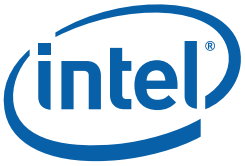


Intel® MPI Library for Windows* OS

Reference Manual

Document Number: 315399-008

Revision: 4.0

World Wide Web: http://www.intel.com

Contents

Disclaimer and Legal Notices

Revision History

Document Number	Revision Number	Description	Revision Date
315399-001	3.1 Beta	Initial release	/07/10/2007
315399-002	3.1	New variables were added. New section “Processor Information Utility” was added. Updated and reviewed for style	/10/02/2007
315399-003	3.1 build 038	Local options were added. Section “GUI Utilities” was updated. Sections “Automatic Tuning Utility”, “Integration with Microsoft* job scheduler”, “Process Identification”, and “Interoperability with OpenMP*” were added	/03/13/2008
315399-004	3.2	Sections “Index” and “Integration with PBS Pro* job scheduler” were added Global options were added Sections “Statistic Gathering Mode”, “Automatic Tuning Utility”, and “Process Pinning” were updated	/09/04/2008
315399-005	3.2 Update 1	Sections “ILP64 Support”, “ User authorization” were added	/03/04/2009
315399-006	3.2 Update 2	Section “Processor Information Utility” was updated	/08/04/2009
315399-007	4.0 Beta	Sections “Processor Information Utility”, “User Authorization”, “Automatic Tuning Utility”, “Process Pinning”, and “Statistics Gathering Mode” were updated. Sections “Compatibility Control”, “Fabrics Control”, and “Glossary” were added. Sections “Device Control”, “RDMA and RDSSM device control”, “Socket Device Control”, and “Graphical Utilities” were deleted.	/11/03/2009
315399-008	4.0	Section “Fabrics Control” and “Installation” were updated. Sections “Dynamic Process Support” and “Graphical Utilities” were added.	/02/16/2009

1 About this Document

This *Reference Manual* provides you with a complete command and tuning reference for the Intel® MPI Library.

The Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, v2 (MPI-2) specification. It provides a standard library across Intel® platforms that:

- Delivers best in class performance for enterprise, divisional, departmental and workgroup high performance computing. The Intel® MPI Library focuses on making applications perform better on IA based clusters.
- Enables to adopt MPI -2 functions as their needs dictate.

The Intel® MPI Library enables you to change or upgrade processors and interconnects as new technology becomes available, and to achieve maximum application performance without changes to the software or to the operating environment.

The library is provided in the following kits:

- *The Intel® MPI Library Runtime Environment* has the tools you need to run programs including, simple multi-purpose daemon (SMPD) and supporting utilities, dynamic (.dll) libraries and documentation.
- *The Intel® MPI Library Development Kit* includes all of the Runtime Environment components plus compilation tools, including compiler commands such as `mpiicc`, include files and modules, debug libraries, trace libraries, and test codes.

1.1 Intended Audience

This *Reference Manual* helps an experienced user understand full functionality of the Intel® MPI Library and get the best possible application performance.

1.2 Using Doc Type Field

This *Reference Manual* contains the following sections

Table 1.2-1 Document Organization

Section	Description
Section 1 About this Document	Section 1 introduces this document
Section 2 Command Reference	Section 2 describes options and variables for compiler commands, job startup commands and MPD daemon commands as well
Section 3 User Authorization	Section 3 describes different user authorizations methods
Section 4 Tuning Reference	Section 4 describes environment variables used to influence program behavior and performance at run time
Section 5 Statistics Gathering Mode	Section 5 describes how to obtain statistics of MPI communication operations

Section 6 ILP64 Support	Section 6 describes support provided for the ILP64 programming model
Section 7 Unified Memory Management	Section 7 describes the unified Intel memory management subsystem (`i_malloc`)
Section 8 Graphical Utilities	Section 8 describes graphical user interface (GUI) utilities distributed with the Intel® MPI Library
Section 9 Index	Section 9 references options and variables names

1.3 Conventions and Symbols

The following conventions are used in this document.

Table 1.3-1 Conventions and Symbols used in this Document

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

1.4 Related Information

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

Product Web Site

Intel® MPI Library Support

Intel® Cluster Tools Products

Intel® Software Development Products

2 Command Reference

2.1 Compiler Commands

(SDK only)

The following table lists available MPI compiler commands and the underlying compilers, compiler families, languages, and application binary interfaces (ABIs) that they support.

Table 2.1-1 The Intel® MPI Library Compiler Drivers

Compiler command	Underlying Compiler	Supported Language(s)	Supported ABI(s)
Common Compilers			
mpicc.bat	cl.exe	C	32/64 bit
mpicxx.bat	cl.exe	C++	32/64 bit
mpifc.bat	ifort.exe	Fortran 77/Fortran 95	32/64 bit
Microsoft* Visual C++* Compilers			
mpicl.bat	cl.exe	C/C++	32/64 bit
Intel® Fortran, C++ Compilers Versions 10.0, 10.1, 11.0 and Higher			
mpiicc.bat	icl.exe	C	32/64 bit
mpiicpc.bat	icl.exe	C++	32/64 bit
mpiifort.bat	ifort.exe	Fortran 77/Fortran 95	32/64 bit

NOTE:

- Compiler commands are available only in the Intel® MPI Library Development Kit.
- Compiler commands are in the *<installdir>**<arch>*\bin directory. For the Intel® 64 architecture in 64-bit-enabled compiler commands are in the *<installdir>**em64t*\bin directory and 32-bit compiler commands are in the *<installdir>**ia32*\bin directory.
- Ensure that the corresponding underlying compilers (32-bit or 64-bit, as appropriate) are already in your PATH.
- To port existing, MPI-enabled applications to the Intel MPI Library, recompile all sources.
- To display mini-help of a compiler command, execute it without any parameters.

2.1.1 Compiler Command Options

-mt_mpi

Use this option to link the thread safe version of the Intel® MPI Library at the following levels: MPI_THREAD_FUNNELED, MPI_THREAD_SERIALIZED, or MPI_THREAD_MULTIPLE. The MPI_THREAD_FUNNELED level is provided by default by the threat safe version of the Intel® MPI Library.

NOTE: If you specify either the `-Qopenmp` or the `-Qparallel` options for the Intel® C/C++ Compiler, the thread safe version of the library is used.

NOTE: If you specify one of the following options for the Intel® Fortran Compiler, the thread safe version of the library is used:

- `-Qopenmp`
- `-Qparallel`
- `-threads`
- `-reentrancy`
- `-reentrancy:threaded`

-profile=<*profile_name*>

Use this option to specify an MPI profiling library. The profiling library is selected using one of the following methods:

- Through the configuration file `<profile_name>.conf` located in the `<installdir>\<arch>\etc` directory. See *Profiles* for details.
- In the absence of the respective configuration file, by linking the library `lib<profile_name>.lib` located in the same directory as the Intel® MPI Library.

-t or -trace

Use the `-t` or `-trace` option to link the resulting executable against the Intel® Trace Collector library.

Use the `-t=log` or `-trace=log` option to link the resulting executable against the logging Intel® MPI Library and the Intel® Trace Collector libraries. The logging libraries trace internal Intel® MPI Library states in addition to the usual MPI function calls.

Include the installation path of the Intel® Trace Collector in the `VT_ROOT` environment variable to use this option.

-check_mpi

Use this option to link the resulting executable against the Intel® Trace Collector correctness checking library. Include the installation path of the Intel® Trace Collector into the `VT_ROOT` environment variable to use this option.

-ilp64

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

NOTE: If you specify the `-i8` option for the Intel® Fortran Compiler, you still have to use the ILP64 option for linkage. See *ILP64 Support* for details.

/Zi or /Z7 or /ZI/debug

Use these options to compile a program in debug mode and link the resulting executable against the debugging version of the Intel® MPI Library. See *Environment Variables*, `I_MPI_DEBUG` for information on how to use additional debugging features with the `/Zi`, `/Z7`, `/ZI` or `debug` builds.

NOTE: The `/ZI` option is only valid for C/C++ compiler.

-O

Use this option to enable optimization.

-echo

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

-show

Use this option to learn how the underlying compiler is invoked. For example, use the following command to see the required compiler flags and options:

```
> mpiicc.bat -show -c test.c
```

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

```
> mpiicc.bat -show -o a.exe test.obj
```

This 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}=<*compiler*>

Use this option to select the underlying compiler.

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

```
> mpiicc.bat –cc=icl.exe -c test.c
```

For this to work, `icl.exe` should be in your path. Alternatively, you can specify the full path to the compiler.

NOTE: This option works only with the `mpiicc.bat` and `mpifc.bat` commands.

-v

Use this option to print the compiler driver script version.

2.1.2 Configuration Files

You can create compiler configuration files using the following file naming convention:

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

where:

<compiler> = {cc,fc}, depending on the language being compiled

<name> = name of underlying compiler

Source this file, if it exists, prior to compiling or linking to enable changes to the environment on a per-compiler-command basis.

2.1.3 Profiles

You can select a profile library through the `-profile` option of the Intel® MPI Library compiler drivers. You can also create your own profile as `<installdir>\<arch>\etc\<profile_name>.conf`

The following variables can be defined there:

`PROFILE_PRELIB` - libraries (and paths) to include before the Intel® MPI Library

`PROFILE_POSTLIB` - libraries to include after the Intel® MPI Library

`PROFILE_INCPATHS` - C preprocessor arguments for any include files

For instance, create a file `<installdir>\<arch>\etc\myprof.conf` with the following lines:

```
SET PROFILE_PRELIB=<path_to_myprof>\lib\myprof.lib
SET PROFILE_INCPATHS=-I"<paths_to_myprof>\include"
```

Use the command-line argument `-profile=myprof` for the relevant compile driver to select this new profile.

2.1.4 Environment Variables

I_MPI_{CC, CXX, FC, F77, F90}_PROFILE

Specify a default profiling library.

Syntax

`I_MPI_{CC, CXX, FC, F77, F90}_PROFILE=<profile_name>`

Arguments

<profile_name>	Specify a default profiling library

Description

Set this variable to select a specific MPI profiling library to be used by default. This has the same effect as if `-profile=<profile_name>` were used as an argument to `mpiicc.bat` or another Intel® MPI Library compiler driver.

I_MPI_{CC, CXX, FC, F77, F90}

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

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

Syntax

`I_MPI_{CC, CXX, FC, F77, F90}=<compiler>`

Arguments

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

Description

Set this 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.

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

VT_ROOT

Set the Intel® Trace Collector installation directory path.

Syntax

`VT_ROOT=<path>`

Arguments

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

Description

Set this 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 is *<installdir>**<arch>*\etc

Description

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

2.2 Job Startup Command

mpiexec

Syntax

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

or

`mpiexec <g-options> <l-options> <executable> : \`

`<l-options> <executable>`

or

`mpiexec –configfile <file>`

Arguments

<g-options>	Global options that apply to all MPI processes
<l-options>	Local options that apply to a single arg-set
<executable>	.\a.exe or path\name of the executable file
<file>	File with command-line options

Description

In the first command-line syntax, run the specified *<executable>* with the specified options. All global and/or local options apply to all MPI processes. A single arg-set is assumed. For example, the following command executes a.out over the specified *<# of processes>*:

```
> mpiexec -n <# of processes> a.exe
```

In the second command-line syntax, divide the command line into multiple arg-sets, separated by colon characters. All the global options apply to all MPI processes, but the various local options and *<executable>* can be specified separately for each arg-set. For example, the following command would run each given executable on a different host:

```
> mpiexec  -n 2 -host host1 a.exe : \
           -n 2 -host host2 b.exe
```

In the third command-line syntax, read the command line from specified *<file>*. For a command with a single arg-set, the entire command should be specified on a single line in *<file>*. For a command with multiple arg-sets, each arg-set should be specified on a single, separate line in *<file>*. Global options should always appear at the beginning of the first line in *<file>*.

Simple multi-purpose daemon (SMPD) service must already be running in order for mpiexec to succeed.

NOTE: If path to executable is not in the PATH on all nodes in the cluster, specify *<executable>* as *<path>*\a.exe rather than a.exe.

2.2.1 Global Options

-machinefile *<machine file>*

Use this option to control the process placement through the *<machine file>*. The number of processes to start is controlled by the option -n as usual.

A machine file is a list of fully qualified or short host names, one name per line. Blank lines and lines that start with '#' as the first character are ignored.

By repeating a host name you will place additional processes on this host. You can also use the following format to avoid repetition of the same host name: *<host name>:<number of processes>*. For example, the following machine file:

```
host1
host1
host2
host2
host3
```

is equivalent to:

```
host1:2
host2:2
host3
```

It is also possible to specify the network interface used for communication for each node: `<host name>:<number of processes> [ifhn=<interface_host_name>]`.

NOTE: The `-machinefile`, `-ppn`, `-rr`, and `-perhost` options are intended for process distribution. Do not use them simultaneously. Otherwise `–machinefile` will take precedence.

-configfile <*filename*>

Use this option to specify the file `<filename>` that contains command-line options. Blank lines and lines that start with '#' as the first character are ignored. For example, a configuration file contains the following commands to run the executables `a.exe` and `b.exe` using the `rdssm` device over `host1` and `host2` respectively:

```
-host host1 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE rdssm -n 2 a.exe
-host host2 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE rdssm -n 2 b.exe
```

To launch a MPI application according to the parameters above, use:

```
> mpiexec –configfile <filename>
```

NOTE: This option may only be used alone. It terminates parsing of the `mpiexec` command line.

-g<*l-option*>

Use this option to apply the named local option `<l-option>` globally. See *Local Options* for a list of all local options. During the application startup, the default value is the `–genvuser` option. The options `–genvnone, -genvuser, -genvall` have the lowest priority, `–genvlist, -genvexcl` have higher priority than the previous set. The `–genv` option has the highest priority. Local options have higher priority then the global options.

-l

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

-tune

Use this option to optimize the Intel® MPI Library performance using the data collected by the `mpitune` utility. If `<configuration_file>` is not mentioned, the best-fit tune options will be selected for the given configurations. Otherwise the given configuration file will be used.

For the Intel® 64 architecture in 64-bit mode the default location of the configuration files are located in the `<installdir>\em64t\etc` directory. For 32-bit mode the files are located in the `<installdir>\ia32\etc` directory. Set the `I_MPI_TUNER_DATA_DIR` environment variable to override the default location.

See *Automatic Tuning Utility* for more details.

-p <*port*> or -port <*port*>

Use this option to specify the SMPD port `mpiexec` should connect to. This option can be useful if SMPD is using a non-default port number.

-hosts n <*host1*> <*num_proc1*> <*host2*> <*num_proc2*> ... <*hostn*> <*num_procn*>

Use this option to specify a particular hosts list and a number of processes on each of the MPI processes in the current arg-set are to be run. For example, the following command line will run the executable `a.exe` on the hosts `host1` and `host2`. Two processes will be run on host1 and one process on host 2 respectively:

```
> mpiexec -hosts 2 host1 2 host2 1 a.exe
```

-logon

Use this option to prompt for your account name and password.

-delegate

Use this option to enable the domain-based authorization with the delegation ability.

-impersonate

Use this option to enable the limited domain-based authorization. You will not be able to open files on remote machines or access mapped network drives.

-pwdfile <*filename*>

Use this option to read the account and password from the file specified. Put account on the first line and password on the second one.

-nopopup_debug

Use this option to disable the system popup dialog if the process crashes.

-exitcodes

Use this option to print the process exit codes when each process exits.

-verbose

Use this option to redirect the `smpd` output to `stdout`.

-localroot

Use this option to launch the root process directly from `mpiexec` if the host is local.

-register [-user n]

Use this option to encrypt the user name and password to the registry.

-remove [-user n]

Use this option to delete the encrypted credentials from the registry. If no user index is specified, all entries are removed.

-timeout <*seconds*>

Use this option to set timeout for the job.

-whoami

Use this option to print the current user name.

-h or –help or --help

Use this option to display the `mpiexec` help message.

2.2.2 Local Options

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

Use this option to set the number of MPI processes to run the current arg-set.

-env <*ENVVAR*> <*value*>

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

-envall

Use this option to propagate all environment variables in the current environment.

-envnone

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

-envlist <*list of env var names*>

Use this option to pass a list of environment variables with their current values.

-envuser

Use this option to propagate all user environment variables to all MPI processes with the exception of the following environment variables: `%ALLUSERSPROFILE%`, `%APPDATA%`, `%CommonProgramFiles%`, `%CommonProgramFiles (x86)%`, `%COMPUTERNAME%`, `%HOMEDRIVE%`, `%HOMEPATH%`, `%NUMBER_OF_PROCESSORS%`, `%OS%`, `%PROCESSOR_ARCHITECTURE%`, `%PROCESSOR_IDENTIFIER%`, `%PROCESSOR_LEVEL%`, `%PROCESSOR_REVISION%`, `%ProfilePath%`, `%ProgramFiles%`, `%ProgramFiles (x86)%`, `%SystemDrive%`, `%SystemRoot%`, `%TEMP%`, `%TMP%`, `%USERDNSDOMAIN%`, `%USERDOMAIN%`, `%USERNAME%`, `%USERPROFILE%`.

This is the default setting.

-envexcl <*list of env var names*>

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

-host <*nodename*>

Use this option to specify a particular `<nodename>` on which the MPI processes in the current arg-set are to be run. For example, the following will run the executable `a.exe` on host `host1` only:

```
> mpiexec –n 2 –host host1 a.exe
```

-path <*directory*>

Use this option to specify the path to `<executable>` that is to be run in the current arg-set. The separator is **;**

-dir <*directory*> or -wdir <*directory*>

Use this option to specify the working directory in which `<executable>` is to be run in the current arg-set.

-map <*drive:\\host\share*>

Use this option to create network mapped drive on nodes before starting `<executable>`. Network drive will be automatically removed after the job completion.

-mapall

Use this option to request creation of all user created network mapped drives on nodes before starting `<executable>`. Network drives will be automatically removed after the job completion.

2.2.3 Environment Variables

I_MPI_DEBUG

Print out debugging information when an MPI program starts running.

Syntax

`I_MPI_DEBUG=<level>`

Arguments

`<level>`	Indicate level of debug information provided
`0`	Print no debugging information. This is the default value
`1`	Output verbose error diagnostics
`2`	Confirm which `I_MPI_DEVICE` was used
`3`	Output effective MPI rank, pid and node mapping table
`4`	Print process pinning information
`5`	Print Intel MPI-specific environment variables
`> 5`	Add extra levels of debug information

Description

Set this variable to control the output of the debugging information.

The `I_MPI_DEBUG` mechanism extends the MPICH2* `MPICH_DBG_OUTPUT` debug mechanism by overriding the current value and setting `MPICH_DBG_OUTPUT=stdout`.

In order to simplify process identification add the `'+'` or `'-'` sign in front of the numerical value for `I_MPI_DEBUG`. This setting produces debug output lines prepended with the MPI process rank, a UNIX* process id, and a host name as defined at the process launch time. For example, the command:

```
> mpiexec -n <# of processes> -env I_MPI_DEBUG +2 a.exe
```

may produce the following output:

```
[rank#pid@hostname]Debug message
```

NOTE: Compiling with `mpiicc.bat /Zi`, `/Z7` or `/Z7` causes considerable amounts of additional debug information to be printed.

I_MPI_JOB_TIMEOUT

Set the `mpiexec` timeout.

Syntax

`I_MPI_JOB_TIMEOUT=<timeout>`

Arguments

<timeout>	Define `mpiexec` timeout period in seconds
<n> ⩾ 0	The default timeout value is zero, meaning no timeout

Description

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

NOTE: Set the `I_MPI_JOB_TIMEOUT` variable in the shell environment before executing the `mpiexec` command. Do not use the `-genv` or `-env` options for setting the *<timeout>* value. Those options are used only for passing variables to the MPI process environment.

I_MPI_SMPD_VERSION_CHECK

Set this environment variable to enable strong SMPD version check. It is valid only for Windows* OS.

Synta

`I_MPI_SMPD_VERSION_CHECK=<arg>`

Arguments

<arg>	Binary indicator
`enable, yes, on, 1`	Check the version and fail in case of mismatch. This is the default value
`disable, no, off, 0`	Only print a warning message in case of a version mismatch and continue working

Description

Set this variable to control SMPD version check. The Intel® MPI Library terminates application if the versions mismatch is found. To disable SMPD version check set the `I_MPI_SMPD_VERSION_CHECK` environment variable to `disable`.

I_MPI_DAT_LIBRARY

Select a particular DAT library to be used.

Syntax

`I_MPI_DAT_LIBRARY=<library>`

Arguments

<library>	Specify the exact library to be used instead of the default `dat.dll`

Description

Set this variable to select a specific DAT library to be used. Specify the full path to the DAT library if it is not located in the dynamic loader search path.

NOTE: Use this variable only if you are going to utilize a DAPL provider.

I_MPI_TUNER_DATA_DIR

Set an alternate path to the directory with the tuning configuration files.

Syntax

`I_MPI_TUNER_DATA_DIR=<path>`

Arguments

<path>	Specify the automatic tuning utility output directory. The default value is *<mpiinstalldir>*\em64t\etc or *<mpiinstalldir>*\ia32\etc

Description

Set this variable to specify an alternate location of the tuning configuration files.

2.2.4 Integration with Microsoft* Job Scheduler

The Intel® MPI Library job startup command `mpiexec` can be called out of Microsoft* Compute Cluster Pack 2003 job scheduler to execute MPI application. In this case the `mpiexec` command automatically inherits the host list, process count and the working directory allocated to the job.

Use the following command to submit MPI job:

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

Make sure `mpiexec` and dynamic libraries are available through `PATH`. The Intel MPI Library environment variables can be registered during the installation process.

2.2.5 Integration with PBS Pro* Job Scheduler

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

Example:

Content of `job` script:

```
REM PBS -l nodes=4:ppn=2
REM PBS -l walltime=1:00:00

cd %PBS_O_WORKDIR%
mpiexec.exe test.exe
```

Use the following command to submit the job:

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

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

2.3 Simple Multi-Purpose Daemon

smpd

Simple multi-purpose daemon.

Syntax

```
smpd.exe   [ -h ] [ --help ] [ -port <port> ] [ -d ] \
           [ -install | -regserever ] [ -start ] [ -stop ] \
           [ -shutdown <hostname> ] [ -status <hostname> ] \
           [ -restart <hostname> ] [ -anyport ] [ -hosts ] [ -sethosts ] \
           [ -set <option_name> <option_value> ] [ -get <option_name> ] \
           [ -traceon <logfilename> [ <hostA> <hostB> ... ]] \
           [ -traceoff [ <hostA> <hostB> ...]] \
           [ -remove | -unregister | -uninstall ] [ -register_spn ]\
           [ -remove_spn ] [ -V ] [ -version ]
```

Arguments

-h | --help	Display a help message
-p <port> | -port <port>	Specify the port that smpd is listening on
-d | -debug	Start smpd in debug mode
-install -regserver	Install the smpd service
-start	Start the smpd service
-stop	Stop the smpd service
-shutdown <hostname>	Shutdown smpd on specified <hostname>
-status <hostname>	Get the smpd status on specified <hostname>
-restart <hostname>	Restart smpd on specified <hostname>
-anyport	Use any port for listening
-hosts	Get the smpd ring list
-sethosts	Set the smpd ring from specified hosts. This settings impact all users
-set <option_name> <option_value>	Register the <option_name> key to the HKEY_LOCAL_MACHINE registry key
-get <option_name>	Get the value of the registered <option_name> key from the HKEY_LOCAL_MACHINE registry key
-traceon <logfilename> <hostA> <hostB> ...	Restart smpd and store the output into provided <logfilename>
-traceoff <hostA> <hostB>	Restart smpd without logging the output
-remove |	Remove smpd service

-unregserver | -uninstall	
-register_spn	Register Service Principle Name (SPN) in the Windows* domain for the cluster node on which this command is executed
-remove_spn	Remove SPN from the Windows* domain for the cluster node on which this command is executed
-V	Get the Intel® MPI Library version info
-version	Display the smpd version information

Description

Simple Multipurpose Daemon* (SMPD) is the Intel® MPI Library process management system for starting parallel jobs. Before running a job, start smpd service on each host and connect them into a ring.

Use the smpd.exe command to install, uninstall, start or stop SMPD service.

Examples:

1. Use the following command to install SMPD service:

```
> smpd.exe -install
```

NOTE: This command must be run by a user with administrator privileges. After that all users will be able to launch MPI jobs using mpiexec.

2. Use the following command to start the SMPD service in debug mode:

```
> smpd.exe -d
```

2.4 Processor Information Utility

cpuinfo

Use the cpuinfo utility to display processor architecture information.

Syntax

```
cpuinfo
```

Description

The cpuinfo utility prints out processor architecture information that can be used to define suitable process pinning settings. The output consists of a header and a number of tables. The output header includes the processor brand name, the processor code name, and the processor architecture. The output includes the following tables:

- **Processor composition table**: describes the processor packages, cores, and threads composition.

  Processors (CPUs) – the number of software executive processor units.
  Packages (sockets) – the number of physical packages and corresponding sockets.
  Cores per package – the number of cores within each package.
  Threads per core – the number of processor units within each core. If the number is equal to one, Simultaneous Multi Threading (SMT) mode is disabled. If the number is larger than one, SMT mode is enabled.

- **Processor identification table**: identifies threads, cores, and packages of each logical processor accordingly.

  Thread Id – unique processor identifier within a core.
  Core Id – unique core identifier within a package.
  Package Id – unique package identifier within a node.

- **Processor placement table**: maps processor packages and cores. It is an inversion of the processor identification table. Each entry contains the information on packages, cores, and processors:

  Package – a physical package identifier.
  Cores – a list of core identifiers that belong to this package.
  Processors – a list of processors that belong to this package. This list order directly corresponds to the core list. A group of processors enclosed in the brackets belongs to one core.

- **Cache sharing table**: lists information of sizes and processors groups, for each cache level.

  Size – cache size in bytes.
  Processors – a list of processor groups enclosed in the parentheses that shared this cache or no sharing otherwise.

NOTE: The architecture information is available on systems based on the IA-32 and Intel® 64 architectures.

Examples

1. `cpuinfo` output for Intel® Xeon® Processor 5400 series:

```
Intel(R) Xeon(TM) Processor (Intel64 Harpertown)

=====  Processor composition  =====
Processors(CPUs)  : 8
Packages(sockets) : 2
Cores per package : 4
Threads per core  : 1

=====  Processor identification  =====
Processor  Thread Id.  Core Id.  Package Id.
0          0           0         1
1          0           0         0
2          0           2         0
3          0           2         1
4          0           1         0
5          0           3         0
6          0           1         1
7          0           3         1

=====  Placement on packages  =====
Package Id.  Core Id.  Processors
1            0,2,1,3   0,3,6,7
0            0,2,1,3   1,2,4,5

=====  Cache sharing  =====
Cache    Size            Processors
L1       32  KB          no sharing
L2       6   MB          (0,6)(1,4)(2,5)(3,7)
```

2. `cpuinfo` output for Intel® Core™ i7 processor with SMT support:

```
Intel(R) Core(TM) i7 Processor (Intel64 Bloomfield)

=====  Processor composition  =====
Processors(CPUs)  : 8
Packages(sockets) : 1
Cores per package : 4
Threads per core  : 2

=====  Processor identification  =====
Processor Thread Id. Core Id.  Package Id.
0         0          0         0
```

```
1          0          1          0
2          0          2          0
3          0          3          0
4          1          0          0
5          1          1          0
6          1          2          0
7          1          3          0
=====  Placement on packages  =====
Package Id   Core Id    Processors
0            0,1,2,3    (0,4)(1,5)(2,6)(3,7)

=====  Cache sharing  =====
Cache   Size             Processors
L1      32  KB          (0,4)(1,5)(2,6)(3,7)
L2      256 KB          (0,4)(1,5)(2,6)(3,7)
L3      8   MB          (0,1,2,3,4,5,6,7)
```

3 User Authorization

3.1 Overview

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

- The password-based authorization
- The domain-based authorization with the delegation ability
- The limited domain-based authorization

The password-based authorization is the typical method of providing remote computer access using your account name and password.

The domain-based authorization methods use the Security Service Provider Interface (SSPI) provided by Microsoft* in a Windows* environment. The SSPI allows domain to authenticate the user on the remote machine in accordance to the domain policies. You do not need to enter and store your account name and password when using such methods.

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

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

3.2 Installation

This feature is supported on Windows clusters under the following Microsoft operation systems: Windows* 2000 Server*, Windows 2003 Server*, and Windows 2008 Server*.

Microsoft's Kerberos Distribution Center* must be enabled on your domain controller. This is the default behavior.

Using the domain-based authorization method with the delegation ability requires specific installation of the domain. You can perform this installation in the following ways:

- Use the IMPI installer if you have domain administrator rights
- Follow the actions described in the topic below

3.2.1 Active Directory* Setup

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

1. Log in on the domain controller under the admin account
2. Enable the delegation for cluster nodes:

    a. Open the **Computers** list in the **Active Directory Users and Computers** administrative utility

    b. Right click on a desired computer object and select **Properties**

    c. If the account is located:

i. in a Windows 2000 functional level domain, check the **Trust computer for delegation** option

ii. in a Windows 2003 functional level domain, select the **Delegation** tab and check the **Trust this computer for delegation to any service (Kerberos only)** option

3. Enable the delegation for users:

   a. Open the **Users** list in the **Active Directory Users and Computers** administrative utility

   b. Right click on a desired user object and select **Properties**

   c. Select the **Account** tab and disable the **Account is sensitive and cannot be delegated** option

4. Register Service Principal Name (SPN) for cluster nodes. Use one of the following methods for registering SPN:

   a. Use the Microsoft*-provided `setspn.exe` utility. For example, execute the following command on the domain controller:
   `setspn.exe –A impi_smpd/<host>:<port>/impi_smpd <host>`
   where
   <host> is a cluster node name
   <port> is a SMPD port. The default value is 8678. Change this number only if your SMPD service uses the non-default port

   b. Log into each desired node under the admin account and execute the `smpd –register_spn` command

NOTE: In case of any issues with the MPI task start, reboot the machine from which the MPI task is started. Alternatively, execute the `klist purge` command there if the Microsoft*-provided `klist.exe` utility is available.

3.3 Environment Variables

I_MPI_AUTH_METHOD

Select a user authorization method.

Syntax

`I_MPI_AUTH_METHOD=<method>`

Arguments

<method>	Define an authorization method
`password`	Use the password-based authorization. This is the default value
`delegate`	Use the domain-based authorization with delegation ability
`impersonate`	Use the limited domain-based authorization. You will not be able to open files on remote machines or access mapped network drives

Description

Set this variable to select a desired authorization method. If this variable is not defined, `mpiexec` uses the password-based authorization method by default.

NOTE: Alternatively, you can change the default behavior by using the `mpiexec –delegate` or `mpiexec –impersonate` options.

4 Tuning Reference

The Intel® MPI Library provides an automatic tuning utility and many environment variables that can be used to influence program behavior and performance at run time.

4.1 Automatic Tuning Utility

mpitune

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

Syntax

```
mpitune  [ -h ] [ -V ] [ -hf <hostsfile> ] [ -od <outputdir> ] [ -d ] \
         [ -i <count> ] [ -s ] [ -a \"<application command line>\" ]
```

or

```
mpitune  [ --help ] [ --version] [ --host-file <hostsfile> ] \
         [ --output-directory <outputdir> ] [ --debug ] \
         [ --iterations <count>] [ --silent ] \
         [ --application \"<application command line>\" ]
```

Arguments

`-a \"<app_cmd_line>\" | --application \"<app_cmd_line>\"`	Switch on the application tuning mode. Quote the full command line as shown
`-of <file-name> | --output-file <file-name>`	Specify the application configuration file to be generated in the application-specific mode. By default, use the `>PWD\app.conf`
`-t \"<test_cmd_line>\" | --test \"<test_cmd_line>\"`	Replace the default Intel® MPI Benchmarks by the indicated benchmarking program in the cluster-specific mode. Quote the full command line as shown
`-cm | --cluster-mode {exclusive|full}`	Set the cluster usage mode `full` – maximum number of tasks will be executed. This is the default mode `exclusive` – only one task will be executed on the cluster at a time
`-d | --debug`	Print out the debug information
`-dl [d1[,d2…[,dN]]] | --device-list [d1[,d2,…[,dN]]]`	Select the device(s) you want to tune. By default, use all devices mentioned in the `<installdir>\<arch>\etc\devices.xml` file
`-fl [f1[,f2…[,fN]]] | --fabric-list [f1[,f2…[,fN]]]`	Select the fabric(s) you want to tune. By default, use all fabrics mentioned in the `<installdir>\<arch>\etc\fabrics.xml` file

-er | --existing-ring	Try to use an existing MPD ring. By default, create a new MPD ring
-hf *<hostsfile>* | --host-file *<hostsfile>*	Specify an alternative host file name. By default, use the >PWD\mpd.hosts
-h | --help	Display a help message
-hr | --host-range *{min:max|min:|:max}*	Set the range of hosts used for testing. The default minimum value is 1. The default maximum value is the number of hosts defined by the mpd.hosts or the existing MPD ring. The min: or :max format will use the default values as appropriate
-i *<count>* | --iterations *<count>*	Define how many times to run each tuning step. Higher iteration counts increase the tuning time, but may also increase the accuracy of the results. The default value is 3
--message-range *{min:max|min:|:max}*	Set the message size range. The default minimum value is 0. The default maximum value is 4194304 (4mb). By default, the values are given in bytes. They can also be given in the following format: 16kb, 8mb or 2gb. The min: or :max format will use the default values as appropriate
-od *<outputdir>* | --output-directory *<outputdir>*	Specify the directory name for all output files. By default, use the current directory. This directory should be accessible from all hosts
-pr *{min:max|min:|:max}* | --ppn-range *{min:max|min:|:max}* | --perhost-range *{min:max|min:|:max}*	Set the maximum number of processes per host. The default minimum value is 1. The default maximum value is the number of cores of the processor. The min: or :max format will use the default values as appropriate
-sf *[file-path]* | --session-file *[file-path]*	Continue the tuning process starting from the state saved in the *file-path* session file
-s | --silent	Suppress all diagnostic output
-td *<dir-path>* | --temp-directory *<dir-path>*	Specify a directory name for the temporary data. By default, use the >PWD\mpitunertemp. This directory should be accessible from all hosts
-tl *<minutes>* | --time-limit *<minutes>*	Set mpitune execution time limit in minutes. The default value is 0, which means no limitations
-mh | --master-host	Dedicate a single host to run mpitune
-V | --version	Print out version information

Deprecated Options

Deprecated Option	New Option
--outdir	-od | --output-directory
--verbose	-d | --debug
--file	-hf | --host-file
--logs	-lf | --log-file
--app	-a | --application

Description

Use the `mpitune` utility to create a set of Intel® MPI Library configuration files that contain optimal settings for a particular cluster or application. You can reuse these configuration files in the `mpiexec` job launcher by using the `-tune` option.

The MPI tuner utility operates in two modes:

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

4.1.1 Cluster-specific Tuning

Run this utility once after the Intel® MPI Library installation and after every cluster configuration change (processor or memory upgrade, network reconfiguration, etc.). Do this under the user account that was used for the Intel® MPI Library installation or set the tuner data directory with the `--output-directory` command.

If there are any configuration files in the `<installdir>\<arch>\etc` directory, the recorded Intel® MPI Library configuration settings will be used automatically by `mpiexec` with the `-tune` option.

For example:

1. Collect configuration settings for the cluster hosts listed in the `.\mpd.hosts` file by using the Intel® MPI Benchmarks

   ```
   > mpitune
   ```

2. Use the optimal recorded values when running on the cluster

   ```
   > mpiexec -tune -n 32 .\myprog
   ```

The job launcher will find a proper set of configuration options based on the execution conditions: communication fabrics, number of hosts and processes, etc. If you have write access permission for `<installdir>\<arch>\etc`, all generated files will be saved in this directory; otherwise the current working directory will be used.

4.1.1.1 Replacing the Default Benchmark

This tuning feature is an extension of the cluster-specific tuning mode in which you specify a benchmarking application that will be used for tuning.

For example:

1. Collect the configuration settings for the cluster hosts listed in the `.\mpd.hosts` file by using the desired benchmarking program

   ```
   > mpitune --test \"benchmark -param1 -param2\"
   ```

2. Use the optimal recorded values for your cluster

   ```
   > mpiexec -tune -n 32 .\myprog
   ```

4.1.2 Application-specific Tuning

Run the tuning process for any kind of MPI application by specifying its command line to the tuner. Performance is measured as inversed execution time of the given application. To reduce the overall tuning time, use the shortest representative application workload if applicable.

For example:

1. Collect configuration settings for the given application

```
> mpitune --application \"mpiexec -n 32 .\myprog\" –of .\myprog.conf
```

2. Use the optimal recorded values for your application

```
> mpiexec –tune .\myprog.conf -n 32 .\myprog
```

Based on the default tuning rules, the automated tuning utility evaluates a full set of the library configuration parameters to minimize the application execution time. By default, all generated files will be saved in the current working directory.

NOTE: The resulting application configuration file contains the optimal Intel® MPI Library parameters for this particular application only. If you want to tune the Intel® MPI Library for the same application in a different configuration (number of hosts, workload, etc.), you may need to rerun the automated tuning utility by using the desired configuration.

The automated tuning utility will overwrite the existing application configuration files by default. You should use a naming convention for your various application files to select the correct file when you need it.

4.1.3 Tuning Utility Output

Upon completion of the tuning process, the Intel® MPI Library tuning utility records the chosen values in the configuration file in the following format:

```
-genv I_MPI_DYNAMIC_CONNECTION 1
-genv I_MPI_ADJUST_REDUCE 1:0-8
```

The Intel MPI Library tuning utility ignores variables having no effect on the application when the difference between probes is at the noise level (1%). In this case, the utility does not set the variable and preserves the default library heuristics.

In the case of the tuning application having significant run-to-run performance variation, the Intel MPI Library tuning utility might select divergent values for the same variable under the same conditions. To improve decision accuracy, increase the number of iterations for each test run with the `-i` command line option. The default value for the iteration number is `3`.

4.2 Process Pinning

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

4.2.1 Process Identification

Two schemes are used to identify logical processors in a system:

1. System-defined logical enumeration

2. 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 to this CPU bit in the kernel affinity bit-mask. Three-level hierarchical identification uses triplets that provide information about processor location and their order. The triplets are hierarchically ordered (package, core, thread).

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

NOTE: Logical and topological enumerations are not the same.

Table 4.2-1 Logical Enumeration

0	4	1	5	2	6	3	7

Table 4.2-2 Hierarchical Levels

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

Table 4.2-3 Topological Enumeration

0	1	2	3	4	5	6	7

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

4.2.2 Environment Variables

I_MPI_PIN

Turn on/off process pinning.

Syntax

`I_MPI_PIN=<arg>`

Arguments

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

Description

Set this variable to turn off the process pinning feature of the Intel® MPI Library.

I_MPI_PIN_PROCESSOR_LIST (I_MPI_PIN_PROCS)

Define a processor subset and mapping rules for MPI processes pinning to separate processors of this subset.

Syntax

I_MPI_PIN_PROCESSOR_LIST=<value>

The variable value has three syntax forms:

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

Deprecated Syntax

I_MPI_PIN_PROCS=<proclist>

NOTE: The postoffset keyword has offset alias.

NOTE: The second form of 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.

The result processor list is used for the consecutive mapping of MPI processes (i-th rank is mapped on the i-th list member).

NOTE: grain, shift, preoffset, and postoffset parameters have unified style of setting.

Arguments

<proclist>	A comma-separated list of logical processor numbers and/or ranges of processors. Process with the i-th rank is pinned on 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>

<procset>	A processor is ordered according to the topological numeration. The default value is allcores
all	All logical processors. The power of this subset is equal to the number of CPU on a node
allcores	All logical processors that belong to different cores. A power of this subset is equal to the number of cores on a node. If HyperThreading is disabled, allcores == all
allsocks	All logical processors that belong to different physical packages/sockets. The power of this subset is equal to the number of sockets on a node

<map>	Pattern used for the process placement
bunch	The processes are mapped in proportion on sockets as close as possible

scatter	The processes are mapped as remotely as possible not to share common resources: FSB, caches, core
spread	The processes are mapped consecutively with the possibility not to share common resources

<grain>	Specify pinning granularity cell for defined procset. Minimal grain is one element of procset. Maximal grain is a number of procset elements in a socket. The grain value must be multiple of the procset power. Otherwise, minimal grain is assumed. The default value is minimal grain
<shift>	Specify the round robin shift of the granularity cells along 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
<preoffset>	Specify cyclic shift of procset on the preoffset value before the round robin shifting. 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 cyclic shift of processor subset 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

<n>	Specify the explicit value of the corresponding parameter. *<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 L1 cache
cache2	Specify the parameter value equal to the amount of the corresponding parameter units that share L2 cache
cache3	Specify the parameter value equal to the amount of the corresponding parameter units that share L3 cache
cache	The largest value among cache1, cache2, and cache3
socket	Specify the parameter value equal to the amount of the corresponding parameter units contained in one physical package/socket
sock	sock = socket
half	Specify the parameter value equal to socket/2
mid	mid = half
third	Specify parameter value equal to socket/3
quarter	Specify parameter value equal to socket/4

octavo	Specify parameter value equal to socket/8

Description

Set the I_MPI_PIN_PROCESSOR_LIST variable to define the processor placement on processors. In order to avoid conflicts with shells, the variable value may be enclosed in quotes.

NOTE: This variable is valid only if I_MPI_PIN is enabled.

The I_MPI_PIN_PROCESSOR_LIST variable has three different variants:

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

   For example, process mapping for I_MPI_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

2. Grain/shift/offset mapping. This method provides cyclic shift of the defined grain along the processor list with step 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)

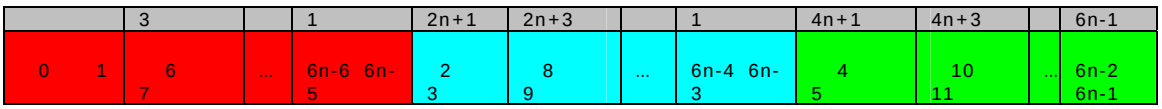
0 1	2n 2n+1		2 3	2n+2 2n+3		. . .	2n-2 2n-1	4n-2 4n-1	
0 1	2 3	4 5	6 7	8 9	10 11	. . .	6n-6 6n-5	6n-4 6n-3	6n-2 6n-1

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	…	2n-2 2n-	2n	2n+2	…	4n-2 4n-	4n	4n+2	… 6n-2



3. Predefined scenario. In this case the most popular schemes of process pinning get unique names and these names are used for selection. Currently there are two such scenarios: `bunch` and `scatter`.

In the `bunch` scenario the processes are mapped proportionally to sockets as closely as possible. This 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 to not share common resources: FSB, caches, cores.

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

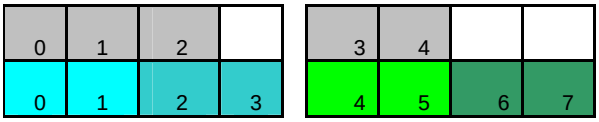
Legend:

gray – MPI processes

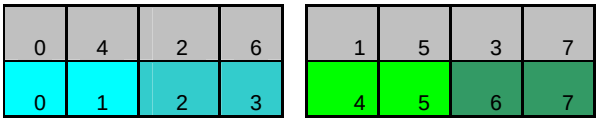
cyan – 1st socket processors

green – 2nd socket processors

The same color – processor pair share one cache



`bunch` scenario for 5 processors



`scatter` scenario for full loading

Examples

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

```
> mpiexec.exe -genv I_MPI_PIN_PROCESSOR_LIST 0,3 \
-n <# of processes> <executable>
```

2. 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:

```
> mpiexec.exe –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>
```

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

```
> mpiexec.exe –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>
```

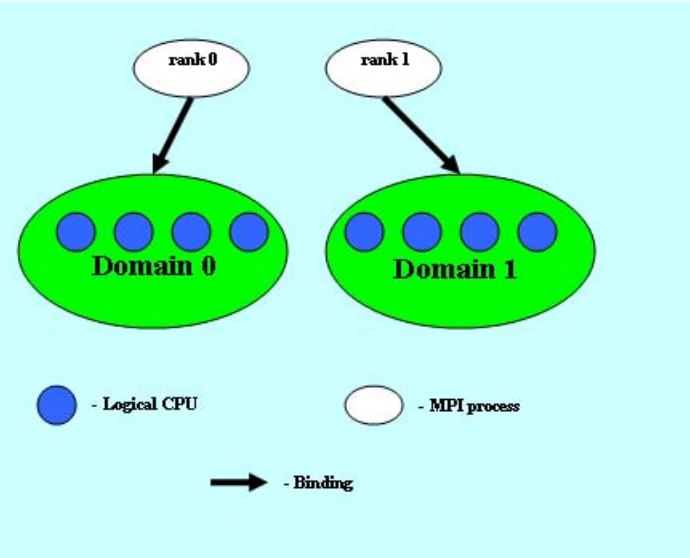
NOTE: If a number of processes is greater than a number of CPUs for pinning, a process list is wrapped on a processor list.

4.2.3 Interoperability with OpenMP*

I_MPI_PIN_DOMAIN

The Intel MPI Library provides an additional environment variable to control process pinning for hybrid Intel MPI/OpenMP* applications.

The 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.



Picture 4.2-1 Domain Example

Each MPI process can create a number of children threads for running within the corresponding domain. The process threads can freely migrate from one logical processor to another within the particular domain. There are no any domains defined by default so they should be defined explicitly.

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

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

The `I_MPI_PIN_DOMAIN` variable has the following syntax forms:

1. Domain description through multi-core terms
2. Domain description through domain size and domain member layout
3. Explicit domain description through bit mask

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 this 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 this node. The recommended value is `socket`
`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 separate domain
cache3	Logical processors that share a particular level 3 cache are arranged into a separate domain
cache	The largest domain among cache1, cache2, and cache3 is selected

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 the positive decimal number *<n>*

<layout>	Ordering of domain members. If *<layout>* is omitted then compact is assumed
platform	Domain members are ordered on the base of 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, etc.). 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, etc.)

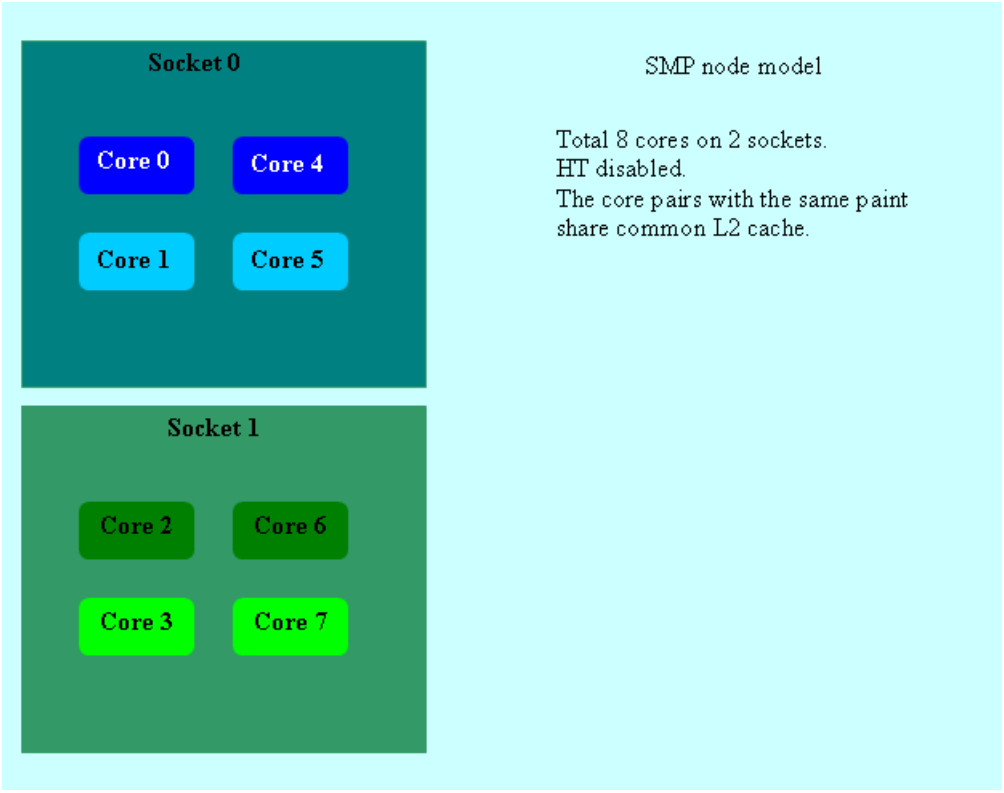
Explicit Domain Mask

I_MPI_PIN_DOMAIN=*<masklist>*

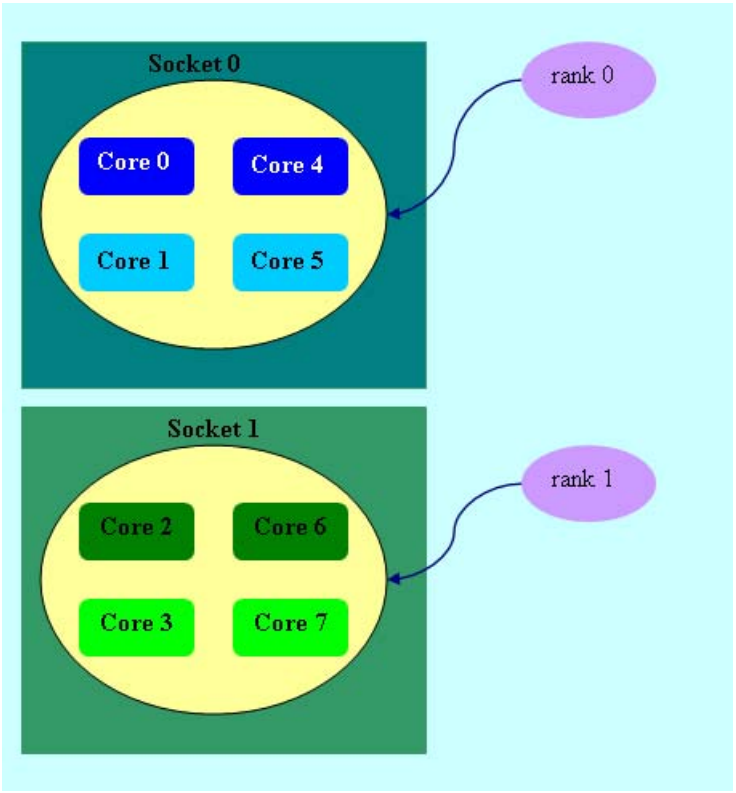
<masklist>	Define domains through the comma separated list of hexadecimal numbers (domain masks)
$[m_1,\ldots,m_n]$	Each m_i number defines one separate domain. The following rule is used: the i^{th} logical processor is included into the domain if the corresponding m_i value is set to 1. All remaining processors are put into a separate domain. BIOS numbering is used

NOTE: In order to pin OpenMP processes/threads inside the domain the corresponding OpenMP feature (KMP_AFFINITY environment variable) may be used.

See the following model of the SMP node in the examples below:

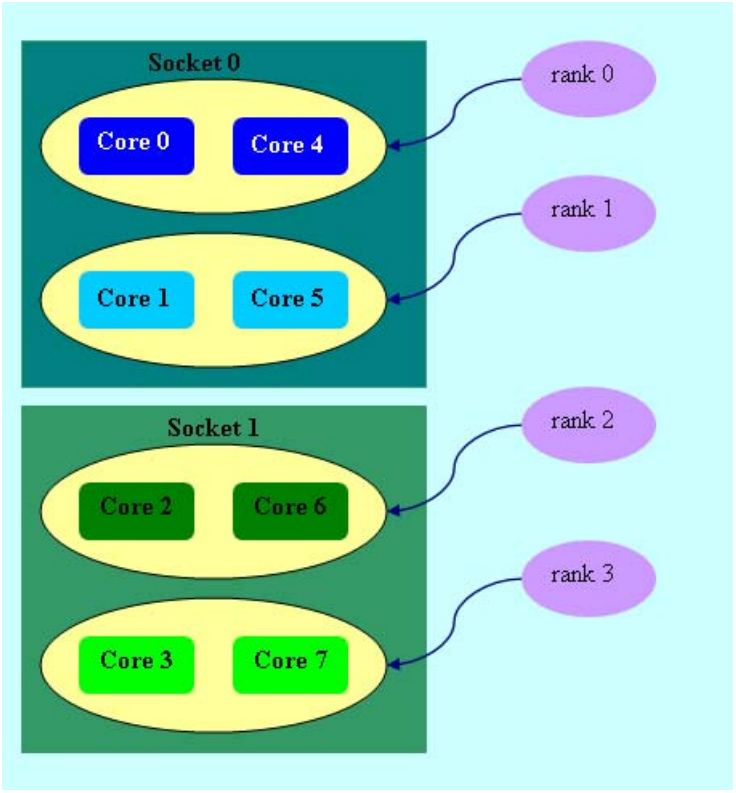


Picture 4.2-2 Node Model



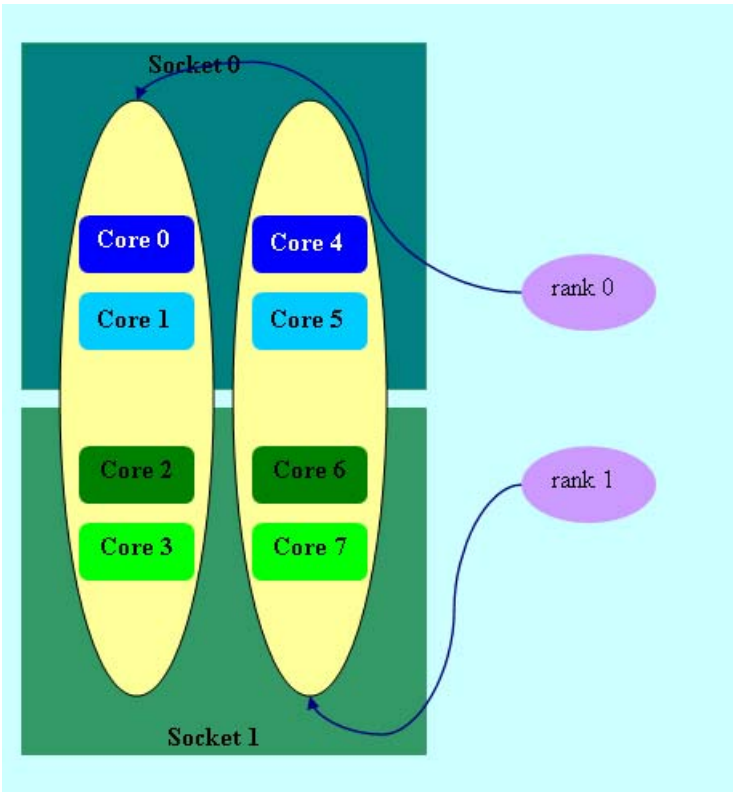
Picture 4.2-3 `mpiexec -n 2 -env I_MPI_PIN_DOMAIN socket .\a.out`

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.



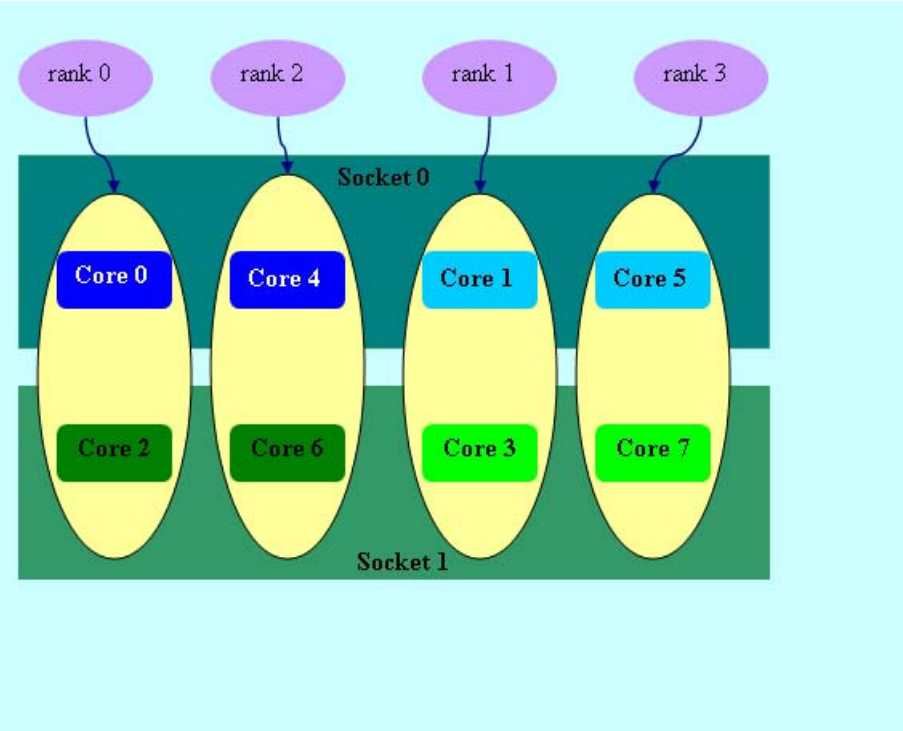
Picture 4.2-4 `mpiexec -n 4 -env I_MPI_PIN_DOMAIN cache2 .\a.out`

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



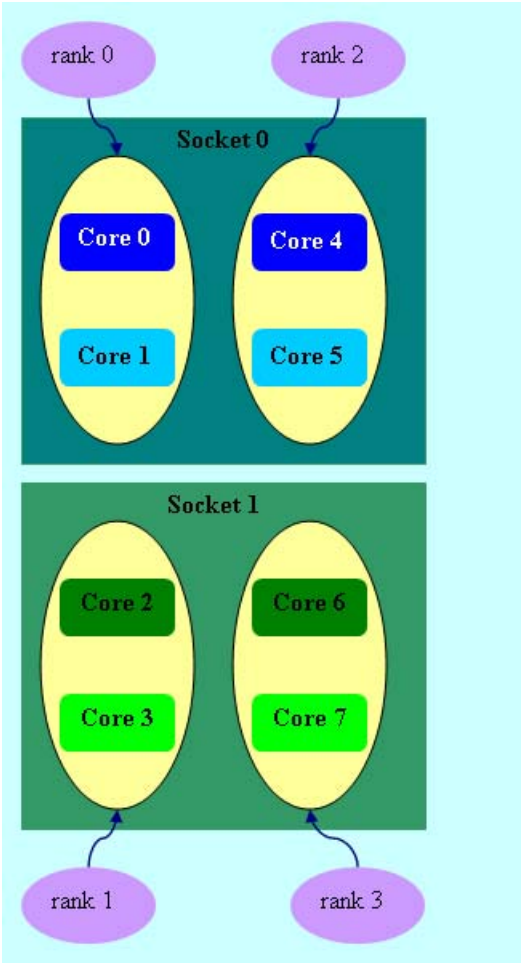
Picture 4.2-5 `mpiexec -n 2 -env I_MPI_PIN_DOMAIN 4:platform .\a.out`

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



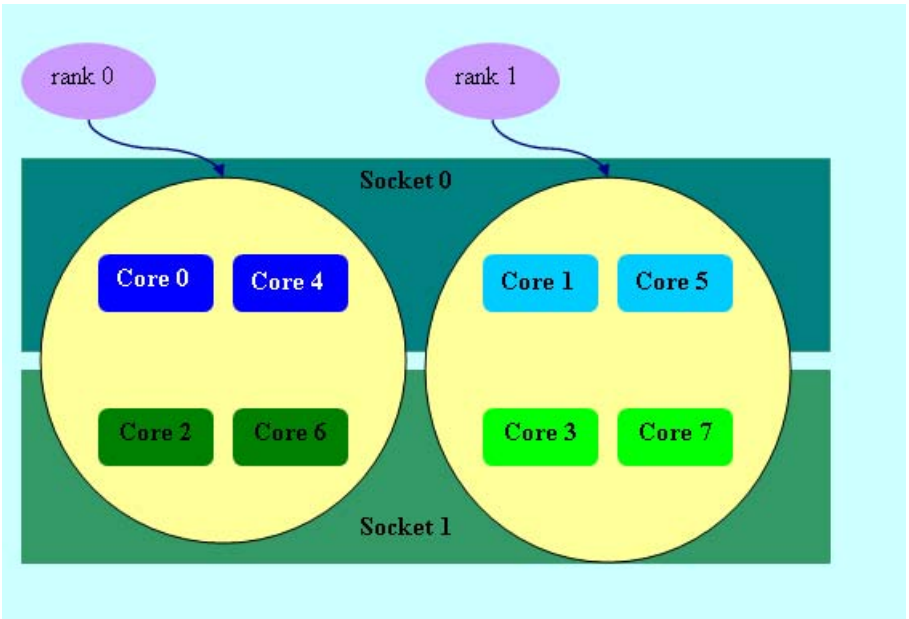
Picture 4.2-6 `mpiexec -n 4 -env I_MPI_PIN_DOMAIN auto:scatter .\a.out`

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



Picture 4.2-7 `mpiexec -n 4 -env I_MPI_PIN_DOMAIN omp:platform .\a.out setenv OMP_NUM_THREADS=2`

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.



Picture 4.2-8 `mpiexec -n 2 -env I_MPI_PIN_DOMAIN [55,aa] .\a.out`

The fist domain is defined by the 0x55 mask. It contains all cores with even numbers {0,2,4,6}. The second domain is defined by the 0xAA mask. It contains all cores with odd numbers {1,3,5,7}.

4.3 Fabrics Control

4.3.1 Communication Fabrics Control

I_MPI_FABRICS

(I_MPI_DEVICE)

Select the particular network fabrics to be used.

Syntax

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

Where `<fabric> := {shm, dapl, tcp}`

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

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

Deprecated Syntax

`I_MPI_DEVICE=<device>[:<provider>]`

Arguments

<fabric>	Define a network fabric
shm	Shared-memory
dapl	DAPL–capable network fabrics, such as InfiniBand*, iWarp*, Dolphin*, and XPMEM* (through DAPL*)
tcp	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand* (through IPoIB*)

Correspondence with I_MPI_DEVICE

<device>	*<fabric>*
sock	tcp
shm	shm
ssm	shm:tcp
rdma	dapl
rdssm	shm:dapl
<provider>	Optional DAPL* provider name (only for the rdma and the rdssm devices) I_MPI_DAPL_PROVIDER=*<provider>*

Use the *<provider>* specification only for the {rdma,rdssm} devices.

For example, to select the OFED* InfiniBand* device, use the following command:

```
$ mpiexec -n <# of processes> \
```

```
-env I_MPI_DEVICE rdssm:OpenIB-cma <executable>
```

For these devices, if <provider> is not specified, the first DAPL* provider in the /etc/dat.conf file is used.

Description

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

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

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

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

For example, to select shared-memory as the chosen fabric, use the following command:

```
$ mpiexec -n <# of processes> -env I_MPI_FABRICS shm <executable>
```

To select shared-memory and DAPL–capable network fabric as the chosen fabric combination, use the following command:

```
$ mpiexec -n <# of processes> -env I_MPI_FABRICS shm:dapl <executable>
```

To enable Intel® MPI Library to select most appropriate fabric combination automatically, use the following command:

```
$ mpiexec -n <# of procs> -perhost <# of procs per host> <executable>
```

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

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

or

```
[0] MPI startup(): tcp data transfer mode
```

NOTE: If the I_MPI_FABRICS variable and the I_MPI_DEVICE variable are set at the same level (command line, environment, configuration files), the I_MPI_FABRICS environment variable has higher priority than the I_MPI_DEVICE variable.

I_MPI_FABRICS_LIST

Define a fabrics list.

Syntax

I_MPI_FABRICS_LIST=<fabrics list>

Where <fabrics list> := <fabric>,…,<fabric>

<fabric> := {dapl, tcp}

Arguments

<fabrics list>	Specify a fabrics list. The default value is dapl, tcp

Description

Set this variable to define a list of fabrics. The library uses the fabrics list to choose the most appropriate fabrics combination automatically. For information on fabric combination, see *I_MPI_FABRICS*.

For example, if I_MPI_FABRICS_LIST=dapl,tcp, I_MPI_FABRICS is not defined and the initialization of DAPL-capable network fabrics fails, the library falls back to TCP-capable network fabric. For information on fallback, see *I_MPI_FALLBACK*.

I_MPI_FALLBACK

(I_MPI_FALLBACK_DEVICE)

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

Syntax

I_MPI_FALLBACK=*<arg>*

Deprecated Syntax

I_MPI_FALLBACK_DEVICE=*<arg>*

Arguments

<arg>	Binary indicator
enable | yes | on | 1	Fall back to the first available fabric. This is the default value if I_MPI_FABRICS(I_MPI_DEVICE) environment variable is not set
disable | no | off | 0	Terminate the job if the library can not initialize the one of the fabrics selected by the I_MPI_FABRICS environment variable. This is the default value if you set I_MPI_FABRICS(I_MPI_DEVICE) environment variable

Description

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

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

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

NOTE: If I_MPI_FABRICS is set and I_MPI_FALLBACK=enable, the library falls back to fabrics with higher numbers in the fabrics list. For example, if I_MPI_FABRICS=dapl, I_MPI_FABRICS_LIST=dapl,tcp, I_MPI_FALLBACK=enable and the initialization of DAPL-capable network fabrics fails, the library falls back to TCP-capable network fabric.

I_MPI_EAGER_THRESHOLD

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

Syntax

I_MPI_EAGER_THRESHOLD=*<nbytes>*

Arguments

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

Description

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

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

I_MPI_INTRANODE_EAGER_THRESHOLD

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

Syntax

I_MPI_INTRANODE_EAGER_THRESHOLD=*<nbytes>*

Arguments

<nbytes>	Define the threshold for DAPL* intra-node communication
> 0	The default *<nbytes>* value is equal to 262144 bytes for all fabrics except shm. For shm, cutover point is equal to the value of I_MPI_SHM_CELL_SIZE environment variable

Description

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

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

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

I_MPI_INTRANODE_DIRECT_COPY

Turn on/off the intranode direct copy communication mode.

Syntax

I_MPI_INTRANODE_DIRECT_COPY=*<arg>*

Arguments

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

Description

Set this variable to specify the communication mode within the node. If the direct copy communication mode is enabled, data transfer algorithms are selected according to the following scheme:

- Messages shorter than or equal to the threshold value of the I_MPI_INTRANODE_EAGER_THRESHOLD variable are transferred using the shared memory.
- Messages larger than the threshold value of the I_MPI_INTRANODE_EAGER_THRESHOLD variable are transferred through the direct process memory access.

I_MPI_SPIN_COUNT

Control the spin count value.

Syntax

I_MPI_SPIN_COUNT=*<scount>*

Arguments

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

Description

Set the spin count limit. The loop for polling the fabric(s) spins *<scount>* times before freeing the processes if no incoming messages are received for processing. Smaller values for *<scount>* cause the Intel® MPI Library to release the processor more frequently.

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

I_MPI_SCALABLE_OPTIMIZATION

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

Syntax

I_MPI_SCALABLE_OPTIMIZATION=*<arg>*

Deprecated Syntax

I_MPI_SOCK_SCALABLE_OPTIMIZATION=*<arg>*

Arguments

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

Description

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

I_MPI_WAIT_MODE

Turn on/off wait mode.

Syntax

I_MPI_WAIT_MODE=<arg>

Arguments

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

Description

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

I_MPI_DYNAMIC_CONNECTION

(I_MPI_USE_DYNAMIC_CONNECTIONS)

Turn on/off the dynamic connection establishment.

Syntax

I_MPI_DYNAMIC_CONNECTION=<arg>

Deprecated Syntax

I_MPI_USE_DYNAMIC_CONNECTIONS=<arg>

Arguments

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

Description

Set this variable to control dynamic connection establishment.

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

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

4.3.2 Shared Memory Control

I_MPI_SHM_CACHE_BYPASS

(I_MPI_CACHE_BYPASS)

Control the message transfer algorithm for the shared memory.

Syntax

I_MPI_SHM_CACHE_BYPASS=<arg>

Deprecated Syntax

I_MPI_CACHE_BYPASS=<arg>

Arguments

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

Description

Set this variable to enable/disable message transfer bypass cache for the shared memory. When enabled, the MPI sends the messages greater than or equal in size to the value specified by the I_MPI_SHM_CACHE_BYPASS_THRESHOLD environment variable through the bypass cache. By default, this feature is enabled on the IA-32 architecture and Intel® 64 architectures. This feature is not supported on IA-64 architecture systems.

I_MPI_SHM_CACHE_BYPASS_THRESHOLDS

(I_MPI_CACHE_BYPASS_THRESHOLDS)

Set the messages copying algorithm threshold.

Syntax

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

Deprecated Syntax

I_MPI_CACHE_BYPASS_THRESHOLDS=<nb_send>,[<nb_recv>,[<nb_send_pk>,[<nb_recv_pk>]]]

Arguments

<nb_send>	Set the threshold for sent messages in the following situations: Processes are pinned on cores that are not located in the same physical processor package Processes are not pinned
⩾ 0	The default <nb_send> value is 16,384 bytes
<nb_recv>	Set the threshold for received messages in the following situations: Processes are pinned on cores that are not located in the same physical processor package Processes are not pinned
⩾ 0	The default <nb_recv> value is 2,097,152 bytes
<nb_send_pk>	Set the threshold for sent messages when processes are pinned on cores located in the same physical processor package
⩾ 0	The default <nb_send_pk> value is -1 (copying bypass cache is disabled)
<nb_recv_pk>	Set the threshold for received messages when processes are pinned on cores located in the same physical processor package

⩾ 0	The default <nb_recv_pk> value is 2,097,152 bytes

Description

Set this variable to control the thresholds for the message copying algorithm. MPI copies messages greater than or equal in size to the defined threshold values so that the messages bypass the cache. The value of -1 disables cache bypass. This variable is valid only when I_MPI_SHM_CACHE_BYPASS is enabled.

I_MPI_SHM_LMT_BUFFER_NUM

(I_MPI_SHM_NUM_BUFFERS)

Change the number of shared memory buffers for Large Message Transfer (LMT) mechanism.

Syntax

I_MPI_SHM_LMT_BUFFER_NUM=<num>

Deprecated Syntax

I_MPI_SHM_NUM_BUFFERS=<num>

Arguments

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

Description

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

I_MPI_SHM_LMT_BUFFER_SIZE

(I_MPI_SHM_BUFFER_SIZE)

Change the size of shared memory buffers for LMT mechanism.

Syntax

I_MPI_SHM_LMT_BUFFER_SIZE=<nbytes>

Deprecated Syntax

I_MPI_SHM_BUFFER_SIZE=<nbytes>

Arguments

<nbytes>	The size of shared memory buffers in bytes
> 0	The default <nbytes> value is equal to 32,768 bytes

Description

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

I_MPI_SHM_CELL_NUM

Change the number of shared memory cells.

Syntax

I_MPI_SHM_CELL_NUM=<num>

Arguments

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

Description

Set this variable to define the number of shared memory cells.

I_MPI_SHM_CELL_SIZE

Change the size of shared memory cell.

Syntax

I_MPI_SHM_CELL_SIZE=<nbytes>

Arguments

<nbytes>	Size of shared memory cell in bytes
> 0	The default <nbytes> value is equal to 65,408 bytes

Description

Set this variable to define the size of shared memory cell.

I_MPI_SHM_FBOX_SIZE

Set the size of shared memory fastbox.

Syntax

I_MPI_SHM_FBOX_SIZE=<nbytes>

Arguments

<nbytes>	Size of shared memory fastbox in bytes
> 0	The default <nbytes> value is equal to 65,408 bytes

Description

Set this variable to define the size of shared memory fastbox.

I_MPI_SHM_SINGLE_SEGMENT_THRESHOLD

(I_MPI_SHM_PROC_THRESHOLD)

Change the shared memory segment(s) allocation mode for the shm device.

Syntax

I_MPI_SHM_SINGLE_SEGMENT_THRESHOLD=<nproc>

Deprecated Syntax

I_MPI_SHM_PROC_THRESHOLD=<nproc>

Arguments

<nproc>	Define the threshold of allocation mode for the shm device
> 0	The default *<nproc>* value depends on the values of the I_MPI_SHM_NUM_BUFFERS and I_MPI_SHM_BUFFER_SIZE

Description

Set this variable to change the allocation mode for the shm device.

The following modes are available for the allocation of the shared memory segment(s) for the shm device:

- If the number of processes started on the system is less than the value specified by *<nproc>*, the static mode is used. In static mode, only one common shared memory segment is allocated for all processes during the initialization stage.
- If the number of processes started on the system is more than the value specified by *<nproc>*, the dynamic mode is used. In dynamic mode, the shared memory segments are allocated for each connection.

The default value depends on the values of the I_MPI_SHM_NUM_BUFFERS and I_MPI_SHM_BUFFER_SIZE environment variables. It is equal to 90 in the case of default settings for I_MPI_SHM_NUM_BUFFERS and I_MPI_SHM_BUFFER_SIZE.

The I_MPI_DYNAMIC_CONNECTION environment variable is not applicable when MPI uses the static allocation mode.

I_MPI_SHM_BYPASS

(I_MPI_INTRANODE_SHMEM_BYPASS, I_MPI_USE_DAPL_INTRANODE)

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

Syntax

I_MPI_SHM_BYPASS=*<arg>*

Deprecated Syntaxes

I_MPI_INTRANODE_SHMEM_BYPASS=*<arg>*

I_MPI_USE_DAPL_INTRANODE=*<arg>*

Arguments

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

Description

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

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

NOTE: This variable is applicable only when shared memory and the network fabric are turned on either by default or by setting the `I_MPI_FABRICS` environment variable to `shm:<fabric>`. This mode is available only for `dapl` and `tcp` fabrics in MPI 4.0.

4.3.3 DAPL-capable Network Fabrics Control

I_MPI_DAPL_PROVIDER

Define the DAPL provider to load.

Syntax

`I_MPI_DAPL_PROVIDER=<name>`

Arguments

`<name>`	Define the name of DAPL provider to load

Description

Set this variable to define the name of DAPL provider to load. This name is also defined in the `dat.conf` configuration file. The DAPL provider name can be also specified inside `I_MPI_FABRICS` variable as `I_MPI_FABRICS=dapl` or `I_MPI_FABRICS=shm:dapl`.

I_MPI_DAT_LIBRARY

Select the DAT library to be used for DAPL provider.

Syntax

`I_MPI_DAT_LIBRARY=<library>`

Arguments

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

Description

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

I_MPI_DAPL_TRANSLATION_CACHE

(I_MPI_RDMA_TRANSLATION_CACHE)

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

Syntax

`I_MPI_DAPL_TRANSLATION_CACHE=<arg>`

Deprecated Syntax

`I_MPI_RDMA_TRANSLATION_CACHE=<arg>`

Arguments

`<arg>`	Binary indicator
`enable | yes | on | 1`	Turn on the memory registration cache. This is the default

disable | no | off | 0	Turn off the memory registration cache

Description

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

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

I_MPI_DAPL_DIRECT_COPY_THRESHOLD

(I_MPI_RDMA_EAGER_THRESHOLD, RDMA_IBA_EAGER_THRESHOLD)

Change the threshold of the DAPL direct-copy protocol.

Syntax

I_MPI_DAPL_DIRECT_COPY_THRESHOLD=*<nbytes>*

Deprecated Syntaxes

I_MPI_RDMA_EAGER_THRESHOLD=*<nbytes>*

RDMA_IBA_EAGER_THRESHOLD=*<nbytes>*

Arguments

<nbytes>	Define the DAPL direct-copy protocol threshold
> 0	The default *<nbytes>* value is equal to 16,456 bytes

Description

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

- Messages shorter than or equal to *<nbytes>* are sent through the internal pre-registered buffers. It involves additional calls of memcpy() function on sender and receiver sides. This approach is faster for short messages.
- Messages larger than *<nbytes>* are sent using the direct copy protocol. It does not use any buffering but involves registration of memory on sender and receiver sides. The data is transferred directly from a sender to a receiver without calling memcpy()function. This approach is faster for large messages.

I_MPI_DAPL_DYNAMIC_CONNECTION_MODE

(I_MPI_DYNAMIC_CONNECTION_MODE, I_MPI_DYNAMIC_CONNECTIONS_MODE)

Choose the algorithm for establishing the DAPL* connections.

Syntax

I_MPI_DAPL_DYNAMIC_CONNECTION_MODE=*<arg>*

Deprecated Syntax

I_MPI_DYNAMIC_CONNECTION_MODE=*<arg>*

I_MPI_DYNAMIC_CONNECTIONS_MODE=*<arg>*

Arguments

<arg>	Mode selector
reject	Deny one of the two simultaneous connection requests. This is the

	default
disconnect	Deny one of the two simultaneous connection requests after both connections have been established

Description

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

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

I_MPI_DAPL_SCALABLE_PROGRESS

(I_MPI_RDMA_SCALABLE_PROGRESS)

Turn on/off scalable algorithm for DAPL read progress.

Syntax

`I_MPI_DAPL_SCALABLE_PROGRESS=<arg>`

Deprecated Syntax

`I_MPI_RDMA_SCALABLE_PROGRESS=<arg>`

Arguments

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

Description

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

I_MPI_DAPL_BUFFER_NUM

(I_MPI_RDMA_BUFFER_NUM, NUM_RDMA_BUFFER)

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

Syntax

`I_MPI_DAPL_BUFFER_NUM=<nbuf>`

Deprecated Syntaxes

`I_MPI_RDMA_BUFFER_NUM=<nbuf>`

`NUM_RDMA_BUFFER=<nbuf>`

Arguments

<nbuf>	Define the number of buffers for each pair in a process group
> 0	The default value is 16

Description

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

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

I_MPI_DAPL_BUFFER_SIZE

(I_MPI_RDMA_BUFFER_SIZE, I_MPI_RDMA_VBUF_TOTAL_SIZE)

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

Syntax

`I_MPI_DAPL_BUFFER_SIZE=<nbytes>`

Deprecated Syntaxes

`I_MPI_RDMA_BUFFER_SIZE=<nbytes>`

`I_MPI_RDMA_VBUF_TOTAL_SIZE=<nbytes>`

Arguments

<nbytes>	Define the size of pre-registered buffers
> 0	The default *<nbytes>* value is equal to 16,640 bytes

Description

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

I_MPI_DAPL_RNDV_BUFFER_ALIGNMENT

(I_MPI_RDMA_RNDV_BUFFER_ALIGNMENT, I_MPI_RDMA_RNDV_BUF_ALIGN)

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

Syntax

`I_MPI_DAPL_RNDV_BUFFER_ALIGNMENT=<arg>`

Deprecated Syntaxes

`I_MPI_RDMA_RNDV_BUFFER_ALIGNMENT=<arg>`

`I_MPI_RDMA_RNDV_BUF_ALIGN=<arg>`

Arguments

<arg>	Define the alignment for the sending buffer
> 0 and a power of 2	The default value is 128

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

I_MPI_DAPL_RDMA_RNDV_WRITE

(I_MPI_RDMA_RNDV_WRITE, I_MPI_USE_RENDEZVOUS_RDMA_WRITE)

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

Syntax

`I_MPI_DAPL_RDMA_RNDV_WRITE=<arg>`

Deprecated Syntaxes

`I_MPI_RDMA_RNDV_WRITE=<arg>`

`I_MPI_USE_RENDEZVOUS_RDMA_WRITE=<arg>`

Arguments

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

Description

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

I_MPI_DAPL_CHECK_MAX_RDMA_SIZE

(I_MPI_RDMA_CHECK_MAX_RDMA_SIZE)

Check the value of the DAPL attribute `max_rdma_size`.

Syntax

`I_MPI_DAPL_CHECK_MAX_RDMA_SIZE=<arg>`

Deprecated Syntax

`I_MPI_RDMA_CHECK_MAX_RDMA_SIZE=<arg>`

Arguments

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

Description

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

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

I_MPI_DAPL_MAX_MSG_SIZE

(I_MPI_RDMA_MAX_MSG_SIZE)

Control message fragmentation threshold.

Syntax

I_MPI_DAPL_MAX_MSG_SIZE=<*nbytes*>

Deprecated Syntax

I_MPI_RDMA_MAX_MSG_SIZE=<*nbytes*>

Arguments

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

Description

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

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

I_MPI_DAPL_CONN_EVD_SIZE

(I_MPI_RDMA_CONN_EVD_SIZE, I_MPI_CONN_EVD_QLEN)

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

Syntax

I_MPI_DAPL_CONN_EVD_SIZE=<*size*>

Deprecated Syntaxes

I_MPI_RDMA_CONN_EVD_SIZE=<*size*>
I_MPI_CONN_EVD_QLEN=<size>

Arguments

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

Description

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

I_MPI_DAPL_SR_THRESHOLD

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

Syntax

I_MPI_DAPL_SR_THRESHOLD=<arg>

Arguments

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

Description

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

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

I_MPI_DAPL_SR_BUF_NUM

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

Syntax

I_MPI_DAPL_SR_BUF_NUM=<nbuf>

Arguments

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

Description

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

I_MPI_DAPL_RDMA_WRITE_IMM

(I_MPI_RDMA_WRITE_IMM)

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

Syntax

I_MPI_DAPL_RDMA_WRITE_IMM=<arg>

Deprecated syntax

I_MPI_RDMA_WRITE_IMM=<arg>

Arguments

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

Description

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

4.3.4 TCP-capable Network Fabrics Control

I_MPI_TCP_NETMASK

(I_MPI_NETMASK)

Choose the network interface for MPI communication over TCP-capable network fabrics.

Syntax

`I_MPI_TCP_NETMASK=<arg>`

Arguments

<arg>	Define the network interface (string parameter)
<interface_mnemonic>	Mnemonic of the network interface: `ib` or `eth`
`ib`	Select IPoIB*
`eth`	Select Ethernet. This is the default value
<interface_name>	Name of the network interface Usually the UNIX* driver name followed by the unit number
<network_address>>	Network address. The trailing zero bits imply netmask
<network_address/netmask>	Network address. The *<netmask>* value specifies the netmask length
<list of interfaces>	A colon separated list of network addresses and interface names

Description

Set this variable to choose the network interface for MPI communication over TCP-capable network fabrics. If you specify a list of interfaces, the first available interface on the node will be used for communication.

Examples

- Use the following setting to select the IP over InfiniBand (IPoIB) fabric:
  `I_MPI_TCP_NETMASK=ib`
- Use the following setting to select the specified network interface for socket communications:
  `I_MPI_TCP_NETMASK=ib0`
- Use the following setting to select the specified network for socket communications. This setting implies the `255.255.0.0` netmask:
  `I_MPI_TCP_NETMASK=192.169.0.0`
- Use the following setting to select the specified network for socket communications with netmask set explicitly:
  `I_MPI_TCP_NETMASK=192.169.0.0/24`
- Use the following setting to select the specified network interfaces for socket communications:
  `I_MPI_TCP_NETMASK=192.169.0.5/24:ib0:192.169.0.0`

4.4 Dynamic Process Support

The Intel® MPI Library provides support for the MPI-2 process model what allows creation and cooperative termination of processes after an MPI application has started. It provides

- a mechanism to establish communication between the newly created processes and the existing MPI application
- a process attachment mechanism to establish communication between two existing MPI applications even when one of them does not spawn the other

The set of hosts indicated within `<machine_file>` (see *mpiexec* for details) is used for placement of spawned processes. The spawned processes are placed onto different hosts in round-robin or per-host fashion. The first spawned process is placed after the last process of the parent group. A specific network fabric combination is selected using the usual fabrics selection algorithm (see *I_MPI_FABRICS* and *I_MPI_FABRICS_LIST* for details).

For example, to run a dynamic application, use the following command:

```
$ mpiexec -n 1 -machinefile smpd.hosts -gwdir <path_to_executable>
-genv I_MPI_FABRICS shm:tcp <spawn_app>
```

In this example, the `spawn_app` spawns 4 dynamic processes. If the `smpd.hosts` contains the following information:

```
host1
host2
host3
host4
```

the original spawing process is placed on `host1`, while the dynamic processes is distributed as follows: 1 – on `host2`, 2 – on `host3`, 3 – on `host4`, and 4 – again on `host1`.

If the `smpd.hosts` contains the following information:

```
host1:2
host2:2
```

the ordinary process is placed on `host1`, while the dynamic processes is distributed as follows: 1 – on `host1`, 2 and 3 – on `host2`, and 4 – on `host1`.

To run a client-server application, use the following command on the server host:

```
$ mpiexec -n 1 -genv I_MPI_FABRICS shm:tcp <server_app> > <port_name>
```

and the following command on the intended client hosts:

```
$ mpiexec -n 1 -genv I_MPI_FABRICS shm:tcp <client_app> < <port_name>
```

To run a simple MPI_COMM_JOIN based application, use the following commands on the intended host:

```
$ mpiexec -n 1 -genv I_MPI_FABRICS shm:tcp <join_server_app> <
<port_number>
```

```
$ mpiexec -n 1 -genv I_MPI_FABRICS shm:tcp <join_client_app> <
<port_number>
```

4.5 Collective Operation Control

Each collective operation in the Intel® MPI Library supports a number of communication algorithms. In addition to highly optimized default settings, the library provides two ways to control the algorithm selection explicitly: the novel I_MPI_ADJUST environment variable family and the deprecated I_MPI_MSG environment variable family. They are described in the following sections.

4.5.1 I_MPI_ADJUST Family

I_MPI_ADJUST_<*opname*>

Control collective operation algorithm selection.

Syntax

I_MPI_ADJUST_<*opname*>=<*algid*>[:<*conditions*>][;<*algid*>:<*conditions*>[…]]

Arguments

<*algid*>	Algorithm identifier
>= 0	The default value of zero selects optimized default settings

<*conditions*>	A comma separated list of conditions. An empty list selects al 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 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. See below.

Table 4.5-1 Environment Variables, Collective Operations, and Algorithms

Environment Variable	Collective Operation	Algorithms
I_MPI_ADJUST_ALLGATHER	MPI_Allgather	1. Recursive doubling algorithm 2. Bruck's algorithm 3. Ring algorithm 4. Topology aware Gatherv + Bcast algorithm
I_MPI_ADJUST_ALLGATHERV	MPI_Allgatherv	1. Recursive doubling algorithm 2. Bruck's algorithm 3. Ring algorithm 4. Topology aware Gatherv + Bcast algorithm

I_MPI_ADJUST_ALLREDUCE	MPI_Allreduce	1. Recursive doubling algorithm 2. Rabenseifner's algorithm 3. Reduce + Bcast algorithm 4. Topology aware Reduce + Bcast algorithm 5. Binomial gather + scatter algorithm 6. Topology aware binominal gather + scatter algorithm 7. Ring algorithm
I_MPI_ADJUST_ALLTOALL	MPI_Alltoall	1. Bruck's algorithm 2. Isend/Irecv + waitall algorithm 3. Pair wise exchange algorithm 4. Plum's algorithm
I_MPI_ADJUST_ALLTOALLV	MPI_Alltoallv	1. Isend/Irecv + waitall algorithm 2. Plum's algorithm
I_MPI_ADJUST_ALLTOALLW	MPI_Alltoallw	1. Isend/Irecv + waitall algorithm
I_MPI_ADJUST_BARRIER	MPI_Barrier	1. Dissemination algorithm 2. Recursive doubling algorithm 3. Topology aware dissemination algorithm 4. Topology aware recursive doubling algorithm 5. Binominal gather + scatter algorithm 6. Topology aware binominal gather + scatter algorithm
I_MPI_ADJUST_BCAST	MPI_Bcast	1. Binomial algorithm 2. Recursive doubling algorithm 3. Ring algorithm 4. Topology aware binomial algorithm 5. Topology aware recursive doubling algorithm 6. Topology aware ring algorithm 7. Shumilin's bcast algorithm
I_MPI_ADJUST_EXSCAN	MPI_Exscan	1. Partial results gathering algorithm 2. Partial results gathering regarding algorithm layout of processes
I_MPI_ADJUST_GATHER	MPI_Gather	1. Binomial algorithm 2. Topology aware binomial algorithm 3. Shumilin's algorithm

I_MPI_ADJUST_GATHERV	MPI_Gatherv	1. Linear algorithm 2. Topology aware linear algorithm
I_MPI_ADJUST_REDUCE_SCATTER	MPI_Reduce_scatter	1. Recursive having algorithm 2. Pair wise exchange algorithm 3. Recursive doubling algorithm 4. Reduce + Scatterv algorithm 5. Topology aware Reduce + Scatterv algorithm
I_MPI_ADJUST_REDUCE	MPI_Reduce	1. Shumilin's algorithm 2. Binomial algorithm 3. Topology aware Shumilin's algorithm 4. Topology aware binomial algorithm
I_MPI_ADJUST_SCAN	MPI_Scan	1. Partial results gathering algorithm 2. Topology aware partial results gathering algorithm
I_MPI_ADJUST_SCATTER	MPI_Scatter	1. Binomial algorithm 2. Topology aware binomial algorithm 3. Shumilin's algorithm
I_MPI_ADJUST_SCATTERV	MPI_Scatterv	1. Linear algorithm 2. Topology aware linear algorithm

The message size calculation rules for the collective operations are described in table below. Here, **n/a** means that the corresponding interval *<l>-<m>* should be omitted.

Table 4.5-2 Message Collective Functions

Collective Function	Message Size Formula
MPI_Allgather	recv_count*recv_type_size
MPI_Allgatherv	total_recv_count*recv_type_size
MPI_Allreduce	count*type_size
MPI_Alltoall	send_count*send_type_size
MPI_Alltoallv	n/a
MPI_Alltoallw	n/a
MPI_Barrier	n/a
MPI_Bcast	count*type_size
MPI_Exscan	count*type_size
MPI_Gather	recv_count*recv_type_size if MPI_IN_PLACE is used, otherwise send_count*send_type_size

MPI_Gatherv	n/a
MPI_Reduce_scatter	total_recv_count*type_size
MPI_Reduce	count*type_size
MPI_Scan	count*type_size
MPI_Scatter	send_count*send_type_size if MPI_IN_PLACE is used, otherwise recv_count*recv_type_size
MPI_Scatterv	n/a

Examples:

1. Use the following settings to select second algorithm for MPI_Reduce operation:
   I_MPI_ADJUST_REDUCE=2
2. Use the following settings to define algorithms for MPI_scatter operation:
   I_MPI_ADJUST_REDUCE_SCATTER=4:0-100,5001-10000;1:101-3200,2:3201-5000;3

   In this case algorithm four will be used for the message sizes from 0 up to 100 bytes and from 5001 to 10000 bytes, algorithm one will be used for the message sizes from 101 up to 3200 bytes, algorithm two will be used for the message sizes from 3201 up to 5000 bytes, and algorithm three will be used for all other messages.

4.5.2 I_MPI_MSG Family

These variables are deprecated and supported mostly for backward compatibility. Use the I_MPI_ADJUST environment variable family whenever possible.

I_MPI_FAST_COLLECTIVES

Control default library behavior during selection of appropriate collective algorithm for specific execution situation.

Syntax

I_MPI_FAST_COLLECTIVES=<*arg*>

Arguments

<*arg*>	Binary indicator
enable | yes | on | 1	Fast collective algorithms are used. This is the default value
disable | no | off | 0	Slower and safer collective algorithms are used

Description

The Intel® MPI Library uses advanced collective algorithms designed for better application performance by default. The implementation makes the following assumptions:

- It is safe to utilize the flexibility of the MPI standard regarding the order of execution of the collective operations to take advantage of the process layout and other opportunities.
- There is enough memory available for allocating additional internal buffers.

Set the I_MPI_FAST_COLLECTIVES variable to disable if you need to obtain results that do not depend on the physical process layout or other factors.

NOTE: Some optimizations controlled by this variable are of an experimental nature. In case of failure, turn off the collective optimizations and repeat the run.

I_MPI_BCAST_NUM_PROCS

Control MPI_Bcast algorithm thresholds.

Syntax

I_MPI_BCAST_NUM_PROCS=<*nproc*>

Arguments

<*nproc*>	Define the number of processes threshold for choosing the MPI_Bcast algorithm
> 0	The default value is 8

I_MPI_BCAST_MSG

Control MPI_Bcast algorithm thresholds.

Syntax

I_MPI_BCAST_MSG=<*nbytes1,nbytes2*>

Arguments

<*nbytes1,nbytes2*>	Define the message size threshold range (in bytes) for choosing the MPI_Bcast algorithm
> 0 nbytes2 >= nbytes1	The default pair of values is 12288,524288

Description

Set these variables to control the selection of the three possible MPI_Bcast algorithms according to the following scheme (See Table 4.5-1 for algorithm descriptions):

1. The first algorithm is selected if the message size is less than <*nbytes1*>, or the number of processes in the operation is less than <*nproc*>.
2. The second algorithm is selected if the message size is greater than or equal to <*nbytes1*> and less than <*nbytes2*>, and the number of processes in the operation is a power of two.
3. If none of the above conditions is satisfied, the third algorithm is selected.

I_MPI_ALLTOALL_NUM_PROCS

Control MPI_Alltoall algorithm thresholds.

Syntax

I_MPI_ALLTOALL_NUM_PROCS=<*nproc*>

Arguments

<*nproc*>	Define the number of processes threshold for choosing the MPI_Alltoall algorithm
> 0	The default value is 8

I_MPI_ALLTOALL_MSG

Control MPI_Alltoall algorithm thresholds.

Syntax

I_MPI_ALLTOALL_MSG=<*nbytes1,nbytes2*>

Arguments

<nbytes1,nbytes2>	Defines the message size threshold range (in bytes) for choosing the MPI_Alltoall algorithm
> 0 nbytes2 >= nbytes1	The default pair of values is 256,32768

Description

Set these variables to control the selection of the three possible MPI_Alltoall algorithms according to the following scheme (See Table 4.5-1 for algorithm descriptions):

1. The first algorithm is selected if the message size is greater than or equal to *<nbytes1>*, and the number of processes in the operation is not less than *<nproc>*.
2. The second algorithm is selected if the message size is greater than *<nbytes1>* and less than or equal to *<nbytes2>*, or if the message size is less than *<nbytes2>* and the number of processes in the operation is less than *<nproc>*.
3. If none of the above conditions is satisfied, the third algorithm is selected.

I_MPI_ALLGATHER_MSG

Control MPI_Allgather algorithm thresholds.

Syntax

I_MPI_ALLGATHER_MSG=*<nbytes1,nbytes2>*

Arguments

<nbytes1,nbytes2>	Define the message size threshold range (in bytes) for choosing the MPI_Allgather algorithm
> 0 nbytes2 >= nbytes1	The default pair of values is 81920,524288

Description

Set this variable to control the selection of the three possible MPI_Allgather algorithms according to the following scheme (See Table 4.5-1 for algorithm descriptions):

1. The first algorithm is selected if the message size is less than *<nbytes2>* and the number of processes in the operation is a power of two.
2. The second algorithm is selected if the message size is less than *<nbytes1>* and number of processes in the operation is not a power of two.
3. If none of the above conditions is satisfied, the third algorithm is selected.

I_MPI_ALLREDUCE_MSG

Control MPI_Allreduce algorithm thresholds.

Syntax

I_MPI_ALLREDUCE_MSG=*<nbytes>*

Arguments

<nbytes>	Define the message size threshold (in bytes) for choosing the MPI_Allreduce algorithm
> 0	The default value is 2048

Description

Set this variable to control the selection of the two possible MPI_Allreduce algorithms according to the following scheme (See Table 4.5-1 for algorithm descriptions):

1. The first algorithm is selected if the message size is less than or equal *<nbytes>*, or the reduction operation is user-defined, or the count argument is less than the nearest power of two less than or equal to the number of processes.
2. If the above condition is not satisfied, the second algorithm is selected.

I_MPI_REDSCAT_MSG

Control the MPI_Reduce_scatter algorithm thresholds.

Syntax

I_MPI_REDSCAT_MSG=*<nbytes1,nbytes2>*

Arguments

<nbytes>	Define the message size threshold range (in bytes) for choosing the MPI_Reduce_scatter algorithm
> 0	The default value is 512,524288

Description

Set this variable to control the selection of the three possible MPI_Reduce_scatter algorithms according to the following scheme (See Table 4.5-1 for algorithm descriptions):

1. The first algorithm is selected if the reduction operation is commutative and the message size is less than *<nbytes2>*.
2. The second algorithm is selected if the reduction operation is commutative and the message size is greater than or equal to *<nbytes2>*, or if the reduction operation is not commutative and the message size is greater than or equal to *<nbytes1>*.
3. If none of the above conditions is satisfied, the third algorithm is selected.

I_MPI_SCATTER_MSG

Control MPI_Scatter algorithm thresholds.

Syntax

I_MPI_SCATTER_MSG=*<nbytes>*

Arguments

<nbytes>	Define the buffer size threshold range (in bytes) for choosing the MPI_Scatter algorithm
> 0	The default value is 2048

Description

Set this variable to control the selection of the two possible MPI_Scatter algorithms according to the following scheme (See Table 4.5-1 for algorithm descriptions):

1. The first algorithm is selected on the intercommunicators if the message size is greater than *<nbytes>*.
2. If the above condition is not satisfied, the second algorithm is selected.

I_MPI_GATHER_MSG

Control MPI_Gather algorithm thresholds.

Syntax

I_MPI_GATHER_MSG=<*nbytes*>

Arguments

<*nbytes*>	Define the buffer size threshold range (in bytes) for choosing the MPI_Gather algorithm
> 0	The default value is 2048

Description

Set this variable to control the selection of the two possible MPI_Gather algorithms according to the following scheme (See Table 4.5-1 for algorithm descriptions):

1. The first algorithm is selected on the intercommunicators if the message size is greater than <*nbytes*>.
2. If the above condition is not satisfied, the second algorithm is selected.

4.6 Compatibility Control

The Intel® MPI Library 4.0 implements the MPI-2.1 standard. The following MPI routines are changed:

- MPI_Cart_create
- MPI_Cart_map
- MPI_Cart_sub
- MPI_Graph_create

If your application depends on the strict pre-MPI-2.1 behavior, set the environment variable I_MPI_COMPATIBILITY to 3.

I_MPI_COMPATIBILITY

Select the runtime compatibility mode.

Syntax

I_MPI_COMPATIBILITY=<*value*>

Arguments

<*value*>	Define compatibility mode
3	Enable the Intel® MPI Library 3.x compatibility mode
4	Disable the Intel® MPI Library 3.x compatibility mode and enable the Intel® MPI Library 4.0 compatible mode. This is the default value

Description

Set this variable to choose the Intel® MPI runtime compatible mode.

5 Statistics Gathering Mode

The Intel® MPI Library has a built-in statistics gathering facility that collects essential performance data without disturbing the application execution. The collected information is output onto a text file. This section describes the environment variables used to control the built-in statistics gathering facility, and provides example output files.

I_MPI_STATS

Control statistics collection.

Syntax

`I_MPI_STATS=[n-] m`

Arguments

`n, m`	Possible stats levels of the output information
`1`	Output the amount of data sent by each process
`2`	Output the number of calls and amount of transferred data
`3`	Output statistics combined according to the actual arguments
`4`	Output statistics defined by a buckets list
`10`	Output collective operation statistics for all communication contexts

Description

Set this variable to control the amount of the statistics information collected and output onto the log file. No statistics are output by default.

NOTE: `n, m` represent the positive integer numbers define range of output information. The statistics from level `n` to level `m` inclusive are output. Omitted *n* value assumes to be `1`.

I_MPI_STATS_SCOPE

Select the subsystem(s) to collect statistics for.

Syntax

`I_MPI_STATS_SCOPE=<subsystem>[:<ops>][;<subsystem>[:<ops>][…]]`

Arguments

<subsystem>	Define the target subsystem(s)
`all`	Collect statistics data for all operations. This is the default value
`coll`	Collect statistics data for all collective operations
`p2p`	Collect statistics data for all point-to-point operations

<ops>	Define the target operations as a comma separated list
`Allgather`	`MPI_Allgather`

Allgatherv	MPI_Allgatherv
Allreduce	MPI_Allreduce
Alltoall	MPI_Alltoall
Alltoallv	MPI_Alltoallv
Alltoallw	MPI_Alltoallw
Barrier	MPI_Barrier
Bcast	MPI_Bcast
Exscan	MPI_Exscan
Gather	MPI_Gather
Gatherv	MPI_Gatherv
Reduce_scatter	MPI_Reduce_scatter
Reduce	MPI_Reduce
Scan	MPI_Scan
Scatter	MPI_Scatter
Scatterv	MPI_Scatterv
Send	Standard transfers (MPI_Send, MPI_Isend, MPI_Send_init)
Bsend	Buffered transfers (MPI_Bsend, MPI_Ibsend, MPI_Bsend_init)
Csend	Point-to-point operations inside the collectives. This internal operation serves all collectives
Rsend	Ready transfers (MPI_Rsend, MPI_Irsend, MPI_Rsend_init)
Ssend	Synchronous transfers (MPI_Ssend, MPI_Issend, MPI_Ssend_init)

Description

Set this variable to select the target subsystem to collects statistics for. All collective and point-to-point operations, including the point-to-point operations performed inside the collectives are covered by default.

Examples

1. The default settings are equivalent to:
   `I_MPI_STATS_SCOPE=coll;p2p`
2. Use the following settings to collect statistics for the `MPI_Bcast`, `MPI_Reduce`, and all point-to-point operations:
   `I_MPI_STATS_SCOPE=p2p;coll:bcast,reduce`
3. Use the following settings to collect statistics for the point-to-point operations inside the collectives:
   `I_MPI_STATS_SCOPE=p2p:csend`

I_MPI_STATS_BUCKETS

Identify a list of ranges for message sizes and communicator sizes that will be used for collecting statistics.

Syntax

`I_MPI_STATS_BUCKETS=<msg>[@<proc>][,<msg>[@<proc>]]…`

Arguments

<msg>	Specify range of message sizes in bytes
<l>	Single value of message size
<l>-<m>	Range from *<l>* to *<m>*

<proc>	Specify range of processes (ranks) for collective operations
<p>	Single value of communicator size
<p>-<q>	Range from *<p>* to *<q>*

Description

Set the I_MPI_STATS_BUCKETS variable to define a set of ranges for message sizes and communicator sizes.

Level 4 of the statistics provides profile information for these ranges.

If I_MPI_STATS_BUCKETS variable is not used, then level 4 statistics is not gathered.

If a range is omitted then the maximum possible range is assumed.

Examples

To specify short messages (from 0 to 1000 bytes) and long messages (from 50000 to 100000 bytes), use the following setting:

```
-env I_MPI_STATS_BUCKETS 0-1000,50000-100000
```

To specify messages that have 16 bytes in size and circulate within four process communicators, use the following setting:

```
-env I_MPI_STATS_BUCKETS "16@4"
```

NOTE: When the *@* symbol is present, the variable value must be enclosed in quotes.

I_MPI_STATS_FILE

Define the statistics output file name.

Syntax

I_MPI_STATS_FILE=*<name>*

Arguments

<name>	Define the statistics output file name

Description

Set this variable to define the statistics output file. The stats.txt file is created in the current directory by default.

The statistics data is blocked and ordered according to the process ranks in the MPI_COMM_WORLD communicator. The timing data is presented in microseconds. For example, with the following settings in effect

```
I_MPI_STATS=4
I_MPI_STATS_SCOPE=p2p;coll:allreduce
```

the statistics output for a simple program that performs only one `MPI_Allreduce` operation may look as follows:

```
Intel(R) MPI Library Version 4.0
____ MPI Communication Statistics ____

Stats level: 4
P2P scope:< FULL >
Collectives scope:< Allreduce >

~~~~ Process 0 of 2 on node svlmpihead01 lifetime = 414.13

Data Transfers
Src     Dst    Amount(MB)   Transfers
-----------------------------------------
000 --> 000    0.000000e+00 0
000 --> 001    7.629395e-06 2
=========================================
Totals         7.629395e-06 2

Communication Activity
Operation      Volume(MB)   Calls
-----------------------------------------
P2P
Csend          7.629395e-06 2
Send           0.000000e+00 0
Bsend          0.000000e+00 0
Rsend          0.000000e+00 0
Ssend          0.000000e+00 0
Collectives
Allreduce       7.629395e-06      2
=========================================

Communication Activity by actual args
P2P
Operation      Dst    Message size Calls
-----------------------------------------------
Csend
1       1      4            2
Collectives
Operation      Context      Algo   Comm size    Message size Calls  Cost(%)
-------------------------------------------------------------------------------
Allreduce
1              0            1      2            4            2       44.96
```

```
================================================================================

~~~~ Process 1 of 2 on node svlmpihead01 lifetime = 306.13

Data Transfers
Src     Dst    Amount(MB)   Transfers
----------------------------------------
001 --> 000    7.629395e-06 2
001 --> 001    0.000000e+00 0
========================================
Totals         7.629395e-06 2

Communication Activity
Operation      Volume(MB)   Calls
----------------------------------------
P2P
Csend          7.629395e-06 2
Send           0.000000e+00 0
Bsend          0.000000e+00 0
Rsend          0.000000e+00 0
Ssend          0.000000e+00 0
Collectives
Allreduce       7.629395e-06      2
========================================

Communication Activity by actual args
P2P
Operation      Dst    Message size Calls
-----------------------------------------
Csend
1       0      4            2
Collectives
Operation      Context      Comm size    Message size Calls  Cost(%)
------------------------------------------------------------------------
Allreduce
1       0      2            4            2      37.93
========================================================================

____ End of stats.txt file ____
```

In the example above all times are measured in microseconds. The message sizes are counted in bytes. **MB** means megabyte equal to 2^{20} or 1 048 576 bytes. The process life time is calculated as a stretch of time between `MPI_Init` and `MPI_Finalize`. The **Cost** field represents a particular collective operation execution time as a percentage of the process life time.

6 ILP64 Support

The term *ILP64* means that int, long, and pointer data entities all occupy 8 bytes. This differs from the more conventional LP64 model in which only long and pointer data entities occupy 8 bytes while int entities stay at 4 byte size. More information on the historical background and the programming model philosophy can be found for example in http://www.unix.org/version2/whatsnew/lp64_wp.html

6.1 Using ILP64

Use the following options to enable the ILP64 interface

- Use the Fortran* compiler driver option `-i8` for separate compilation and the `-ilp64` option for separate linkage. For example,

```
> mpiifort -i8 -c test.f
> mpiifort -ilp64 -o test test.o
```

- Use the `mpiexec -ilp64` option to preload the ILP64 interface. For example,

```
> mpiexec -ilp64 -n 2 .\myprog
```

6.2 Known Issues and Limitations

- Datatype counts and other arguments with values larger than 2^{31}-1 are not supported.
- Special MPI types `MPI_FLOAT_INT`, `MPI_DOUBLE_INT`, `MPI_LONG_INT`, `MPI_SHORT_INT`, `MPI_2INT`, `MPI_LONG_DOUBLE_INT`, `MPI_2INTEGER` are not changed and still use a 4-byte integer field.
- Predefined communicator attributes `MPI_APPNUM`, `MPI_HOST`, `MPI_IO`, `MPI_LASTUSEDCODE`, `MPI_TAG_UB`, `MPI_UNIVERSE_SIZE`, and `MPI_WTIME_IS_GLOBAL` are returned by the functions `MPI_GET_ATTR` and `MPI_COMM_GET_ATTR` as 4-byte integers. The same holds for the predefined attributes that may be attached to the window and file objects.
- Do not use the `-i8` option to compile MPI callback functions, such as error handling functions, user-defined reduction operations, etc.
- You have to use a special ITC library if you desire to use the Intel Trace Collector with the Intel MPI ILP64 executable files. If necessary, the Intel MPI `mpiifort` compiler driver will select the correct ITC library automatically.
- Use the `mpif.h` file instead of the MPI module in Fortran90 applications. The Fortran module supports 32-bit `INTEGER` size only.
- There is currently no support for C and C++ applications.

7 Unified Memory Management

Intel® MPI Library provides a way to replace the memory management subsystem by a user defined package. The following function pointers may optionally be set by the user:

- `i_malloc`
- `i_calloc`
- `i_realloc`
- `i_free`

These pointers also affect the C++ new and delete operators.

The respective standard C library functions are used by default.

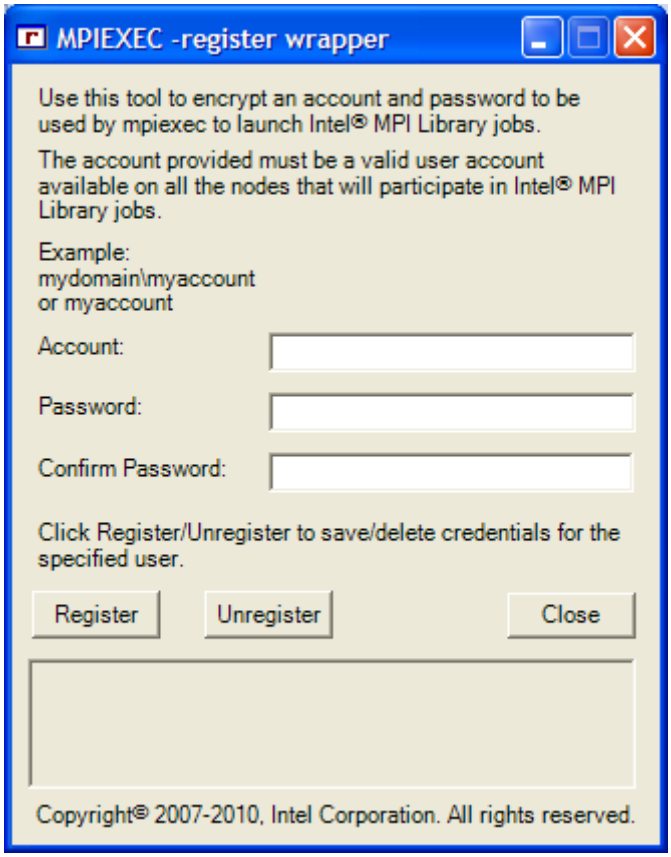
To use the unified memory management subsystem, link your application against the `libimalloc.dll`.

The following contrived source code snippet illustrates the usage of the unified memory subsystem:

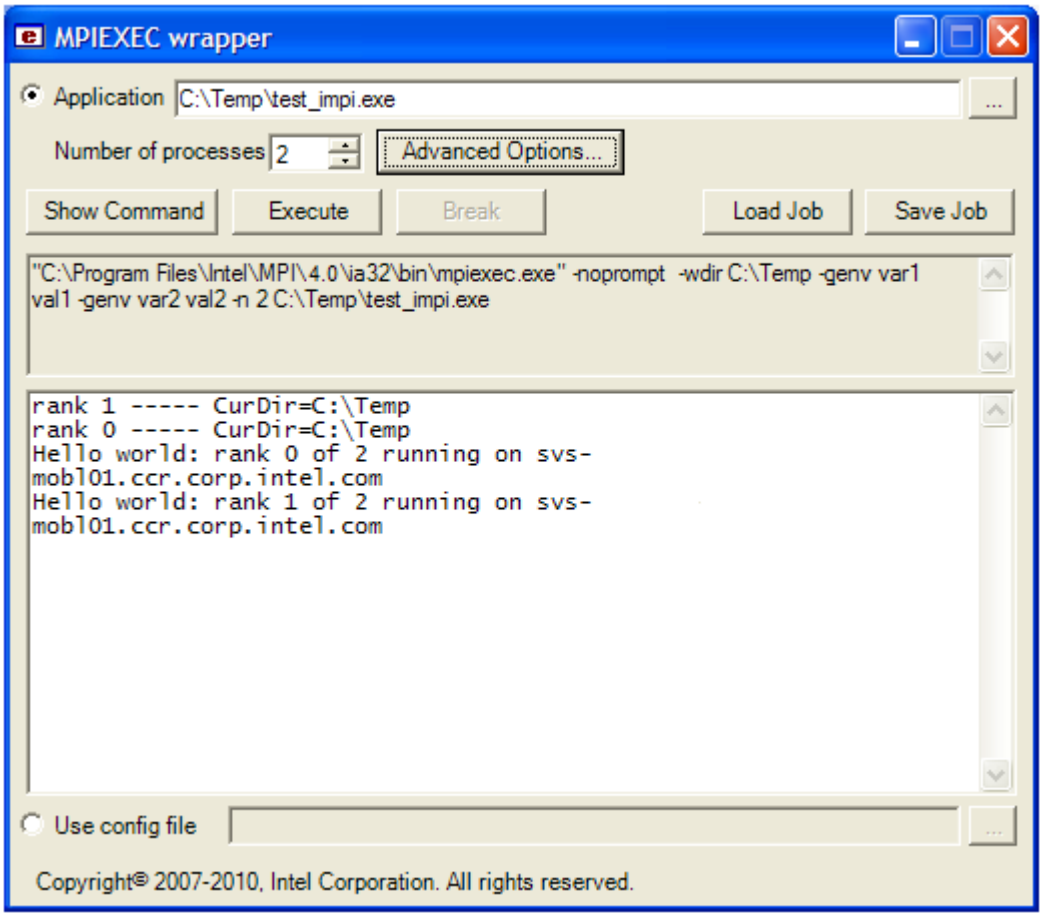
```
#include <i_malloc.h>
#include <my_malloc.h>

int main( int argc, int argv )
{
    // override normal pointers
    i_malloc = my_malloc;
    i_calloc = my_calloc;
    i_realloc = my_realloc;
    i_free = my_free;

#ifdef _WIN32
    // also override pointers used by DLLs
    i_malloc_dll = my_malloc;
    i_calloc_dll = my_calloc;
    i_realloc_dll = my_realloc;
    i_free_dll = my_free;
#endif

    // now start using Intel(R) libraries
}
```

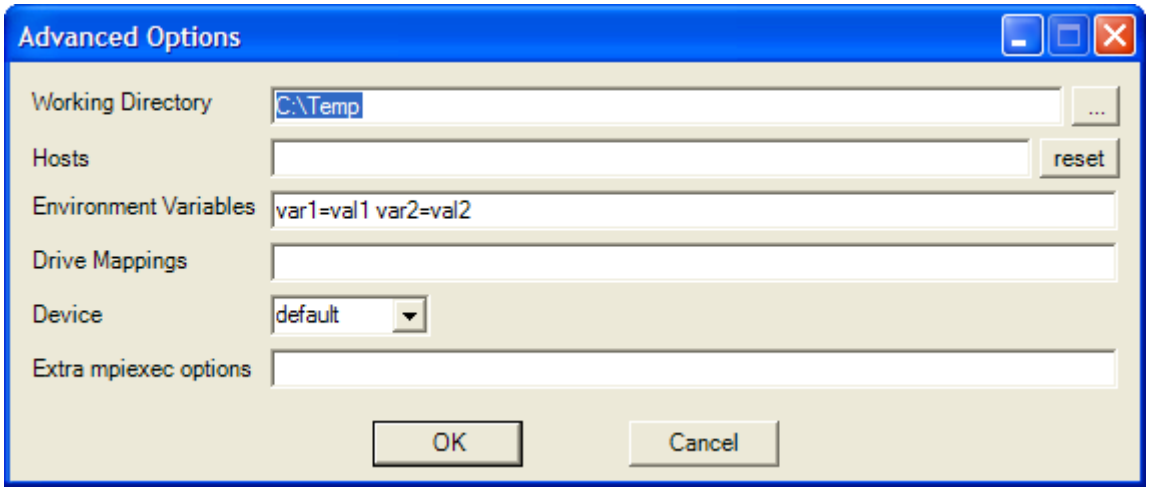
8 Graphical Utilities

The Intel® MPI Library provides three graphical utilities: `wmpiregister`, `wmpiexec`, and `wmpiconfig`. These utilities simplify using the Intel® MPI Library under Windows* OS.



Use the `wmpiregister` utility to encrypt and store your account name and a password. The specified account and the password are used for all subsequent MPI jobs you start. The first time you use this utility, you need to enter an account name and a password during the first `mpiexec` invocation.

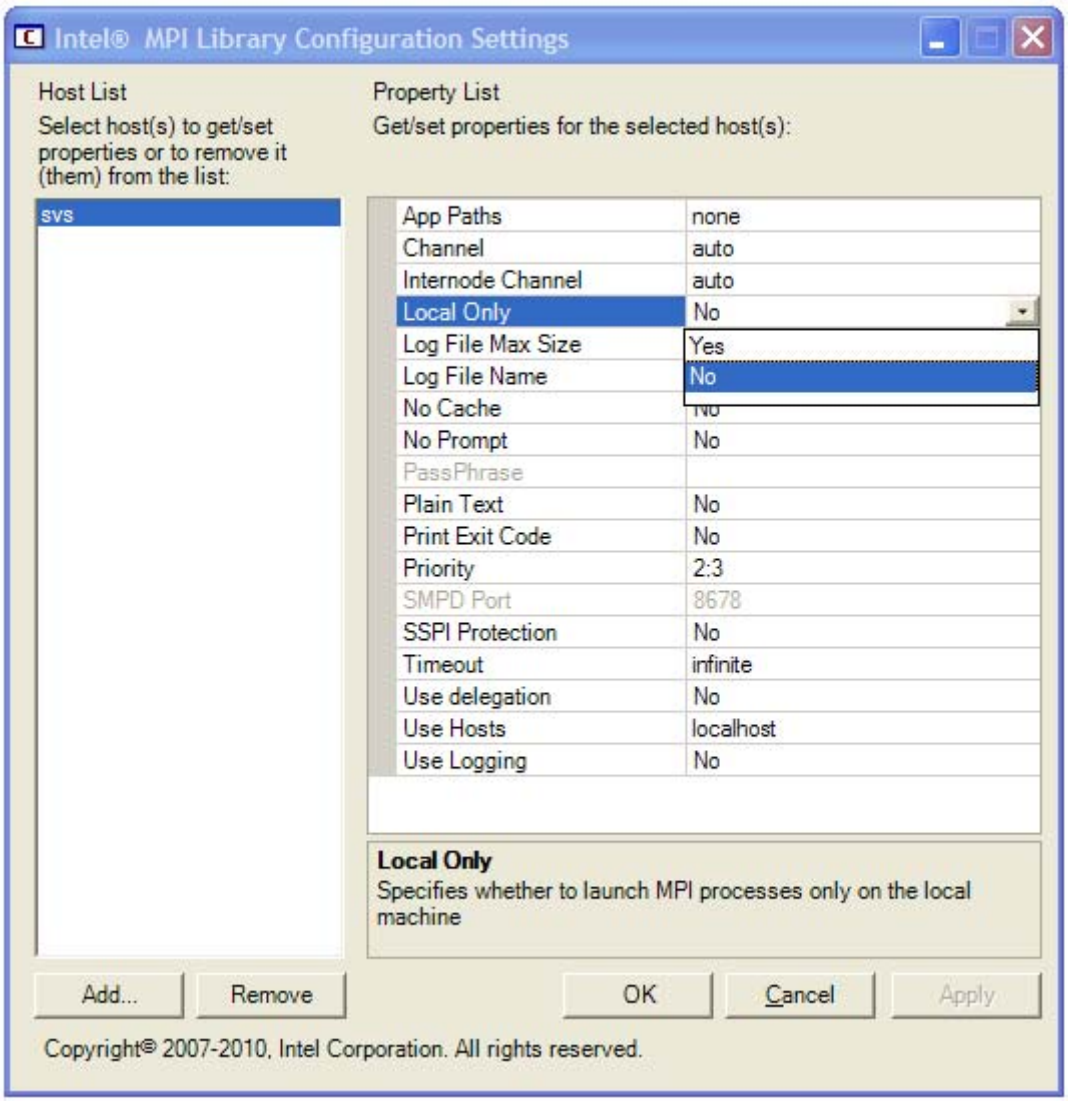




Use the `wmpiexec` utility as a graphical interface to the `mpiexec` command. This utility allows you to:

1. Describe the job by specifying the following:
    - an application to run
    - a number of instances
    - host names
    - a communication device to be used
    - a working directory for the MPI processes
    - environment variables to be set for the MPI processes
    - drive mappings to be used
    - extra MPI options for `wmpiexec`

2. Save the job description using the **Save Job** button (optional).
3. Load the job description using the **Load Job** button (optional).
4. View the actual `mpiexec` command line using the **Show Command** button.
5. Launch the job using the **Execute** button.
6. Break the job execution using the **Break** button.



Use the `wmpiconfig` utility to view/change the Intel® MPI Library settings for different hosts. This affects every job run on that host. The work with the `wmpiconfig` utility can be split into three steps:

1. Select the host(s) for which you want to change the Intel® MPI Library settings, and add them to the host list using the **Add** button.
2. Select host(s) in the host list to view the properties. If more than one host name is selected, intersection of the properties is displayed.
3. Change properties for the selected host(s) and press the **Apply** button in confirmation.

9 Glossary

hyper-threading technology	A feature within the IA-32 family of processors, where each processor core provides the functionality of more than one logical processor.
logical processor	The basic modularity of processor hardware resource that allows a software executive (OS) to dispatch task or execute a thread context. Each logical processor can execute only one thread context at a time.
multi-core processor	A physical processor that contains more than one processor core.
multi-processor platform	A computer system made of two or more physical packages.
processor core	The circuitry that provides dedicated functionalities to decode, execute instructions, and transfer data between certain sub-systems in a physical package. A processor core may contain one or more logical processors.
physical package	The physical package of a microprocessor capable of executing one or more threads of software at the same time. Each physical package plugs into a physical socket. Each physical package may contain one or more processor cores.
processor topology	Hierarchical relationships of “shared vs. dedicated” hardware resources within a computing platform using physical package capable of one or more forms of hardware multi-threading.

10 Index