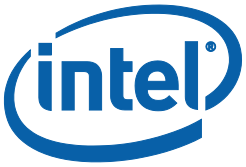


Intel® MPI Library for Linux* OS

Developer Reference

Update 6, software version 2021.1 Beta

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.

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.

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

What's New

This document reflects the updates for Intel® MPI Library 2019 Update 6 release for Linux* OS:

The following latest changes in this document were made:

Intel MPI Library 2019 Update 6 (software version 2021.1)

- Added `I_MPI_PMI_VALUE_LENGTH_MAX` to Other Environment Variables.
- Added support for non-blocking collectives, more blocking collectives, and `HCOLL` collectives to Autotuning.
- Reworked directory layout:
  - Removed `intel64/`.
  - `Mpivars.[c]sh` and `mpi modulefile` moved to `env/`.
  - `Mpivars.[c]sh` renamed to `vars.[c]sh`.
- Removed deprecated symbolic links.
- Removed static libraries for debug configurations.

Intel MPI Library 2019 Update 5

- Added `I_MPI_SHM_HEAP`, `I_MPI_SHM_HEAP_VSIZE`, `I_MPI_SHM_HEAP_CSIZE`, `I_MPI_SHM_HEAP_OPT`, `I_MPI_WAIT_MODE`, `I_MPI_THREAD_YIELD`, `I_MPI_PAUSE_COUNT`, `I_MPI_THREAD_SLEEP` to Other Environment Variables.

- Added `I_MPI_ADJUST_<opname>_LIST`, `I_MPI_COLL_EXTERNAL` to I_MPI_ADJUST Family Environment Variables.
- Updated Autotuning and Tuning Environment Variables.

Intel MPI Library 2019 Update 4

- Added new Autotuning functionality description and environment variables to Environment Variables for Autotuning.
- Added new variables `I_MPI_TUNING`, `I_MPI_TUNING_BIN`, and `I_MPI_TUNING_BIN_DUMP` to Tuning Environment Variables.
- Added arguments for `I_MPI_PLATFORM` in Other Environment Variables.
- Added new `-tune`, `-hosts-group` options to Global Options.
- Added new environment variables `I_MPI_JOB_STARTUP_TIMEOUT`, `I_MPI_HYDRA_NAMESERVER` to Hydra Environment Variables
- Added new transports to `I_MPI_SHM` in Shared Memory Control.
- Removed `-unmask` and `-gumask` options.

Intel MPI Library 2019 Update 3

- Added new option `-norpath` to Compilation Command Options.
- Added new options `-silent-abort`,`-nameserver` and environment variables `I_MPI_SILENT_ABORT`,`I_MPI_HYDRA_NAMESERVER` to Hydra Environment Variables.
- Added new variables `I_MPI_MALLOC`,`I_MPI_EXTRA_FILESYSTEM`,`I_MPI_STATS` to Other Environment Variables.
- Updated the `-validate` option description.
- Added new argument for the `-s` *<spec>* option.
- Removed the `-whoami` option.
- Removed 14 outdated variables from I_MPI_ADJUST Family Environment Variables.

Intel MPI Library 2019 Update 2

- Bug fixes.

Intel MPI Library 2019 Update 1

- Added new variable `I_MPI_CBWR` to I_MPI_ADJUST Family Environment Variables.
- Restored `I_MPI_PLATFORM` and `I_MPI_PLATFORM_CHECK` (Other Environment Variables).
- Adjusted description of the `-configfile` option in Global Options and `-wdir` option in Local Options.
- Added new variable `I_MPI_VAR_CHECK_SPELLING` to Other Environment Variables.
- Added new variable `I_MPI_HYDRA_SERVICE_PORT` to Hydra Environment Variables.
- Added `I_MPI_SHM_FILE_PREFIX_4K`, `I_MPI_SHM_FILE_PREFIX_2M`, and `I_MPI_SHM_FILE_PREFIX_1G` variables to Shared Memory Control.

Intel MPI Library 2019

- Document overhaul to align with supported functionality.
- Removed the `I_MPI_HARD_FINALIZE`, `I_MPI_MIC`, `I_MPI_ENV_PREFIX_LIST`, `I_MPI_TUNE*`, `I_MPI_ENV_PREFIX_LIST`,`I_MPI_JOB_FAST_STARTUP`,`I_MPI_FALLBACK`, `I_MPI_DAPL*`, `I_MPI_LARGE_SCALE_THRESHOLD`, `I_MPI_OFA*`, `I_MPI_TCP*`, `I_MPI_TMI*` environment variables.
- Removed the `-hostos` option from Local Options.

- Added the `I_MPI_OFI_LIBRARY_INTERNAL` environment variable to OFI-capable Network Fabrics Control.
- Added an option for setting `MPI_UNIVERSE_SIZE` to Global Options.
- Added new collective operations to I_MPI_ADJUST Family Environment Variables.
- Added new variables `I_MPI_SHM_CELL_EXT_SIZE` and `I_MPI_SHM_CELL_EXT_NUM_TOTAL` to Shared Memory Control.
- Added impi_info utility.
- Updated mpitune utility.
- Updated the topic Environment Variables for Asynchronous Progress Control.
- Added environment variables for Multi-EP (`I_MPI_THREAD_SPLIT`, `I_MPI_THREAD_RUNTIME`, `I_MPI_THREAD_MAX`, `I_MPI_THREAD_ID_KEY`).
- Examples are now available as a part of Intel® MPI Library Developer Guide.

Intel MPI Library 2018 Update 3

- Added new algorithms for `I_MPI_ADJUST_ALLREDUCE` to I_MPI_ADJUST Family.

Intel MPI Library 2018 Update 2

- Improved shm performance with collective operations (`I_MPI_THREAD_YIELD`).
- Bug fixes .

Intel MPI Library 2018 Update 1

- Added the environment variable `I_MPI_STARTUP_MODE` in Other Environment Variables

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`
  -

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)

Related Information

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

Command Reference

Compilation Commands

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

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: release, debug, release_mt, or debug_mt . The ordinary 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.

Compilation Command Options

-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>/intel64/etc/mpi*<compiler>*-*<name>*.conf
where:

- *<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/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. The default value is `libVTmc.so`. Using this option has the same effect as the `-profile=vtmc` 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.

-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.

-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.

-v

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

-norpath

Use this option to disable rpath for the compiler wrapper for the Intel® MPI Library.

mpirun

Launches an MPI job and provides integration with job schedulers.

Syntax

`mpirun` *<options>*

Arguments

<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.

mpiexec.hydra

Launches an MPI job using the Hydra process manager.

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 using the Hydra process manager.
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.

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 ':'.

-tune *<filename>*

Use this option to specify the file name that contains the tuning data in a binary format.

-usize *<usize>*

Use this option to set `MPI_UNIVERSE_SIZE`, which is available as an attribute of the `MPI_COMM_WORLD`.

<size>	Define the universe size
`SYSTEM`	Set the size equal to the number of cores passed to mpiexec through the hostfile or the resource manager.
`INFINITE`	Do not limit the size. This is the default value.
<value>	Set the size to a numeric value ≥ 0.

-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. For example:

```
$ cat ./machinefile
node0:2
node1:2
node0:1
```

-hosts-group

Use this option to set node ranges using brackets, commas, and dashes (like in Slurm* Workload Manager).

For more details, see the `I_MPI_HYDRA_HOST_FILE` environment variable in Hydra Environment Variables.

-silent-abort

Use this option to disable abort warning messages.

For more details, see the `I_MPI_SILENT_ABORT` environment variable in Hydra Environment Variables.

-nameserver

Use this option to specify the nameserver in the hostname:port format.
For more details, see the `I_MPI_HYDRA_NAMESERVER` environment variable in Hydra Environment Variables.

-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.

Note

The option does not work for localhost.

-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.

-mps

Use this option to collect only MPI and OpenMP* statistics from your MPI application using Application Performance Snapshot. Unlike the `-aps` option, `-mps` doesn't collect hardware metrics. The option is equivalent to:

```
$ mpirun -n 2 aps -c mpi,omp ./myapp
```

-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 with one executable per line. Blank lines and lines that start with '#' are ignored. Other options specified in the command line are treated as global.
You can specify global options in configuration files loaded by default (`mpiexec.conf` in `<installdir>/intel64/etc`, `~/.mpiexec.conf`, and `mpiexec.conf` in the working directory). The remaining options can be specified in the command line.

-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.

-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.

-ilp64

Use this option to preload the ILP64 interface. 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.
none	Do not direct standard output to any 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.

-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.

-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.

-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 *<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`, `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.
`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.

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.

Note

The option does not work for localhost.

-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.

---

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.

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.

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). Use the `-remote` argument for ranks to use the tool on remote nodes only.

Note

The tool attached to an MPI process may be executed without having access to stdin. To pass input to it, run a rank under the tool directly, for example: `-gtool "gdb --args:0"`

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.

-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`.

cpuinfo

Provides information on processors used in the system.

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        |
|_________|________|______|________|_______|__________|
```

impi_info

Provides information on available Intel® MPI Library environment variables.

Syntax

```
impi_info <options>
```

Arguments

`<options>`	List of options.
`-a | -all`	Show all IMPI variables.
`-h | -help`	Show a help message.
`-v | -variable`	Show all available variables or description of the specified variable.
`-c | -category`	Show all available categories or variables of the specified category.

Description

The `impi_info` utility provides information on environment variables available in the Intel MPI Library. For each variable, it prints out the name, the default value, and the value data type. By default, a reduced list of variables is displayed. Use the `-all` option to display all available variables with their descriptions.
The example of the `impi_info` output:

```
$ ./impi_info

| NAME                                  | DEFAULT VALUE | DATA TYPE    |
=========================================================================
| I_MPI_THREAD_SPLIT                    | 0             | MPI_INT      |
| I_MPI_THREAD_RUNTIME                  | none          | MPI_CHAR     |
| I_MPI_THREAD_MAX                      | -1            | MPI_INT      |
| I_MPI_THREAD_ID_KEY                   | thread_id     | MPI_CHAR     |
```

mpitune

Tunes the Intel® MPI Library parameters for the given MPI application.

Syntax

```
mpitune <options>
```

Arguments

<mpitune options>	**List of options.**
`-c | --config-file <file>`	Specify a configuration file to run a tuning session.
`-d | --dump-file <file>`	Specify a file that stores the collected results. The option is used in the `analyze` mode.
`-m | --mode {collect | analyze}`	Specify the `mpitune` mode. The supported modes are `collect` and `analyze`: • the `collect` mode runs the tuning process and saves results in temporary files; • the `analyze` mode transforms temporary files into a JSON-tree, which is used by the Intel® MPI Library, and generates a table that represents

	algorithm values in a human-readable format.
`-h | --help`	Display the help message.
`-v | --version`	Display the product version.

Description

The `mpitune` utility allows you to automatically adjust Intel® MPI Library parameters, such as collective operation algorithms, to your cluster configuration or application.
The tuner iteratively launches a benchmarking application with different configurations to measure performance and stores the results of each launch. Based on these results, the tuner generates optimal values for the parameters being tuned.

Note

Starting with the Intel® MPI Library Update 4 release, you must specify two mpitune configuration files, which differ in their mode and dump-file fields. A simpler alternative may be to use a single configuration file with templates inside for mode and dump-file fields, which should be defined via the command line.

The configuration files should specify all tuner parameters, which are passed to the tuner with the `--config-file` option. All configuration file examples are available at `<installdir>/etc/tune_cfg`. Please note that configuration files for Intel® MPI Benchmarks are already created.
The tuning process consists of two steps: data collection (the `collect` mode) and data analysis (the `analyze` mode):

```
$ mpitune -m analyze -c /path/to/config_file1
$ mpitune -m collect -c /path/to/config_file2
```

Another variant of the launch is:

```
$ mpitune -m analyze -c /path/to/config_file1

$ mpitune -m collect -c /path/to/config_file1 -d path/to/dump-file
```

where the path to the dump-file received in the first step is used in the config file with templates inside.

The tuning results are presented as a JSON tree and can be added to the library with the `I_MPI_TUNING` environment variable.

MPI Options Support

The following MPI options are available within the utility:

<MPI options>	List of options.
-f <filename>	Specify a file containing host names.
-hosts <hostlist>	Specify a comma-separated list of hosts.
-np <value>	Specify the number of processes.
-ppn <n>	Specify the number of processes per node.

For example:

```
$ mpitune -np 2 -ppn 1 -hosts HOST1,HOST2 -m analyze -c /path/to/config_file1
$ mpitune -np 2 -ppn 1 -hosts HOST1,HOST2 -m collect -c /path/to/config_file2
```

See Also

Developer Guide, section *Analysis and Tuning > MPI Tuning.*

mpitune Configuration Options

Application Options

-app

Sets a template for the command line to be launched to gather tuning results. The command line can contain variables declared as `@<var_name>@`. The variables are defined further on using other options.
For example:

```
-app: mpirun -np @np@ -ppn @ppn@ IMB-MPI1 -msglog 0:@logmax@ -npmin @np@
@func@
```

Note

The application must produce output (in `stdout` or file or any other destination) that can be parsed by the tuner to pick the value to be tuned and other variables. See the `-app-regex` and `-app-regex-legend` options below for details.

-app-regex

Sets a regular expression to be evaluated to extract the required values from the application output. Use regular expression groups to assign the values to variables. Variables and groups associations are set using the `-app-regex-legend` option.
For example, to extract the `#bytes` and `t_max[usec]` values from this output:

```
#bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]
0      1000          0.06         0.06         0.06
1      1000          0.10         0.10         0.10
```

use the following configuration:

```
-app-regex: (\d+)\s+\d+\s+[\d.+-]+\s+([\d.+-]+)
```

-app-regex-legend

Specifies a list of variables extracted from the regular expression. Variables correspond to the regular expression groups. The tuner uses the last variable as the performance indicator of the launch. Use the `-tree-opt` to set the optimization direction of the indicator.
For example:

```
-app-regex-legend: size,time
```

-iter

Sets the number of iterations for each launch with a given set of parameters. Higher numbers of iterations increase accuracy of results.

For example:

```
-iter: 3
```

Search Space Options

Use these options to define a search space, which is a set of combinations of Intel® MPI Library parameters that the target application uses for launches. The library parameters are generally configured using run-time options or environment variables.

Note

A search space line can be very long, so line breaking is available for all the search space options. Use a backslash to break a line (see examples below).

-search

Defines the search space by defining variables declared with the `-app` option and by setting environment variables for the application launch.
For example:

```
-search: func=BCAST, \
         np=4,ppn={1,4},{,I_MPI_ADJUST_BCAST=[1,3]},logmax=5
```

The `-app` variables are defined as `<var1>=<value1>[,<var2>=<value2>][,...]`. The following syntax is available for setting values:

Syntax	Description	Examples
`<value>`	Single value. Can be a number or a string.	`4`
`{<value1>[,<value2>][,...]}`	List of independent values.	`{2,4}`
`[<start>,<end>[,<step>]]`	Linear range of values with the default step of `1`.	`[1,8,2]` – expands to `{1,2,4,6,8}`
`(<start>,<end>[,<step>])`	Exponential range with the default step of `2`.	`(1,16)` – expands to `{1,2,4,8,16}`

To set environment variables for the command launch, use the following syntax:

Syntax	Description	Examples
`<variable>=<value>`	Single variable definition. Any type of the syntax above can be used for the value: single values, lists or ranges.	`I_MPI_ADJUST_BCAST=3` `I_MPI_ADJUST_BCAST=[1,3]`
`{,<variable>=<value>}`	A special case of the syntax above. When set this way, the variable default value is first used in an application launch.	`{,I_MPI_ADJUST_BCAST=[1,3]}`
`<prefix>{<value1>[,<value2>][,...]}`	Multi-value variable definition. Prefix is a common part for all the values, commonly the variable name. A value can be a singular value or a combination of values in the format: `<prefix>(<value1>[,<value2>][,...])`. Prefix is optional and a value in the	`I_MPI_ADJUST_ALLREDUCE{=1,=2,(=9,_KN_RADIX=(2,8))}` See below for a more complete example.

	combination is a string, which can utilize the list and range syntax above.	

The following example shows a more complex option definition:

```
I_MPI_ADJUST_BCAST{=1,=2,(=9,_KN_RADIX=(2,8)),(={10,11},_SHM_KN_RADIX=[2,8,
2])}
```

This directive consecutively runs the target application with the following environment variables set:

```
I_MPI_ADJUST_BCAST=1
I_MPI_ADJUST_BCAST=2
I_MPI_ADJUST_BCAST=9,I_MPI_ADJUST_BCAST_KN_RADIX=2
I_MPI_ADJUST_BCAST=9,I_MPI_ADJUST_BCAST_KN_RADIX=4
I_MPI_ADJUST_BCAST=9,I_MPI_ADJUST_BCAST_KN_RADIX=8
I_MPI_ADJUST_BCAST=10,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=2
I_MPI_ADJUST_BCAST=10,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=4
I_MPI_ADJUST_BCAST=10,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=6
I_MPI_ADJUST_BCAST=10,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=8
I_MPI_ADJUST_BCAST=11,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=2
I_MPI_ADJUST_BCAST=11,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=4
I_MPI_ADJUST_BCAST=11,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=6
I_MPI_ADJUST_BCAST=11,I_MPI_ADJUST_BCAST_SHM_KN_RADIX=8
```

-search-excl

Excludes certain combinations from the search space. The syntax is identical to that of the `-search` option. For example:

```
-search-excl: I_MPI_ADJUST_BCAST={1,2}
```

or

```
-search-excl: func=BCAST,np=4,ppn=1,I_MPI_ADJUST_BCAST=1
```

-search-only

Defines a subset of the search space to search in. Only this subset is used for application launches. The syntax is identical to the `-search` option.
This option is useful for the second and subsequent tuning sessions on a subset of parameters from the original session, without creating a separate configuration file.

Output Options

Use these options to customize the output. The tuner can produce output of two types:

- table – useful for verifying the tuning results, contains values from all the application launches
- tree – an internal output format, contains the optimal values

-table

Defines the layout for the resulting output table. The option value is a list of variables declared with the `-app` option, which are joined in colon-separated groups. Each group denotes a specific part of the table.
For example:

```
-table: func:ppn,np:size:*:time
```

The last group variables (`time`) are rendered in table cells. The second last group variables are used for building table columns (`*`, denotes all the variables not present the other variable groups). The third last group variables are used for building table rows (`size`). All other variable groups are used to make up the table label. Groups containing several variables are complex groups and produce output based on all the value combinations.
For example, the option definition above can produce the following output:

```
Label:                                                  "func=BCAST,ppn=2,np=2"


Legend:
set 0: ""
set 1: "I_MPI_ADJUST_BCAST=1"
set 2: "I_MPI_ADJUST_BCAST=2"
set 3: "I_MPI_ADJUST_BCAST=3"
Table:
           | set 0       | set 1       | set 2       | set 3
-----------|-------------|-------------|-------------|------------
"size=0"   | "time=0.10" | "time=0.08" | "time=0.11" | "time=0.10"
           | "time=0.12" | "time=0.09" | "time=0.12" | "time=0.11"
           |             | "time=0.10" |             |
-----------|-------------|-------------|-------------|------------
"size=4"   | "time=1.12" | "time=1.11" | "time=1.94" | "time=1.72"
           | "time=1.35" | "time=1.18" | "time=1.97" | "time=1.81"
           | "time=1.38" | "time=1.23" | "time=2.11" | "time=1.89"
-----------|-------------|-------------|-------------|------------
"size=8"   | "time=1.21" | "time=1.10" | "time=1.92" | "time=1.72"
           | "time=1.36" | "time=1.16" | "time=2.01" | "time=1.75"
           | "time=1.37" | "time=1.17" | "time=2.24" | "time=1.87"
-----------|-------------|-------------|-------------|------------

...
```

Cells include only unique values from all the launches for the given parameter combination. The number of launches is set with the `-iter` option.

-table-ignore

Specifies the variables to ignore from the `-table` option definition.

-tree

Defines the layout for the resulting tree of optimal values of the parameter that is tuned (for example, collective operation algorithms). The tree is rendered as a JSON structure. The option value is a list of variables declared with the `-app` option, which are joined in colon-separated groups. Each group denotes a specific part of the tree. Groups containing several variables are complex groups and produce output based on all the value combinations.
Example:

```
-tree: func:ppn,np:size:*:time
```

The first two groups (`func` and `ppn,np`) make up the first two levels of the tree. The last group variables (`time`) are used as the optimization criteria and are not rendered. The second last group contains variables to be optimized (`*`, denotes all the variables not present the other variable groups). The third last group variables are used to split the search space into intervals based on the optimal values of parameters from the next group (for example, `I_MPI_ADJUST_<operation>` algorithm numbers).
For example, the option definition above can produce the following output:

```
{
    "func=BCAST":
    {
        "ppn=1,np=4":
        {
            "size=0":
                {"I_MPI_ADJUST_BCAST": "3"},
            "size=64":
                {"I_MPI_ADJUST_BCAST":                                  "1"},
            "size=512":
                {"I_MPI_ADJUST_BCAST":                                  "2"},
            ...
        }
    }
}
```

This tree representation is an intermediate format of tuning results and is ultimately converted to a string that the library can understand. The conversion script is specified with `-tree-postprocess` option.

-tree-ignore

Specifies the variables to ignore from the `-tree` option definition.

-tree-intervals

Specifies the maximum number of intervals where the optimal parameter value is applied. If not specified, any number of intervals is allowed.

-tree-tolerance

Specifies the tolerance level. Non-zero tolerance (for example, 0.03 for 3%) joins resulting intervals with the performance indicator value varying by the specified tolerance.

-tree-postprocess

Specifies an executable to convert the resulting JSON tree to a custom format.

-tree-opt

Specifies the optimization direction. The available values are `max` (default) and `min`.

-tree-file

Specifies a log file where the tuning results are saved.

-tree-view

Specify the mode to present the json-tree. The available values are “simple” and “default”. The “default” mode enables an interpolation mechanism; the “simple” mode disables the interpolation mechanism. The resulting tree contains message sizes used during the launch.

-mode

Specifies the mpitune mode. The available values are “collect” for gathering data and “analyze” for converting this data to a JSON-tree. Note that the -mode field can be defined in the configuration file as macros @-mode@, although the real value must be defined in the command line.

-dump-file

Specifies the path for the dump-file, which is returned by mpitune after the first iteration. The first iteration can be initialized by way of “” (an nempty string). Note that the -dump-file field can be defined in the configuration file as macros @-dump-file@, although the real value must be defined in the command line.

Environment Variable Reference

Compilation Environment Variables

I_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>
```

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}

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>`

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	Multi-threaded optimized library (with the global lock). This is the default value

dbg	Multi-threaded debug library (with the global lock)
opt_mt	Multi-threaded optimized library (with per-object lock for the thread-split model)
dbg_mt	Multi-threaded debug library (with per-object lock for the thread-split model)

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.

Hydra Environment Variables

I_MPI_HYDRA_HOST_FILE

Set the host file to run the application.

Syntax

`I_MPI_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_HOSTS_GROUP

Set node ranges using brackets, commas, and dashes.

Syntax

`I_MPI_HYDRA_HOSTS_GROUP=`*<arg>*

Argument

<arg>	Set a node range.

Description

Set this variable to be able to set node ranges using brackets, commas, and dashes (like in Slurm* Workload Manager). For example:

```
I_MPI_HYDRA_HOSTS_GROUP="hostA[01-05],hostB,hostC[01-05,07,09-11]"
```

You can set node ranges with the `-hosts-group` option.

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

Set the timeout period for `mpiexec.hydra`.

Syntax

`I_MPI_JOB_TIMEOUT=`*<timeout>*
`I_MPI_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_STARTUP_TIMEOUT

Set the `mpiexec.hydra` job startup timeout.

Syntax

`I_MPI_JOB_STARTUP_TIMEOUT=`*<timeout>*

Arguments

<timeout>	Define `mpiexec.hydra` startup 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 if some processes are not launched. The *<timeout>* value should be greater than zero.

I_MPI_JOB_TIMEOUT_SIGNAL

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

Syntax

`I_MPI_JOB_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

Control signal propagation.

Syntax

`I_MPI_JOB_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
`jmi`	Use Job Manager Interface (tighter integration)

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.

Note

If the launcher (blaunch, lsf, pdsh, pbsdsh) falls back to ssh, pass the arguments with the invocation of ssh.

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 | 0`	Do not use `fork` for the local processes. This is default value for the `sge` 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` and `pdsh` bootstrap servers.

I_MPI_HYDRA_RMK

Use the specified value as the resource management kernel to obtain data about available nodes, externally set process counts.

Syntax

`I_MPI_HYDRA_RMK=<arg>`

Arguments

<arg>	String parameter
<rmk>	Resource management kernel. The supported values are `slurm`, `ll`, `lsf`, `sge`, `pbs`, `cobalt`.

Description

Set this environment variable to use the resource management kernel.

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_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>`

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> >=	• The default value is `-1` if less than 128 nodes are used. This value also means that

0	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_TMPDIR

Specify a temporary directory.

Syntax

`I_MPI_TMPDIR=`*<arg>*

Arguments

<arg>	String parameter
<path>	Temporary directory. The default value is `/tmp`

Description

Set this environment variable to specify a directory for temporary files.

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.

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_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_PORT_RANGE

Specify a range of allowed port numbers.

Syntax

I_MPI_PORT_RANGE=*<range>*

Arguments

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

Description

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

I_MPI_SILENT_ABORT

Control abort warning messages.

Syntax

`I_MPI_SILENT_ABORT=<arg>`

Argument

<arg>	Binary indicator
`enable | yes | on | 1`	Do not print abort warning message
`disable | no | off | 0`	Print abort warning message. This is the default value

Description

Set this variable to disable printing of abort warning messages. The messages are printed in case of the MPI_Abort call.

You can also disable printing of these messages with the `-silent-abort` option.

I_MPI_HYDRA_NAMESERVER

Specify the nameserver.

Syntax

`I_MPI_HYDRA_NAMESERVER=<arg>`

Argument

<arg>	String parameter
<hostname>:<port>	Set the hostname and the port.

Description

Set this variable to specify the nameserver for your MPI application in the following format:

```
I_MPI_HYDRA_NAMESERVER = hostname:port
```

You can set the nameserver with the `-nameserver` option.

I_MPI_ADJUST Family Environment Variables

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.

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 12. NUMA aware SHM-based (SSE4.2) 13. NUMA aware SHM-based (AVX2) 14. NUMA aware SHM-based (AVX512)
`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 7. Reduce scatter allgather 8. SMP 9. Nreduce
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 5. SMP 6. Tree knominal 7. Tree kary
I_MPI_ADJUST_IEXSCAN	MPI_Iexscan	1. Recursive doubling 2. SMP
I_MPI_ADJUST_IGATHER	MPI_Igather	1. Binomial 2. Knomial
I_MPI_ADJUST_IGATHERV	MPI_Igatherv	1. Linear 2. Linear ssend
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	1. Recursive Doubling 2. SMP
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.

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_<opname>_LIST

Syntax

`I_MPI_ADJUST_<opname>_LIST=<algid1>[-<algid2>][,<algid3>][,<algid4>-<algid5>]`

Description

Set this environment variable to specify the comma-separated list of ranges. The list has to be ordered.

Note: Setting an empty string disables autotuning for the <opname> collective.

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_COLL_EXTERNAL

Syntax

`I_MPI_COLL_EXTERNAL=<arg>`

Arguments

<arg>	Binary indicator.
enable | yes | on | 1	Enable the external collective operations functionality.
disable | no | off | 0	Disable the external collective operations functionality. This is the default value.

Description

Set this environment variable to enable external collective operations. The mechanism allows to enable HCOLL. The functionality enables the following collective operations: `I_MPI_ADJUST_ALLREDUCE=24, I_MPI_ADJUST_BARRIER=11, I_MPI_ADJUST_BCAST=16, I_MPI_ADJUST_REDUCE=13, I_MPI_ADJUST_ALLGATHER=6, I_MPI_ADJUST_ALLTOALL=5, I_MPI_ADJUST_ALLTOALLV=5, I_MPI_ADJUST_SCAN=3, I_MPI_ADJUST_EXSCAN=3, I_MPI_ADJUST_GATHER=5, I_MPI_ADJUST_GATHERV=4, I_MPI_ADJUST_SCATTER=5, I_MPI_ADJUST_SCATTERV=4, I_MPI_ADJUST_ALLGATHERV=5, I_MPI_ADJUST_ALLTOALLW=2, I_MPI_ADJUST_REDUCE_SCATTER=6, I_MPI_ADJUST_REDUCE_SCATTER_BLOCK=4, I_MPI_ADJUST_IALLGATHER=5, I_MPI_ADJUST_IALLGATHERV=5, I_MPI_ADJUST_IGATHERV=3, I_MPI_ADJUST_IALLREDUCE=9, I_MPI_ADJUST_IALLTOALLV=2, I_MPI_ADJUST_IBARRIER=2, I_MPI_ADJUST_IBCAST=5, I_MPI_ADJUST_IREDUCE=4.`

I_MPI_CBWR

Control reproducibility of floating-point operations results across different platforms, networks, and topologies in case of the same number of processes.

Syntax

`I_MPI_CBWR=<arg>`

Arguments

<arg>	CBWR compatibility mode	Description
0	None	Do not use CBWR in a library-wide mode. CNR-safe communicators may be created with MPI_Comm_dup_with_info explicitly. This is the default value.
1	Weak mode	Disable topology aware collectives. The result of a collective operation does not depend on the rank placement. The mode guarantees results reproducibility across different runs on the same cluster (independent of the rank placement).

2	Strict mode	Disable topology aware collectives, ignore CPU architecture, and interconnect during algorithm selection. The mode guarantees results reproducibility across different runs on different clusters (independent of the rank placement, CPU architecture, and interconnection)

Description

Conditional Numerical Reproducibility (CNR) provides controls for obtaining reproducible floating-point results on collectives operations. With this feature, Intel MPI collective operations are designed to return the same floating-point results from run to run in case of the same number of MPI ranks.

Control this feature with the I_MPI_CBWR environment variable in a library-wide manner, where all collectives on all communicators are guaranteed to have reproducible results. To control the floating-point operations reproducibility in a more precise and per-communicator way, pass the {"I_MPI_CBWR", "yes"} key-value pair to the MPI_Comm_dup_with_info call.

Note

Setting the I_MPI_CBWR in a library-wide mode using the environment variable leads to performance penalty.

CNR-safe communicators created using MPI_Comm_dup_with_info always work in the strict mode. For example:

```
MPI_Info                                                    hint;

MPI_Comm                    cbwr_safe_world,                cbwr_safe_copy;

MPI_Info_create(&hint);

MPI_Info_set(hint,                  "I_MPI_CBWR",                   "yes");

MPI_Comm_dup_with_info(MPI_COMM_WORLD,      hint,      &      cbwr_safe_world);

MPI_Comm_dup(cbwr_safe_world, & cbwr_safe_copy);
```

In the example above, both cbwr_safe_world and cbwr_safe_copy are CNR-safe. Use cbwr_safe_world and its duplicates to get reproducible results for critical operations.

Note that MPI_COMM_WORLD itself may be used for performance-critical operations without reproducibility limitations.

Tuning Environment Variables

Tuning Environment Variables

I_MPI_TUNING_MODE

Select the tuning method.

Syntax

`I_MPI_TUNING_MODE=<arg>`

Argument

<arg>	Description
none	Disable tuning modes. This is the default value.
auto	Enable autotuner.
auto:application	Enable autotuner with application focused strategy (alias for auto).
auto:cluster	Enable autotuner without application specific logic. This is typically performed with the help of benchmarks (for example, IMB-MPI1) and proxy applications.

Description

Set this environment variable to enable the autotuner functionality and set the autotuner strategy.

I_MPI_TUNING_BIN

Specify the path to tuning settings in a binary format.

Syntax

`I_MPI_TUNING_BIN=<path>`

Argument

<path>	A path to a binary file with tuning settings. By default, Intel® MPI Library uses the binary tuning file located at <$I_MPI_ROOT/intel64/etc>.

Description

Set this environment variable to load tuning settings in a binary format.

I_MPI_TUNING_BIN_DUMP

Specify the file for storing tuning settings in a binary format.

Syntax

`I_MPI_TUNING_BIN_DUMP=<filename>`

Argument

<filename>	A file name of a binary that stores tuning settings. By default, the path is not specified.

Description

Set this environment variable to store tuning settings in a binary format.

I_MPI_TUNING

Load tuning settings in a JSON format.

Syntax

`I_MPI_TUNING=<path>`

Argument

<path>	A path to a JSON file with tuning settings.

Description

Set this environment variable to load tuning settings in a JSON format.

Note

The tuning settings in the JSON format are produced by the mpitune utility.

By default, Intel® MPI library loads tuning settings in a binary format. If it is not possible, Intel MPI Library loads the tuning file in a JSON format specified through the `I_MPI_TUNING` environment variable.
Thus, to enable JSON tuning, turn off the default binary tuning: `I_MPI_BIN=""`. If it is not possible to load tuning settings from a JSON file and in a binary format, the default tuning values are used.
You do not need to turn off binary or JSON tuning settings if you use `I_MPI_ADJUST` family environment variables. The algorithms specified with `I_MPI_ADJUST` environment variables always have priority over binary and JSON tuning settings.

See Also

Autotuning

Environment Variables for Autotuning

Autotuning

Autotuning

Tuning is very dependent on the specifications of the particular platform. Intel carefully determines the tuning parameters for a limited set of platforms, and makes them available for autotuning using the `I_MPI_TUNING_MODE` environment variable.
A full list of the platforms supported with the `I_MPI_TUNING_MODE` environment variable is available in Tuning Environment Variables. This variable has no effect on platforms not included in this list. For such platforms, use I_MPI_TUNING_AUTO Family Environment Variables directly to find the best settings.
The autotuner functionality lets you automatically find the best algorithms for collective operations . The autotuner search space can be modified by `I_MPI_ADJUST_<opname>_LIST` variables from I_MPI_ADJUST Family Environment Variables.
The collectives currently available for autotuning are: `MPI_Allreduce`, `MPI_Bcast`, `MPI_Barrier`, `MPI_Reduce`, `MPI_Gather`, `MPI_Scatter`, `MPI_Alltoall`, `MPI_Allgatherv`, `MPI_Reduce_scatter`, `MPI_Reduce_scatter_block`, `MPI_Scan`, `MPI_Exscan`, `MPI_Iallreduce`, `MPI_Ibcast`, `MPI_Ibarrier`, `MPI_Ireduce`, `MPI_Igather`, `MPI_Iscatter`, `MPI_Ialltoall`, `MPI_Iallgatherv`, `MPI_Ireduce_scatter`, `MPI_Ireduce_scatter_block`, `MPI_Iscan`, `MPI_Iexscan`.

To get started with the tuner, follow these steps:

1. Launch the application with the autotuner enabled and specify the dump file, which stores results:

```
I_MPI_TUNING_MODE=auto
I_MPI_TUNING_BIN_DUMP=<tuning_results.dat>
```

2. Launch the application with the tuning results generated at the previous step:

```
I_MPI_TUNING_BIN=<tuning_results.dat>
```

3. Or use the `-tune` Hydra option.
4. If you experience performance issues, see Environment Variables for Autotuning.

For example:

1.

```
$                          export                          I_MPI_TUNING_MODE=auto
$                        export                          I_MPI_TUNING_AUTO_SYNC=1
$                      export                        I_MPI_TUNING_AUTO_ITER_NUM=5
$               export               I_MPI_TUNING_BIN_DUMP=./tuning_results.dat
$ mpirun -n 128 -ppn 64 IMB-MPI1 allreduce -iter 1000,800 -time 4800
```

3.

```
$                  export                  I_MPI_TUNING_BIN=./tuning_results.dat
$ mpirun -n 128 -ppn 64 IMB-MPI1 allreduce -iter 1000,800 -time 4800
```

Note
To tune collectives on a communicator identified with the help of Application Performance Snapshot (APS), execute the following variable at step 1: `I_MPI_TUNING_AUTO_COMM_LIST=comm_id_1, … , comm_id_n`.

See Also

Environment Variables for Autotuning

I_MPI_TUNING_AUTO Family Environment Variables

I_MPI_TUNING_AUTO_STORAGE_SIZE

Define size of the per-communicator tuning storage.

Syntax

`I_MPI_TUNING_AUTO_STORAGE_SIZE=<size>`

Argument

<size>	Specify size of the communicator tuning storage. The default size of the storage is 512

	Kb.

Description

Set this environment variable to change the size of the communicator tuning storage.

I_MPI_TUNING_AUTO_ITER_NUM

Specify the number of autotuner iterations.

Syntax

`I_MPI_TUNING_AUTO_ITER_NUM=<number>`

Argument

<number>	Define the number of iterations. By default, it is 1.

Description

Set this environment variable to specify the number of autotuner iterations. The greater iteration number produces more accurate results.

Note
To check if all possible algorithms are iterated, make sure that the total number of collective invocations for a particular message size in a target application is at least equal the value of `I_MPI_TUNING_AUTO_ITER_NUM` multiplied by the number of algorithms.

I_MPI_TUNING_AUTO_WARMUP_ITER_NUM

Specify the number of warmup autotuner iterations.

Syntax

`I_MPI_TUNING_AUTO_WARMUP_ITER_NUM=<number>`

Argument

<number>	Define the number of iterations. By default, it is 1.

Description

Set this environment variable to specify the number of autotuner warmup iterations. Warmup iterations do not impact autotuner decisions and allow to skip additional iterations, such as infrastructure preparation.

I_MPI_TUNING_AUTO_SYNC

Enable the internal barrier on every iteration of the autotuner.

Syntax

`I_MPI_TUNING_AUTO_SYNC=<arg>`

Argument

<arg>	Binary indicator

`enable | yes | on | 1`	Align the autotuner with the IMB measurement approach.
`disable | no | off | 0`	Do not use the barrier on every iteration of the autotuner. This is the default value.

Description

Set this environment variable to control the IMB measurement logic. Setting this variable to 1 may lead to overhead due to an additional MPI_Barrier call.

I_MPI_TUNING_AUTO_COMM_LIST

Control the scope of autotuning.

Syntax

`I_MPI_TUNING_AUTO_COMM_LIST=<comm_id_1, ..., comm_id_n>`

Argument

<comm_id_n, ...>	Specify communicators to be tuned.

Description

Set this environment variable to specify communicators to be tuned using their unique id. By default, the variable is not specified. In this case, all communicators in the application are involved into the tuning process.

Note
To get the list of communicators available for tuning, use the Application Performance Snapshot (APS) tool, which supports per communicator profiling starting the 2019 Update 4 release.

For example:

1. Source `apsvars.sh`:

```
$ source <path_to_aps>/apsvars.sh
```

2. Gather APS statistics:

```
$                          export                          MPS_STAT_LEVEL=5
$                        export                        APS_COLLECT_COMM_IDS=1
mpirun -aps -n 128 -ppn 64 IMB-MPI1 allreduce -npmin 128 -iter 1000,800 -time
4800
```

3. Generate an APS report:

```
$ aps-report aps_result_20190228/ -lFE
```

4. Get the results:

```
|         Communicators         used         in         the         application
|-----------------------------------------------------------------------------
| Communicator Id      Communicator Size      Time (Rank Average)(sec)  Ranks
|-----------------------------------------------------------------------------
  4611686018431582688    4                     1.80  (0.45)              0,1,2,3
|-----------------------------------------------------------------------------
  4611686018431582208    4                     0.59  (0.15)              0,1,2,3
|-----------------------------------------------------------------------------
  4611686018429485552    2                       0.51   (0.25)               0,1
|-----------------------------------------------------------------------------
  4611686018429485520    2                       0.01   (0.00)               0,1
|-----------------------------------------------------------------------------
  4611686018431582672    4                     0.00  (0.00)              0,1,2,3
|-----------------------------------------------------------------------------
```

5. Specify the communicators to be tuned:

```
$            export            I_MPI_TUNING_AUTO_COMM_LIST=4611686018431582688

$                           export                              MPS_STAT_LEVEL=5

$                         export                           APS_COLLECT_COMM_IDS=1

$                           export                              I_MPI_TUNING_AUTO=1

$ mpirun -aps -n 128 -ppn 64 IMB-MPI1 allreduce -iter 1000,800 -time 4800
```

I_MPI_TUNING_AUTO_COMM_DEFAULT

Mark all communicators with the default value.

Syntax

`I_MPI_TUNING_AUTO_COMM_DEFAULT=<arg>`

Argument

<arg>	Binary indicator
`enable | yes | on | 1`	Mark communicators.
`disable | no | off | 0`	Do not mark communicators. This is the default value.

Description

Set this environment variable to mark all communicators in an application with the default value. In this case, all communicators will have the identical default comm_id equal to -1.

I_MPI_TUNING_AUTO_COMM_USER

Enable communicator marking with a user value.

Syntax

`I_MPI_TUNING_AUTO_COMM_USER=<arg>`

Argument

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

Description

Set this environment variable to enable communicator marking with a user value. To mark a communicator in your application, use the MPI_Info object for this communicator that contains a record with the comm_id key. The key must belong the `0...UINT64_MAX` range .

I_MPI_TUNING_AUTO_ITER_POLICY

Control the iteration policy logic.

Syntax

`I_MPI_TUNING_AUTO_ITER_POLICY=<arg>`

Argument

<arg>	Binary indicator
`enable | yes | on | 1`	Reduce the number of iterations with a message size increase after 64Kb (by half). This is the default value.
`disable | no | off | 0`	Use the `I_MPI_TUNING_AUTO_ITER_NUM` value. This value affects warmup iterations.

Description

Set this environment variable to control the autotuning iteration policy logic.

I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD

Control the message size limit for the `I_MPI_TUNING_AUTO_ITER_POLICY` environment variable.

Syntax

`I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD=<arg>`

Argument

<arg>	Define the value. By default, it is 64KB.

Description

Set this environment variable to control the message size limit for the autotuning iteration policy logic (`I_MPI_TUNING_AUTO_ITER_POLICY`).

I_MPI_TUNING_AUTO_POLICY

Choose the best algorithm identification strategy.

Syntax

`I_MPI_TUNING_AUTO_POLICY=<arg>`

Argument

<arg>	Description
max	Choose the best algorithm based on a maximum time value. This is the default value.
min	Choose the best algorithm based on a minimum time value.
avg	Choose the best algorithm based on an average time value.

Description

Set this environment variable to control the autotuning strategy and choose the best algorithm based on the time value across ranks involved into the tuning process.

Process Pinning

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

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.

Logical Enumeration

0	4	1	5	2	6	3	7

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

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.

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`

Note

If I_MPI_PIN_ORDER is not specified and one of the sockets (NUMA-nodes) is not used, for better performance the 'bunch' order will automatically be used instead of the default ‘compact’ order.

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 process pinning

Description

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

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.

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	2n+2	...	4n-2	4n	4n+2	...	6n-

			2n-1	2n+1	2n+3		4n-1	4n+1	4n+3		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

cy an - 1st socket processors

gre en - 2nd socket processors

Same color defines a processor pair sharing a cache

0	1	2			3	4		
0	1	2	3		4	5	6	7

`bunch` scenario for 5 processes

0	4	2	6		1	5	3	7
0	1	2	3		4	5	6	7

`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 job manager's 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 uses its own 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

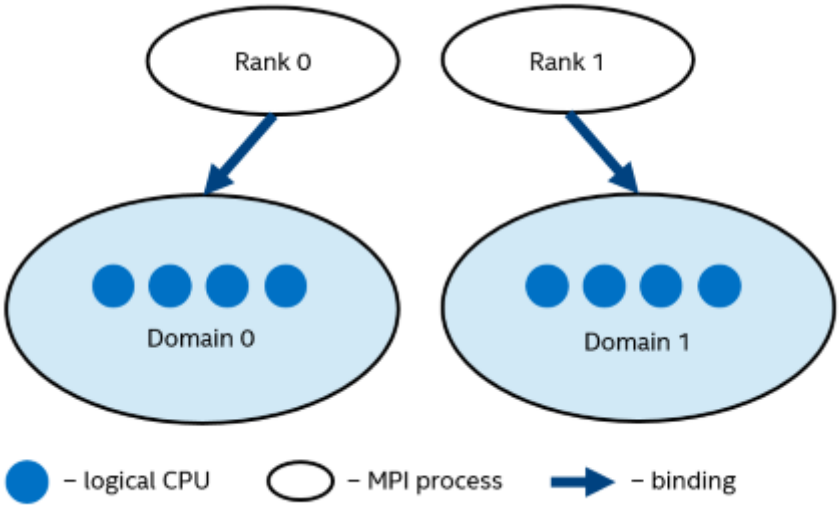
If you set `I_MPI_PIN_RESPECT_HCA=enable` , the Hydra process launcher uses the location of IBA HCA on each intended host for applying Intel MPI Library pinning capability.
If you set `I_MPI_PIN_RESPECT_HCA=disable`, the Hydra process launcher does not use the location of IBA HCA on each intended host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

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 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=[55,aa]`

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

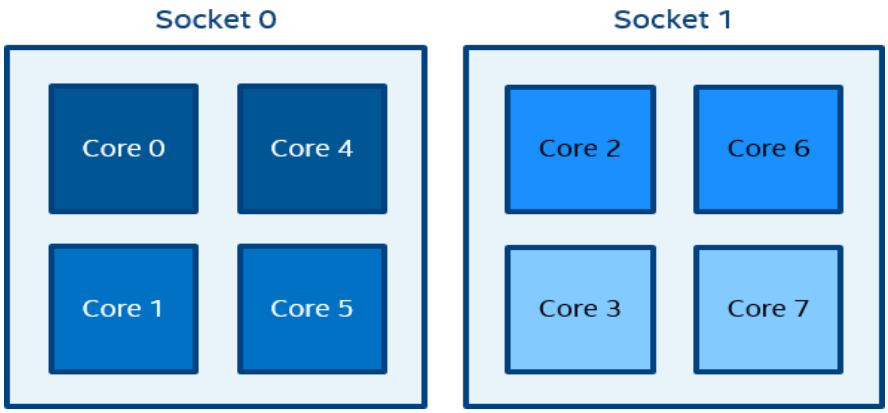
The following configurations are effectively the same as if pinning is not applied:

- 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).

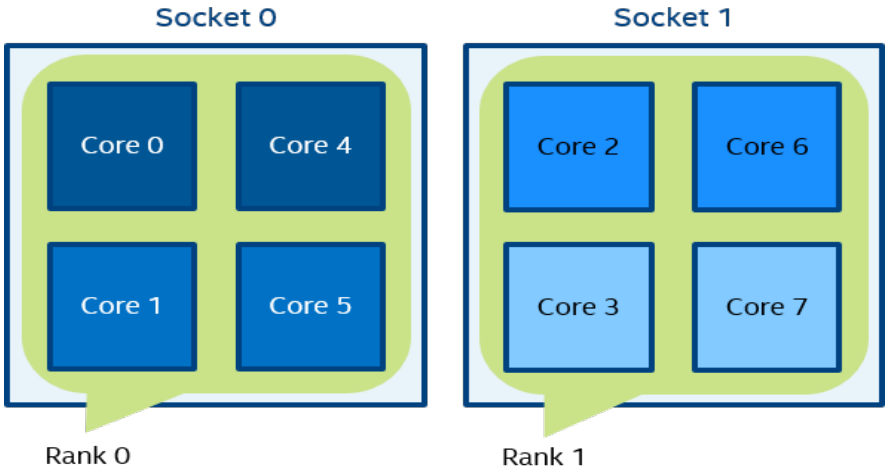
See the following model of a symmetric multiprocessing (SMP) node in the examples:

Figure 2 Model of a Node



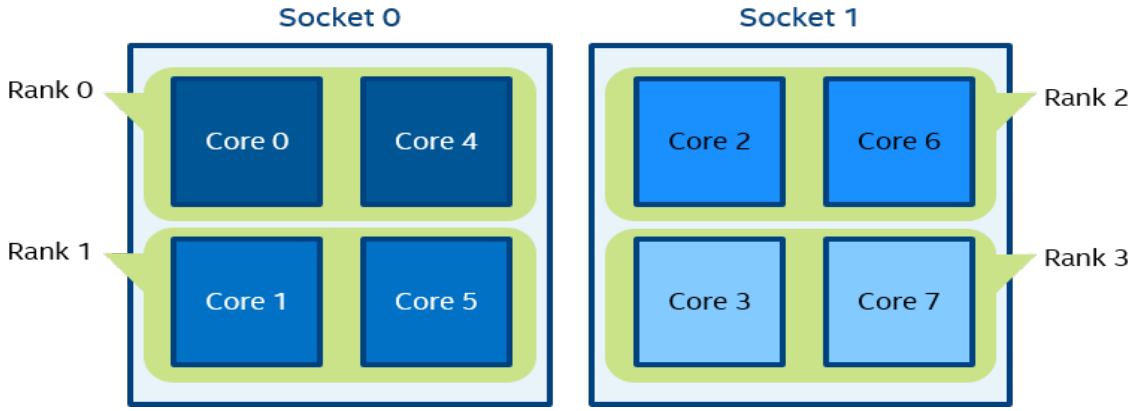
The figure above represents the SMP node model with a total of 8 cores on 2 sockets. Intel® Hyper-Threading Technology is disabled. Core pairs of the same color share the L2 cache.

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



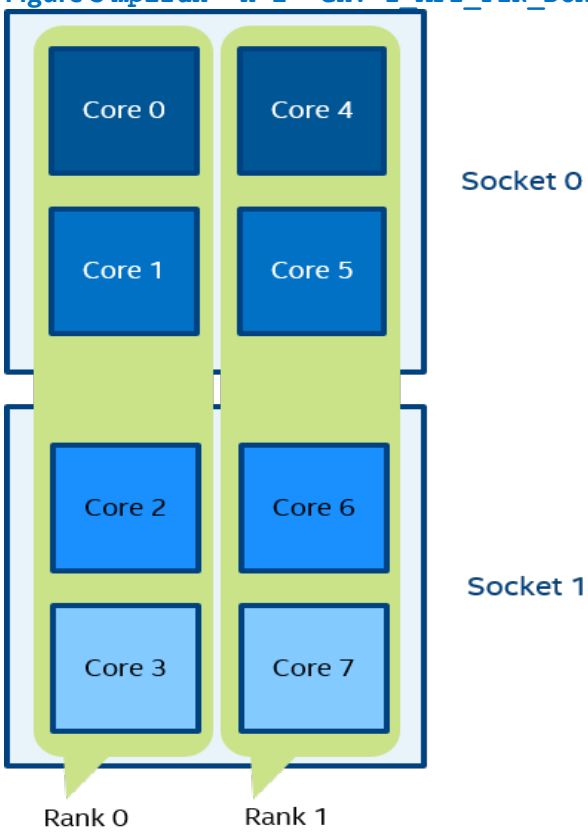
In Figure 3, two domains are defined according to the number of sockets. Process rank 0 can migrate on all cores on the 0-th socket. Process rank 1 can migrate on all cores on the first socket.

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



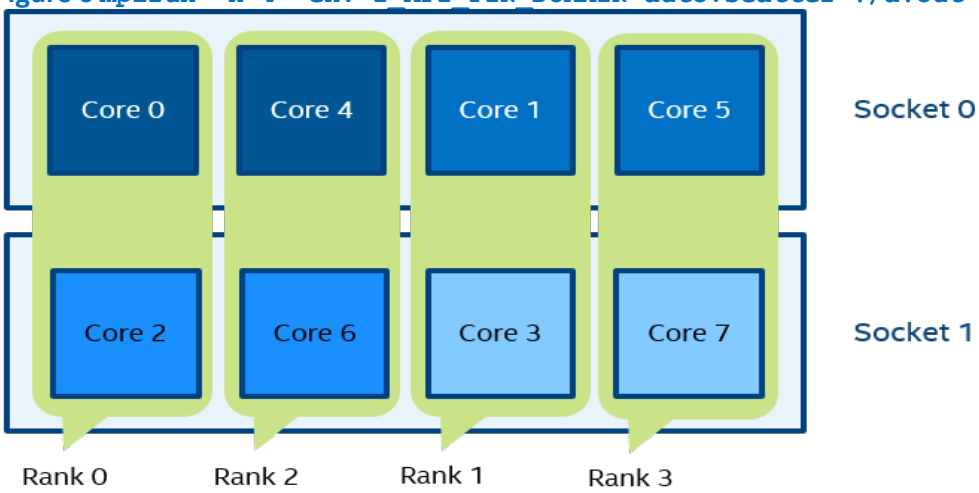
In Figure 4, four domains are defined according to the amount of common L2 caches. Process rank 0 runs on cores {0,4} that share an L2 cache. Process rank 1 runs on cores {1,5} that share an L2 cache as well, and so on.

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



In Figure 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.

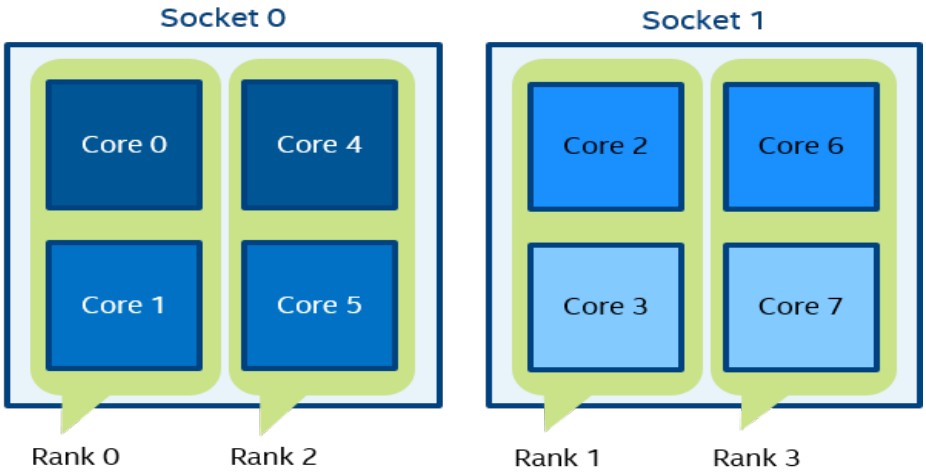
Figure 6 `mpirun -n 4 -env I_MPI_PIN_DOMAIN auto:scatter ./a.out`



In Figure 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.

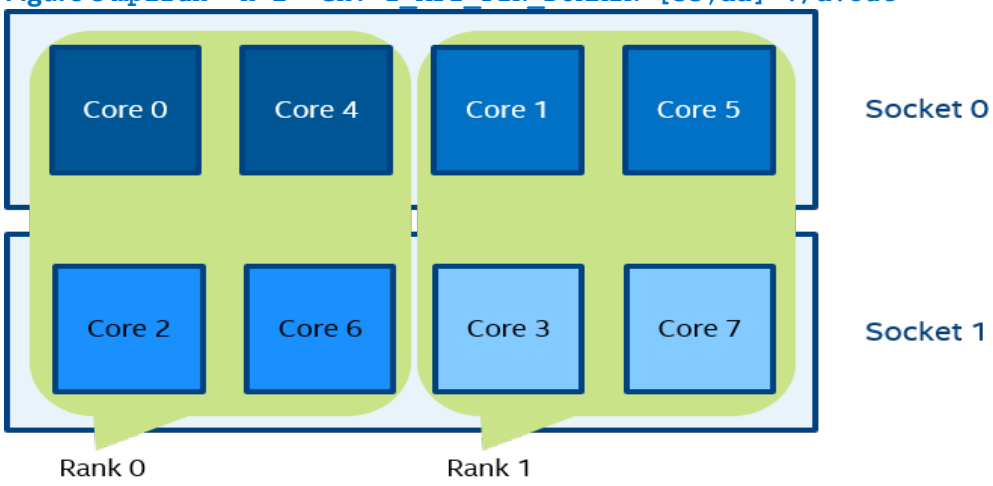
Figure 7 setenv OMP_NUM_THREADS=2

```
mpirun -n 4 -env I_MPI_PIN_DOMAIN omp:platform ./a.out
```



In Figure 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 8 `mpirun -n 2 -env I_MPI_PIN_DOMAIN [55,aa] ./a.out`



In Figure 8 (the example for `I_MPI_PIN_DOMAIN=<masklist>`), the first domain is defined by the 55 mask. It contains all cores with even numbers {0,2,4,6}. The second domain is defined by the AA 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, whenever possible.
`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

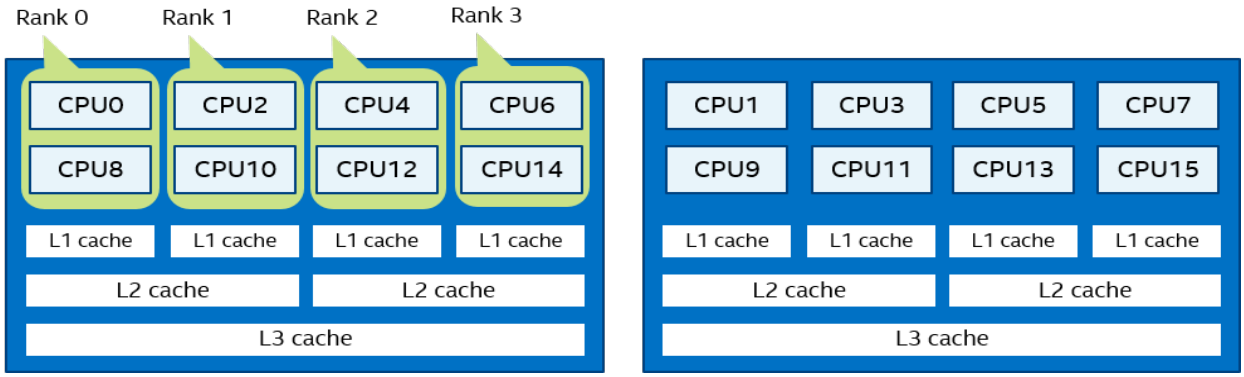
For the following configuration:

- 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
```

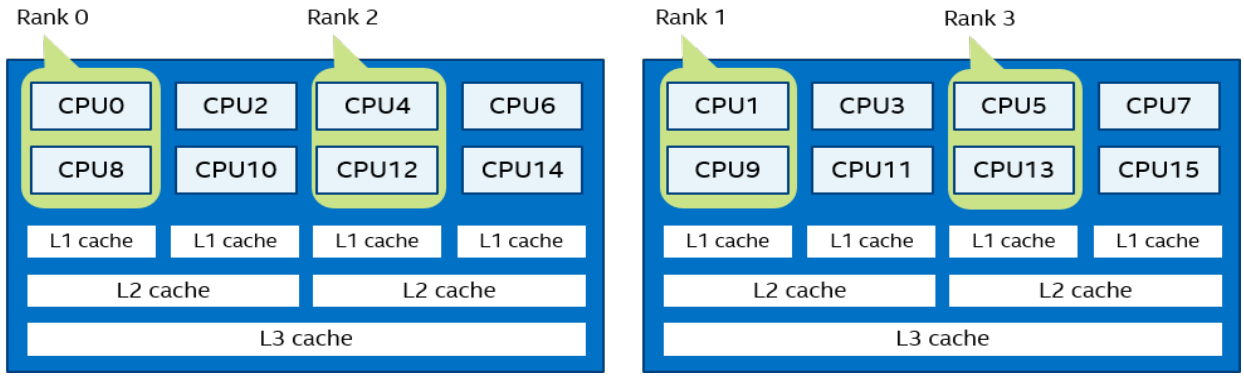
Figure 9 Compact Order Example



Scatter order:

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

Figure 10 Scatter Order Example



Spread order:

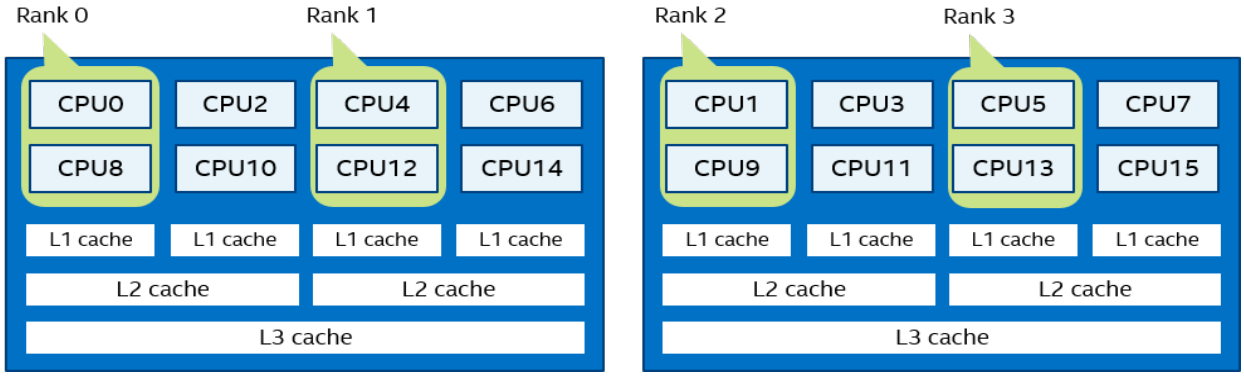
```
I_MPI_PIN_DOMAIN=2
I_MPI_PIN_ORDER=spread
```

Note

For I_MPI_PIN_ORDER=spread, the order will be switched to 'compact' if:

- there are not enough CPUs to emplace all domains
- different domains share the L1 cache

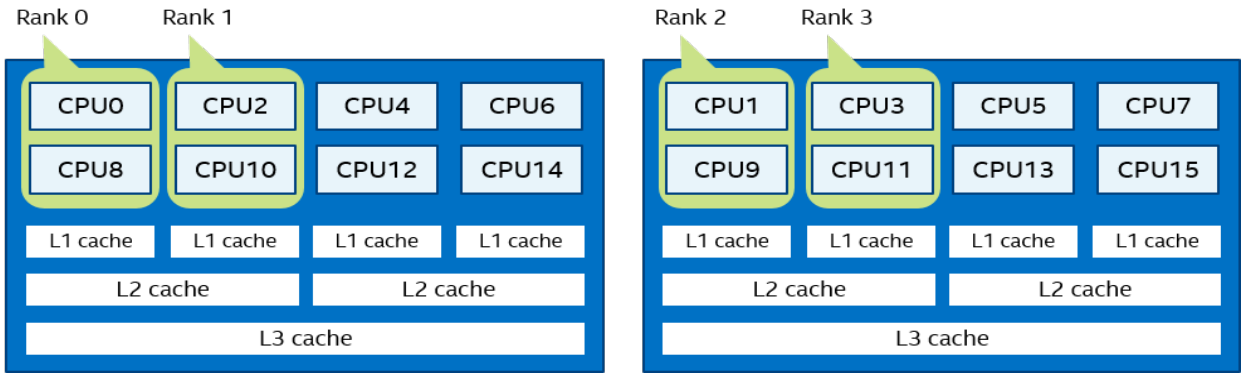
Figure 11 Spread Order Example



Bunch order:

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

Figure 12 Bunch Order Example



Environment Variables for Fabrics Control

Communication Fabrics Control

I_MPI_FABRICS

Select the particular fabrics to be used.

Syntax

```
I_MPI_FABRICS=ofi | shm:ofi | shm
```

Arguments

<fabric>	Define a network fabric.
shm	Shared memory transport (used for intra-node communication only).
ofi	OpenFabrics Interfaces* (OFI)-capable network fabrics, such as Intel® True Scale

	Fabric, Intel® Omni-Path Architecture, InfiniBand*, and Ethernet (through OFI API).

Description

Set this environment variable to select a specific fabric combination.

The default values are `shm:ofi` for the regular mode and `ofi` for the multiple endpoints mode. In the multiple endpoints mode, the default value `ofi` cannot be changed.

Note

DAPL, TMI, and OFA fabrics are deprecated.

Note

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

Shared Memory Control

I_MPI_SHM

Select a shared memory transport to be used.

Syntax

```
I_MPI_SHM=<transport>
```

Arguments

`<transport>`	Define a shared memory transport solution.
`disable | no | off | 0`	Do not use shared memory transport.
`auto`	Select a shared memory transport solution automatically.
`bdw_sse`	The shared memory transport solution tuned for Intel® microarchitecture code name Broadwell. The SSE4.2. instruction set is used.
`bdw_avx2`	The shared memory transport solution tuned for Intel® microarchitecture code name Broadwell. The AVX2 instruction set is used.
`skx_sse`	The shared memory transport solution tuned for Intel® Xeon® processors based on Intel® microarchitecture code name Skylake. The CLFLUSHOPT and SSE4.2 instruction set is used.
`skx_avx2`	The shared memory transport solution tuned for Intel® Xeon® processors based on Intel® microarchitecture code name Skylake. The CLFLUSHOPT and AVX2 instruction set is used.
`skx_avx512`	The shared memory transport solution tuned for Intel® Xeon® processors based on Intel® microarchitecture code name Skylake. The CLFLUSHOPT and AVX512 instruction set is used.

knl_ddr	The shared memory transport solution tuned for Intel® microarchitecture code name Knights Landing.
knl_mcdram	The shared memory transport solution tuned for Intel® microarchitecture code name Knights Landing. Shared memory buffers may be partially located in the Multi-Channel DRAM (MCDRAM).
clx_sse	The shared memory transport solution tuned for Intel® Xeon® processors based on Intel® microarchitecture code name Cascade Lake. The CLFLUSHOPT and SSE4.2 instruction set is used.
clx_avx2	The shared memory transport solution tuned for Intel® Xeon® processors based on Intel® microarchitecture code name Cascade Lake. The CLFLUSHOPT and AVX2 instruction set is used.
clx_avx512	The shared memory transport solution tuned for Intel® Xeon® processors based on Intel® microarchitecture code name Cascade Lake. The CLFLUSHOPT and AVX512 instruction set is used.

Description

Set this environment variable to select a specific shared memory transport solution.

Automatically selected transports:

- bdw_avx2 for Intel® microarchitecture code name Haswell, Broadwell and Skylake
- skx_avx2 for Intel® Xeon® processors based on Intel® microarchitecture code name Skylake
- ckx_avx2 for Intel® Xeon® processors based on Intel® microarchitecture code name Cascade Lake
- knl_mcdram for Intel® microarchitecture code name Knights Landing and Knights Mill
- bdw_sse for all other platforms

The value of I_MPI_SHM depends on the value of I_MPI_FABRICS as follows: if I_MPI_FABRICS is ofi, I_MPI_SHM is disabled. If I_MPI_FABRICS is shm:ofi, I_MPI_SHM defaults to auto or takes the specified value.

I_MPI_SHM_CELL_FWD_SIZE

Change the size of a shared memory forward cell.

Syntax

I_MPI_SHM_CELL_FWD_SIZE=<nbytes>

Arguments

<nbytes>	The size of a shared memory forward cell in bytes
> 0	The default <nbytes> value depends on the transport used and should normally range from 64K to 1024K.

Description

Forward cells are in-cache message buffer cells used for sending small amounts of data. Lower values are recommended. Set this environment variable to define the size of a forward cell in the shared memory transport.

I_MPI_SHM_CELL_BWD_SIZE

Change the size of a shared memory backward cell.

Syntax

`I_MPI_SHM_CELL_BWD_SIZE=<nbytes>`

Arguments

`<nbytes>`	The size of a shared memory backward cell in bytes
> 0	The default `<nbytes>` value depends on the transport used and should normally range from 64K to 1024K.

Description

Backward cells are out-of-cache message buffer cells used for sending large amounts of data. Higher values are recommended. Set this environment variable to define the size of a backwrad cell in the shared memory transport.

I_MPI_SHM_CELL_EXT_SIZE

Change the size of a shared memory extended cell.

Syntax

`I_MPI_SHM_CELL_EXT_SIZE=<nbytes>`

Arguments

`<nbytes>`	The size of a shared memory extended cell in bytes
> 0	The default `<nbytes>` value depends on the transport used and should normally range from 64K to 1024K.

Description

Extended cells are used in the imbalanced applications when forward and backward cells are run out. An extended cell does not have a specific owner - it is shared between all ranks on the computing node. Set this environment variable to define the size of an extended cell in the shared memory transport.

I_MPI_SHM_CELL_FWD_NUM

Change the number of forward cells in the shared memory transport (per rank).

Syntax

`I_MPI_SHM_CELL_FWD_NUM=<num>`

Arguments

`<num>`	The number of shared memory forward cells
> 0	The default value depends on the transport used and should normally range from 4 to 16.

Description

Set this environment variable to define the number of forward cells in the shared memory transport.

I_MPI_SHM_CELL_BWD_NUM

Change the number of backward cells in the shared memory transport (per rank).

Syntax

`I_MPI_SHM_CELL_BWD_NUM=<num>`

Arguments

`<num>`	The number of shared memory backward cells
> 0	The default value depends on the transport used and should normally range from 4 to 64.

Description

Set this environment variable to define the number of backward cells in the shared memory transport.

I_MPI_SHM_CELL_EXT_NUM_TOTAL

Change the total number of extended cells in the shared memory transport.

Syntax

`I_MPI_SHM_CELL_EXT_NUM_TOTAL=<num>`

Arguments

`<num>`	The number of shared memory backward cells
> 0	The default value depends on the transport used and should normally range from 2K to 8K.

Description

Set this environment variable to define the number of extended cells in the shared memory transport.

Note

This is not “per rank” number, it is total number of extended cells on the computing node.

I_MPI_SHM_CELL_FWD_HOLD_NUM

Change the number of hold forward cells in the shared memory transport (per rank).

Syntax

`I_MPI_SHM_CELL_FWD_HOLD_NUM=<num>`

Arguments

`<num>`	The number of shared memory hold forward cells
> 0	The default value depends on the transport used and must be less than `I_MPI_SHM_CELL_FWD_NUM`.

Description

Set this environment variable to define the number of forward cells in the shared memory transport a rank can hold at the same time. Recommended values are powers of two in the range between 1 and 8.

I_MPI_SHM_MCDRAM_LIMIT

Change the size of the shared memory bound to the multi-channel DRAM (MCDRAM) (size per rank).

Syntax

`I_MPI_SHM_MCDRAM_LIMIT=<nbytes>`

Arguments

`<nbytes>`	The size of the shared memory bound to MCDRAM per rank
1048576	This is the default value.

Description

Set this environment variable to define how much MCDRAM memory per rank is allowed for the shared memory transport. This variable takes effect with `I_MPI_SHM=knl_mcdram` only.

I_MPI_SHM_SEND_SPIN_COUNT

Control the spin count value for the shared memory transport for sending messages.

Syntax

`I_MPI_SHM_SEND_SPIN_COUNT=<count>`

Arguments

`<count>`	Define the spin count value. A typical value range is between 1 and 1000.

Description

If the recipient ingress buffer is full, the sender may be blocked until this spin count value is reached. It has no effect when sending small messages.

I_MPI_SHM_RECV_SPIN_COUNT

Control the spin count value for the shared memory transport for receiving messages.

Syntax

`I_MPI_SHM_RECV_SPIN_COUNT=<count>`

Arguments

`<count>`	Define the spin count value. A typical value range is between 1 and 1000000.

Description

If the receive is non-blocking, this spin count is used only for safe reorder of expected and unexpected messages. It has no effect on receiving small messages.

I_MPI_SHM_FILE_PREFIX_4K

Change the mount point of the 4 KB pages size file system (`tmpfs`) where the shared memory files are created.

Syntax

`I_MPI_SHM_FILE_PREFIX_4K=<path>`

Arguments

`<path>`	Define the path to the existed mount point of the 4 KB pages size file system (`tmpfs`). By

	default, the path is not set.

Description

Set this environment variable to define a new path to the shared memory files. By default, the shared memory files are created at `/dev/shm/`.
This variable affects shared memory transport buffers and RMA windows.

Example

```
I_MPI_SHM_FILE_PREFIX_4K=/dev/shm/intel/
```

I_MPI_SHM_FILE_PREFIX_2M

Change the mount point of the 2 MB pages size file system (`hugetlbfs`) where the shared memory files are created.

Syntax

`I_MPI_SHM_FILE_PREFIX_2M=<path>`

Arguments

`<path>`	Define the path to the existed mount point of the 2 MB pages size file system (`hugetlbfs`). By default, the path is not set.

Description

Set this environment variable to enable 2 MB huge pages on the Intel MPI Library.

The variable affects shared memory transport buffers. It may affect RMA windows as well if the windows size is greater than or equal to 2 MB.

Example

```
I_MPI_SHM_FILE_PREFIX_2M=/dev/hugepages
```

Note

The root privileges are required to configure the huge pages subsystem. Contact your system administrator to obtain permission.

I_MPI_SHM_FILE_PREFIX_1G

Change the mount point of the 1 GB pages size file system (`hugetlbfs`) where the shared memory files are created.

Syntax

`I_MPI_SHM_FILE_PREFIX_1G=<path>`

Arguments

`<path>`	Define the path to the existed mount point of the 1 GB pages size file system (`hugetlbfs`). By default, the path is not set.

Description

Set this environment variable to enable 1 GB huge pages on the Intel MPI Library.

The variable affects shared memory transport buffers. It may affect RMA windows as well if the windows size is greater than or equal to 1 GB.

Example

```
I_MPI_SHM_FILE_PREFIX_1G=/dev/hugepages1G
```

Note

The root privileges are required to configure the huge pages subsystem. Contact your system administrator to obtain permission.

OFI*-capable Network Fabrics Control

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` <<<<<<< HEAD environment variable. If you set the wrong name for an available provider, use `FI_LOG_LEVEL=debug` to get a hint to set the name correctly.
======= environment variable. If you set a wrong name of an available provider, use `FI_LOG_LEVEL=debug` to get a hint to correct the name.
>>>>>>> 9bf233a4b6f0b2ff79ee96c6b6427f461e650274

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_LIBRARY_INTERNAL

Control the usage of libfabric* shipped with the Intel® MPI Library.

Syntax

`I_MPI_OFI_LIBRARY_INTERNAL=<arg>`

Arguments

<arg>	Binary indicator
`enable	yes	on	1`	Use libfabric from the Intel MPI Library
`disable	no	off	0`	Do not use libfabric from the Intel MPI Library

Description

Set this environment variable to disable or enable usage of libfabric from the Intel MPI Library. The variable must be set before sourcing the `mpivars.[c]sh` script.

Example

```
$ export I_MPI_OFI_LIBRARY_INTERNAL=1
$ source <installdir>/intel64/bin/mpivars.sh
```

Setting this variable is equivalent to passing the `-ofi_internal` option to the `mpivars.[c]sh` script.

For more information, refer to the Intel® MPI Library Developer Guide, section *Running Applications > Libfabric* Support.*

Environment Variables for Memory 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`.

Environment Variables for Asynchronous Progress Control

Note

This feature is supported for the `release_mt` and `debug_mt` library configurations only. To specify the configuration, run the following command:
`$ source <installdir>/intel64/bin/mpivars.sh release_mt`

I_MPI_ASYNC_PROGRESS

Control the usage of progress threads.

Syntax

`I_MPI_ASYNC_PROGRESS=<arg>`

Arguments

<arg>	Binary indicator
`disable | no | off | 0`	Disable asynchronous progress threads for each rank. This is the default value.
`enable | yes | on | 1`	Enable asynchronous progress threads.

Description

Set this environment variable to enable asynchronous progress. If disabled, the `I_MPI_ASYNC_PROGRESS_*` knobs are ignored.

I_MPI_ASYNC_PROGRESS_THREADS

Control the number of asynchronous progress threads.

Syntax

`I_MPI_ASYNC_PROGRESS_THREADS=<arg>`

Arguments

<nthreads>	Define the number of progress threads. The default value is 1.

Description

Set this environment variable to control the number of asynchronous progress threads for each rank.

I_MPI_ASYNC_PROGRESS_PIN

Control the asynchronous progress threads pinning.

Syntax

`I_MPI_ASYNC_PROGRESS_PIN=<arg>`

Arguments

<arg>	Comma-separated list of logical processors
<CPU list>	Pin all progress threads of local processes to the listed CPUs. By default, N progress threads are pinned to the last N logical processors.

Description

Set this environment variable to control pinning for all progress threads of local processes.

Example

```
I_MPI_ASYNC_PROGRESS_THREADS=2

I_MPI_ASYNC_PROGRESS_PIN="0,1,2,3,4,5"
```

In case of three MPI processes per node, progress threads of the first process are pinned to 0, 1, second are pinned to 2, 3, and third are pinned to 4, 5.

Note

Exclude selected processors for progress threads from pinning of computational threads to avoid oversubscription of cores.

I_MPI_ASYNC_PROGRESS_ID_KEY

Set the MPI info object key that is used to explicitly define the progress thread id for a communicator.

Syntax

`I_MPI_ASYNC_PROGRESS_ID_KEY=<arg>`

Arguments

<key>	MPI info object key. The default value is `thread_id`.

Description

Set this environment variable to control the MPI info object key that is used to define the progress thread id for a communicator. The progress thread id is used for work distribution between progress threads. By default, communication goes over the first progress thread.

For more information and examples, refer to the Intel® MPI Library Developer Guide, section *Additional Supported Features > Asynchronous Progress Control.*

Environment Variables for Multi-EP

Note

This feature is supported for the `release_mt` and `debug_mt` library configurations only. To specify the configuration, run the following command:
`$ source <installdir>/intel64/bin/mpivars.sh release_mt`

I_MPI_THREAD_SPLIT

Syntax

`I_MPI_THREAD_SPLIT=<value>`

Arguments

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

Description

Use this environment variable to control the `I_MPI_THREAD_SPLIT` programming model.

I_MPI_THREAD_RUNTIME

Syntax

`I_MPI_THREAD_RUNTIME=<value>`

Arguments

<value>	Thread runtime
`generic`	Enable runtime support (for example, pthreads, TBB). This is the default value if OpenMP* cannot be detected at runtime
`openmp`	Enable OpenMP* runtime support. This is the default value if OpenMP is detected at runtime.

Description

Use this environment variable to control threading runtime support.

Note

This knob works if `I_MPI_THREAD_SPLIT` model support is enabled.

I_MPI_THREAD_MAX

Syntax

`I_MPI_THREAD_MAX=<int>`

Arguments

`<int>`	The maximum number of threads per rank. The default value is `omp_get_max_threads()` if `I_MPI_THREAD_RUNTIME` is set to `openmp`, 1 otherwise

Description

Use this environment variable to set the maximum number of threads to be used in each process concurrently.

I_MPI_THREAD_ID_KEY

Syntax

`I_MPI_THREAD_ID_KEY=<string>`

Arguments

`<string>`	Define the MPI info object key. The default value is `thread_id`

Description

Use this environment variable to set the MPI info object key that is used to explicitly define the logical thread number `thread_id`.

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 the level of debug information provided
`0`	Output no debugging information. This is the default value.
`1,2`	Output libfabric* version and provider.
`3`	Output effective MPI rank, `pid` and node mapping table.
`4`	Output process pinning information.
`5`	Output environment variables specific to Intel® MPI Library.
`> 5`	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_STATS

Collect MPI statistics from your application using Application Performance Snapshot.

Syntax

`I_MPI_STATS=<level>`

Arguments

<level>	Indicate the level of statistics collected
`1,2,3,4,5`	Specify the level to indicate amount of MPI statistics to be collected by Application Performance Snapshot (APS). The full description of levels is available in the official APS documentation.

Description

Set this variable to collect MPI-related statistics from your MPI application using Application Performance Snapshot. The variable creates a new folder `aps_result_<date>-<time>` containing statistics data. To analyze the collected data, use the `aps` utility. For example:

```
$                              export                                  I_MPI_STATS=5
$                mpirun                -n                2                ./myApp
$ aps-report aps_result_20171231_235959
```

I_MPI_STARTUP_MODE

Select a mode for the Intel® MPI Library process startup algorithm.

Syntax

`I_MPI_STARTUP_MODE=<arg>`

Arguments

<arg>	String value
`pmi_shm`	Use shared memory to reduce the number of PMI calls. This mode is enabled by default.
`pmi_shm_netmod`	Use the `netmod` infrastructure for address exchange logic in addition to PMI and shared memory.

Description

The `pmi_shm` and `pmi_shm_netmod` modes reduce the application startup time. The efficiency of the modes is more clearly observed with the higher `-ppn` value, while there is no improvement at all with `-ppn 1`.

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_PMI_PMI_VALUE_LENGTH_MAX

Control the length of the value buffer in PMI on the client side.

Syntax

`I_MPI_PMI_VALUE_LENGTH_MAX=<length>`

Arguments

<length>	Define the value of the buffer length in bytes.
<n> > 0	The default value is -1, which means do not override the value received from the `PMI_KVS_Get_value_length_max()` function.

Description

Set this environment variable to control the length of the value buffer in PMI on the client side. The length of the buffer will be minimum of `I_MPI_PMI_VALUE_LENGTH_MAX` and `PMI_KVS_Get_value_length_max()`.

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_REMOVED_VAR_WARNING

Print out a warning if a removed environment variable is set.

Syntax

`I_MPI_REMOVED_VAR_WARNING=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Print out the warning. This is the default value
`disable | no | off | 0`	Do not print the warning

Description

Use this environment variable to print out a warning if a removed environment variable is set. Warnings are printed regardless of whether `I_MPI_DEBUG` is set.

I_MPI_VAR_CHECK_SPELLING

Print out a warning if an unknown environment variable is set.

Syntax

`I_MPI_VAR_CHECK_SPELLING=<arg>`

Arguments

<arg>	Binary indicator
`enable | yes | on | 1`	Print out the warning. This is the default value
`disable | no | off | 0`	Do not print the warning

Description

Use this environment variable to print out a warning if an unsupported environment variable is set. Warnings are printed in case of removed or misprinted environment variables.

I_MPI_LIBRARY_KIND

Specify the Intel® MPI Library configuration.

Syntax

`I_MPI_LIBRARY_KIND=<value>`

Arguments

<value>	Binary indicator
`release`	Multi-threaded optimized library (with the global lock). This is the default value
`debug`	Multi-threaded debug library (with the global lock)
`release_mt`	Multi-threaded optimized library (with per-object lock for the thread-split model)
`debug_mt`	Multi-threaded debug library (with per-object lock for the thread-split model)

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. To ensure that the desired configuration is set, check the `LD_LIBRARY_PATH` variable.

Example

```
$ export I_MPI_LIBRARY_KIND=debug
```

Setting this variable is equivalent to passing an argument directly to the `mpivars.[c]sh` script:

Example

```
$ . <installdir>/intel64/bin/mpivars.sh release
```

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
`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
`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
`skx`	Optimize for the Intel® Xeon® Processors E3 V5 and Intel® Xeon® Scalable Family series, and other Intel® Architecture processors formerly code named Skylake
`clx`	Optimize for the 2nd Generation Intel® Xeon® Scalable Processors, and other Intel® Architecture processors formerly code named Cascade Lake.
`clx-ap`	Optimize for the 2nd Generation Intel® Xeon® Scalable Processors, and other Intel® Architecture processors formerly code named Cascade Lake AP Note: The explicit `clx-ap` setting is ignored if the actual platform is not Intel.

Description

Set this environment variable to use the predefined platform settings. The default value is a local platform for each node.

The variable 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_MALLOC

Control the Intel® MPI Library custom allocator of private memory.

Syntax

`I_MPI_MALLOC=<arg>`

Argument

<arg>	Binary indicator
1	Enable the Intel MPI Library custom allocator of private memory. Use the Intel MPI custom allocator of private memory for MPI_Alloc_mem/MPI_Free_mem.
0	Disable the Intel MPI Library custom allocator of private memory. Use the system-provided memory allocator for MPI_Alloc_mem/MPI_Free_mem.

Description

Use this environment variable to enable or disable the Intel MPI Library custom allocator of private memory for MPI_Alloc_mem/MPI_Free_mem.

By default, `I_MPI_MALLOC` is enabled for `release` and `debug` Intel MPI library configurations and disabled for `release_mt` and `debug_mt` configurations.

Note

If the platform is not supported by the Intel MPI Library custom allocator of private memory, a system-provided memory allocator is used and the `I_MPI_MALLOC` variable is ignored.

I_MPI_SHM_HEAP

Control the Intel® MPI Library custom allocator of shared memory.

Syntax

`I_MPI_SHM_HEAP=<arg>`

Argument

<arg>	Binary indicator
1	Use the Intel MPI custom allocator of shared memory for MPI_Alloc_mem/MPI_Free_mem.
0	Do not use the Intel MPI custom allocator of shared memory for MPI_Alloc_mem/MPI_Free_mem.

Description

Use this environment variable to enable or disable the Intel MPI Library custom allocator of shared memory for `MPI_Alloc_mem/MPI_Free_mem`.

By default, I_MPI_SHM_HEAP is disabled. If enabled, it can improve performance of the shared memory transport because in that case it is possible to make only one memory copy operation instead of two copy-in or copy-out memory copy operations. If both I_MPI_SHM_HEAP and I_MPI_MALLOC are enabled, the shared memory allocator is used first. The private memory allocator is used only when required volume of shared memory is not available.

Details

By default, the shared memory segment is allocated on tmpfs file system on the /dev/shm/ mount point. Starting from Linux kernel 4.7, it is possible to enable transparent huge pages on the shared memory. If Intel MPI Library shared memory heap is used, it is recommended to enable transparent huge pages on your system. To enable transparent huge pages on /dev/shm, please contact your system administrator or execute the following command:

```
sudo mount -o remount,huge=advise /dev/shm
```

In order to use another tmpfs mount point instead of /dev/shm/, use I_MPI_SHM_FILE_PREFIX_4K, I_MPI_SHM_FILE_PREFIX_2M, and I_MPI_SHM_FILE_PREFIX_1G.

Note

If your application does not use MPI_Alloc_mem/MPI_Free_mem directly, you can override standard malloc/calloc/realloc/free procedures by preloading the libmpi_shm_heap_proxy.so library:

```
export LD_PRELOAD=$I_MPI_ROOT/intel64/lib/libmpi_shm_heap_proxy.so
export I_MPI_SHM_HEAP=1
```

In this case, the malloc/calloc/realloc is a proxy for MPI_Alloc_mem and free is a proxy for MPI_Free_mem.

Note

If the platform is not supported by the Intel MPI Library custom allocator of shared memory, the I_MPI_SHM_HEAP variable is ignored.

I_MPI_SHM_HEAP_VSIZE

Change the size (per rank) of virtual shared memory available for the Intel MPI Library custom allocator of shared memory.

Syntax

I_MPI_SHM_HEAP_VSIZE=*<size>*

Argument

<size>	The size (per rank) of shared memory used in shared memory heap (in megabytes).
>0	If shared memory heap is enabled for MPI_Alloc_mem/MPI_Free_mem, the default value is 4096.

Description

Intel MPI Library custom allocator of shared memory works with fixed size virtual shared memory. The shared memory segment is allocated on `MPI_Init` and cannot be enlarged later.
The `I_MPI_SHM_HEAP_VSIZE=0` completely disables the Intel MPI Library shared memory allocator.

I_MPI_SHM_HEAP_CSIZE

Change the size (per rank) of shared memory cached in the Intel MPI Library custom allocator of shared memory.

Syntax

`I_MPI_SHM_HEAP_CSIZE=<size>`

Argument

<size>	The size (per rank) of shared memory used in Intel MPI Library shared memory allocator (in megabytes).
`>0`	It depends on the available shared memory size and number of ranks. Normally, the size is less than `256`.

Description

Small values of `I_MPI_SHM_HEAP_CSIZE` may reduce overall shared memory consumption. Larger values of this variable may speed up `MPI_Alloc_mem/MPI_Free_mem`.

I_MPI_SHM_HEAP_OPT

Change the optimization mode of Intel MPI Library custom allocator of shared memory.

Syntax

`I_MPI_SHM_HEAP_OPT=<mode>`

Argument

<mode>	Optimization mode
`rank`	In this mode, each rank has its own dedicated amount of shared memory. This is the default value when `I_MPI_SHM_HEAP=1`
`numa`	In this mode, all ranks from NUMA-node use the same amount of shared memory.

Description

It is recommended to use `I_MPI_SHM_HEAP_OPT=rank` when each rank uses the same amount of memory, and `I_MPI_SHM_HEAP_OPT=numa` when ranks use significantly different amounts of memory.
Usually, the `I_MPI_SHM_HEAP_OPT=rank` works faster than `I_MPI_SHM_HEAP_OPT=numa` but the `numa` optimization mode may consume smaller volume of shared memory.

I_MPI_WAIT_MODE

Control the Intel® MPI Library optimization for oversubscription mode.

Syntax

`I_MPI_WAIT_MODE=<arg>`

Argument

<arg>	Binary indicator
1	Optimize MPI application to work in the normal mode (1 rank on 1 CPU)
0	Optimize MPI application to work in the oversubscription mode (multiple ranks on 1 CPU). This is the default value if a number of process-per-node is less than a number of CPU on the node. In other cases, 1 is the default.

Description

It is recommended to use this variable in the oversubscription mode.

I_MPI_THREAD_YIELD

Control the Intel® MPI Library thread yield customization during MPI busy wait time.

Syntax

`I_MPI_THREAD_YIELD=<arg>`

Argument

<arg>	Binary indicator
0	Do nothing for thread yield during the busy wait (spin wait). This is the default value when `I_MPI_WAIT_MODE=0`
1	Do the `pause` processor instruction for `I_MPI_PAUSE_COUNT` during the busy wait.
2	Do the `shied_yield()` system call for thread yield during the busy wait. This is the default value when `I_MPI_WAIT_MODE=1`
3	Do the `usleep()` system call for `I_MPI_THREAD_SLEEP` number of microseconds for thread yield during the busy wait.

Description

It is recommended to use `I_MPI_THREAD_YIELD=0` or `I_MPI_THREAD_YIELD=1` in the normal mode and `I_MPI_THREAD_YIELD=2` or `I_MPI_THREAD_YIELD=3` in the oversubscription mode.

I_MPI_PAUSE_COUNT

Control the Intel® MPI Library pause count for the thread yield customization during MPI busy wait time.

Syntax

`I_MPI_PAUSE_COUNT=<arg>`

Argument

<arg>	Description
>=0	Pause count for thread yield customization during MPI busy wait time. The default value is 0. Normally, the value is less than 100.

Description

This variable is applicable when `I_MPI_THREAD_YIELD=1`. Small values of `I_MPI_PAUSE_COUNT` may increase performance, while larger values may reduce energy consumption.

I_MPI_THREAD_SLEEP

Control the Intel® MPI Library thread sleep microseconds timeout for thread yield customization while MPI busy wait progress.

Syntax

`I_MPI_THREAD_SLEEP=<arg>`

Argument

<arg>	Description
>=0	Thread sleep microseconds timeout. The default value is 0. Normally, the value is less than 100.

Description

This variable is applicable when `I_MPI_THREAD_YIELD=3`. Small values of `I_MPI_PAUSE_COUNT` may increase performance in the normal mode, while larger values may increase performance in the oversubscription mode

I_MPI_EXTRA_FILESYSTEM

Control native support for parallel file systems.

Syntax

`I_MPI_EXTRA_FILESYSTEM=<arg>`

Argument

<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.

Description

Use this environment variable to enable or disable native support for parallel file systems.

Miscellaneous

Java* Bindings for MPI-2 Routines

Intel® MPI Library provides an experimental feature to enable support for Java* MPI applications. Intel MPI Library provides Java bindings for a subset of MPI-2 routines.

You can find all supported MPI routines 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.

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 comm)	MPI_Gatherv
	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,	MPI_Scatter

	Datatype recvType, int root, Comm comm)	
	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

	MPI_Group_intersection(Group group1, Group group2)	
	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>	MPI_Send

	buffer, int count, Datatype type, int dest, int tag, Comm comm)	
	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, Comm comm)	MPI_Isend
	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

Notices and Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at www.intel.com.

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.