, int count, Datatype type, int dest, int tag, Comm comm)`	`MPI_Send`
	`static void send(<java array> buffer, int count, Datatype type, int dest, int tag, Comm comm)`	`MPI_Send`
	`static Status recv(Buffer buf, int count, Datatype type, int source, int tag, Comm comm)`	`MPI_Recv`
	`static Status recv(<java array> buf, int count, Datatype type, int source, int tag, Comm comm)`	`MPI_Recv`
	`static Request isend(Buffer buffer, int count, Datatype type, int dest, int tag,`	`MPI_Isend`

	Comm comm)	
	static Request isend(<java array> buffer, int count, Datatype type, int dest, int tag, Comm comm)	MPI_Isend
	static Request irecv(Buffer buf, int count, Datatype type, int source, int tag, Comm comm)	MPI_Irecv
	static Request irecv(<java array> buf, int count, Datatype type, int source, int tag, Comm comm)	MPI_Irecv
	static Status sendRecv(Buffer sendbuf, int sendcount, Datatype sendtype, int senddest, int sendtag, Buffer recvbuf, int recvcount, Datatype recvtype, int recvsource, int recvtag, Comm comm)	MPI_Sendrecv
Request	Status Wait()	MPI_Wait
	static Status[] waitAll(int count, Request[] reqs)	MPI_Waitall
	static Status waitAny(int count, Request[] reqs, int[] index)	MPI_Waitany
	static Status[] waitSome(int count, Request[] reqs, int[] outcount, int[] indexes)	MPI_Waitsome
	boolean test(Status status)	MPI_Test

4.11.1. Running Java* MPI Applications

Use the following 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 <mpirun options> java <main class of your java application>
   ```

   For example:

   ```
   $ mpirun -n 8 -ppn 1 -genv I_MPI_FABRICS shm:tcp -f ./hostfile java
   mpi.samples.Allreduce
   ```

Sample Java MPI applications are available in the *<install_dir>*`/test` folder.

4.11.2. Developing Java* MPI Applications

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 to 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 so on, the memory allocation happens on the native layer and is not tracked by garbage collector. Thus, this memory should be released explicitly. The pointers to the allocated memory are kept in a special pool and can be de-allocated using one of the following methods:
  - `entity.free()`: frees the memory backing the `entity` Java object, which can be an instance of `Comm`, `Group`, and so on.
  - `AllocablePool.remove(entity)`: frees the memory backing the `entity` Java object, which can be an instance of `Comm`, `Group`, and so on.
  - `AllocablePool.cleanUp()`: explicitly de-allocates the memory backing all Java MPI objects created by that moment.
  - `MPI.Finalize()`: implicitly de-allocates the memory backing all Java MPI objects and that was not explicitly de-allocated by that moment.

4.12. Other Environment Variables

I_MPI_DEBUG

Print out debugging information when an MPI program starts running.

Syntax

`I_MPI_DEBUG=<level>[,<flags>]`

Arguments

<level>	Indicate level of debug information provided
0	Output no debugging information. This is the default value.
1	Output verbose error diagnostics.
2	Confirm which `I_MPI_FABRICS` was used and which Intel® MPI Library configuration was used.
3	Output effective MPI rank, `pid` and node mapping table.
4	Output process pinning information.
5	Output Intel MPI-specific environment variables.
6	Output collective operation algorithms settings.

> 6	Add extra levels of debug information.

<flags>	Comma-separated list of debug flags
pid	Show process id for each debug message.
tid	Show thread id for each debug message for multithreaded library.
time	Show time for each debug message.
datetime	Show time and date for each debug message.
host	Show host name for each debug message.
level	Show level for each debug message.
scope	Show scope for each debug message.
line	Show source line number for each debug message.
file	Show source file name for each debug message.
nofunc	Do not show routine name.
norank	Do not show rank.
flock	Synchronize debug output from different process or threads.
nobuf	Do not use buffered I/O for debug output.

Description

Set this environment variable to print debugging information about the application.

NOTE

Set the same <level> value for all ranks.

You can specify the output file name for debug information by setting the I_MPI_DEBUG_OUTPUT environment variable.

Each printed line has the following format:

[<identifier>] <message>

where:

- <identifier> is the MPI process rank, by default. If you add the '+' sign in front of the <level> number, the <identifier> assumes the following format: rank#pid@hostname. Here, rank is the MPI process rank, pid is the UNIX* process ID, and hostname is the host name. If you add the '–' sign, <identifier> is not printed at all.
- <message> contains the debugging output.

The following examples demonstrate possible command lines with the corresponding output:

```
$ mpirun -n 1 -env I_MPI_DEBUG=2 ./a.out
...
[0] MPI startup(): shared memory data transfer mode
```

The following commands are equal and produce the same output:

```
$ mpirun -n 1 -env I_MPI_DEBUG=+2 ./a.out
$ mpirun -n 1 -env I_MPI_DEBUG=2,pid,host ./a.out
...
[0#1986@mpicluster001] MPI startup(): shared memory data transfer mode
```

NOTE

Compiling with the `-g` option adds a considerable amount of printed debug information.

I_MPI_DEBUG_OUTPUT

Set output file name for debug information.

Syntax

`I_MPI_DEBUG_OUTPUT=<arg>`

Arguments

<arg>	String value
`stdout`	Output to `stdout`. This is the default value.
`stderr`	Output to `stderr`.
`<file_name>`	Specify the output file name for debug information (the maximum file name length is 256 symbols).

Description

Set this environment variable if you want to split output of debug information from the output produced by an application. If you use format like *%r*, *%p* or *%h*, rank, process ID or host name is added to the file name accordingly.

I_MPI_PRINT_VERSION

Print library version information.

Syntax

`I_MPI_PRINT_VERSION=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Print library version information.
`disable | no | off | 0`	No action. This is the default value.

Description

Set this environment variable to enable/disable printing of Intel® MPI library version information when an MPI application starts running.

I_MPI_OUTPUT_CHUNK_SIZE

Set the size of the `stdout/stderr` output buffer.

Syntax

`I_MPI_OUTPUT_CHUNK_SIZE=<size>`

Arguments

<size>	Define output chunk size in kilobytes
<n> > 0	The default chunk size value is 1 KB

Description

Set this environment variable to increase the size of the buffer used to intercept the standard output and standard error streams from the processes. If the *<size>* value is not greater than zero, the environment variable setting is ignored and a warning message is displayed.

Use this setting for applications that create a significant amount of output from different processes. With the `-ordered-output` option of `mpiexec.hydra`, this setting helps to prevent the output from garbling.

NOTE

Set the `I_MPI_OUTPUT_CHUNK_SIZE` environment variable in the shell environment before executing the `mpiexec.hydra`/`mpirun` command. Do not use the `-genv` or `-env` options for setting the *<size>* value. Those options are used only for passing environment variables to the MPI process environment.

I_MPI_PMI_EXTENSIONS

Turn on/off the use of the Intel® MPI Library Process Management Interface (PMI) extensions.

Syntax

`I_MPI_PMI_EXTENSIONS=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the PMI extensions
`disable | no | off | 0`	Turn off the PMI extensions

Description

The Intel® MPI Library automatically detects if your process manager supports the PMI extensions. If supported, the extensions substantially decrease task startup time. Set `I_MPI_PMI_EXTENSIONS` to `disable` if your process manager does not support these extensions.

I_MPI_PMI_LIBRARY

Specify the name to third party implementation of the PMI library.

Syntax

`I_MPI_PMI_LIBRARY=<name>`

Arguments

<name>	Full name of the third party PMI library

Description

Set `I_MPI_PMI_LIBRARY` to specify the name of third party PMI library. When you set this environment variable, provide full name of the library with full path to it.

I_MPI_JOB_FAST_STARTUP

Turn on/off the faster Intel® MPI Library process startup algorithm for the TMI and OFI fabrics.

Syntax

`I_MPI_JOB_FAST_STARTUP=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Turn on the algorithm for fast startup on the TMI and OFI fabrics. This is the default value
`disable | no | off | 0`	Turn off the algorithm for fast startup

Description

The new algorithm significantly reduces the application startup time when run over the TMI or OFI fabric. The algorithm efficiency is more clearly observed with the higher `-ppn` value, while there is no improvement at all with `-ppn 1`.

I_MPI_HARD_FINALIZE

Turn on/off the hard (ungraceful) process finalization algorithm.

Syntax

`I_MPI_HARD_FINALIZE=<arg>`

Argument

<arg>	Binary indicator
`enable | yes | on | 1`	Enable hard finalization. This is the default value for the OFI and TMI fabrics.
`disable | no | off | 0`	Disable hard finalization. This is the default value for all other fabrics.

Description

The hard (ungraceful) finalization algorithm may significantly reduce the application finalization time.

TOTALVIEW

Select a particular TotalView* executable file to use.

Syntax

`TOTALVIEW=<path>`

Arguments

<path>	Path/name of the TotalView* executable file instead of the default `totalview`

Description

Set this environment variable to select a particular TotalView* executable file.

I_MPI_PLATFORM

Select the intended optimization platform.

Syntax

`I_MPI_PLATFORM=<platform>`

Arguments

<platform>	Intended optimization platform (string value)
`auto[:min]`	Optimize for the oldest supported Intel® Architecture Processor across all nodes. This is the default value
`auto:max`	Optimize for the newest supported Intel® Architecture Processor across all nodes
`auto:most`	Optimize for the most numerous Intel® Architecture Processor across all nodes. In case of a tie, choose the newer platform
`uniform`	Optimize locally. The behavior is unpredictable if the resulting selection differs from node to node
`none`	Select no specific optimization
`htn | generic`	Optimize for the Intel® Xeon® Processors 5400 series and other Intel® Architecture processors formerly code named Harpertown
`nhm`	Optimize for the Intel® Xeon® Processors 5500, 6500, 7500 series and other Intel® Architecture processors formerly code named Nehalem
`wsm`	Optimize for the Intel® Xeon® Processors 5600, 3600 series and other Intel® Architecture processors formerly code named Westmere
`snb`	Optimize for the Intel® Xeon® Processors E3, E5, and E7 series and other Intel® Architecture processors formerly code named Sandy Bridge
`ivb`	Optimize for the Intel® Xeon® Processors E3, E5, and E7 V2 series and other Intel®

	Architecture processors formerly code named Ivy Bridge
`hsw`	Optimize for the Intel® Xeon® Processors E3, E5, and E7 V3 series and other Intel® Architecture processors formerly code named Haswell
`bdw`	Optimize for the Intel® Xeon® Processors E3, E5, and E7 V4 series and other Intel® Architecture processors formerly code named Broadwell
`knl`	Optimize for the Intel® Xeon Phi™ processor and coprocessor formerly code named Knights Landing

Description

Set this variable to use the predefined platform settings. It is available for both Intel® and non-Intel microprocessors, but it may utilize additional optimizations for Intel microprocessors than it utilizes for non-Intel microprocessors.

NOTE

The values `auto:min`, `auto:max` and `auto:most` may increase the MPI job startup time.

I_MPI_PLATFORM_CHECK

Turn on/off the optimization setting similarity check.

Syntax

`I_MPI_PLATFORM_CHECK=<arg>`

Argument

<arg>	Binary indicator
`enable | yes | on | 1`	Turns on the optimization platform similarity check. This is the default value
`disable | no | off | 0`	Turns off the optimization platform similarity check

Description

Set this variable to check the optimization platform settings of all processes for similarity. If the settings are not the same on all ranks, the library terminates the program. Disabling this check may reduce the MPI job startup time.

I_MPI_THREAD_LEVEL_DEFAULT

Set this environment variable to initialize the MPI thread environment for the multi-threaded library if `MPI_Init()` call is used for initialization.

Syntax

`I_MPI_THREAD_LEVEL_DEFAULT=<threadlevel>`

Arguments

<threadlevel>	Define the default level of thread support

SINGLE | single	Set the default level of thread support to MPI_THREAD_SINGLE
FUNNELED | funneled	Set the default level of thread support to MPI_THREAD_FUNNELED. This is the default value if MPI_Init() call is used for initialization
SERIALIZED | serialized	Set the default level of thread support to MPI_THREAD_SERIALIZED
MULTIPLE | multiple	Set the default level of thread support to MPI_THREAD_MULTIPLE

Description

Set I_MPI_THREAD_LEVEL_DEFAULT to define the default level of thread support for the multi-threaded library if MPI_Init() call is used for initialization.

NOTE

The environment variable I_MPI_THREAD_LEVEL_DEFAULT is equivalent to the environment variable MPICH_THREADLEVEL_DEFAULT.

I_MPI_YARN

Set this variable when running on a YARN*-managed cluster.

Arguments

<value>	Binary indicator
enable | yes | on | 1	Enable YARN support
disable | no | off | 0	Disable YARN support. This is the default value.

Description

Set this environment variable to make Hydra request resources from the YARN cluster manager prior to running an MPI job. Use this functionality only when you launch MPI on a YARN-managed cluster with Llama* installed (for example on cluster with the Cloudera* Distribution for Hadoop*).

Usage Example

Verify that YARN is configured to work properly with Llama (refer to the Llama documentation for the specific configuration details), and the Apache* Thrift* installation is available on the cluster.

1. Make sure Llama is started on the same host where YARN is running, or start it by issuing the following command as the llama user:

   ```
   $ llama [--verbose &]
   ```

2. Make sure passwordless SSH is configured on the cluster.
3. Set the I_MPI_YARN environment variable:

   ```
   $ export I_MPI_YARN=1
   ```

4. Either set I_MPI_THRIFT_PYTHON_LIB to point to Thrift's Python* modules, or add these modules explicitly to PYTHONPATH.

5. Set `I_MPI_LLAMA_HOST`/`I_MPI_LLAMA_PORT` to point to the Llama server host/port (by default, it is `localhost:15000`, so you can skip this step, if launching MPI from the same host where the Llama service is running).
6. Launch an MPI job as usual (do not specify hosts or machine file explicitly - the resources will be automatically allocated by YARN):

```
$ mpirun -n 16 -ppn 2 [<other mpirun options>] <executable>
```

I_MPI_LIBRARY_KIND

Specify the Intel® MPI Library configuration.

Syntax

`I_MPI_LIBRARY_KIND=<value>`

Arguments

<value>	Intel® MPI Library configuration
`debug`	Single-threaded debug library
`debug_mt`	Multi-threaded debug library
`release`	Single-threaded optimized library
`release_mt`	Multi-threaded optimized library. This is the default value

Description

Use this variable to set an argument for the `mpivars.[c]sh` script. This script establishes the Intel® MPI Library environment and enables you to specify the appropriate library configuration.

Setting this variable is equivalent to passing an argument directly to the `mpivars.[c]sh` script.

Example

```
$ export I_MPI_LIBRARY_KIND=debug
$ . <installdir>/intel64/bin/mpivars.sh
```

To ensure that the desired configuration is set, check the `LD_LIBRARY_PATH` variable.

I_MPI_SLURM_EXT

Enable faster job startup with the SLURM* job scheduler.

Syntax

`I_MPI_SLURM_EXT=<value>`

Arguments

<value>	Binary indicator
`enable | yes | on | 1`	Enable the faster job startup algorithm. This is the default value.
`disable | no | off | 0`	Use the standard startup algorithm.

Description

When you run an MPI job under the SLURM job scheduler on a large scale, the job startup process may take very long. Set the `I_MPI_SLURM_EXT` variable to use an optimized startup algorithm to speed up the process.

5. Glossary

cell	A pinning resolution in descriptions for pinning property.
hyper-threading technology	A feature within the IA-64 and Intel® 64 family of processors, where each processor core provides the functionality of more than one logical processor.
logical processor	The basic modularity of processor hardware resource that allows a software executive (OS) to dispatch task or execute a thread context. Each logical processor can execute only one thread context at a time.
multi-core processor	A physical processor that contains more than one processor core.
multi-processor platform	A computer system made of two or more physical packages.
processor core	The circuitry that provides dedicated functionalities to decode, execute instructions, and transfer data between certain sub-systems in a physical package. A processor core may contain one or more logical processors.
physical package	The physical package of a microprocessor capable of executing one or more threads of software at the same time. Each physical package plugs into a physical socket. Each physical package may contain one or more processor cores.
processor topology	Hierarchical relationships of "shared vs. dedicated" hardware resources within a computing platform using physical package capable of one or more forms of hardware multi-threading.

6. Index