MPICH2 Windows Development Guide*

Version 1.3.1

Mathematics and Computer Science Division
Argonne National Laboratory

Pavan Balaji
Darius Buntinas
Ralph Butler
Anthony Chan
David Goodell
William Gropp
Jayesh Krishna
Rob Latham
Ewing Lusk
Guillaume Mercier
Rob Ross
Rajeev Thakur

Past Contributors:
David Ashton
Brian Toonen

November 17, 2010

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, SciDAC Program, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

Contents

1 Introduction

This manual describes how to set up a Windows machine to build and test MPICH2 on.

2 Build machine

Build a Windows XP or Windows Server 2003 machine. This machine should have access to the internet to be able to download the MPICH2 source code.

3 Test machine

Build a Windows XP or Windows Server 2003 machine on a 32bit CPU. Also build a Windows Server 2003 X64 machine to test the Win64 distribution.

4 Software

This section describes the software necessary to build MPICH2.

4.1 Packages

To build MPICH2 you will need:

1. Microsoft Visual Studio 2005
2. The latest version of Microsoft .NET framework
3. Microsoft Platform SDK
4. Cygwin - full installation
5. Intel Fortran compiler IA32
6. Intel Fortran compiler EMT64
7. Java SDK

Microsoft Visual Studio 2005 can be found on the CDs from an MSDN subscription.

The Platform SDK can also be found on the MSDN CDs or downloaded from Microsoft.com. The latest version as of the writing of this document was Platform SDK - Windows Server 2003 SP1. The platform SDK usually has an up-to-date version of headers and libraries.

The Intel Fortran compilers need to be installed after Developer Studio and the PSDK because they integrate themselves into those two products. The regular IA32 compiler needs to be installed and the EMT64 compiler needs to be installed. They are two separate packages and they require a license file to use. The license file is for a single user on a single machine.

Cygwin needs to be installed to get svn, perl and ssh. By default the Cygwin installer might not install all the required packages, so make sure that the required packages are selected during the install. MPICH2 also requires autoconf version 2.62 or above. The OpenPA library used by MPICH2 requires the automake package. Select to use the DOS file format when installing Cygwin.

Assuming you installed Cygwin to the default `c:\cygwin` directory, add `c:\cygwin\bin` to your PATH environment variable. This is required so the automated scripts can run tools like ssh and perl without specifying the full path.

The Java SDK needs to be installed so the logging library can be compiled. After installing the SDK set the JAVA_HOME environment variable to point to the installation directory.

Run the following command from a command prompt to change the Windows script engine from GUI mode to console mode:

```
cscript //H:cscript
```

5 Building MPICH2

This section describes how to make various packages once you have a working build machine.

5.1 Visual Studio automated 32bit build

The easist way to build an MPICH2 distribution is to use the Visual Studio environment and the makewindist.bat script from the top level of the mpich2 source tree. You can check out mpich2 from SVN or you can simply copy this batch file from the distribution. The batch file knows how to check out mpich2 so it the only file required to make a distribution.

The product GUIDs need to be changed when a new release is created. To do this run "`perl update_windows_version <new_version>`". Run this script with mpich2/maint as the current directory so the project files can be found. Example:

```
perl update_windows_version 1.0.8
```

Or you can modify the project files by hand. Edit mpich2/maint/mpich2i.vdproj. The ProductCode and PackageCode entries need to be changed to use new GUIDs. Under Unix or Windows, uuidgen can be used to generate a new GUID. The ProductVersion entry needs to be changed to match the version of MPICH2. Once the version and GUIDs have been updated, commit the changes to mpich2i.vdproj to SVN. Now you can build a distribution.

Bring up a build command prompt by selecting Start→Progams→Microsoft Visual Studio 2005→Visual Studio 2005 Tools→Visual Studio 2005 Command Prompt.

Change directories to wherever you want to create the distribution. mpich2 will be checked out under the current directory. Run the makewindist batch file:

```
makewindist.bat --with-checkout
```

The batch file executes the following steps:

1. Check out trunk from the MPICH2 svn repository.
2. Run `maint/updatefiles` to generate the autogenerated files
3. Run "`winconfigure.wsf --cleancode`" to configure mpich2 for Windows and output all the generated files like mpi.h and the fortran interface files, etc.

4. Run the Visual Studio command line tool to build all the components of MPICH2. This includes each of the channels - sock, nemesis, ssm, shm, and the multi-threaded sock channel. Two versions of each channel are built, the regular release build and the rlog profiled version. The mpi wrapper channel selector dll is built and three Fortran interfaces are built, one for each set of common symbol types and calling conventions. mpiexec and smpd are built along with the Windows GUI tools and the Cygwin libraries. (These are the Cygwin link libraries to use the Windows native build of MPICH2, not a Unix-style build of MPICH2 under Cygwin.)

5. Package up everthing into `maint\ReleaseMSI\mpich2.msi`.

When the batch file is finished you will be left with a mpich2.msi file that can be used to install MPICH2 on any Win32 machine. This file can be re-named to match the release naming conventions.

5.1.1 Automated build from the source distribution

Follow the steps mentioned below to build MPICH2 from a source tarball.

1. unzip/untar the source distribution
2. Open a Visual Studio Command Prompt
3. cd into the mpich2xxx directory
4. execute "`winconfigure.wsf --cleancode`"
5. execute "`makewindist.bat --with-curdir`"

5.1.2 Building without Fortran

If you don't have a Fortran compiler you can use winconfigure.wsf to remove the Fortran projects. Execute `winconfigure.wsf --remove-fortran --cleancode` Then you can build the projects without Fortran support. If you want to use the `makewindist.bat` script you will need to remove the Fortran lines from it before executing it.

5.2 Platform SDK builds

The makefile in the `mpich2\winbuild` directory builds a distribution based on the compilers specified in the environment. The following targets can all be built with this mechanism:

- Win64 X64
- Win64 IA64
- Win32 x86

Follow the steps below to build MPICH2.

1. Open a Cygwin bash shell and check out mpich2:

   `svn checkout https://svn.mcs.anl.gov/repos/mpi/mpich2/trunk mpich2`.
2. cd into mpich2 directory
3. run `maint/updatefiles`
4. Open a Visual Studio command prompt
5. From within the Visual Studio command prompt run `winconfigure.wsf --cleancode`

To build the Win64 X64 distribution do the following:

1. Bring up a build command prompt from the PSDK. It can be found here: Start→Programs →Microsoft Platform SDK for Windows Server 2003 SP1→Open Build Environment Window→ Windows Server 2003 64-bit Build Environment→Set Win Svr 2003 x64 Build Env (Retail)
2. Run `\Program Files\Intel\Fortran\compiler80\Ia32e\Bin\ifortvars.bat`
3. cd into `mpich2\winbuild`
4. run `build.bat 2>&1 | tee build.x64.out`

For building the installer for Win64 x64, open the mpich2 solution file, mpich2.sln, using Visual Studio 2005 and build the Installerx64 solution. The installer, `mpich2.msi` will be available at `mpich2\maint\ReleaseMSIx64` directory.

The Visual Studio 2005 compiler provides a Cross tools command prompt for building X64 applications. However the current makefile depends on environment variables not available with the Cross tools command prompt.

To build the Win64 IA64 distribution do the following:

1. Bring up a build command prompt from the PSDK. It can be found here: Start→Programs →Microsoft Platform SDK for Windows Server 2003 SP1→Open Build Environment Window→ Windows Server 2003 64-bit Build Environment→Set Win Svr 2003 IA64 Build Env (Retail)
2. Run `\Program Files\Intel\Fortran\compiler80\Itanium\Bin\ifortvars.bat`
3. cd into `mpich2\winbuild`
4. run `build.bat 2>&1 | tee build.ia64.out`

To build the Win32 x86 distribution do the following:

1. Bring up a build command prompt from the PSDK. It can be found here: Start→Programs →Microsoft Platform SDK for Windows Server 2003 SP1→Open Build Environment Window→ Windows 2000 Build Environment→Set Windows 2000 Build Environment (Retail)
2. Run `\Program Files\Intel\Fortran\compiler80\Ia32\Bin\ifortvars.bat`
3. cd into `mpich2\winbuild`
4. run `build.bat 2>&1 | tee build.x86.out`

6 Distributing MPICH2 builds

If you built an .msi file using the Visual Studio build process 5.1 then all you have to do is rename the `mpich2.msi` file to something appropriate like `mpich2-1.0.3-1-win32-ia32.msi`

If you built using the Platform SDK build process 5.2 then the output files are left in their build locations and need to be collected and put in a zip file for distributing. This process should be automated with a script.

7 Testing MPICH2

Run the `testmpich2.wsf` script to checkout mpich2, build it, install it, checkout the test suites, build them, run the test suites, and collect the results in a web page.

7.1 Testing from scratch

Explain the use of testmpich2.wsf.

Run "`testmpich2.wsf`" without any parameters and it will create a `testmpich2` subdirectory and check out into that directory mpich2 and the test suites - c++, mpich, intel and mpich2. It will then build mpich2 and all the tests from the test suites. Then it will run the tests and place a summary in `testmpich2\summary\index.html`.

7.2 Testing a built mpich2 directory

Explain how to run `testmpich2.wsf` if you have the mpich2 source tree on a machine and you have already built all of mpich2.

Here is a sample batch file to test mpich2 that has already been built in c:\mpich2:

```
testmpich2.wsf /mpich2:c:\mpich2 /make- /configure- /buildbatch
pushd testmpich2\buildMPICH
call mpich_cmds.bat
popd
pushd testmpich2\buildCPP
call cpp_cmds.bat
popd
pushd testmpich2\buildINTEL
call intel_cmds.bat
popd
```

```
pushd testmpich2\buildMPICH2
call mpich2_cmds.bat
popd
testmpich2.wsf /mpich2:c:\mpich2 /make- /configure- /summarize
```

7.3 Testing an existing installation

Explain the use of testmpich2.wsf to test an existing installation, one that was installed with the .msi distribution.

8 Development issues

This section describes development issues that are particular to the Windows build.

Whenever a .h.in file is created on the Unix side, winconfigure.wsf needs to be updated to create the .h file from the .h.in file. Copy and paste an existing section in winconfigure.wsf that already does this and rename the file names.

When new definitions are added to the .h.in files these definitions, usually in the form HAVE_FOO or USE_FOO, need to be added to the AddDefinitions function in winconfigure.wsf. Simply add new cases to the big case statement as needed. winconfigure.wsf warns you of definitions that are not in the case statement.

Whenever a @FOO@ substitution is added on the Unix side, winconfigure.wsf needs to be updated to handle the substitution. Find the ReplaceAts function in winconfigure.wsf and add the substitution to the big case statement. winconfigure.wsf warns you of new substitutions that have not been added to the case statement.

9 Runtime environment

This section describes the MPICH2 environment that is particular to Windows.

9.1 User credentials

mpiexec must have the user name and password to launch MPI applications in the context of that user. This information can be stored in a secure encrypted manner for each user on a machine. Run `mpiexec -register` to save your username and password. Then mpiexec will not prompt you for this information.

This is also true for a nightly build script. The user context under which the script is run must have saved credentials so mpiexec doesn't prompt for them. So scripts won't hang, mpiexec provides a flag, `-noprompt`, that will cause mpiexec to print out errors in cases when it normally would prompt for user input. This can also be specified in the environment with the variable MPIEXEC_NOPROMPT.

You can also save more than one set of user credentials. Add the option `-user n` to the `-register`, `-remove`, `-validate`, and `mpiexec` commands to specify a saved user credential other than the default. The parameter `n` is a non-zero positive number. For example this will save credentials in slot 1:

```
mpiexec -register -user 1
```

And this command will use the user 3 to launch a job:

```
mpiexec -user 3 -n 4 cpi.exe
```

User credentials can also be specified in a file using the `-pwdfile filename` option to mpiexec. Put the username on the first line of the file and the password on the second line. If you choose this option you should make sure the file is only readable by the current user.

9.2 MPICH2 channel selection

MPICH2 for Windows comes with multiple complete implementations of MPI. These are called channels and each build represents a different transport mechanism used to move MPI messages. The default channel (sock) uses sockets for communication. There is channel that use only shared memory (shm). There are two channels that uses both sockets and shared

memory (nemesis, ssm). And there is a thread-safe version of the sockets channel (mt). We recommend users to use the sock, mt or nemesis channels. The shm and ssm channels will soon be deprecated.

The short names for the channels are: sock, nemesis, shm, ssm, mt.

These channels can be selected at runtime with an environment variable: MPICH2_CHANNEL. The following is an example that uses the nemesis channel instead of the default sockets channel:

```
mpiexec -env MPICH2_CHANNEL nemesis -n 4 myapp.exe
or
mpiexec -channel nemesis -n 4 myapp.exe
```

If you specify `auto` for the channel then mpiexec will automatically choose a channel for you.

```
mpiexec -channel auto -n 4 myapp.exe
```

The rules are:

1. If numprocs is less than 8 on one machine, use the shm channel
2. If running on multiple machines, use the ssm channel. This channel can be changed using winconfigure.

9.3 MPI apps with GUI

Many users on Windows machines want to build GUI apps that are also MPI applications. This is completely acceptable as long as the application follows the rules of MPI. MPI_Init must be called before any other MPI function and it needs to be called soon after each process starts. The processes must be started with mpiexec but they are not required to be console applications.

The one catch is that MPI applications are hidden from view so any Windows that a user application brings up will not be able to be seen. mpiexec has an option to allow the MPI processes on the local machine to be able to bring up GUIs. Add -localroot to the mpiexec command to enable this capability. But even with this option, all GUIs from processes on remote machines will be hidden.

So the only GUI application that MPICH2 cannot handle by default would be a video-wall type application. But this can be done by running smpd.exe by hand on each machine instead of installing it as a service. Log on to each machine and run "`smpd.exe -stop`" to stop the service and then run "`smpd.exe -d 0`" to start up the smpd again. As long as this process is running you will be able to run applications where every process is allowed to bring up GUIs.

9.4 Security

MPICH2 can use Microsoft's SSPI interface to launch processes without using any user passwords. This is the most secure way to launch MPI jobs but it requires the machines to be configured in a certain way.

- All machines must be part of a Windows domain.
- Each machine must have delegation enabled.
- Each user that will run jobs must be allowed to use delegation.

If the machines are set up this way then an administrator can set up MPICH2 for passwordless authentication. On each node, a domain administrator needs to execute the following: "`smpd -register_spn`".

Then a user can add the `-delegate` flag to their mpiexec commands and the job startup will be done without any passwords. Example:

```
mpiexec -delegate -n 3 cpi.exe
```

With SSPI enabled you can also control access to nodes with job objects.

First the nodes need to be set up so that only SSPI authentication is allowed. An administrator can run the following on each node:

1. `smpd.exe -set sspi_protect yes`
2. `smpd.exe -set jobs_only yes`
3. `smpd.exe -restart`

These settings mean that authentication must be done through SSPI and mpiexec commands will only be accepted for registered jobs.

To register jobs an administrator or a scheduler running with administrator privileges can execute the following command:

```
mpiexec.exe -add_job <name> <domain\username> [-host <hostname>]
```

This adds a job called "name" for the specified user on either the local or specified host. Any name can be used but it must not collide with another job with the same name on the same host. The command must be executed for each host that is to be allocated to the user.

Then when the job has finished or the allotted time has expired for the user to use the nodes the following command can be executed:

```
mpiexec.exe -remove_job <name> [-host <hostname>]
```

This command removes the job from the local or specified host. Any processes running on the host under the specified job name will be terminated by this command.

So `-add_job` and `-remove_job` can be used by a scheduler to create a window when a user is allowed to start jobs on a set of nodes.

When the window is open the user can run jobs using the job name. First the user must run:

```
mpiexec.exe -associate_job <name> [-host <hostname>]
```

This will associate the user's token with the job object on the local or specified host. This must be done for all of the hosts allocated to the user. Then the user can issue mpiexec commands. The mpiexec commands are of the usual format except they must contain one extra option - "`-job <name>`". This job name must match the job allocated by the `-add_job` command. So a typical command would look like this:

```
mpiexec.exe -job foo -machinefile hosts.txt -n 4 myapp.exe
```

Multiple mpiexec commands can be issued until the `-remove_job` command is issued. This allows the users to issue multiple mpiexec commands and

multiple MPI_Comm_spawn commands all using the same job name until the job is removed from the nodes.

The rationale for the design where an adminstrator can create and destroy jobs but the user must first associate the job with his own token before running jobs is so that the administrator does not need to know the user's password. In order for an administrator to do both the job allocation and association he would have to call LogonUser with the user name and password for each user that submits a job request.

9.5 Firewalls

Windows comes with a default firewall that is usually turned on by default. Firewalls block all TCP ports by default which renders MPICH2 applications inoperable because the default communication mechanism used by MPICH2 are sockets on arbitrary ports assigned by the operating system. This can be solved in several ways:

- Turn off the firewall completely.
- MPICH2 applications can be limited to a range of TCP ports using the MPICH_PORT_RANGE environment variable. If you set your firewall to allow the same port range then MPICH2 applications will run.
- Leave the Windows firewall on and allow exceptions for your MPICH2 applications. This can be done through the Security Center module of the Windows Control Panel. Click the Windows Firewall option in the Security Center to bring up the properties page and select the Exceptions tab. Here you can add each MPICH2 application to exempt. Note that this exception includes the path to the executable so if you move the executable you will have to exempt the new location. This solution obviously only will work for a small number of applications since managing a large list would be difficult. Make sure you add mpiexec.exe and the smpd.exe process manager to this exception list.

9.6 MPIEXEC options

This section describes all the options to mpiexec.exe

- `-add_job job_name domain\user [-host hostname]` Create a job object on the local or specified host for the specified user. Administrator privileges are required to execute this command.

- `-associate_job job_name [-host hostname]` Associate the current user token with the specified job on the local or specified host. The current user must match the user specifed by the `-add_job job_name username` command.

- `-binding process_binding_scheme` This option is currently available only under Windows. It allows the user to specify a process binding scheme for the MPI processes. Currently `auto` and `user` are the supported binding schemes. Using `auto` as the process binding scheme the process manager will choose the process binding scheme automatically taking into account the load on system resources like caches. The `user` binding scheme can be used to provide a user defined binding for the MPI processes. The supported formats for specifying the binding schemes are provided below.

  ```
  -binding auto
  ```

  ```
  -binding user:core1,core2
  ```

  where `core1` and `core2` represent the logical processor ids. The logical processor ids specified are used in a round robin fashion to bind the MPI processes to the logical processors. Some examples are provided below.

  ```
  mpiexec -n 4 -binding auto cpi.exe
  ```

  In the above example the process manager binds the MPI processes to the available cores automatically as mentioned above.

  ```
  mpiexec -n 3 -binding user:1,3 cpi.exe
  ```

  In the example above the process manager binds MPI process with rank 0 to logical processor 1, rank 1 to logical processor 3, rank 2 to logical processor 1.

- `-channel channel_name` This option is only available under Windows and allows the user to select which channel implementation of MPICH2 to select at runtime. The current channels supported are sock, mt, ssm, and shm. These represent the sockets, multi-threaded sockets, sockets plus shared memory, and shared memory channels. The shared memory channels only work on one node. The sockets, multi-threaded sockets, and sockets plus shared memory channels work on multiple

nodes. There are also profiled versions of the channels that produce RLOG files for each process when selected. They are named p, mtp, ssmp, and shmp. See the section on channel selection for additional information.

- `-configfile filename` Use the specified job configuration file to launch the job. Each line in the file represents a set of options just like you would enter them on the `mpiexec` command line. The one difference is that there are no colons in the file. The colons are replaced by new-lines.

- `-delegate` Specify that you want to use passwordless SSPI delegation to launch processes. The machines must be configured to use SSPI as described in the section on security.

- `-dir drive:\my\working\directory` Specify the working directory for the processes.

- `-env variable value` Specify an environment variable and its value to set in the processes' environments. This option can be specified multiple times.

- `-exitcodes` Specify that the exit code of each process should be printed to stdout as each processes exits.

- `-file filename` Use the specified implementation specific job configuration file. For Windows this option is used to specify the old MPICH 1.2.5 configuration file format. This is useful for users who have existing configuration files and want to upgrade to MPICH2.

- `-genvlist a,b,c,d...` Specify a list of environment variables to taken from the environment local to mpiexec and propagated to the launched processes.

- `-hide_console` Detach from the console so that no command prompt window will appear and consequently not output will be seen.

- `-host hostname` Specify that the processes should be launched on a specific host.

- `-hosts n host1 host2 host3 ...` Specify that the processes should be launched on a list of hosts. This option replaces the `-n x` option.

- `-hosts n host1 m1 host2 m2 host3 m3 ...` Specify that the processes should be launched on a list of hosts and how many processes should be launched on each host. The total number of processes launched is m1 + m2 + m3 + ... mn.
- `-impersonate` Specify that you want to use passwordless SSPI impersonation to launch processes. This will create processes on the remote machines with limited access tokens. They wil not be able to open files on remote machines or access mapped network drives.
- `-job job_name` Specify that the processes should be launched under the specifed job object. This can only be used after successful calls to `-add_job` and `-associate_job`.
- `-l` This flag causes mpiexec to prefix output to stdout and stderr with the rank of the process that produced the output. (This option is the lower-case L not the number one)
- `-localonly x` or `-localonly` Specify that the processes should only be launched on the local host. This option can replace the `-n x` option or be used in conjunction with it when it is only a flag.
- `-localroot` Specify that the root process should be launched on the local machine directly from mpiexec bypassing the smpd process manager. This is useful for applications that want to create windows from the root process that are visible to the interactive user. The smpd process manager creates processes in a hidden service desktop where you cannot interact with any GUI.
- `-log` This option is a short cut to selecting the MPE wrapper library to log the MPI application. When the job finishes there will be a .clog2 file created that can be viewed in Jumpshot.
- `-logon` Prompt for user credentials to launch the job under.
- `-machinefile filename` Use the specified file to get host names to launch processes on. Hosts are selected from this file in a round robin fashion. One host is specified per line. Extra options can be specified. The number of desired processes to launch on a specific host can be specified with a colon followed by a number after the host name: `hostname:n`. This is usefull for multi-CPU hosts. If you want to specify the interface that should be used for MPI communication to the

host you can add the `-ifhn` flag. A sample machinefile is provided below for reference.

```
# Comment line
# Run two procs on hostname1
hostname1:2
# Run four procs on hostname2 but use 192.168.1.100
# as the interface
hostname2:4 -ifhn 192.168.1.100
```

The interface can also be specified using the `ifhn=` option. The following line is valid in a machinefile.

```
#Using ifhn= option to specify the interface
hostname1:2 ifhn=192.168.1.101
```

- `-map drive:\\host\share` Specify a network mapped drive to create on the hosts before launching the processes. The mapping will be removed when the processes exit. This option can be specified multiple times.
- `-mapall` Specify that all network mapped drives created by the user executing mpiexec command will be created on hosts before launching the processes. The mappings will be removed when the processes exit.
- `-n x` or `-np x` Specify the number of processes to launch.
- `-nopm` This flag is used in conjunction with the `-rsh` flag. With this flag specified there need not be any smpd process manager running on any of the nodes used in the job. `mpiexec` provides the PMI interface and the remote shell command is used to start the processes. Using these flags allows jobs to be started without any process managers running but the MPI-2 dynamic process functions like MPI_Comm_spawn are consequently not available.
- `-noprompt` Prevent mpiexec for prompting for information. If user credentials are needed to launch the processes mpiexec usually prompts for this information but this flag causes an error to be printed out instead.
- `-p port` Short version of the `-port` option.

- `-path search_path` Specify the search path used to locate executables. Separate multiple paths with semicolons. The path can be mixed when using both Windows and Linux machines. For example: `-path c:\temp;/home/user` is a valid search path.

- `-phrase passphrase` Specify the passphrase used to authenticate with the smpd process managers.

- `-plaintext` Specify that user credentials should go over the wire unencrypted. This is required if both Linux and Windows machines are used in the same job because the Linux machines cannot encrypt and decrypt the data created by the Windows machines.

- `-pmi_server num_processes or -pmiserver num_processes` This option specified by itself connects to the local smpd process manager and starts a PMI service. This service is used by MPICH2 processes to communicate connection information to each other. This option is only good for a single MPICH2 job. The input parameter is the number of processes in the job. `mpiexec` immediately outputs three lines of data. The first line is the host name. The second line is the port it is listening on and the third line is the name of the PMI KVS. A process manager that can set environment variables and launch processes but does not implement the PMI service can use this option to start jobs. Along with the other PMI environment variables the process manager must set PMI_HOST to the host name provided, PMI_PORT to the port provided and PMI_KVS and PMI_DOMAIN to the KVS name provided. It is the responsibility of the process manager to set the other environment variables correctly like PMI_RANK and PMI_SIZE. See the document on the smpd PMI implementation for a complete list of the environment variables. When the job is finished the PMI server will exit. This option can be executed in separate command simultaneously so that multiple jobs can be executed at the same time.

- `-port port` Specify the port where the smpd process manager is listening.

- `-priority class[:level]` Specify the priority class and optionally the thread priority of the processes to be launched. The class can be 0,1,2,3, or 4 corresponding to idle, below, normal, above, and high. The level can be 0,1,2,3,4, or 5 corresponding to idle, lowest, below, normal, above, highest. The default is 2:3.

- `-pwdfile filename` Specify a file to read the user name and password from. The user name should be on the first line and the password on the second line.

- `-quiet_abort` Use this flag to prevent extensive abort messages to appear. Instead the job will simply exit with minimal error output.

- `-register [-user n]` Encrypt a user name and password into the Windows registry so that it can be automatically retrieved by mpiexec to launch processes with. If you specify a user index then you can save more than one set of credentials. The index should be a positive non-zero number and does not need to be consecutive.

- `-remove [-user n]` Remove the encrypted credential data from the Registry. If multiple entries are saved then use the `-user` option to specify which entry to remove. `-user all` can be specified to delete all entries.

- `-remove_job job_name [-host hostname]` Remove a job object on the local or specified host. Any processes running under this job will be terminated. Administrator privileges are required to execute this command.

- `-rsh or -ssh` Use the remote shell command to execute the processes in the job instead of using the smpd process manager. The default command is "`ssh -x`" no matter whether `-rsh` or `-ssh` is used. If this is the only flag specified then an smpd process manager must be running on the local host where `mpiexec` is executed. `mpiexec` contacts the local smpd process to start a PMI service required by the MPI job and then starts the processes using the remote shell command. On the target machines the application "`env`" must be available since it is used to set the appropriate environment variables and then start the application. The remote shell command can be changed using the environment variable MPIEXEC_RSH. Any command can be used that takes a host name and then everything after that as the user command to be launched. Note that you need to specify a fully qualified file name of the executable when running your job with the `-rsh` option. If you like to use relative paths set the working directory for the job using the `-wdir` option of mpiexec.

- `-smpdfile filename` Specify the location of the smpd configuration

file. The default is `~/.smpd`. This is a Unix only option. Under Windows the settings are stored in the Windows Registry.

- `-timeout seconds` Specify the maximum number of seconds the job is allowed to run. At the end of the timeout period, if the job has not already exited then all processes will be killed.
- `-user n` Specify which encrypted credentials to retrieve from the Registry. The corresponding entry must have been previously saved using the `-register -user n` option.
- `-validate [-user n] [-host hostname]` Validate that the saved credentials can be used to launch a process on the local or specified host. If more that one credentials has been saved then the `-user` option can be used to select which user credentials to use.
- `-verbose` Output trace data for mpiexec. Only useful for debugging.
- `-wdir drive:\my\working\directory` `-wdir` and `-dir` are synonyms.
- `-whoami` Print out the current user name in the format that mpiexec and smpd expect it to be. This is useful for users who use a screen name that is different from their user name.

9.7 SMPD process manager options

This section describes some of the options for the smpd process manager.

smpd.exe runs as a service under Windows. This is required so that it can start processes under multiple user credentials. Only services have the privileges necessary to log on users and start processes for them. Since this is a privileged operation administrator rights are required to install the smpd service. This is what the default installer package does.

But smpd can be run in other ways for debugging or single user use.

If you have smpd.exe installed first execute `smpd.exe -stop` to stop the service.

Then you can run it by hand for single user mode or for debugging. The flag for debugging single user mode is `-d debug_output_level`.

If you run it like this you will get full trace output:

```
smpd.exe -d
```

If you run it like this you will get no output except for errors:

```
smpd.exe -d 0
```

Here are all the options to smpd.exe:

- `-install or -regserver` Install the smpd service. Requires administrator privileges.
- `-remove or -uninstall or -unregserver` Uninstall the smpd service. Requires administrator privileges.
- `-start` Start the smpd service. Requires administrator privileges.
- `-stop` Stop the smpd service. Requires administrator privileges.
- `-restart` Stop and restart the smpd service. Requires administrator privileges.
- `-register_spn` Register the Service Prinicipal Name for the smpd service of the local machine on the domain controller. Requires DOMAIN administrator privileges. This is used in conjunction with passwordless SSPI authentication described in the section on security.
- `-remove_spn` Remove the Service Prinicipal Name from the domain controller for the smpd service of the local machine. Requires DOMAIN administrator privileges.
- `-traceon filename [hostA hostB ...]` Turn on the trace logging of the smpd service on the local or specified hosts and set the output to the specified file. The file location must be available on the local drive of each of the hosts. It cannot be located on a remote machine.
- `-traceoff [hostA hostB ...]` Turn off the trace logging of the smpd service on the local or specified hosts.
- `-port n` Listen on the specified port number. If this option is not specified then smpd listens on the default port (8676).
- `-anyport` Listen on any port assigned by the OS. smpd immediately prints out the port that it has been assigned.

- `-phrase passphrase` Use the specified passphrase to authenticate connections to the smpd either by mpiexec or another smpd process.
- `-getphrase` Prompt the user to input the passphrase. This is useful if you don't want to specify the phrase on the command line.
- `-noprompt` Don't prompt the user for input. If there is missing information, print an error and exit.
- `-set option value` Set the smpd option to the specified value. For example, `smpd -set logfile c:\temp\smpd.log` will set the log file to the specified file name. `smpd -set log yes` will turn trace logging on and `smpd -set log no` will turn it off.
- `-get option` Print out the value of the specified smpd option.
- `-hosts` Print the hosts that mpiexec and this smpd will use to launch processes on. If the list is empty then processes will be launched on the local host only.
- `-sethosts hostA hostB ...` Set the hosts option to a list of hosts that mpiexec and smpd will use to launch processes on.
- `-d [level] or -debug [level]` Start the smpd in debug or single user mode with the optionally specified amount of output. For example, `smpd -d` will start the smpd will lots of trace output and `smpd -d 0` will start the smpd with no output except for errors.
- `-s` Only available on Unix systems. This option starts the smpd in single user daemon mode for the current user.
- `-smpdfile filename` On Unix systems the smpd options are stored in a file that is readable only by the current user (chmod 600). This file stores the same information that would be stored in the Windows registry like the port and passphrase. The default file is named `~/.smpd` if this option is not specified.
- `-shutdown` Shutdown a running smpd that was started by `smpd -s` or `smpd -d`.
- `-printprocs` On a Windows machine you can run `smpd -printprocs` and it will print out the processes started and stopped by smpd on the current host. The format of the output is `+/-pid cmd`. Plus means a process was started and minus means the process has exited. The

process id is specified next and then the rest of the line is the command that was launched.

- `-enum or -enumerate` Print the smpd options set on the local host.
- `-version` Print the smpd version and exit.
- `-status [-host hostname]` Print the status of the smpd on the local or specified host.
- `-help` Print a brief summary of the options to smpd.

9.8 Debugging jobs by starting them manually

This section describes how to start a job by hand without the use of a process manager so the job can be stepped through with a debugger.

You can launch an MPICH2 job by hand if you set the minimum required environment variables for each process and then start the processes yourself (or in a debugger).

Here is a script that sets the environment variables so that a job can be started on the local machine: The file is called `setmpi2.bat`

```
if ‘‘%1’’ == ‘‘’’ goto HELP
if ‘‘%2’’ == ‘‘’’ goto HELP
set PMI_ROOT_HOST=%COMPUTERNAME%
set PMI_ROOT_PORT=9222
set PMI_ROOT_LOCAL=1
set PMI_RANK=%1
set PMI_SIZE=%2
set PMI_KVS=mpich2
goto DONE
:HELP
REM usage: setmpi2 rank size
:DONE
```

For example, to debug a two process job bring up two separate command prompts. In the first prompt execute `setmpi2.bat 0 2` and in the second prompt execute `setmpi2.bat 1 2`. Then run your application always starting the root process first. The root process must call MPI_Init before any

of the other processes because it is the process that listens on the port specifed by the environment variable PMI_ROOT_PORT. Simply execute `myapp.exe` from each command prompt to run your job. Or better yet run each process in a debugger. If you have the Microsoft developer studio installed you can run the following from each command prompt: `devenv.exe myapp.exe`. This will bring up a debugger for each process. Then you can step through each process and debug it. Remember that the first process must call MPI_Init before any of the rest of the processes do. You can restart the processes at any time as long as you restart all of them.

The script can be modified to launch on multiple hosts by changing the line:

```
set PMI_ROOT_HOST=%COMPUTERNAME%
```

to set the variable to the hostname where the root process will be started instead of the local host name.

The limitation of this method of starting processes is that MPI-2 spawning operations are not supported. If your application calls MPI_Comm_spawn it will produce an error.

9.9 Debugging jobs using MPI Cluster Debugger

This section describes how to debug MPI jobs using the MPI Cluster Debugger available with Visual Studio suites. Follow the steps below to debug your MPI application (myapp.exe) locally on your machine using Visual Studio 2010

1. Select Visual Studio application project properties (Select "`myapp.exe properties`" item from the "`Project`" menu)
2. In Debugging section (Select "`Configuration Properties`" and then select "`Debugging`" category), select "`MPI Cluster debugger`" as the debugger to launch from the drop down menu.
3. Set the "`Run Environment`" item to "`localhost/NUM_PROCS_TO_LAUNCH`" (e.g. localhost/3 to launch 3 MPI processes)
4. Set the "`Application Arguments`" item, if your MPI application has any arguments (To specify input to your application redirect it from a text file)

5. Set the "`MPIExec Command`" item to point to MPICH2's mpiexec (`C:\Program Files\MPICH2\bin\mpiexec.exe`)
6. Click OK to submit the project property changes
7. Insert break points in the Visual Studio explorer window
8. Press F5 to start debugging your code. Note that multiple windows pop up showing a view of the processes launched and the corresponding call stacks. The debugger automatically switches between MPI processes, if required, when it hits a breakpoint.

9.10 Environment variables

This section describes the environment variables used by MPICH2 and smpd.

- `MPICH_ABORT_ON_ERROR` Call abort() when an error happens instead of returning an error and calling MPID_Abort. useful for unix where calling abort() creates a core file.
- `MPICH_PRINT_ERROR_STACK` Print the entire error stack when an error occurs (currently this is the default)
- `MPICH_CHOP_ERROR_STACK` Split the error stack output at the character position specified. A value of 79 would cause carriage returns to be inserted after the 79th character.
- `MPICH_WARNINGS` Print runtime warnings (unmatched messages at MPI_Finalize, unreleased resources, etc)
- `MPICH_SOCKET_BUFFER_SIZE` socket buffer size
- `MPICH_SOCKET_RBUFFER_SIZE` socket receive buffer size
- `MPICH_SOCKET_SBUFFER_SIZE` socket send buffer size
- `MPICH_SOCKET_NUM_PREPOSTED_ACCEPTS` number of accepts posted for MPIDU_Sock_listen
- `MPICH_PORT_RANGE` Range of ports to use for sockets: min..max or min,max

- `MPICH_INTERFACE_HOSTNAME` hostname to use to connect sockets
- `MPICH_NETMASK` bitmask to select an ip subnet: ip/numbits, ie 192.0.0.0/8
- `MPIEXEC_TIMEOUT` job timeout in seconds
- `MPIEXEC_LOCALONLY` launch job processes on the local machine only
- `MPIEXEC_NOPROMPT` Don't prompt for user input for missing information, print an error instead.
- `MPIEXEC_SMPD_PORT` Connect to smpd on the specified port.

  The following two only affect mpiexec for smpd if -rsh is on the command line:

- `MPIEXEC_RSH` rsh command to use, default is "ssh -x"
- `MPIEXEC_RSH_NO_ESCAPE` create an rsh command compatible with Cygwin's ssh
- `MPICH_SPN` Service Principal Name used for passwordless authentication
- `SMPD_DBG_OUTPUT` Print debugging output
- `SMPD_DBG_LOG_FILENAME` name of logfile to send output to
- `SMPD_MAX_LOG_FILE_SIZE` maximum number of bytes the logfile can grow to before it is truncated
- `MPICH_DBG_OUTPUT` stdout, memlog or file. determines where debugging output goes
- `MPI_DLL_NAME` name of the dll that contains the MPI and PMPI interfaces
- `MPI_DLL_PATH` path of the dll that contains the MPI and PMPI interfaces
- `MPICH2_CHANNEL` short name of the channel used to create the full name of the MPI dll (ie. ib becomes mpich2ib.dll)
- `MPI_WRAP_DLL_NAME` name of the dll that contains only the MPI interface, not the PMPI interface
- `MPICH_TRMEM_INITZERO` used by the memory tracing package

- MPICH_TRMEM_VALIDATE used by the memory tracing package
- MPITEST_DEBUG used by the test suite
- MPITEST_VERBOSE used by the test suite
- PATH used by smpd to search for executables under Unix.

SMPD options specified on the command line can also be specified in the environment by prefixing SMPD_OPTION_ to the option name and saving it as an environment variable.

- SMPD_OPTION_APP_PATH
- SMPD_OPTION_LOGFILE
- SMPD_OPTION_NOCACHE
- SMPD_OPTION_PHRASE
- SMPD_OPTION_SSPI_PROTECT
- SMPD_OPTION_MAX_LOGFILE_SIZE
- SMPD_OPTION_PLAINTEXT
- SMPD_OPTION_PORT
- SMPD_OPTION_TIMEOUT
- SMPD_OPTION_EXITCODES
- SMPD_OPTION_PRIORITY
- SMPD_OPTION_LOCALONLY
- SMPD_OPTION_NOPROMPT
- SMPD_OPTION_CHANNEL
- SMPD_OPTION_HOSTS
- SMPD_OPTION_DELEGATE
- SMPD_OPTION_INTERNODE_CHANNEL

- SMPD_OPTION_LOG
- SMPD_OPTION_NO_DYNAMIC_HOSTS

Variables to control debugging output when enabled:

- MPICH_DBG
- MPICH_DBG_CLASS
- MPICH_DBG_FILENAME
- MPICH_DBG_LEVEL
- MPICH_DBG_OUTPUT
- MPICH_DBG_RANK
- MPICH_DEBUG_ITEM

The following variables affect the MPE logging library:

- MPE_LOGFILE_PREFIX name of the clog file to create without the extension
- MPE_DELETE_LOCALFILE true,false - delete or not the local clog file
- MPE_LOG_OVERHEAD I think this one adds an event to the clog files representing the time it takes to write a clog buffer to disk
- CLOG_BLOCK_SIZE number of bytes in a clog block
- CLOG_BUFFERED_BLOCKS number of blocks
- MPE_CLOCKS_SYNC yes/no - synchronize clocks

  directories to store temporary files:

- MPE_TMPDIR
- TMPDIR
- TMP
- TEMP

PMI environment variables created by smpd are described in the smpd documentation:

- PMI_DLL_NAME name of the PMI dll to load (replaces the default smpd functions)
- PMI_NAMEPUB_KVS name of the key-val-space where MPI service names are stored for MPI_Lookup_name()
- PMI_ROOT_HOST
- PMI_ROOT_PORT
- PMI_ROOT_LOCAL
- PMI_SPAWN
- PMI_KVS
- PMI_DOMAIN
- PMI_RANK
- PMI_SIZE
- PMI_CLIQUE
- PMI_APPNUM
- PMI_SMPD_ID
- PMI_SMPD_KEY
- PMI_SMPD_FD
- PMI_HOST
- PMI_PORT
- PMI_APPNUM

Used by the process managers other than smpd:

- MPIEXEC_DEBUG
- MPIEXEC_MACHINES_PATH

- MPIEXEC_PORTRANGE
- MPIEXEC_PREFIX_STDERR
- MPIEXEC_PREFIX_STDOUT
- MPIEXEC_REMSHELL
- MPIEXEC_USE_PORT

9.11 Compiling

This section describes how to set up a project to compile an MPICH2 application using Visual Studio 2005 and Visual Studio 6.0.

9.11.1 Visual Studio 6.0

Visual C++ 6.0 cannot handle multiple functions with the same type signature that only differ in their return type. So you must define HAVE_NO_VARIABLE_RETURN_TYPE_SUPPORT in your project.

1. Create a project and add your source files.
2. Bring up the settings for the project by hitting Alt F7. Select the Preprocessor Category from the C/C++ tab. Enter HAVE_NO_VARIABLE_RETURN_TYPE_SUPPORT into the Preprocessor box. Enter C:\Program Files\MPICH2\include into the "Additional include directories" box.
3. Select the Input Category from the Link tab. Add cxx.lib and mpi.lib to the Object/library modules box. Add C:\Program Files\MPICH2\lib to the "Additional library path" box.
4. Compile your application.

9.11.2 Visual Studio 2005

You can use the example projects provided with Visual Studio 2005 and use it as a guide to create your own projects.

1. Create a project and add your source files.

2. Bring up the properties dialog for your project by right clicking the project name and selecting Properties.
3. Navigate to Configuration Properties::C/C++::General
4. Add `C:\Program Files\MPICH2\include` to the "Additional Include Directories" box.
5. Navigate to Configuration Properties::Linker::General
6. Add `C:\Program Files\MPICH2\lib` to the "Aditional Library Directories" box.
7. Navigate to Configuration Properties::Linker::Input
8. Add `cxx.lib` and `mpi.lib` and `fmpich2.lib` to the "Additional Dependencies" box. If your application is a C application then it only needs `mpi.lib`. If it is a C++ application then it needs both `cxx.lib` and `mpi.lib`. If it is a Fortran application then it only needs one of the `fmpich2[s,g].lib` libraries. The fortran library comes in three flavors `fmpich2.lib`, `fmpich2s.lib` and `fmpich2s.lib`. `fmpich2.lib` contains all uppercase symbols and uses the C calling convention like this: `MPI_INIT`. `fmpich2s.lib` contains all uppercase symbols and uses the stdcall calling convention like this: `MPI_INIT@4`. `fmpich2g.lib` contains all lowercase symbols with double underscores and the C calling convention like this: `mpi_init__`. Add the library that matches your Fortran compiler.
9. Compile your application.

9.11.3 Cygwin gcc

You can compile your MPI programs using gcc/g++ from Cygwin and the MPICH2 header files/libraries installed with MPICH2 on windows. Compile using the header files in `C:\Program Files\MPICH2\include` and link using the libs, lib*.a, in `C:\Program Files\MPICH2\lib`. Note that you should use the "-localroot" option when running programs compiled using gcc/g++ from Cygwin.

9.12 Performance Analysis

MPICH2 includes the Multi-Processing Environment (MPE), which is a suite of performance analysis tools comprising profiling libraries, utility programs, a set of graphical tools, and a collective checking library.

The first set of tools to be used with user MPI programs is profiling libraries which provide a collection of routines that create log files. These log files can be created manually by inserting MPE calls in the MPI program, or automatically by linking with the appropriate MPE libraries, or by combining the above two methods. Currently, MPE offers the following four profiling libraries.

1. Tracing Library: Traces all MPI calls. Each MPI call is preceded by a line that contains the rank in `MPI_COMM_WORLD` of the calling process, and followed by another line indicating that the call has completed. Most send and receive routines also indicate the values of count, tag, and partner (destination for sends, source for receives). Output is to standard output.

2. Animation Libraries: A simple form of real-time program animation that requires X window routines (Currently not available on windows).

3. Logging Libraries: The most useful and widely used profiling libraries in MPE. These libraries form the basis for generating log files from user MPI programs. Several different log file formats are available in MPE. The default log file format is CLOG2. It is a low overhead logging format, a simple collection of single timestamp events. The old format ALOG, which is not being developed for years, is not distributed here. The powerful visualization format is SLOG-2, stands for Scalable LOGfile format version II, which is a total redesign of the original SLOG format. SLOG-2 allows for much improved scalability for visualization purpose. A CLOG2 file can be easily converted to SLOG-2 file through the new SLOG-2 viewer, Jumpshot-4.

4. Collective and datatype checking library: An argument consistency checking library for MPI collective calls. It checks for datatype, root, and various argument consistency in MPI collective calls (Currently not available on Windows).

The set of utility programs in MPE includes log format converter (e.g.

clogTOslog2) and logfile viewer and convertor (e.g. Jumpshot). These new tools, clog2TOslog2 and Jumpshot (Jumpshot-4) replace old tools, clog2slog, slog_print and logviewer (i.e. Jumpshot-2 and Jumpshot-3).

9.12.1 Tracing MPI calls using the MPIEXEC Wrapper

A developer can trace MPI calls by using the tracing functionality of the mpiexec wrapper (wmpiexec.exe). A step by step process is given below.

1. Launch the mpiexec wrapper application (wmpiexec.exe).
2. After launching the mpiexec wrapper, select the application that you would like to run and select the number of processes. Now click on the "more options" checkbox to show the extended options for mpiexec.
3. Check the "produce clog2 file" checkbox so that the clog2 file is generated when the application is run.
4. Check "run in an separate window" checkbox to enable your program to run in a separate window (for user interaction).
5. Run your application by clicking on the "Execute" button.
6. Once the application exits, click on the "Jumpshot" button to launch Jumpshot (the logfile viewer).
7. Open your logfile (the default name of the logfile is `<APPLICATION NAME>.clog2`) using Jumpshot. Jumpshot will ask for converting the logfile to slog2 format. Click "Convert" button in Jumpshot to convert your logfile to slog2 format.
8. Now click on "OK" button in Jumpshot to view the logfile.

9.12.2 Tracing MPI calls from the command line

1. Run your application using the "-log" option to the mpiexec command.
2. Launch Jumpshot using the java command. (eg: `java -jar "c:\Program Files\MPICH2\bin\jumpshot.jar"`)
3. Follow the steps mentioned in the previous section to convert the logfile to slog2 format and view the log.

9.12.3 Customizing logfiles

In addition to using the predefined MPE logging libraries to log all MPI calls, MPE logging calls can be inserted into the user's MPI program to define and log states. These states are called user-defined states. States may be nested, allowing one to define a state describing a user routine that contains several MPI calls, and display both the user-defined state and the MPI operations contained within it.

The simplest way to insert user-defined states is as follows:

1. Get handles from MPE logging library. `MPE_Log_get_state_eventIDs` must be used to get unique event IDs (MPE logging handles). This is important if you are writing a library that uses the MPE logging routines from the MPE system. Hardwiring the eventIDs is considered a bad idea since it may cause eventID confict and so the practice isn't supported. The older MPE library provides `MPE_Log_get_event_number`, which is still being supported but has been deprecated; users are strongly urged to use `MPE_Log_get_state_eventIDs` instead.
2. Set the logged state's characteristics. `MPE_Describe_state` sets the name and color of the states.
3. Log the events of the logged states. `MPE_Log_event` is called twice to log the user-defined states.

Below is a simple example that uses the three steps outlined above.

```
int eventID_begin, eventID_end;
...
MPE_Log_get_state_eventIDs( &eventID_begin, &eventID_end );
...
MPE_Describe_state( eventID_begin, eventID_end, "Multiplication", "red" );
...
MyAmult( Matrix m, Vector v )
{
    /* Log the start event along with the size of the matrix */
    MPE_Log_event( eventID_begin, 0, NULL );
    ... Amult code, including MPI calls ...
    MPE_Log_event( eventID_end, 0, NULL );
}
```

The logfile generated by this code will have the MPI routines nested within the routine `MyAmult`.

Besides user-defined state, MPE2 also provides support for user-defined events, which can be defined through use of `MPE_Log_get_solo_eventID` and `MPE_Describe_event`. For more details, see cpilog.c.

For undefined user-defined state (where the corresponding `MPE_Describe_state` has not been issued), the new Jumpshot (Jumpshot-4) may display the legend name as "UnknownType-INDEX" where INDEX is the internal MPE category index.

An example program, cpilog.c, is provided in the "examples" directory of your MPICH2 installation. This program can be used as a reference for customizing logfiles.