ANL/MCS-TM-ANL-01/x Rev x

Installation and User's Guide to MPICH, a Portable Implementation of MPI Version 1.2.5 The ch_nt device for workstations and clusters of Microsoft Windows machines

by

David Ashton, William Gropp, and Ewing Lusk

MATHEMATICS AND
COMPUTER SCIENCE
DIVISION

Contents

Abstract **1**

1 Introduction **1**

2 System Requirements **1**

3 Quick Start **2**
3.1 Downloading `MPICH` . . . 2
3.2 Installing . . . 2
3.3 Configuring . . . 2
3.4 Making and running an example . . . 3

4 Programming Tools **3**
4.1 Compiling and linking with Microsoft Developer Studio (VC++ 6.x) . . . 3
4.2 Compiling with VC++ from the command line . . . 4
4.3 Compiling and linking with Fortran . . . 4
4.4 Compiling and linking with gcc or g77 . . . 5
4.5 Debugging . . . 5
4.5.1 The `printf` Approach . . . 6
4.5.2 Error handlers . . . 6
4.5.3 Starting processes manually . . . 6
4.5.4 Attaching a debugger to a running program . . . 7
4.6 Log and tracefile tools . . . 7
4.6.1 Logging C applications . . . 7
4.6.2 Logging Fortran applications . . . 7
4.6.3 Jumpshot . . . 8
4.7 Performance measurements . . . 8

5 Details **9**
5.1 mpirun for mpd . . . 9
5.1.1 Usage . . . 10
5.1.2 Configuration files for mpirun . . . 10
5.1.3 Command line options for mpirun . . . 11
5.2 MPIRegister tool . . . 12
5.3 Configuration tool . . . 13
5.3.1 Create the host list . . . 13
5.3.2 Select the options to configure . . . 13
5.3.3 Apply the settings . . . 15
5.4 Update tool . . . 15
5.4.1 Create the host list . . . 15
5.4.2 Select the new binaries . . . 16
5.4.3 Apply the updates . . . 16
5.5 Runtime environment variable options . . . 17
5.6 Job manager tool . . . 19
5.7 MPD process launcher . . . 20
5.7.1 Quick reference . . . 20
5.7.2 Command line options . . . 22

5.7.3 Console commands 24
5.8 MPICH and threads 33
5.9 Rebuilding the MPICH dlls from the source distribution 33
5.9.1 Download the source distribution 33
5.9.2 Build 33

6 Documentation **35**

7 In Case of Trouble **36**
7.1 Things to try first 36
7.2 Submitting bug reports 36
7.3 The Most Common Problems 37

8 FAQ **37**
8.1 Error 64 - GetQueuedCompletenessStatus failed 37
8.2 No more connections 37
8.3 my windows don't show up 38
8.4 mpirun options don't work 38
8.5 mpirun in a bash shell doesn't work 38
8.6 Does MPICH work on Windows98? 39
8.7 Can I run on both Windows and Linux at the same time? 39

A History of MPICH **39**

B File Manifest **40**

C Automated installation **40**

D Distribution files **42**

E MSDEV Project settings **42**

References **47**

Abstract

MPI (Message-Passing Interface) is a standard specification for message-passing libraries. MPICH is a portable implementation of the full MPI-1.2 specification for a wide variety of parallel and distributed computing environments. MPICH contains, along with the MPI library itself, a programming environment for working with MPI programs. The programming environment includes a startup mechanism and a profiling library for studying the performance of MPI programs. This document describes how to install and use MPICH on Micrsoft Windows systems.

This document describes how to obtain, install, and use MPICH [7], the portable implementation of the MPI Message-Passing Standard. This document describes version 1.2.5.

1 Introduction

MPICH is a freely available implementation of the MPI standard that runs on a wide variety of systems. The details of the MPICH implementation are described in [7]; related papers include [5] and [6].

Major Features of MPICH:

- Full MPI 1.2 compliance, *including* cancel of sends.
- MPMD programs.
- Multiple Fortran bindings.
- Parts of MPI-2 are also supported:
  - Most of MPI-IO is supported through the ROMIO implementation (See '`romio/README`' for details).
  - Support for `MPI_INIT_THREAD` (but only for `MPI_THREAD_SINGLE` and `MPI_THREAD_FUNNELLED`).
  - Miscellaneous new `MPI_Info` and `MPI_Datatype` routines.
- MPICH also includes components of a parallel programming environment, including
  - Tracing and logfile tools based on the MPI profiling interface, including a scalable logfile format (SLOG).
  - Parallel performance visualization tools (`jumpshot`).
  - Extensive correctness and performance tests.
  - Both large and small application examples.

2 System Requirements

MPICH for Windows requires the following:

- WindowsNT4/2000/XP Professional or Server (Win9x/ME are not supported)
- The ability to make TCP/IP socket connections between all hosts.

To use the default installation you must be able to run '`mpich.nt.1.2.4.exe`' while logged on as an administrator. If you do not have administrator privileges on your machine, you will need to get a system administrator to run setup for you.

If you don't have administrator privileges and just want to evaluate `MPICH` see Section 5.7.1. If you want to run under Windows9x in a limited fashion see FAQ 8.6.

Known compilers that may be used with the distributed libraries are MS VC++ 6.x, MS VC++.NET, Compaq Visual Fortran 6.x, Intel Fortran, gcc, and g77.

To compile the mpich dlls from the source distribution, which is not required because the dlls come pre-compiled, you need Visual C++ 6.x and Visual Fortran 6.x.

3 Quick Start

Here is a set of steps for setting up `MPICH`. Details and instructions for a more thorough tour of `MPICH`'s features, including installing, validating, benchmarking, and using the performance evaluation tools, are given in the following sections.

3.1 Downloading `MPICH`

The first step is to download `MPICH` .

The easiest way to get `MPICH` is to use the web page `www.mcs.anl.gov/mpi/mpich/download.html`; you can also use anonymous ftp from `ftp.mcs.anl.gov` in directory '`pub/mpi/nt`'. Get the file '`mpich.nt.1.2.4.exe`'.

3.2 Installing

1. Logon to your machine using an account with adminstrator privileges. You may have to get your systems administrators to do this part for you.
2. Execute '`mpich.nt.1.2.4.exe`', selecting all the defaults.
3. Repeat on each of your machines. (See Appendix C for a command line installation method that can be used in a script)

3.3 Configuring

If you installed on one machine only, you may skip this step.

Before some of the automatic features of `MPICH` can work, you must configure it.

1. Invoke the configuration tool on one host in your cluster:
   start->programs->MPICH->mpd->MPICH Configuration tool.
2. Add the hosts where you installed `MPICH` to the list with the Add or Select button.
3. Click Apply to set the hosts configuration option. This option saves a list of hosts in the Windows registry from which mpirun can draw names when it needs to select hosts to launch processes on.
4. Click OK to exit.

For convenience you should add the launcher tools bin directory to your path. For the default installation this will be `C:\Program Files\MPICH\mpd\bin`

3.4 Making and running an example

There are sample MPI applications in the `MPICH\SDK\examples\nt` directory. They can be built with Microsoft Visual Studio 6.x, Visual Fortran 6.x, gcc and g77.

1. Open the MSDEV workspace file found in `MPICH\SDK\examples\nt\examples.dsw`.
2. Build the Debug target of the cpi project.
3. Copy `MPICH\SDK\examples\nt\basic\Debug\cpi.exe` to a shared directory or to the same place on all the machines in your cluster. For example you could copy `cpi.exe` to `c:\temp\cpi.exe` on all the nodes.
4. Open a command prompt and change to the directory where you placed `cpi.exe`.
5. Execute '`MPICH\mpd\bin\mpirun.exe -np 4 cpi`'.

4 Programming Tools

This section describes the use of some tools.

4.1 Compiling and linking with Microsoft Developer Studio (VC++ 6.x)

1. Create a new project.
2. Add `MPICH\SDK\include` to the include path.
3. Add `MPICH\SDK\lib` to the library path.
4. Add the /MTd compiler switch to the Debug target and /MT to the Release target.
5. Add `ws2_32.lib` to the library option. Add `mpich.lib` to the Release target and `mpichd.lib` to the Debug target.
6. Add your source files.

7. Build.

8. Copy the executable and use `mpirun` to run the application.

A graphical illustration of where to make the compiler settings can be found in Appendix E

4.2 Compiling with VC++ from the command line

If you want to compile from the command line instead of using the MS Integrated Development Environment, here are the compile and link commands copied from the cpi project in the examples directory.

1. Bring up a command prompt.

2. Execute '`vcvars32.bat`' to set up the environment variables for VC++.

3. compile '`example.c`':

   Debug target: execute this:

   ```
   cl.exe /nologo /MTd /W3 /GX /Od /I "C:\Program Files\MPICH\SDK\include"
   /D WIN32 /D _DEBUG /D _CONSOLE /D _MBCS /GZ /c example.c
   ```

   Release target: execute this:

   ```
   cl.exe /nologo /MT /W3 /GX /O2 /I "C:\Program Files\MPICH\SDK\include" /D
   WIN32 /D NDEBUG /D _CONSOLE /D _MBCS /c example.c
   ```

4. link '`example.obj`':

   Debug target: execute this:

   ```
   link.exe ws2_32.lib mpichd.lib kernel32.lib user32.lib gdi32.lib winspool.lib
   comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib
   odbccp32.lib /nologo /subsystem:console /debug /machine:I386 /out:"example.exe"
   /pdbtype:sept /libpath:"C:\Program Files\MPICH\SDK\lib" example.obj
   ```

   Release target: execute this:

   ```
   link.exe ws2_32.lib mpich.lib kernel32.lib user32.lib gdi32.lib winspool.lib
   comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib
   odbccp32.lib /nologo /subsystem:console /machine:I386 /out:"example.exe"
   /libpath:"C:\Program Files\MPICH\SDK\lib" example.obj
   ```

   Depending on what functions '`example.c`' uses, you may not need all the libraries specified above.

4.3 Compiling and linking with Fortran

To compile with Visual Fortran, follow the same steps as in the Visual C++ Section 4.1. Visual Fortran uses the same linker as MS Visual C++.

The `MPICH` dlls contain the following interfaces for Fortran:

1. `MPI_INIT@4` - example symbol in the mpich dll

   Uppercase externals using the standard calling convention with mixed string length parameters. This is the default for Visual Fortran. But, it doesn't allow passing strings to any of the message passing functions like MPI_SEND because strings cause the function signature to change. To use strings you must use the C calling convention.

2. `MPI_INIT`

   Uppercase externals using the C calling convention with string length parameters at the end of the list. This is the default for the Intel Fortran compiler. You can also use this interface with Visual Fortran by adding the following compiler flags: `/iface:cref /iface:nomixed_str_len_arg`

3. `mpi_init__`

   Lowercase double underscore externals using the C calling convention with string length parameters at the end of the argument list. This is the default for g77.

If your Fortran compiler is not compatible with any of these interfaces then you will have to re-build the mpich libraries from the source distribution, '`mpich.nt.1.2.4.src.exe`', and change the linkage to match your compiler. See Section 5.9 for details on re-building the mpich libraries.

4.4 Compiling and linking with gcc or g77

1. Create a makefile.
2. Add `-I.../MPICH/SDK.gcc/include`
3. Add `-L.../MPICH/SDK.gcc/lib`
4. Add `-lmpich`
5. Add the rules for your source files.
6. make
7. Copy the executable and use `mpirun` to run the application.

4.5 Debugging

Debugging parallel programs is notoriously difficult. Parallel programs are subject not only to the usual kinds of bugs but also to new kinds having to do with timing and synchronization errors. Often, the program "hangs," for example when a process is waiting for a message to arrive that is never sent or is sent with the wrong tag. Parallel bugs often disappear precisely when you adds code to try to identify the bug, which is particularly frustrating. In this section we discuss several approaches to parallel debugging.

4.5.1 The `printf` Approach

Just as in sequential debugging, you often wish to trace interesting events in the program by printing trace messages. Usually you wish to identify a message by the rank of the process emitting it. This can be done explicitly by putting the rank in the trace message.

It is recommended that you call fflush(stdout) after your printf statements to ensure the output gets forwarded to the root without delay.

4.5.2 Error handlers

The MPI Standard specifies a mechanism for installing one's own error handler, and specifies the behavior of two predefined ones, MPI_ERRORS_RETURN and MPI_ERRORS_ARE_FATAL.

4.5.3 Starting processes manually

You can start each process in a parallel job by hand by setting the appropriate environment variables. Each process needs the following variables:

1. MPICH_JOBID=some short unique string to identify the job
2. MPICH_NPROC=total number of processes in the job
3. MPICH_IPROC=rank of the current process
4. MPICH_ROOT=host:port where the root process will live and listen

If you set these by hand then you can run each process in a debugger.

Here is an example to run a two process job from two command prompts on the machines Fry and Jazz:
On Fry

```
C:\Temp>set MPICH_JOBID=fry.123
C:\Temp>set MPICH_IPROC=0
C:\Temp>set MPICH_NPROC=2
C:\Temp>set MPICH_ROOT=fry:12345
C:\Temp>netpipe.exe
```

On Jazz

```
C:\Temp>set MPICH_JOBID=fry.123
C:\Temp>set MPICH_IPROC=1
C:\Temp>set MPICH_NPROC=2
C:\Temp>set MPICH_ROOT=fry:12345
C:\Temp>netpipe.exe
```

If you want to debug netpipe.exe, execute '`msdev netpipe.exe`' instead of simply '`netpipe`'.

4.5.4 Attaching a debugger to a running program

You can often attach the MSDEV debugger to a running process locally. Visual C++.NET has the ability to debug processes remotely. See the MSDEV help utility for details.

4.6 Log and tracefile tools

The Windows distribution of `MPICH` comes with the MPE library for profiling applications and the Jumpshot java tool for visualizing the generated log files.

4.6.1 Logging C applications

The MPE library is used for logging information about the execution of each MPI call to a log file for later analysis. This library may be accessed when linking the program. For example, to create a log file of a program such as `cpi`, all that needs to be done is to insert the mpe library in the link statement before the mpich library: '`mpe.lib mpich.lib`'

The log file will be written to a file with the name '`cpi.exe.clog`' or '`cpi.exe.slog`', depending on the value of the environment variable `MPE_LOG_FORMAT` (clog is the default). A clog file can be converted to the slog format using the '`clog2slog.exe`' tool found in the `MPICH\SDK\profiling` directory. Only small slog files can be created directly. If the log file is going to be large, you must create a clog file and then convert it to an slog file. SLOG files can be graphically displayed using the Jumpshot program, described in Section 4.6.3.

Here is an example:

1. Build cpi from the examples\nt directory selecting the PDebug target. This project has '`mped.lib mpichd.lib`' in the link command.
2. mpirun -np 4 cpi.exe
3. clog2slog cpi.exe.clog
4. java -jar jumpshot3.jar
5. File=>Select Logfile, choose cpi.exe.slog

4.6.2 Logging Fortran applications

The Fortran interface in the mpich dlls cannot be profiled. In order to profile a Fortran application, the application must be re-linked with the static mpich libraries. The static libraries can be built from the source distribution '`mpich.nt.1.2.4.src.exe`' using the '`mpich.static.dsw`' workspace in the mpich root directory, or downloaded from `ftp.mcs.anl.gov` in directory '`pub/mpi/nt/binaries`'. The debug static libraries are: '`mpichsd.lib`', '`pmpichsd.lib`' and '`mpichfsd.lib`'. The release static libraries are: '`mpichs.lib`', '`pmpichs.lib`' and '`mpichfs.lib`'. You also need the '`mpdutil.lib`' and '`crypt.lib`' libraries from the mpd workspace that can be built or downloaded.

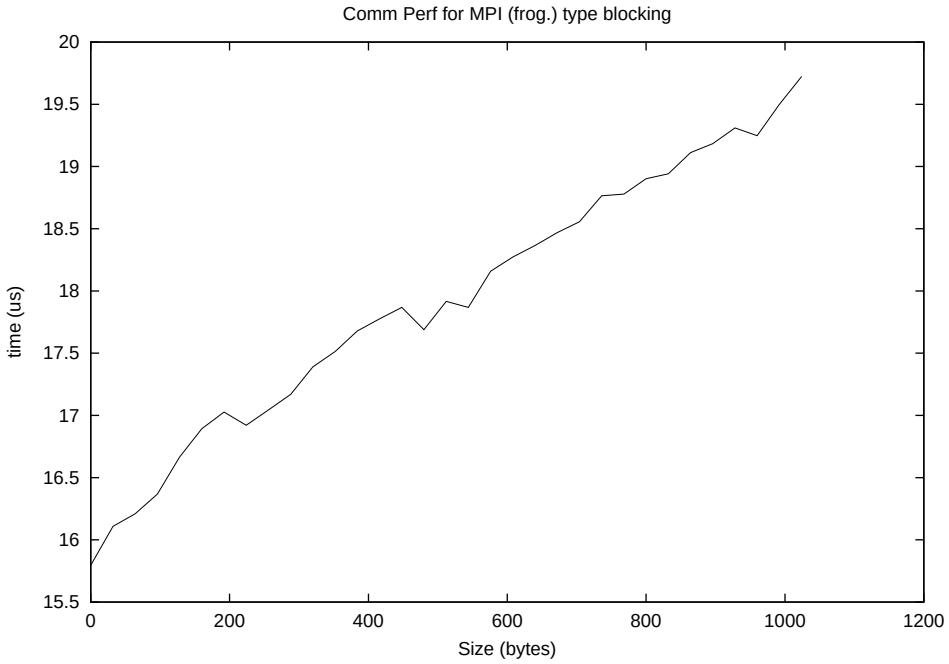

Figure 1: Sample output from `mpptest`

For debug targets, add '`mpichfsd.lib mped.lib mpichsd.lib pmpichsd.lib mpdutil.lib crypt.lib`' to the link command.

For release targets, add '`mpichfs.lib mpe.lib mpichs.lib pmpichs.lib mpdutil.lib crypt.lib`' to the link command.

After building with the static libraries, run the application and view the log file using the steps described in the previous section.

4.6.3 Jumpshot

Jumpshot is a program for displaying logfiles produced using the MPE logging library. Jumpshot is described in more detail in the *MPE Installation and User's manual* [2]. Jumpshot is a Java program and requires a functioning Java environment. You can view an slog logfile such as '`cpi.exe.slog`' in the example above by using '`jumpshot3.jar`' found in the `MPICH\Jumpshot` directory. Shortcuts to the Jumpshot manual and jumpshot3.jar are found at: start->programs->MPICH->Jumpshot

4.7 Performance measurements

The `mpich\sdk\examples\nt\mpptest` directory contains a sophisticated tool for measuring latency and bandwidth for `MPICH` on your system. After building mpptest from the examples workspace in MSDEV Studio, simply change to the `mpichsdk\examples\nt\mpptest` directory and do

```
mpirun -np 2 mpptest -gnuplot > out.gpl
```

The file 'out.gpl' will then contain the necessary gnuplot commands. The file 'mppout.gpl' will contain the data. To view the data with gnuplot, use:

```
gnuplot out.gpl
```

or use

```
load 'out.gpl'
```

from within gnuplot. You can use

```
gnuplot
set term postscript eps
set output "foo.eps"
load 'out.gpl'
```

to create an Encapsulated Postscript graph such as the one in Figure 1.

The program mpptest has a wide variety of capabilities; the option -help will list them. For example, mpptest can automatically pick message lengths to discover any sudden changes in behavior and can investigate the ability to overlap communication with computation. More information is available at http://www.mcs.anl.gov/mpi/mpptest.

gnuplot can be found here: ftp://ftp.dartmouth.edu/pub/gnuplot

Benchmarking can be very tricky. Some common benchmarking errors are discussed at http://www.mcs.anl.gov/mpi/mpptest/hownot.html. The paper [10] discusses these issues at more length.

5 Details

This section covers details of the specific pieces of MPICH. Read this section if you need more detailed control over configuring, building, installing, or operating MPICH.

5.1 mpirun for mpd

mpirun is the tool that communicates with the mpd process launcher to start MPI applications. mpirun comes in two flavors - mpirun and guimpirun. mpirun is the command line version of the tool and guimpirun is the graphical version. The command line tool was developed first and then the gui tool was ported from the command line version. For this reason, the command line tool is more stable.

You will want to add MPICH\mpd\bin to your path to have access to mpirun and the other tools from a command prompt.

5.1.1 Usage

```
mpirun [-mpirun options] configfile [args ...]
mpirun -np #processes [-mpirun options] executable [args ...]
```

Bracketed sections are optional (don't type the `[]` characters).

5.1.2 Configuration files for mpirun

The configuration file format is as follows:

```
exe c:\somepath\myapp.exe
 OR \\host\share\somepath\myapp.exe
[args arg1 arg2 arg3 ...]
[env VAR1=VAL1|VAR2=VAL2|...|VARn=VALn]
[dir drive:\some\path]
[map drive:\\host\share]
hosts
hostA #procs [path\myapp.exe]
hostB #procs [\\host\share\somepath\myapp2.exe]
hostC #procs
...
```

Bracketed lines are optional (don't inlucde the `[]` characters). The # character will comment out a line. You may specify a path to an executable on each host line, thus enabling MPMD programming. If you do not specify a path, then the default is used from the exe line.

Here are two sample configuration files:

```
exe c:\temp\myapp.exe
hosts
fry 1
jazz 2
```

This one shows a more complicated scenario:

```
exe c:\temp\slave.exe
env MINX=0|MAXX=2|MINY=0|MAXY=2
args -i c:\temp\cool.points
hosts
fry 1 c:\temp\master.exe
fry 1
#light 1
jazz 2
```

This configuration file would launch one instance of '`master.exe`' on fry and three instances of '`slave.exe`', one on fry and two on jazz. Host light would be ignored because of the # character. Each process would have four environment variables set. Each process would receive "`-i c:\temp\cool.points`" as command line arguments.

5.1.3 Command line options for mpirun

`-np #procs`
: Launch `#procs` processes. mpirun uses the list of hosts stored in the registry by the configuration tool to choose hosts to start processes on. If there is no list in the registry all the processes are launched on the local host.

`-machinefile filename`
: This tells mpirun to use the hosts in '`filename`' when determining where to launch processes. Use this in conjunction with '`-np x`' to launch processes on a specific set of machines. Put one host per line in the file. Empty lines are discarded and lines starting with # are ignored. You can specify a number after the host name to recommend how many processes to launch on the host. This is useful if you want to launch more than one process on a multi-CPU machine. mpirun will cycle through this list until all the processes are launched, repeating hosts if necessary. Example file:

```
ccnode01
ccnode02 2
ccnode03 4
ccnode04
```

`-localonly`
: This flag causes all the processes to be launched on the local machine using the shared memory device.

`-localonly -tcp`
: Add the -tcp switch to force the use of sockets instead of shared memory

`-env "var1=val1|var2=val2|var3=val3|...varn=valn"`
: This will set the environment variables specified in the string before each process is launched. Remember to quote the string so the command prompt doesn't interpret the vertical bar as a pipe command.

`-logon`
: This option will cause mpirun to prompt for an account and password. If you use mpiregister.exe to encrypt an account and password into the registry, -logon will override the use of that user.

`-map drive:\\host\share`
: This option will map a drive on the hosts where the processes are launched. The mappings are removed after the processes exit. This option can be repeated multiple times. example: `-map z:\\myserver\myhome`

`-dir drive:\some\path`
: This sets the working directory for the launched processes. If this option is not specified the current directory is used.

`-hosts n host1 host2 ...  hostn`

`-hosts n host1 m1 host2 m2 ...  hostn mn`
: Specify the hosts to launch on. In the second form, the number of processes is m1 + m2 + ... + mn.

`-pwdfile filename`
: Specify a file containing an account and password used to launch processes under. The first line of the file must be the account name and the second line must be the password.

`-exitcodes`
: This option causes mpirun to print out the exit code of each process as it exits.

`-noprompt`
: This option prevents mpirun from prompting for user credentials if they have not been stored in the registry.

`-priority class:level`
: This option set the process run priority. The class can be a value from 0 to 4 representing idle, below, normal, above, and high priority classes (realtime priority is not allowed). The level can be a value from 0 to 5 representing idle, lowest, below, normal, above, and higest. The values corresponding to below and above are only supported on Window2000 and XP. An example would be `-priority 3:4`. The default is 2:3.

`-mpduser`
: Use this option to launch a job in the context of the user registered with mpd. All the mpd's must have been installed with the mpduser option, an account must be set on each mpd, and the mpduser option must be enabled on each mpd. If the mpd's are configured correctly, mpirun will not use the current user's credentials to launch the job but instead launch in the context of the registered mpd user.

5.2 MPIRegister tool

MPIRegister.exe is a tool to encrypt an account and password into the registry for the current user. It is found in the '`MPICH\mpd\bin`' directory. The information it stores is used by mpirun to launch applications in the context of the specified user. If you don't use mpiregister then mpirun will prompt for an account and password each time it is invoked.

Usage:

- MPIRegister
- MPIRegister -remove
- MPIRegister -validate [-nocache -host h -port p -phrase x]

First it prompts for an account. Enter the name in the form [Domain \]Account where the domain name is optional (ie mcs\ashton or ashton). Then it prompts for the password twice. Finally it asks if you wish to make the action persistent. If you say yes then the data will be saved to the hard drive. If you say no then the data will only remain in memory. This means that you may run mpirun various times and it will not prompt you for an

account and password. But, when you re-boot the machine and use mpirun again, it will prompt for an account and password.

The -remove option will delete the information from the registry.

The -validate option will connect to an mpd and attempt to validate the user credentials by logging on the user with the supplied password. This option must be used only after the user credentials have been saved. It does not prompt for credentials. The host, port and phrase options are optional and refer to where the mpd is located. If they are not specified the local host and default mpd port and passphrase are used. The -nocache option causes the mpd to ignore any cached user handles.

5.3 Configuration tool

The configuration tool is a graphical interface to the registry settings that control some of runtime configurable options to `MPICH`.

In order to run an application on various hosts without specifying the hosts in a configuration file, the launcher needs to know all the hosts it has been installed on. MPIConfig.exe can find the hosts where the launcher has been installed and write this list of hosts to the registry. With this information mpirun can pick hosts from the list in the registry when determining where to launch processes.

A shortcut to the configuration tool is found here:
start->programs->MPICH->mpd->MPICH Configuration tool.
See Figure 2 for a snapshot of the tool.

5.3.1 Create the host list

Section 1 of the dialog has a list that needs to be filled with the hostnames where mpd has been installed. Use the buttons and/or type host names directly into the edit box to create the list.

Add button: Adds the host name from the edit box to the list

Select button: Bring up a dialog to select host names from the network.

5.3.2 Select the options to configure

Section 2 of the dialog contains the list of options that can be configured.

- If you check the hosts box, mpiconfig makes a group out of all the selected hosts and writes this list of host names to the registry on each host. When mpirun is executed with the -np option from any host in a group, hosts will be selected from the list in round robin fashion.
- The launch timeout specifies how long MPIRun will wait before it determines that it is unable to launch a process. The time is in seconds.

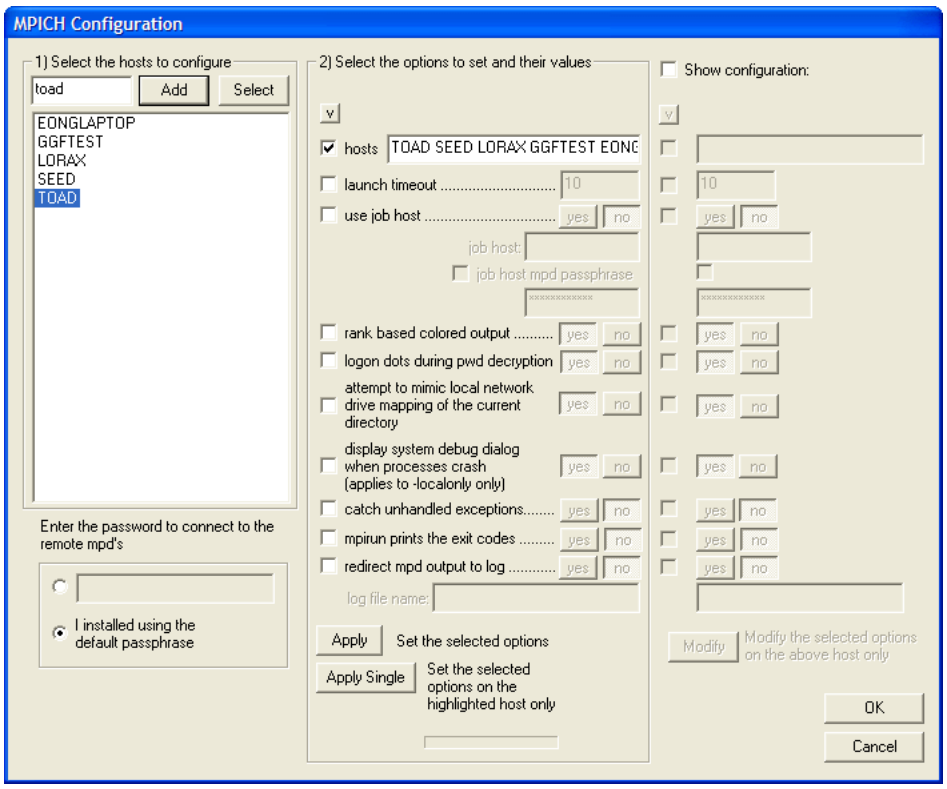

Figure 2: MPICH Configuration tool

- Specify a job host if you want mpirun to send job information to an mpd running on the specified host. With this enabled, you can use the MPIJob tools to view where jobs are running, what state they are in, and you can kill errant jobs. Since the job database is stored in the mpd running on the specified host, the host you input here must have an mpd installed and running on it.

- The rank based colored output checkbox enables/disables colorized output. There are 32 colors that are given out to the ranks in round robin fashion.

- The logon dots check is for mpirun to display .... while decrypting passwords. It is essentially a progress indicator to tell the user that mpirun has not hung.

- The mimic local drive mapping is for when mpirun is executed from a network mapped drive. With this option enabled, mpirun tries to make the same mapping on the remote hosts where it launches processes. For example you might have the Z: drive mapped to \\myserver\myhome. If you execute mpirun from the Z:\ directory, mpirun will attempt to map Z: to \\myserver\myhome on the remote machines before launching processes.

- The 'display system dialog ...' check only applies to jobs run on a single machine with the -localonly option to mpirun. Disable this option to prevent the system from showing dialog boxes when processes crash. This is useful if you have a script running jobs and you don't want it to hang waiting for an error dialog box to be closed.

- The 'catch unhandled exceptions' option causes mpd to watch each process as it is running and report if an unhandled exception has caused a process to exit. Normally, processes simply exit when an unhandled exception occurs.
- The 'mpirun prints the exitcodes' option causes mpirun to print the exit code of each process as it exits. Lines like:

  ```
  [rank 2 exit code: 0]
  ```

  will be inserted in the standard output of mpirun.
- The 'redirect mpd output to log' causes mpd to redirect all of its internal output messages to the log file specified. The log file must reside on the local hard drive. This file can get large quickly if you run a lot of jobs.

5.3.3 Apply the settings

- Apply button:

  All the hosts are contacted and the selected settings are set.
- Apply single:

  The selected host from the host list is contacted and the selected settings are set. This option would be useful to set an option on a single node without affecting the settings on the rest of the nodes.
- Password

  If you installed mpd manually and set the passphrase to something other than the default, enter it here.
- Show host configuration:

  Check this box to show the configuration of the currently selected host.
- Modify:

  You can change individual options on the currently selected host by checking the option you want to change and then hit the modify button.

5.4 Update tool

This tool allows an administrator to update mpds and the mpich dlls on a cluster of machines. See Figure 3 for a snapshot of the tool.

5.4.1 Create the host list

First create the list of hosts to update by adding them to the host list using the Add and Select buttons.

- Add: Adds the host name from the edit box to the list
- Select: Bring up a dialog to select host names from the network.

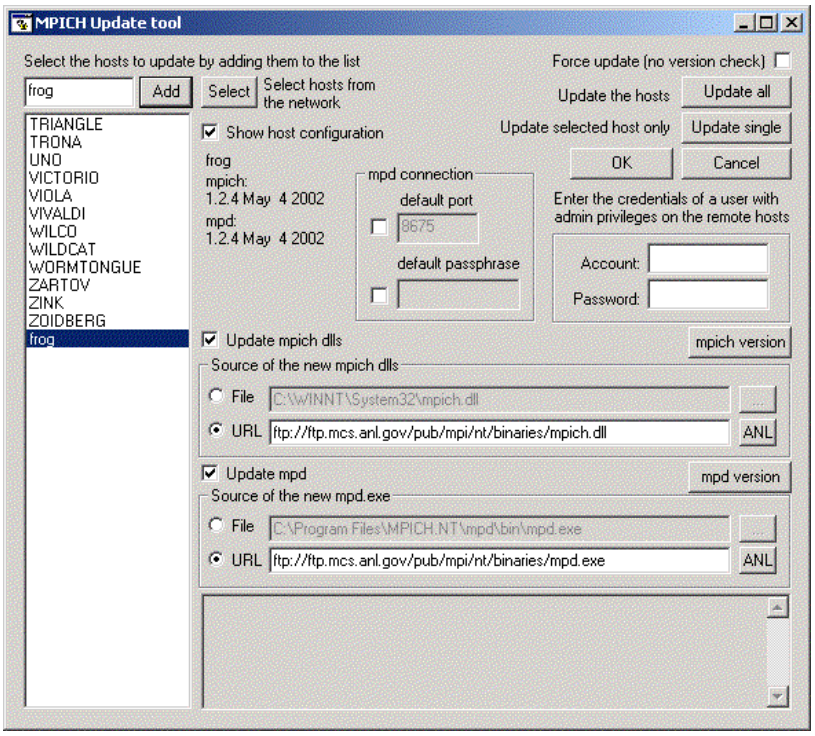

Figure 3: MPICH Update tool

5.4.2 Select the new binaries

Select the checkboxes you want to update. You can update the mpds, the mpich dlls, or both. Then select the source of the new versions.

- File

  Select this option if the new file you want use as the source for updating is located on your file system

- URL

  Select this option if the new file you want use as the source for updating is located on a web site or an ftp site. The ANL button will set the field to Argonne's ftp site where the most current version will be posted.

  Warning: When you select this option, the downloaded file may be stored in the web browser cache. You may need to empty your browser cache to force it to download the latest file.

5.4.3 Apply the updates

Enter the connection information and then choose one of the update options to apply the updates.

1. Enter the connection information

MPD Port

If mpd was installed with a port other than the default, enter it here.

MPD Password

If you installed mpd manually and set the passphrase to something other than the default, enter it here.

Account and Password:

Enter the credentials here of a user who can stop and start the mpd service on each machine (ie an adminstrator)

2. Update the hosts

   Update:

   Update the files on all the hosts in the list.

   Update single:

   Update the files on the selected host only.

   Force update:

   Check this box to suppress version checking when updating the files. With this box unchecked, only files with versions older than the new one will be updated.

   Show host configuration:

   Check this box to show the version of the currently selected host.

5.5 Runtime environment variable options

There are some lesser used options available to MPICH which can be used to fine tune the performance to a specific machine. The following environment variables may be used to set runtime options:

`MPICH_NETMASK`
: Set this variable to the subnet ip address and mask you want to use to select the appropriate network adaptor on multi-nic hosts. The form is: IP/Mask, eg 192.0.0.0/8. The subnet ip address is the partial ip address specifying which bits represent the subnet part of an ip address. The mask part represents which bits are signifigant to determine the subnet. For example, 20 = 255.255.240.0

`MPICH_USE_POLLING`
: Set this variable to 1 to enable polling. The default is to use event objects to wait on. Polling has lower latency but burns the CPU and can decrease performance in certain situations.

`MPICH_SINGLETHREAD`
: Set this variable to 1 to cause the shared memory and via devices to be single threaded. Single threaded devices have much lower latency but they obey different progress rules than the multithreaded versions. They only make progress on messge passing when MPI calls are made. The multi-threaded devices make progress on messages asynchronously when the message arives.

MPICH_SHMQSIZE
: This value is the size, in bytes, of the shared memory queue for each process. The default value is 1MB.

MPICH_MAXSHMMSG
: This value is the largest message, measured in bytes, that can be put in the shared memory queue. This value must be less than or equal to MPICH_SHMQSIZE. Messages larger than this value are copied directly from the address space of the sender to the receiver. The default value is 15k.

MPICH_LONGVLONGTHRESH

MPICH_TCPLONGVLONGTHRESH

MPICH_SHMLONGVLONGTHRESH
: This value is the message size, in bytes, when the message sending protocol changes from eager to rendezvous. The default value is for shm is 20k and tcp is 100k. Use the first variable to set both thresholds or use the protocol specific version to set an individual threshold.

MPICH_NUMCOMMPORTS
: This value is the number of completion port threads that will be launched by each process to handle all socket communication. The default is 2.

MPICH_VI_USE_POLLING
: Set this variable to 1 to enable polling in the VIA device. The default is to use the wait interface. Polling has lower latency but burns the CPU and can decrease performance in certain situations.

MPICH_VERBOSE
: Set this variable to 1 to cause MPICH to spit out loads of internal state as the application runs. This would be useful to report suspected bugs in MPICH.

Examples:

High performance ping pong shared memory test:

```
mpirun -localonly 2
  -env "MPICH_USE_POLLING=1|MPICH_SINGLETHREAD=1"
  netpipe.exe
```

High performance ping pong via test

```
mpirun -np 2
  -env "MPICH_USE_POLLING=1|MPICH_SINGLETHREAD=1|MPICH_VI_CLIQUES=*"
  netpipe.exe
```

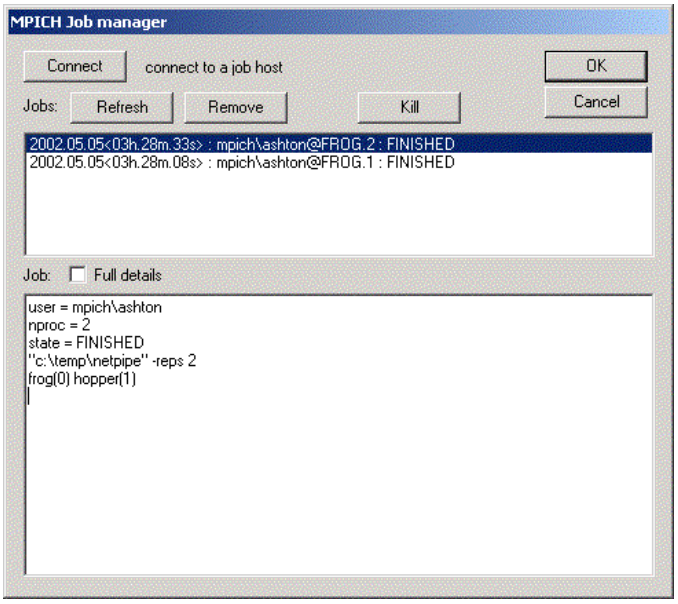

Figure 4: Snapshot of the MPIJob tool

5.6 Job manager tool

The job manager tool allows a user to see the jobs that have been or are currently running (if job logging has been enabled using the configure tool). This is an in memory database that is lost if the machine is rebooted or the mpd that hosts the data is restarted. There is a gui and command line version of the tool.

Connect:
Connect to the job host.

Refresh:
Read the jobs on the job host.

Remove:
Delete the currently selected job. This does not kill the processes, it only removes the data from the mpd job database.

Kill:
Connect to the hosts where the processes are running and kill all the processes.

Jobs window
This window shows the list of jobs. Jobs are listed in the format "timestamp : user@jobid : state" The timestamp is in the format, "year.month.day<hourh.minutem.seconds>"

Job window
This window shows the details of the selected job.

Full details
With this box checked, the complete information of all the processes launched in a job is displayed.

The command line version of mpijob is used this way:

```
mpijob -jobs [jobhost]
mpijob jobid [-full] [jobhost]
mpijob -killjob jobid [jobhost]
mpijob -clear [all, before timestamp, or jobid] [jobhost]
mpdjob -tofile filename [all, before timestamp, or jobid] [jobhost]

timestamp = yyyy.mm.dd<hh.mm.ss>
```

5.7 MPD process launcher

MPD is a process manager for clusters of computers running WindowsNT/2000/XP. It can run in three different modes:

- A service that launches processes in the context of multiple connecting users. This is the default installation.
- A service that launches processes in the context of a single user.
- A command line program started manually on all the nodes. This can be useful for evaluation purposes or for users who do not have the ability to install services on their machines. This option acts like single user mode.

5.7.1 Quick reference

NOTE: The binary distribution of mpich.nt, '`mpich.nt.1.2.4.exe`', installs mpd for you using the setup program. You can use Add/Remove programs to remove it. If you want to install mpd by hand, use the following information:

Default Installation - multi-user

1. Copy '`mpd.exe`' to all the nodes.
2. Logon to each node with an account that has Administrator privileges.
3. Execute `mpd -install` from a command prompt on each node.
4. From a single node, run '`mpiconfig.exe`'
   (a) Click Select to find the hosts where you installed mpd.
   (b) Click Add to add these hosts to the list. If you can't see all the hosts where you installed mpd, add them manually with the edit box above the list.
   (c) Click Set to set the global options on each machine.
5. Compile a sample application like cpi from the `examples\nt` directory.
6. Copy '`cpi.exe`' to all the nodes or place it in a shared directory.

7. Run '`mpirun -np 4 cpi`'

Single user Installation - all jobs run in the security context of a specified user.

1. Copy '`mpd.exe`' to all the nodes.
2. Logon to each node with an account that has Administrator privileges.
3. Execute `mpd -install -account` *`domain\username`* `-getphrase` on each node. Input the password of the user and a passphrase for the mpds.
4. If this user has administrator privileges, run '`mpiconfig.exe`'
   (a) Click Select to find the hosts where you ran `mpd -install`
   (b) Click Add to add these hosts to the list. If you can't see all the hosts where you installed mpd, add them manually with the edit box above the list.
   (c) Click Set to set the hosts on each machine.
5. Compile a sample application like cpi
6. Copy '`cpi.exe`' to all the nodes or place it in a shared directory.
7. Run '`mpirun -np 4 cpi`'

No rights installation - evaluation, interactive usage

1. Copy '`mpd.exe`' to all the nodes.
2. Execute `mpd -d` on each node.
3. Compile a sample application like cpi
4. Copy '`cpi.exe`' to all the nodes or place it in a shared directory.
5. Create a machine file and fill it with the host names where mpd is running.
6. Run '`mpirun -np 4 -machinefile` *`file`* `cpi`'

Uninstall

1. Execute `mpd -remove` from each node or type "stop" into the window where `mpd -d` is running.
2. Delete all the files.

5.7.2 Command line options

Installation

mpd -install -regserver -interact -phrase x -getphrase -account x -password x -port x -mpduser

- The options -install and -regserver are synonyms to install the service.
- The option -interact allows the service to start applications with access to the desktop window. Don't use this option unless you have to show windows while your application is running.
- The option -phrase x allows you to set the passphrase for mpd authentication. When a remote machine connects to the mpd, this phrase is used to encrypt a challenge response string to authenticate the remote user. The option -getphrase causes the mpd to prompt for the passphrase to be entered. This is useful if you don't want to pass the phrase on the command line. If -phrase x or -getphrase are not specified, the default passphrase is used.
- The -port option allows you to specify a port for the mpd to listen on. This must be the same on all the nodes. If you do not specify this option, the default is used, 8675.
- The options -account x and -password x allow the user to install the service in single user mode. In single user mode, all processes launched are placed in the security context of the specified user no matter who connects to the mpd. The advantage of this mode is that no passwords are needed again after install time. The disadvantage is that any user who knows the mpd passphrase can launch processes in the context of the installed user. If -account x is specified but -password is not, the user will be prompted to enter a password. Accounts should be specified in the form "Domain\User".
- The option -mpduser in conjunction with -install installs the mpd with the ability to accept anonymous launch requests that are launched in the context of a registered mpd user. The option enables the following mpd console commands: setmpduser, clrmpduser, enablempduser and disablempduser.

mpd -remove -unregister -uninstall

The options -remove, -unregister and -uninstall are all synonyms to un-install the service.

mpd -d -startalone -port x -phrase x -getphrase

The -d option allows a user to run an mpd from the command line. This is for debugging the mpd or for users who need to execute the mpd manually. The option -startalone tells the mpd to ignore the installed settings and not attempt to connect to a ring of mpd's. The options -port x and -phrase x or -getphrase allow the user to specify what port to listen on and what passphrase to use. The option -getphrase will cause the mpd to prompt for a passphrase. If these options are not specified, mpd will use the default port and passphrase. While the mpd is running it will spit out information as messages pass through and commands are issued. You can enter "stop" to stop the mpd or "quit" to forcibly exit the process.

mpd -update -mpd x -host x -hostfile x -account x -password x -singleuser -port x -phrase x -getphrase

Before you upgrade a set of running mpds, make sure there are not any jobs running. The upgrade process will terminate any running processes managed by the mpds. The -update option is used to upgrade running mpds after a new version has been downloaded. Specify the location of the new mpd.exe with the -mpd x option where x is the full path to the newly downloaded mpd.exe including the executable name (mpd.exe). If you do not specify this option, the mpd will use itself as the upgrade module. So if you do not specify -mpd, make sure you execute the new mpd and not the old mpd. Specify the host to upgrade with the -host x option or a list of hosts to upgrade in a file with the -hostfile x option. You need to specify a user with enough privileges to be able to stop and start services on the specified hosts with the -account x and -password x options. If you do not provide a password on the command line, you will be prompted to enter one. The options -port x and -phrase x or -getphrase allow you to specify what port to connect to and what passphrase to use. If you specify the option -getphrase, you will be prompted to enter a passphrase. If these options are not specified, the default port and passphrase will be used. If you do not specify any options, you will be prompted to enter a host, account, and password. You cannot upgrade mpds running from a command prompt: mpd -d. If you installed the mpds in single user mode, the user must have administrator privileges in order to upgrade this way. Use the -singleuser option instead of an account and password in this case because single user mode does not require a user logon. You can always upgrade manually by doing the following on each node: 1) Execute `mpd -stop` to stop the running mpd. 2) Delete the old mpd.exe and place the new version in its place - the exact same place. The name and path of mpd.exe cannot change. 3) Execute `mpd -start` to start the new mpd. If you want to move mpd.exe to a new location you will have to uninstall (mpd -r emove) and then re-install (mpd -install ...).

Session

mpd -console -port x -phrase x -getphrase

mpd -console host -port x -phrase x -getphrase

The -console option creates a console session with an mpd on the local host or the host specified. If the mpd is listening on a port other than the default, the option -port x can be used to specify what port to connect to. If -phrase x or -getphrase are not specified the mpd searches for the passphrase specified at install time and then reverts to the default passphrase if mpd is not installed on the local host.

Management

mpd -start

The option -start starts an installed mpd on the local host. Note: the mpd is started automatically at installation time.

mpd -stop

The option -stop stops an installed mpd on the local host. Note: when the mpd stops, it will kill all running processes that it has launched.

mpd -restart
The option -restart stops and restarts the mpd on the local host.

mpd -restart host
The option -restart host connects to 'host' and restarts the mpd on that host.

mpd -clean
The option -clean removes all the settings from the registry. The next time the mpd is started it will revert to the default values.

Information

mpd -v -version
The options -v and -version will both print out version information.

mpd -h -? -help
The options -h -? or -help will print out a list of the most common command line options.

5.7.3 Console commands

This section describes all the commands that can be issued to a mpd in a console session. Console sessions are established by executing '`mpd -console`' or '`mpd -console` *`host`*' as described in the previous section.

DATABASE OPERATIONS

A ring of mpds can maintain a set of in-memory databases. The databases store key/value pairs of strings. The data is distributed around the ring. Puts occur locally while gets travel around the ring. The data is not persistent, so when a ring is taken down, all the databases are destroyed. Mpd is not a database application. This capability is provided so parallel applications can pass small amounts of data between processes. It is not intended to hold user data.

dbcreate
This command creates a database and returns the name.

Return values:

The name of the database or `DBS_FAIL` if an error occurred.

dbcreate name or name=x
This command creates a database with the specified name. DBS_SUCCESS is returned if the database is created or if it already exists.

Return values:

`DBS_SUCCESS` or `DBS_FAIL`

dbdestroy name or name=x
This command removes an entire database.

Return values:

`DBS_SUCCESS` or `DBS_FAIL`

dbput name:key:value or name=x key=x value=x

This command inputs a key/value pair into the specified database. The key values need to be unique. It is not allowed to call dbput with the same key more than once into the same database.

Return values:

`DBS_SUCCESS` or `DBS_FAIL`

dbget name:key or name=x key=x

This command retrieves the value of the key in the specified database.

Return values:

The value of the key in the specified database or `DBS_FAIL`

dbdelete name:key or name=x key=x

This command deletes a key from the specified databse.

Return values:

`DBS_SUCCESS` or `DBS_FAIL`

dbfirst name or name=x

This command starts the iterator on the specified database. It returns the first key/value pair in the database or `DBS_END` if the database is empty.

Return values:

`key=value`, or `DBS_END` or `DBS_FAIL`

dbnext name or name=x

This command returns the next key/value pair in the specified database. Repeat this command until it returns `DBS_END` to iterate through the entire database. You must call `dbfirst` before this command to start the iteration.

Return values:

`key=value`, `DBS_END`, or `DBS_FAIL`

dbfirstdb

This command starts the iterator on the entire database namespace. It returns the name of the first database in the space or `DBS_END` if there are no existing databases.

Return values:

`name=name` or `DBS_END`

dbnextdb

This command returns the name of the next database in the namespace. Repeat this command until it returns `DBS_END` to iterate through the names of all available databases. You must call `dbfirstdb` before this command to start the iteration.

Return values:

`name=name` or `DBS_END`

PROCESS OPERATIONS

launch h=host c=cmd e=env m=map d=dir a=account p=password 0=stdin 1=stdout 2=

With the default installation, the mpds run as services on each of the nodes. When a launch command reaches the requested node, the mpd uses the account and password parameters to launch the requested process in the security context of that user. If the mpd has been started in single user mode, then the mpd runs in the security context of a single user. The mpd runs in single user mode if it was installed with a specific username and password or if it is run from a command prompt, `mpd -d ...`. When in single user mode all processes are launched in the security context of the same user. There is no need to pass the account and password. If they are provided they are ignored.

- h=host

  the hostname to launch the process on. If this option is not specified the process is launched on the local host.

- c=cmd

  the path to the executable plus any arguments. For example: c=c:\my\favorite\path\myapp.ex arg1 arg2 or c=\\somehost\someshare\some\path\someapp.exe arg1 arg2.

- e=env

  a string of environment variables to set. Single quote this list and separate values by the vertical bar character. Example: e='var1=val1—var2=val2—var3=val3'

- m=map

  network drive mapping option in the form: 'drive:\\host\share'. Multiple mappings can be specified separated by semicolons.
  eg. m=y:\\myhost\myfiles;z:\\myhost\myhome)

- d=dir

  the working directory to launch the process in

- a=account p=password

  the account and password used to set the security context for the launched process. If the mpd is in single user mode, these parameters are not necessary and they are ignored.

- 0=stdin 1=stdout 2=stderr 12=stdouterr 012=stdinouterr k=rank

  these options specify where to connect the standard input, output, and error of the launched process. The format is host:port. For example, 012=somehost:1234, would connect to the host "somehost" on port 1234 three times to redirect standard input, output and error. When connecting to this host, the mpd sends a five byte message first. The first byte is a 0, 1, or a 2 to signify stdin, stdout or stderr. The next 4 bytes are the integer specified by the k=rank option. If no k option is specified a value of zero is sent by default.

Return values:

launchid

geterror launchid

This command returns the current error message for the launch command corresponding to the provided launchid. ERROR_SUCCESS signifies that the launch was successful. If the launch command is still in progress, the return value is LAUNCH_PENDING. If there is an error associated with the launch, the specific error message will be returned.

Return values:

`ERROR_SUCCESS`, `LAUNCH_PENDING`, "specific error message"

getpid launchid

This command blocks until the process associated with launchid has been started. If there was an error in the startup of the process, this command will return -1 and geterror can be used to get the specific error message.

Return values:

process id or -1

getexitcode launchid

This command returns the exit code of the process associated with launchid. If the process is still running, ACTIVE is returned. If there was a error launching the process, FAIL will be returned and geterror can be called to retrieve the error message.

Return values:

exitcode, `ACTIVE`, `FAIL`

getexitcodewait launchid

This command blocks until the process associated with launchid exits. If there is an error and the process was not launched, FAIL will be returned and geterror can be called to retrieve the error message.

Return values:

exitcode, `FAIL`

freeprocess launchid

This command frees the local structures used to store the process id, exit code and state. freeprocess should be called after there is no need to get any further information about a process, usually after a successful call to one of the getexitcode commands. After this command, the launchid becomes invalid and cannot be used in any other calls.

Return values:

nothing

kill launchid

This command kills the process associated with launchid. `kill launchid` will only work if there is a valid process id in the local launchid structure. If the state is LAUNCH_PENDING or an error has occurred, kill will not succeed. This is important because if a launch command is issued and immediately followed by a kill command, the kill will not succeed if the state of the process is LAUNCH_PENDING. To guarantee that the process has started before trying to kill it you should call getpid first.

Return values:

nothing

kill host=x pid=x

This command attempts to kill the process on host x associated with pid x. kill can only kill processes launched by the mpd.

Return values:

nothing

killall

This command travels around the ring and attempts to kill all the processes launched by the mpds.

Return values:

nothing

ps

This command returns a list of the active processes started by all the mpds in the ring.

Return values:

pid:command line...

RING OPERATIONS

The ring operations manage a ring of mpds. By default, an mpd is installed on a node in a single ring with itself.

hosts

This command travels around the ring of mpds and gets the name of each host.

Return value:

hostA,hostB,hostC,...

extract

This command removes the current host from the ring and forms a new ring with itself.

Return values:

nothing

insert *host*

This command merges the ring that the console session is attached to with the ring that *host* belongs to. If the current host and the specified host belong to the same ring, this command causes the ring to split into two rings.

Return values:

nothing

set nodes

This command marches around the ring of mpds and lets everyone know who their neighbors are. Then when the mpds shutdown and startup again, they try to connect to their neighbors. This way the ring always tries to remain intact.

Return values:

nothing

set key=value

This command sets key/value pairs in the mpd section of the registry. For example, `set temp=c:\temp` sets the temp directory for mpd files on all the nodes.

Return values:

nothing

shutdown

This command stops the mpd and closes the console connection. Do not use this command unless you want to stop the mpd. You will have to run `mpd -start` to get the mpd to start again. Use `done` to close a console session.

Return values:

nothing

restart

This command restarts the mpd and closes the console connection. You will have to reconnect to the mpd. Use `done` to close a console session.

Return values:

```
Restarting mpd...
```

exitall

This command shuts down all the mpds in the ring and closes the console connection. Do not use this command unless you want to stop the mpds. After this command completes you will have to go to each node and start the mpds again (mpd -start). Use `done` to close a console session.

Return values:

nothing

done or quit

This command closes the current console session with the mpd.

Return values:

nothing

LOCAL OPERATIONS

The local operations only act on the mpd and host that the console is connected to. This is unlike the ring operations above that act on all the hosts and mpds in the ring.

lset key=value

This command sets the key/value pair in the mpd section of the registry on the host local to the mpd only. The command does not traverse the ring like the `set key=value` command does.

Return values:

nothing

lget key

This command gets the value of the key provided from the mpd section of the registry on the host local to the mpd.

Return values:

value

ldelete key

This command removes the specified key from the mpd section of the registry on the host local to the mpd.

Return values:

nothing

version

This command returns the version numbers and date of the mpd the console session is connected to. For example: 1.2.3 Mar 2 2002

Return values:

release.major.minor date

mpich version

This command returns the version numbers and date of the mpich dll on the host the console session is connected to. For example: 1.2.4 Apr 12 2002

Return values:

release.major.minor date

config

This command returns all the mpd registry key/value pairs on the host local to the mpd.

Return values:

key=value...

print

This command spits out a lot of internal state useful only for debugging mpd.

Return values:

internal state

stat *param*

This command reports internal information on the specified parameter where *param* can be one of the following:

- ps - running processes, command line, environment variables, working directory, mpich rank, io redirection
- launch - launch structures, id, process id, process state
- config - mpd registry settings
- context - open contexts, contexts are socket connections to the mpd - both internal and external
- tmp - temporary files
- barrier - outstanding barriers
- forwarders - forwarders open on this node, input port and output host:port
- cached - cached user handles

Return values:

internal state

setdbgoutput *filename*

This command redirects the output of the running mpd to a file. This is useful for logging all the commands and operations that mpd processes and should only be used for debugging mpd.

Return values:

SUCCESS, FAIL

canceldbgoutput

This command cancels the redirection of mpd output.

Return values:

SUCCESS, FAIL

setmpduser a=*account* p=*password*

This command sets the mpd user account and password for anonymous launch requests. If you don't supply either of the two parameters you will be prompted to enter them.

Return values:

SUCCESS, FAIL - error msg

clrmpduser

This command removes the mpd user credentials and disables anonymous launch requests.

Return values:

SUCCESS, FAIL - error msg

enablempduser

This command enables anonymous launch requests.

Return values:

SUCCES, FAIL - error msg

disablempduser

This command disables anonymous launch requests.

Return values:

SUCCESS, FAIL - error msg

FILE OPERATIONS

The file operations are for moving files between hosts, creating temporary files and one custom function for mpich.nt

fileinit account=x password=x

This command is the first command that must be issued before the other commands can be used. File operations are done under the security context of this user. If the password option is omitted, you will be prompted to input the password.

Return values:

nothing

putfile local=fullfilename remote=fullfilename replace=yes/no createdir=yes/no

This command copies the file described by the local option to the location described by the remote option. Both the local and remote options must specify complete paths including file names. The replace and createdir options refer to the remote file. replace=yes overwrites the remote file if it exists. createdir=yes causes the path described by the remote option to be created if it doesn't exist. If replace and createdir are not specified, the defaults are replace=yes and createdir=yes.

Return values:

`SUCCESS` or "error message"

getfile remote=fullfilename local=fullfilename replace=yes/no createdir=yes/no

This command copies the file described by the remote option to the location described by the local option. Both the remote and local options must specify complete paths including file names. The replace and createdir options refer to the local file. replace=yes overwrites the local file if it exists. createdir=yes causes the path described by the local option to be created if it doesn't exist. If replace and createdir are not specified, the default values are replace=yes createdir=no.

Return values:

`SUCCESS` or "error message"

getdir path=fullpath

This command returns the names of the folders and files found in the directory on the remote host described by the path option. The path must be a full path. The file names are returned preceded by their file size like this: 1022 cathy.txt

Return values:

folders and sizes and filenames or and error message: `ERROR: error message...`

createtmpfile host=x

This command creates a temporary file on the host specified and returns the file name.

Return values:

filename

deletetmpfile host=x file=x

This command deletes the file described by the file option on the host described by the host option. The file option must specify the complete path to the file.

Return values:

`SUCCESS` or `FAIL`

mpich1readint host=x file=x

This command is a custom function provided to allow mpd to launch mpich 1.2.3 applications and earlier. It reads an integer from the file that was written by the root process during MPI_Init.

Return values:

integer

5.8 `MPICH` and threads

The `MPICH` implementation of MPI is currently not threadsafe. It may, however, be possible to use `MPICH` in a threaded application as long as all `MPICH` calls are made by a single thread.

5.9 Rebuilding the `MPICH` dlls from the source distribution

This section describes how to download and build the mpich dlls and tools from the source distribution.

5.9.1 Download the source distribution

The first step is to download the source distribution of `MPICH` .

The easiest way to get `MPICH` is to use the web page `www.mcs.anl.gov/mpi/mpich/download.html`; you can also use anonymous ftp from `ftp.mcs.anl.gov` in directory '`pub/mpi/nt`'. Get the file '`mpich.nt.1.2.4.src.exe`'.

Execute '`mpich.nt.1.2.4.src.exe`' and choose a directory to unzip the files into. An `MPICH` subdirectory will be created in the directory you choose. The default is C:\Program Files\mpich\mpich. (The extra mpich sub-directory is to prevent the source distribution from colliding with the binary distribution.)

5.9.2 Build

You must have MS Visual C++ 6 and Compaq(Digital) Visual Fortran 6 in order to build without any modifications.

Load the workspace in Visual C++ IDE and select build=>batch build=>rebuild all.

There are several workspaces available:

- **mpich.dsw**

  This project builds a shared memory/via/tcp device. It also contains the MPI-2 file functions provided by Romio for the NTFS file system. There are three targets to the project:

  - Debug/Release builds the C interface and three Fortran interfaces for g77, Visual Fortran, and Intel. The Intel interface has MPI functions but not PMPI functions.
  - Debug/ReleaseNoFortran excludes the Fortran files from the build in case you don't have a Fortran compiler
  - Debug/ReleaseCDECLStrLenEnd builds the C interface and one Fortran interface. Use this project if you need to change the settings to match your Fortran compiler.

  If you have a FORTRAN compiler other than Digital Visual Fortran you will have to make some code changes. You will have to change farg.f to use the corresponding `getarg` and `nargs` calls provided by your compiler and set the USE lines to the appropriate modules.

- mpe**mpe.dsw**

  The mpe workspace contains two projects used for logging mpich applications. The mpe library is used to profile an application. If you link to '`mpe.lib`' before '`mpich.lib`' your application will log all the MPI functions and output a clog file. '`clog2slog.exe`' is a tool that converts the clog file to an slog file format that can be viewed with Jumpshot.

- mpid\\nt_server\\winmpd**mpi2.dsw**

  The mpi2 workspace contains the projects necessary to build the MPD launcher, the configuration tool, the update tool, and other associated libraries.

  mpid\\nt_server\\winmpd\\mpich1**mpich1.dsw**

  The mpich1 workspace contains the projects necessary to build mpirun, mpiregister, and guimpirun. These executables are specific to the mpd launcher.

- mpid\\nt_server\\remoteshell**remoteshell.dsw**

  The RemoteShell workspace contains projects necessary to build the DCOM launcher. Only the Debug and ReleaseMinDependency versions have been tested. The UNICODE versions may not work. The RemoteShellServer is the DCOM launcher. mpirun, mpiconfig and mpiregister are tools that work with the DCOM launcher.

Note:

You must compile the projects in the mpi2 workspace before compiling the mpich workspace because the mpich dlls depend on the mpd libraries.

Note:

In order to compile cleanly you need to use the header files and libraries from the Platform SDK instead of the VC++ distribution. The files that come with VC6 are very old. The platform sdk has the most recent header files. If you do not want to use the platform sdk files, you will have to do the following to get the projects to compile:

1. Adjust the mpich project

   In the mpich project add the following definintion for all configurations: USE_VC6_HEADERS

2. Adjust the mpirun project

   In the mpirun project of the mpich1 workspace remove the definition: WSOCK2_BEFORE_WINDOWS.

You can change these definitions from the project settings dialog: Bring up the project settings (Alt F7). Go to C/C++, Category:[Preprocessor].

With these changes you can compile all the workspaces using the default libraries and header files provided with MSDEV Visual C++ 6.x.

6 Documentation

This distribution of `MPICH` comes with complete `man` pages for the MPI routines. The '`mpich/www`' directory contains HTML versions of the man pages for MPI. All documentation is also available on the web at `www.mcs.anl.gov/mpi/mpich/docs.html`.

Information about MPI is available from a variety of sources. Some of these, particularly WWW pages, include pointers to other resources.

- The Standard itself:
  - As a Technical report [3].
  - As Postscript and HTML at `www.mpi-forum.org`, for both MPI-1 and MPI-2.
  - As a journal article in the Fall 1994 issue of the Journal of Supercomputing Applications [12] for MPI-1 and as a journal article in the International Journal of High Performance Computing Applications in 1998.
- MPI Forum discussions
  - The MPI Forum email discussions and both current and earlier versions of the Standard are available from `www.netlib.org`. MPI-2 discussions are available at `www.mpi-forum.org`.
- Books:
  - *Using MPI: Portable Parallel Programming with the Message-Passing Interface, Second Edition*, by Gropp, Lusk, and Skjellum [8].

- *Using MPI-2: Advanced Features of the Message-Passing Interface*, by Gropp, Lusk, and Thakur [9].
- *MPI—The Complete Reference: Volume 1, The MPI Core*, by Snir, et al. [14].
- *MPI—The Complete Reference: Volume 2, The MPI-2 Extensions*, by Gropp, et al. [4].
- *Parallel Programming with MPI*, by Peter S. Pacheco [13].

- Newsgroup:
  - `comp.parallel.mpi`
- Mailing lists:
  - `mpi-comments@mpi-forum.org`: The MPI Forum discussion list.
  - `mpi-impl@mcs.anl.gov`: The implementors' discussion list.
  - `mpi-bugs@mcs.anl.gov` is the address to which you should report problems with `mpich`.
- Implementations available from the web:
  - `MPICH` is available from `http://www.mcs.anl.gov/mpi/mpich` or by anonymous `ftp` from `ftp.mcs.anl.gov` in the directory '`pub/mpi/mpich`', file '`mpich.tar.gz`'.
  - Links to other implementations are available at `www.mcs.anl.gov/mpi/implementations.html`.
- Test code repository:
  - `ftp://ftp.mcs.anl.gov/pub/mpi/mpi-test`

7 In Case of Trouble

This section describes some of the problems that you may run into and some solutions, along with information on submitting bug reports.

7.1 Things to try first

If something goes wrong, the first thing to do is to read the error message carefully and see if you can do something to fix it. After that, check Section 7.3 for common problems. If you still can't find a solution to your problem, submit a bug report and we will try to help you.

7.2 Submitting bug reports

Send any problem that you can not solve by checking this section to `mpi-bugs@mcs.anl.gov`.

Please include:

- The version of `MPICH` (e.g., 1.2.5)
- The operating system configuration you are running under.
- The output of running your program with the `-mpiversion` argument (e.g., `mpirun -np 1 a.exe -mpiversion`)

If you have more than one problem, please send them in separate messages; this simplifies our handling of problem reports.

The rest of this section contains some information on trouble-shooting `MPICH`. Some of these describe problems that are peculiar to some environments and give suggested work-arounds. Each section is organized in question and answer format, with questions that relate to more than one environment (workstation, operating system, etc.) first, followed by questions that are specific to a particular environment. Problems with workstation clusters are collected together as well. To make it easier to find solutions, the most common problems are described first.

7.3 The Most Common Problems

This section describes some of the most common problems encountered when building and using `MPICH`. See also Section 8 which covers frequently asked questions, including some additional problems.

8 FAQ

8.1 Error 64 - GetQueuedCompletenessStatus failed

Q) Why do I get this error:
GetQueuedCompletenessStatus failed, The specified network name is no longer available.

A) Error 64 is the generic "connection aborted" message for I/O completion ports and is usually the result of another error.

If there is another error in the output, it is probably the real cause of the error.

A common cause of error 64 all by itself is when one process exits while data is still being transmitted. Make sure all your MPI_Isend's and MPI_Irecv's are matched with corresponding MPI_Wait calls. If all your communication is matched up, make sure a process isn't crashing or exiting prematurely.

8.2 No more connections

Q) Why do I get this error:
LaunchProcess failed, CreateProcessAsUser failed, No more connections can be made to this remote computer at this time because there are already as many connections as the computer can accept.

A) This error usually occurs when you try to launch an executable from a shared directory on WindowsNT Workstation, Windows 2000 Professional, or WindowsXP Professional. The professional versions of Windows as apposed to the server editions have limitations on the file sharing capabilities. Place the executable on a network share on a server machine or copy the executable to the local drive of each machine to resolve this problem.

8.3 my windows don't show up

Q) I'm having a problem with mpirun. When I use the command-line interface my application loads fine and works. When I try using a configuration file or use guimpirun, for some reason, my application is unable to create a window.

A)

1. The process launcher for MPICH, mpd, runs as a service. When it launches processes they are put in their own hidden desktop. Any windows these processes bring up are hidden from view. If you must be able to see your windows, you can allow processes to share the default desktop by re-installing mpd with the interact option. Execute `mpd -remove` to uninstall and then execute `mpd -install -interact` to re-install.

   This will not work for a terminal services session. This will only allow windows to show up on the default logon desktop (the monitor directly connected to the host).

   There may be permission issues if a user is logged on to a machine and a different user attempts to launch a process on the same machine. So '-interact' is not the default nor recommended method of installation.

2. But sometimes I can see my windows, even with the default installation. This is true. If mpirun determines that you are only running processes on the local machine, it bypasses mpd and launches the processes in the current context - thus allowing you to see your windows. When mpirun parses a configuration file, it always use mpd. '`guiMPIRun`' always uses mpd.

8.4 mpirun options don't work

Q) Why doesn't the mpirun option do what the help pages say it shoud do?

A) mpirun options must be specified before the name of the executable. Any options specified after the executable will be passed as arguments to the executable and not parsed as mpirun options. For example: `mpirun -np 5 myapp.exe -machinefile filename` will not use the machine file specified by `filename` because mpirun considers this an argument to the application.

8.5 mpirun in a bash shell doesn't work

Q) Why doesn't mpirun work in the cygwin bash shell?

A) The cygwin environment has problems with the Windows API function CreateProcess. A workaround was introduced in mpich.nt.1.2.2 Oct 10, 2001. This and more recent versions

of mpirun function in a bash shell running in a command prompt. MPIRun does not work in the XFree86 windowing environment.

8.6 Does MPICH work on Windows98?

Q) Can I run MPICH applications on Windows9x/ME?

A) In a limited way yes. The TCP/IP device for Windows has code that only runs on WindowsNT/2000/XP, but you can use the `-localonly` option to '`mpirun`' on a Win9x machine (See Section 5.1.3). This means you can run multiple processes on a single Win9x machine but you cannot run applications across multiple Win9x machines. This capability is provided so you can compile and test programs on a single Win9x machine and then run the code on an NT cluster at some other time. To install on a Win9x machine, download the source distribution, unzip the contents, use mpirun from the bin directory and make sure the dlls in the lib directory are in your path. Help files are in the www directory, `www\index.html`.

8.7 Can I run on both Windows and Linux at the same time?

Q) Can I run a single MPICH application on multiple Windows and Unix/Linux boxes?

A) No. You can get the same MPI code to compile under Linux and Windows. But the resulting executables will not be able to used in the same MPI job. The MPICH code for establishing and managing socket connections between hosts is different for unix and Windows and is not compatible.

Appendices

A History of MPICH

MPICH was developed during the MPI standards process to provide feedback to the MPI Forum on implementation and usability issues. With the release of the MPI standard, MPICH was designed to provide an implementation of the MPI standard that could replace the proprietary message-passing systems on the massively parallel computers of the day, such as the Intel Paragon, IBM SP, and TMC CM5. MPICH used an early version of the abstract device interface (ADI), based on the Chameleon [11] portability system, to provide a light-weight implementation layer. To enable development on desktop systems, a device layered on top of the P4 [1] system was used. Over time, other devices were developed; as systems have vanished (e.g., the TMC CM5 and the Ncube), these devices have been dropped from our distribution. Because MPICH used P4 for workstation networks, MPICH has supported both MIMD programming and heterogeneous clusters from the very beginning.

Because MPICH was designed to enable ports to other systems, many parallel computer vendors and research groups have used MPICH as the basis for their implementation. Many users are now familar only with the version of MPICH that uses the ch_p4 device for work-

station and Beowulf clusters. However, MPICH continues to support other systems and continues to serve as a platform for research into MPI implementations.

B File Manifest

This section briefly describes the files and directories at the top level of the MPICH source tree.

COPYRIGHT Copyright statement. This code is free but not public domain. It is copyrighted by the University of Chicago and Mississippi State University.

README Basic information and instructions for configuring.

doc Assorted tools for producing documentation, together with this manual.

examples Directory containing further directories of example MPI programs. Of particular note are basic, with a few small examples to try first, test, with a test suite for exercising MPICH, and perftest, containing benchmarking code.

include The include files, both user and system.

bin Contains the programs, such as mpirun, used to run MPI programs.

man Man pages for MPI, MPE, and internal routines.

mpe The source code for the MPE extensions for logging and X graphics. The contrib directory contains examples. Best are the mandel and mastermind subdirectories. The profiling subdirectory contains the profiling subsystem, including a system for automatically generating the "wrappers" for the MPI profiling interface. MPE also includes the performance visualization programs, such as jumpshot (see Section 4.6.3).

mpid The source code for the various "devices" that customize MPICH for a particular machine, operating system, and environment.

romio The ROMIO parallel I/O system, which includes an implementation of most of the MPI-2 parallel I/O standard.

src The source code for the portable part of MPICH. There are subdirectories for the various parts of the MPI specification.

util Utility programs and files.

www HTML versions of the man pages.

C Automated installation

Instructions to get 'mpich.nt.1.2.4.exe' up and running on a cluster:

1. Download the zipped package - 'mpich.nt.1.2.4.zip'.

2. Don't run setup yet. Unzip the contents to a temporary location in a shared directory.
3. Use notepad to edit '`setup.iss`'.
4. Find the line:

   ```
   szDir=C:\Program Files\MPICH
   ```

5. Change it to whatever directory you want.
6. Find the lines:

   ```
   Component-count=7
   Component-0=runtime dlls
   Component-1=mpd
   Component-2=SDK
   Component-3=Help
   Component-4=SDK.gcc
   Component-5=RemoteShell
   Component-6=Jumpshot
   ```

7. Delete the components you don't want installed and adjust the count and numbers.

   A typical setting for a non-interactive node would be as follows:

   ```
   Component-count=2
   Component-0=runtime dlls
   Component-1=mpd
   ```

   A typical setting for an interactive node would be as follows:

   ```
   Component-count=5
   Component-0=runtime dlls
   Component-1=mpd
   Component-2=SDK
   Component-3=Help
   Component-4=Jumpshot
   ```

8. From each host in the cluster, execute the following command: `\\myhost\myshare\setup -s -f1\\myhost\myshare\setup.iss`

   Here is an example. I unzipped '`mpich.nt.1.2.4.zip`' into the directory c:\temp on a machine called FRY. I edited '`setup.iss`' as explained above and then typed the following from a command prompt on a machine called FRENCH:

   ```
   C:\>\\fry\c$\temp\setup -s -f1\\fry\c$\temp\setup.iss
   ```

   Note: c$ is an administrative share for c: on fry

   Note: There is no space between `-f1` and `\\myhost\...`

9. Finally, delete the files you unzipped from the archive.

D Distribution files

There are several files that can be downloaded to get the mpich distribution. The two major categories are the binary and source distributions. The binary distribution contains the pre-built dlls and libraries ready to link and run applications with. The source distribution contains the same binaries along with all the source code. The source distribution does not contain an installer though.

The binary distribution comes in two forms:

1. '`mpich.nt.1.2.4.exe`'

   This is the preferred download because it contains an installer to copy an install everything automatically.

2. '`mpich.nt.1.2.4.zip`'

   The zipped file contains the same files as the .exe file but you have to unzip the files and run setup.exe yourself.

The source distribution comes in three forms:

1. '`mpich.nt.1.2.4.src.exe`'

   The source distribution contains a self-extractor to copy all the files to the location of your choice. There is no setup program so you will have to read the Section 5.7.1 on how to install the mpd launcher manually.

2. '`mpich.nt.1.2.4.src.zip`'

   This is a zipped only version of the source distribution.

3. '`mpich.nt.1.2.4.tar.gz`'

   This is a gzipped version of the source distribution.

E MSDEV Project settings

Here are the steps to creating a new mpich.nt project with MS Developer Studio 6 after you have installed mpich.nt:

1. Open MS Developer Studio - Visual C++
2. Create a new project with whatever name you want in whatever directory you want. The easiest one is a Win32 console application with no files in it. See figure 5
3. Finish the new project wizard.
4. Go to Project->Settings or hit Alt F7 to bring up the project settings dialog box.
5. Change the settings to use the multithreaded libraries.
   Change the settings for both Debug and Release targets. See figure 6

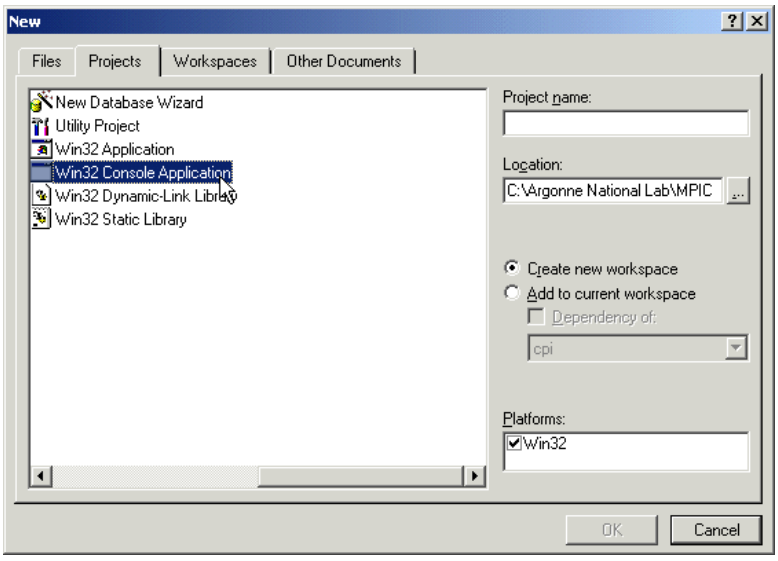

Figure 5: New project

6. Set the include path for all target configurations: This should be Program Files\MPICH\SDK\include. See figure 7

7. Set the lib path for all target configurations: This should be Program Files\MPICH\SDK\lib. See figure 8

8. Add the ws2_32.lib library to all configurations (This is the Microsoft Winsock2 library. It's in your default library path). Add mpich.lib to the release target and mpichd.lib to the debug target. See figure 9

9. Close the project settings dialog box.

10. Add your source files to the project. See figure 10

11. Build

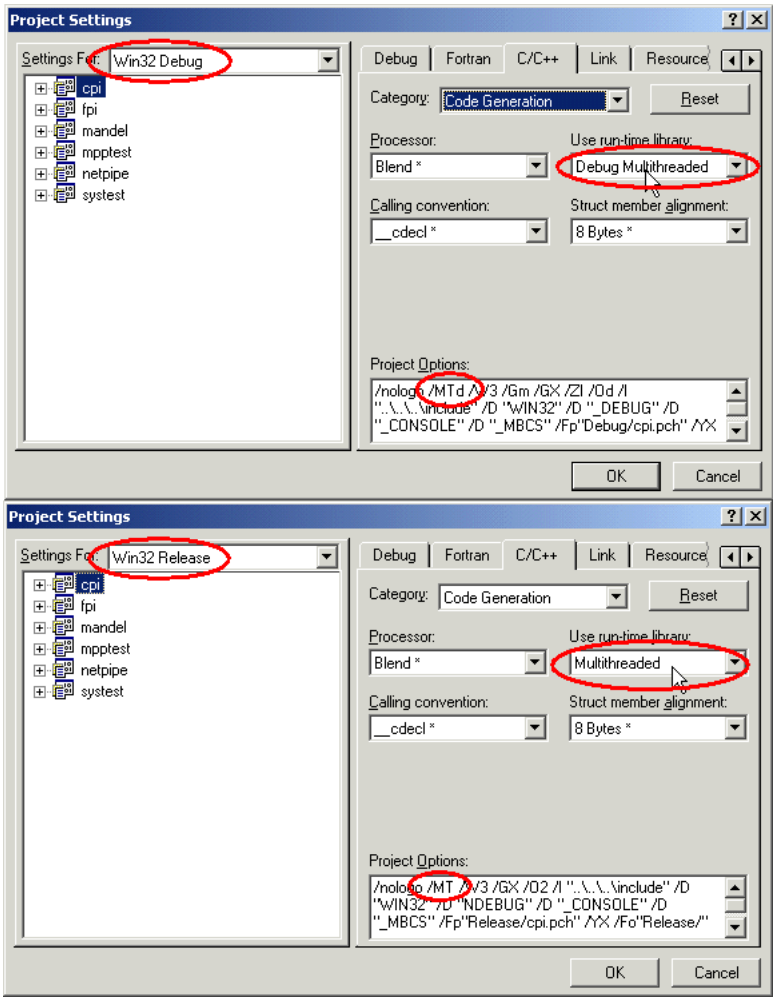

Figure 6: Compiler flags

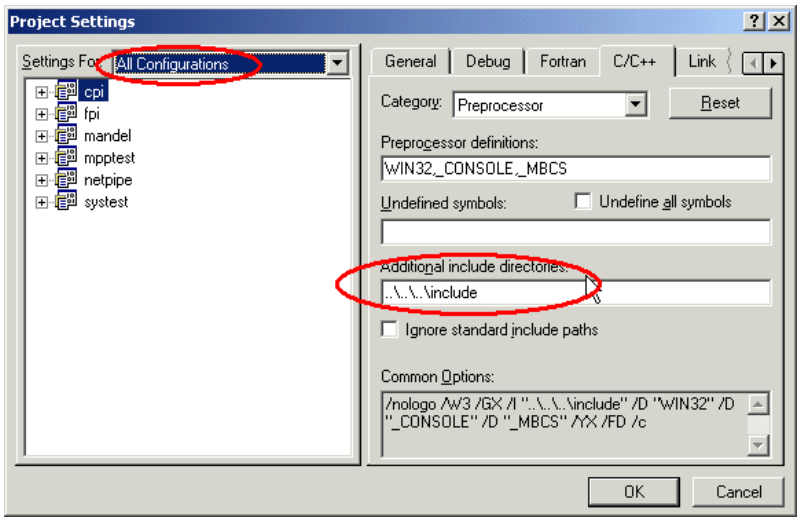

Figure 7: Include path

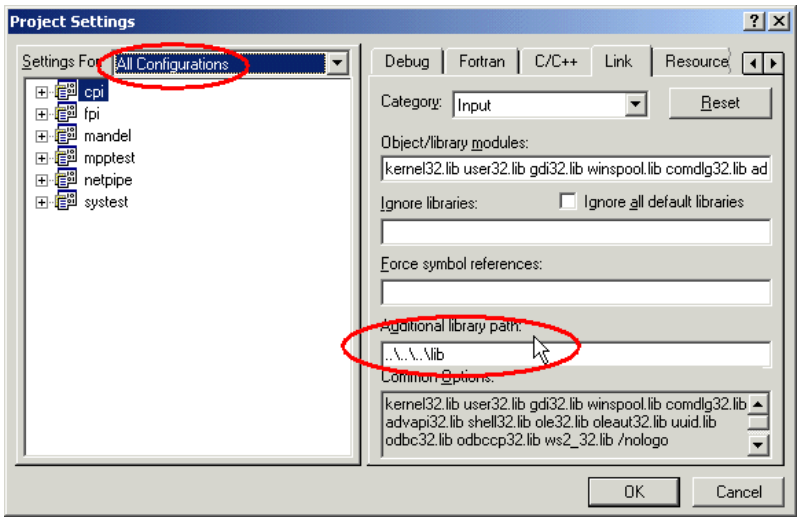

Figure 8: Library path

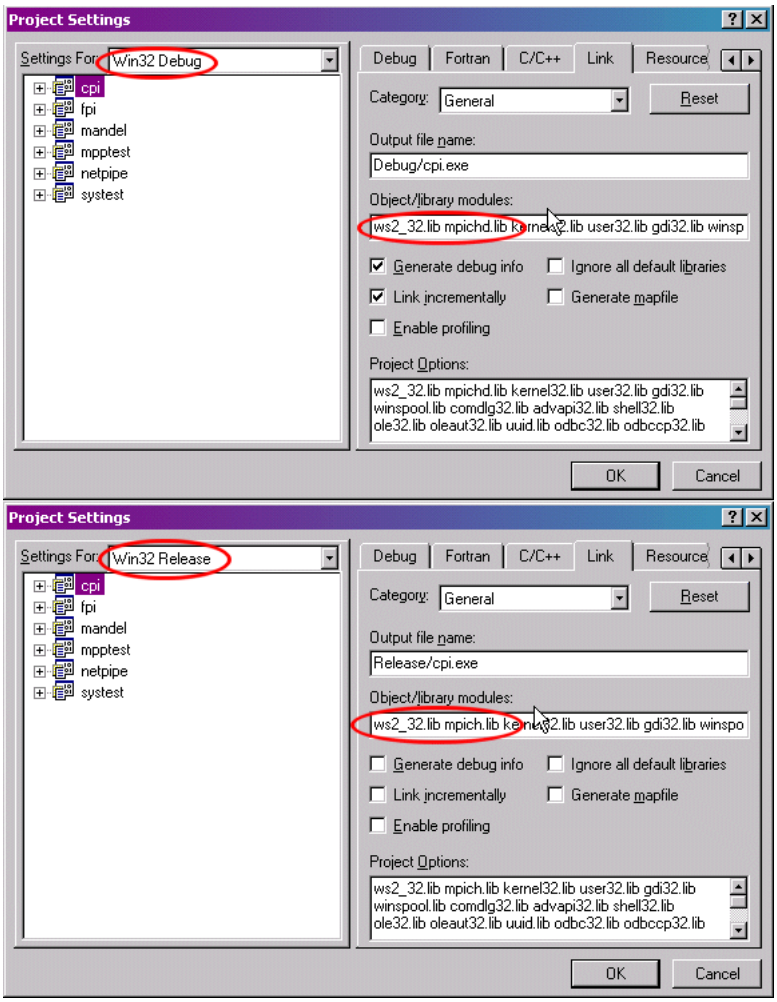

Figure 9: Libraries

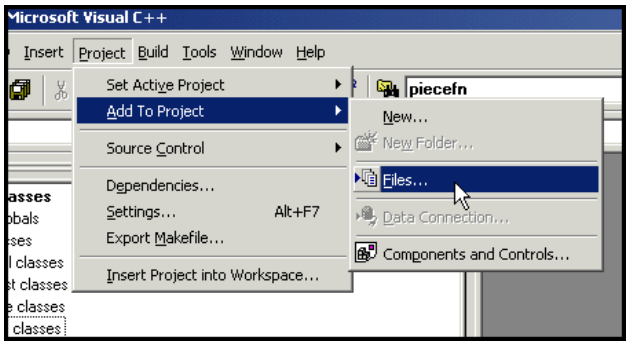

Figure 10: Add source files

References

[1] James Boyle, Ralph Butler, Terrence Disz, Barnett Glickfeld, Ewing Lusk, Ross Overbeek, James Patterson, and Rick Stevens. *Portable Programs for Parallel Processors.* Holt, Rinehart, and Winston, New York, NY, 1987.

[2] Anthony Chan, William Gropp, and Ewing Lusk. User's guide for `mpe` extensions for mpi programs. Technical Report ANL-98/xx, Argonne National Laboratory, 1998. The updated version is at `ftp://ftp.mcs.anl.gov/pub/mpi/mpeman.ps`.

[3] Message Passing Interface Forum. MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, TN, 1994.

[4] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI—The Complete Reference: Volume 2, The MPI-2 Extensions.* MIT Press, Cambridge, MA, 1998.

[5] William Gropp and Ewing Lusk. A high-performance MPI implementation on a shared-memory vector supercomputer. *Parallel Computing*, 22(11):1513–1526, January 1997.

[6] William Gropp and Ewing Lusk. Sowing MPICH: A case study in the dissemination of a portable environment for parallel scientific computing. *IJSA*, 11(2):103–114, Summer 1997.

[7] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI Message-Passing Interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[8] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface,* 2nd edition. MIT Press, Cambridge, MA, 1999.

[9] William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface.* MIT Press, Cambridge, MA, 1999.

[10] William D. Gropp and Ewing Lusk. Reproducible measurements of MPI performance characteristics. In Jack Dongarra, Emilio Luque, and Tomàs Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1697 of *Lecture Notes in Computer Science*, pages 11–18. Springer Verlag, 1999.

[11] William D. Gropp and Barry Smith. Chameleon parallel programming tools users manual. Technical Report ANL-93/23, Argonne National Laboratory, Argonne, IL, March 1993.

[12] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414, 1994.

[13] Peter S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufman, 1997.

[14] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI—The Complete Reference: Volume 1, The MPI Core,* 2nd edition. MIT Press, Cambridge, MA, 1998.