ANL/MCS-TM-ANL-96/6

User's Guide for `mpich`, a Portable Implementation of MPI

by

William Gropp and Ewing Lusk

ARGONNE NATIONAL LABORATORY
UNIVERSITY OF CHICAGO

MATHEMATICS AND
COMPUTER SCIENCE
DIVISION

Contents

This *User's Guide* corresponds to Version 1.1.1 of `mpich`. It was processed by LaTeX on July 6, 1998.

Abstract

MPI (Message-Passing Interface) is a standard specification for message-passing libraries. MPICH is a portable implementation of the full MPI specification for a wide variety of parallel and distributed computing environments. This paper describes how to build and run MPI programs using the MPICH implementation of MPI.

There are a number of major features in Release 1.1.1 of MPICH :

- The Romio subsystem implements a large part of the MPI-2 standard for parallel I/O. For details on what types of file systems runs on and current limitations, see the Romio documentation in `romio/doc`.
- The MPI-2 standard C++ bindings are available for the MPI-1 functions.
- A new device, Globus, is available. See Section 3.2.
- A new program visualization program, called `Jumpshot`, is available as an alternative to the `upshot` and `nupshot` programs.

1 Introduction

MPICH is a freely available implementation of the MPI standard that runs on a wide variety of systems. This document assumes that MPICH has already been installed; if not, you should first read *Installation Guide to MPICH, a Portable Implementation of MPI* [3]. For concreteness, this document assumes that the MPICH implementation is installed into '`/usr/local/mpich`' and that you have added '`/usr/local/mpich/bin`' to your path. If MPICH is installed somewhere else, you should make the appropriate changes.

In addition, you need to know what systems MPICH has been built and installed for. You need to know the *architecture* and the *device*. The architecture indicates the kind of processor and/or OS; examples are `sun4` and `intelnx`. The device indicates how MPICH performs communication between processes and how processes are created; examples are `ch_nx` (for the Intel Paragon, an MPP), `ch_p4` (for workstation clusters), and `globus` (for "computational grids" that include both MPPs and workstations, and that may span multiple sites). The libraries and special commands for each architecture/device pair are provided in the directory '`/usr/local/mpich/lib/<architecture>/<device>`'. For example, the directory for the `sun4` architecture and `ch_p4` device is in '`/usr/local/mpich/lib/sun4/ch_p4`'. This directory should also be in your path (or you should use the full path for some commands; we'll indicate which). (Note that `sun4` is MPICH's name for SunOS before Solaris; `solaris` is used for Solaris also known as SunOS version 5.)

This approach makes it easy to have MPICH available for several different parallel machines. For example, you might have a workstation cluster version and a massively parallel version.

2 Linking and running programs

MPICH provides tools that simplify creating MPI executables. Because MPICH programs may require special libraries and compile options, you should use the commands that MPICH provides for compiling and linking programs. These commands are `mpicc` for C programs and `mpif77` for Fortran programs.

2.1 Scripts to Compile and Link Applications

The MPICH implementation provides four commands for compiling and linking C (`mpicc`), Fortran 77 (`mpif77`), C++ (`mpiCC`), and Fortran 90 (`mpif90`) programs.

You may use these commands *instead of* the '`Makefile.in`' versions, particularly for programs contained in a small number of files. In addition, they have a simple interface to the profiling and visualization libraries described in [7]. In addition, the following special options are supported:

-mpilog Build version that generates MPE log files.

-mpitrace Build version that generates traces.

-mpianim Build version that generates real-time animation.

-show Show the commands that would be used without actually running them.

Use these commands just like the usual C or Fortran compiler. For example,

```
mpicc -c foo.c
mpif77 -c foo.f
```

and

```
mpicc -o foo foo.o
mpif77 -o foo foo.o
```

Similarly for C++ and Fortran 90 programs. Commands for the linker may include additional libraries. For example, to use routines from the C math library library, use

```
mpicc -o foo foo.o -lm
```

Combining compilation and linking in a single command, as shown here,

```
mpicc -o foo foo.c
mpif77 -o foo foo.f
```

may also be used.

These commands are set up for a specific architecture and MPICH device and are located in the directory that contains the MPI libraries. For example, if the architecture is `sun4` and the device is `ch_p4`, these commands may be found in '`/usr/local/mpich/lib/sun4/ch_p4`' (assuming that MPICH is installed in '`/usr/local/mpich`').

2.2 Running with mpirun

To run an MPI program, use the `mpirun` command, which is located in '`/usr/local/mpich/bin`'. For almost all systems, you can use the command

```
mpirun -np 4 a.out
```

to run the program '`a.out`' on four processors. The command `mpirun -help` gives you a complete list of options, which may also be found in Appendix B.

On exit, `mpirun` returns a status of zero unless `mpirun` detected a problem, in which case it returns a non-zero status (currently, all are one, but this may change in the future).

When using the p4 device in workstation clusters, multiple architectures may be handled by giving multiple `-arch` and `-np` arguments. For example, to run a program on 2 sun4s and 3 rs6000s, with the local machine being a sun4, use

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program
```

This assumes that program will run on both architectures. If different executables are needed, the string '%a' will be replaced with the arch name. For example, if the programs are `program.sun4` and `program.rs6000`, then the command is

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program.%a
```

If instead the execuables are in different directories; for example, '`/tmp/me/sun4`' and '`/tmp/me/rs6000`', then the command is

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 /tmp/me/%a/program
```

It is important to specify the architecture with `-arch` *before* specifying the number of processors. Also, the *first* `arch` command must refer to the processor on which the job will be started. Specifically, if `-nolocal` is not specified, then the first `-arch` must refer to the processor from which mpirun is running.

When running on multiple machines using the globus device, `mpirun` is also used, but different techniques are used to control how executables are selected. See Section 3.2 for details.

2.3 More detailed control

For more control over the process of compiling and linking programs for MPICH, you should use a 'Makefile'. Rather than modify your 'Makefile' for each system, you can use a makefile template and use the command 'mpireconfig' to convert the makefile template into a valid 'Makefile'. To do this, start with the file 'Makefile.in' in '/usr/local/mpich/examples'. Modify this 'Makefile.in' for your program and then enter

```
mpireconfig Makefile
```

(not `mpireconfig Makefile.in`). This creates a 'Makefile' from 'Makefile.in'. Then enter:

```
make
```

3 Special features of different systems

MPI makes it relatively easy to write portable parallel programs. However, one thing that MPI does not standardize is the environment within which the parallel program is running. There are three basic types of parallel environments: parallel computers, clusters of workstations, and integrated distributed environments, which we will call "computational grids", that include parallel computers and workstations, and that may span multiple sites. Naturally, a parallel computer (usually) provides an integrated, relatively easy way of running parallel programs. Clusters of workstations and grid environments, on the other hand, usually have no standard way of running a parallel program and will require some additional setup. The MPICH implementation is designed to hide these differences behind the `mpirun` script; however, if you need special features or options or if you are having problems running your programs, you will need to understand the differences between these systems. In the following, we describe the special features that apply for workstation clusters, grids (as supported by the globus device), and certain parallel computers.

3.1 Workstation clusters

Most massively parallel processors (MPPs) provide a way to start a program on a requested number of processors; `mpirun` makes use of the appropriate command whenever possible. In contrast, workstation clusters require that each process in a parallel job be started individually, though programs to help start these processes exist (see 3.1.3 below). Because workstation clusters are not already organized as an MPP, additional information is required to make use of them. MPICH should be installed with a list of participating workstations in the file 'machines.<arch>' in the directory '/usr/local/mpich/util/machines'. This file is used by `mpirun` to choose processors to run on. (Using heterogeneous clusters is discussed below.) The rest of this section discusses some of the details of this process, and how you can check for problems. These instructions apply to only the `ch_p4` device.

3.1.1 Checking your machines list

Use the script '`tstmachines`' in '`/usr/local/mpich/lib/<arch>/<device>`' to ensure that you can use all of the machines that you have listed. This script performs an `rsh` and a short directory listing; this tests that you both have access to the node and that a program in the current directory is visible on the remote node. If there are any problems, they will be listed. These problems *must* be fixed before proceeding.

The only argument to `bin/tstmachines` is the name of the architecture; this is the same name as the extension on the machines file. For example,

```
/usr/local/mpich/bin/tstmachines sun4
```

tests that a program in the current directory can be executed by all of the machines in the `sun4` machines list. This program is silent if all is well; if you want to see what it is doing, use the `-v` (for verbose) argument:

```
/usr/local/mpich/bin/tstmachines -v sun4
```

The output from this command might look like

```
Trying true on host1.uoffoo.edu ...
Trying true on host2.uoffoo.edu ...
Trying ls on host1.uoffoo.edu ...
Trying ls on host2.uoffoo.edu ...
Trying user program on host1.uoffoo.edu ...
Trying user program on host2.uoffoo.edu ...
```

3.1.2 Using the Secure Shell

The *Installation Guide* explains how to set up your environment so that the `ch_p4` device on networks will use the secure shell `ssh` instead of `rsh`. This is useful on networks where for security reasons the use of `rsh` is discouraged.

3.1.3 Using the Secure Server

Because each workstation in a cluster (usually) requires that a new user log into it, and because this process can be very time-consuming, MPICH provides a program that may be used to speed this process. This is the *secure server*, and is located in '`serv_p4`' in the directory '`/usr/local/mpich/lib/<arch>/<device>`' [1]. The script '`chp4_servs`' in the same directory may be used to start '`serv_p4`' on those workstations that you can `rsh` programs on. You can also start the server by hand and allow it to run in the background; this is appropriate on machines that do not accept `rsh` connections but on which you have accounts.

[1] The `globus` device does not use the secure server. It uses a security model implemented using the GSS API. See section 3.2 Security and the `globus` device

Before you start this server, check to see if the secure server has been installed for general use; if so, the same server can be used by everyone. In this mode, root access is required to install the server. If the server has not been installed, then you can install it for your own use without needing any special privileges with

```
chp4_servs -port=1234
```

This starts the secure server on all of the machines listed in the file '/usr/local/mpich/util/machines/machines.<arch>'.

The port number, provided with the option -port=, must be different from any other port in use on the workstations.

To make use of the secure server for the ch_p4 device, add the following definitions to your environment:

```
setenv MPI_USEP4SSPORT yes
setenv MPI_P4SSPORT 1234
```

The value of MPI_P4SSPORT must be the port with which you started the secure server. When these environment variables are set, mpirun attempts to use the secure server to start programs that use the ch_p4 device. (There are command line arguments to mpirun that can be used instead of these environment variables; mpirun -help will give you more information.)

3.1.4 Heterogeneous networks and the ch_p4 device

A heterogeneous network of workstations is one in which the machines connected by the network have different architectures and/or operating systems. For example, a network may contain 3 Sun SPARC (sun4) workstations and 3 SGI IRIX workstations, all of which communicate via the TCP/IP protocol. The mpirun command may be told to use all of these with

```
mpirun -arch sun4 -np 3 -arch IRIX -np 3 program.%a
```

While the ch_p4 device supports communication between workstations in heterogeneous TCP/IP networks, it does not allow the coupling of multiple multicomputers. To support such a configuration, you should use the globus device. See the following section for details.

The special program name program.%a allows you to specify the different executables for the program, since a Sun executable won't run on an SGI workstation and vice versa. The %a is replaced with the architecture name; in this example, program.sun4 runs on the Suns and program.IRIX runs on the SGI IRIX workstations. You can also put the programs into different directories; for example,

```
mpirun -arch sun4 -np 3 -arch IRIX -np 3 /tmp/%a/program
```

For even more control over how jobs get started, we need to look at how mpirun starts a parallel program on a workstation cluster. Each time mpirun runs, it constructs and uses a new file of machine names for just that run, using the machines file as input. (The new file is called PIyyyy, where yyyy is the process identifier.) If you specify -keep_pg on your mpirun invocation, you can use this information to see where mpirun ran your last few jobs. You can construct this file yourself and specify it as an argument to mpirun. To do this for ch_p4, use

```
mpirun -p4pg pgfile myprog
```

where pfile is the name of the file. The file format is defined below.

This is necessary when you want closer control over the hosts you run on, or when mpirun cannot construct it automatically. Such is the case when

- You want to run on a different set of machines than those listed in the machines file.
- You want to run different executables on different hosts (your program is not SPMD).
- You want to run on a heterogeneous network, which requires different executables.
- You want to run all the processes on the same workstation, simulating parallelism by time-sharing one machine.
- You want to run on a network of shared-memory multiprocessors and need to specify the number of processes that will share memory on each machine.

The format of a ch_p4 procgroup file is a set of lines of the form

```
<hostname>  <#procs>  <progname>  [<login>]
```

An example of such a file, where the command is being issued from host sun1, might be

```
sun1   0  /users/jones/myprog
sun2   1  /users/jones/myprog
sun3   1  /users/jones/myprog
hp1    1  /home/mbj/myprog    mbj
```

The above file specifies four processes, one on each of three suns and one on another workstation where the user's account name is different. Note the 0 in the first line. It is there to indicate that no *other* processes are to be started on host sun1 than the one started by the user by his command.

You might want to run all the processes on your own machine, as a test. You can do this by repeating its name in the file:

```
sun1 0 /users/jones/myprog
sun1 1 /users/jones/myprog
sun1 1 /users/jones/myprog
```

This will run three processes on `sun1`, communicating via sockets.

To run on a shared-memory multiprocessor, with 10 processes, you would use a file like:

```
sgimp  9  /u/me/prog
```

Note that this is for 10 processes, one of them started by the user directly, and the other nine specified in this file. This requires that MPICH was configured with the option `-comm=shared`; see the installation manual for more information.

If you are logged into host `gyrfalcon` and want to start a job with one process on `gyrfalcon` and three processes on `alaska`, where the `alaska` processes communicate through shared memory, you would use

```
local    0  /home/jbg/main
alaska   3  /afs/u/graphics
```

3.1.5 Using special switches

In some installations, certain hosts can be connected in multiple ways. For example, the "normal" Ethernet may be supplemented by a high-speed FDDI ring. Usually, alternate host names are used to identify the high-speed connection. All you need to do is put these alternate names in your `machines/machines.xxxx` file. In this case, it is important not to use the form `local 0` but to use the name of the local host. For example, if hosts `host1` and `host2` have ATM connected to `host1-atm` and `host2-atm` respectively, the correct `ch_p4` procgroup file to connect them (running the program '`/home/me/a.out`') is

```
host1-atm 0 /home/me/a.out
host2-atm 1 /home/me/a.out
```

3.2 Computational grids: the `globus` device

The globus device (see `http://www.globus.org/mpi`) supports the execution of MPI programs on "computational grids" that may include parallel computers and workstations, and that may span multiple sites. In such grid environments, different sites may support different security mechanisms and different process creation mechanisms. The globus device hides these low-level details from you, allowing you to start programs with `mpirun` as on MPPs and workstation clusters. The globus device also provides other convenient features, such as remote access to files and executable staging. These features are provided by using services supplied by the Globus toolkit: see `http://www.globus.org` for details.

The Globus device requires that special servers be running on the computers where processes are to be created. In our discussion of how to use the globus device, we assume that we are using the globus device on GUSTO, an international network of resources on which the various Globus servers are already installed and running. If possible, we recommend that you use the globus device in this environment. If you wish to use the globus device in other situations, please send email to `developers@globus.org`. Details of how to run MPI programs using the Globus device on the GUSTO environment are in Appendix C.

3.3 MPPs

Each MPP is slightly different, and even systems from the same vendor may have different ways for running jobs at different installations. The `mpirun` program attempts to adapt to this, but you may find that it does not handle your installation. One step that you can take is to use the `-t` option to `mpirun`. This shows how `mpirun` would start your MPI program without actually doing so. Often, you can use this information, along with the instructions for starting programs at your site, to discover how to start the program. Please let us know (`mpi-bugs@mcs.anl.gov`) about any special needs.

3.3.1 IBM SP

Using `mpirun` with the IBM SP computers can be tricky, because there are so many different (and often mutually exclusive) ways of running programs on them. The `mpirun` distributed with MPICH works on systems using the Argonne scheduler (sometimes called EASY) and with systems using the default resource manager values (i.e., those not requiring the user to choose an `RMPOOL`). If you have trouble running an MPICH program, try following the rules at your installation for running an MPL or POE program (if using the `ch_mpl` device) or for running p4 (if using the `ch_p4` device).

3.3.2 Intel Paragon

Using `mpirun` with an Intel Paragon can be tricky, because there are so many different (and often mutually exclusive) ways of running programs. The `mpirun` distributed with MPICH works with Paragons that provide a default compute partition. There are some options, `-paragon...`, for selecting other forms. For example, `-paragonpn compute1` specifies the pre-existing partition named `compute1` to run on.

3.4 Symmetric Multiprocessors (SMPs)

On many of the shared-memory implementations (device `ch_shmem`, MPICH reserves some shared memory in which messages are transferred back and forth. By default, MPICH reserves roughly four MBytes of shared memory. You can change this with the environment variable `MPI_GLOBMEMSIZE`. For example, to make it 8 MB, enter

```
setenv MPI_GLOBMEMSIZE 8388608
```

Large messages are transfered in pieces, so `MPI_GLOBMEMSIZE` does not limit the maximum message size but increasing it may improve performance.

4 Sample MPI programs

The MPICH distribution contains a variety of sample programs, which are located in the MPICH source tree.

mpich/examples/basic contains a few short programs in Fortran, C, and C++ for testing the simplest features of MPI.

mpich/examples/test contains multiple test directories for the various parts of MPI. Enter "make testing" in this directory to run our suite of function tests.

mpich/examples/test/lederman tests created by Steve Huss-Lederman of SRC. See the README in that directory.

mpich/examples/perftest Performance benchmarking programs. See the script `runmpptest` for information on how to run the benchmarks. These are relatively sophisticated.

mpich/mpe/contrib/mandel A Mandelbrot program that uses the MPE graphics package that comes with mpich. It should work with any other MPI implementation as well, but we have not tested it. This is a good demo program if you have a fast X server and not too many processes.

mpich/mpe/contrib/mastermind A program for solving the Mastermind puzzle in parallel. It can use graphics (`gmm`) or not (`mm`).

Additional examples from the book *Using MPI* [4] are available by anonymous ftp and through the World Wide Web at `ftp://info.mcs.anl.gov/pub/mpi/using/`. At the web site `ftp://info.mcs.anl.gov/pub/mpi` can also be found tutorial material containing other examples.

5 The MPE library of useful extensions

It is anticipated that MPICH will continue to accumulate extension routines of various kinds. Some of them may ultimately become part of an extended MPI Standard. In the meantime, we keep them in a library we call `mpe`, for MultiProcessing Environment. Currently the main components of the `mpe` library are

- A set of routines for creating logfiles for examination by `upshot` or `Jumpshot`.
- A shared-display parallel X graphics library.
- Routines for sequentializing a section of code being executed in parallel.
- Debugger setup routines.

5.1 Creating logfiles

You can create customized logfiles for viewing with `upshot` or `Jumpshot` by calls to the various `mpe` logging routines. For details, see the `mpe` man pages. A profiling library exists that automatically logs all calls to MPI functions. To find out how to link with a profiling library that produces log files automatically, see Section 5.6.

To be added in later editions of this *User's Guide*:

- All `mpe` logging routines
- Format of logfiles
- An example logfile

5.1.1 Parallel X Graphics

The available graphics routines are shown in Table 1. For arguments, see the `man` pages.

	Control Routines
`MPE_Open_graphics`	(collectively) opens an X display
`MPE_Close_graphics`	Closes a X11 graphics device
`MPE_Update`	Updates an X11 display
	Output Routines
`MPE_Draw_point`	Draws a point on an X display
`MPE_Draw_points`	Draws points on an X display
`MPE_Draw_line`	Draws a line on an X11 display
`MPE_Draw_circle`	Draws a circle
`MPE_Fill_rectangle`	Draws a filled rectangle on an X11 display
`MPE_Draw_logic`	Sets logical operation for new pixels
`MPE_Line_thickness`	Sets thickness of lines
`MPE_Make_color_array`	Makes an array of color indices
`MPE_Num_colors`	Gets the number of available colors
`MPE_Add_RGB_color`	Add a new color
	Input Routines
`MPE_Get_mouse_press`	Get current coordinates of the mouse
`MPE_Get_drag_region`	Get a rectangular region

Table 1: MPE graphics routines.

You can find an example of the use of the `mpe` graphics library in the directory `mpich/mpe/contrib/mandel`. Enter

```
make
mpirun -np 4 pmandel
```

to see a parallel Mandelbrot calculation algorithm that exploits several features of the mpe graphics library.

5.1.2 Other mpe routines

Sometimes during the execution of a parallel program, you need to ensure that only a few (often just one) processor at a time is doing something. The routines MPE_Seq_begin and MPE_Seq_end allow you to create a "sequential section" in a parallel program.

The MPI standard makes it easy for users to define the routine to be called when an error is detected by MPI. Often, what you'd like to happen is to have the program start a debugger so that you can diagnose the problem immediately. In some environments, the error handler in MPE_Errors_call_dbx_in_xterm allows you to do just that. In addition, you can compile the mpe library with debugging code included. (See the -mpedbg configure option.)

5.2 Profiling libraries

The MPI profiling interface provides a convenient way for you to add performance analysis tools to any MPI implementation. We demonstrate this mechanism in MPICH, and give you a running start, by supplying three profiling libraries with the MPICH distribution.

5.2.1 Accumulation of time spent in MPI routines

The first profiling library is simple. The profiling version of each MPI_Xxx routine calls PMPI_Wtime (which delivers a time stamp) before and after each call to the corresponding PMPI_Xxx routine. Times are accumulated in each process and written out, one file per process, in the profiling version of MPI_Finalize. The files are then available for use in either a global or process-by-process report. This version does not take into account nested calls, which occur when MPI_Bcast, for instance, is implemented in terms of MPI_Send and MPI_Recv.

5.2.2 Logfile creation

The second profiling library generates *logfiles*, which are files of timestamped events. During execution, calls to MPI_Log_event are made to store events of certain types in memory, and these memory buffers are collected and merged in parallel during MPI_Finalize. During execution, MPI_Pcontrol can be used to suspend and restart logging operations. (By default, logging is on. Invoking MPI_Pcontrol(0) turns it off; MPI_Pcontrol(1) turns it back on again.) You can analyze the logfile produced at the end with a variety of tools.

5.2.3 Real-time animation

The third library does a simple form of real-time program animation. The MPE graphics library contains routines that allow a set of processes to share an X display that is not associated with any one specific process. Our prototype uses this capability to draw arrows that represent message traffic as the program runs.

5.3 upshot

One tool that we use is called `upshot`, which is a derivative of Upshot [6], written in Tcl/Tk. A screen dump of Upshot in use is shown in Figure 1. It shows parallel time lines with

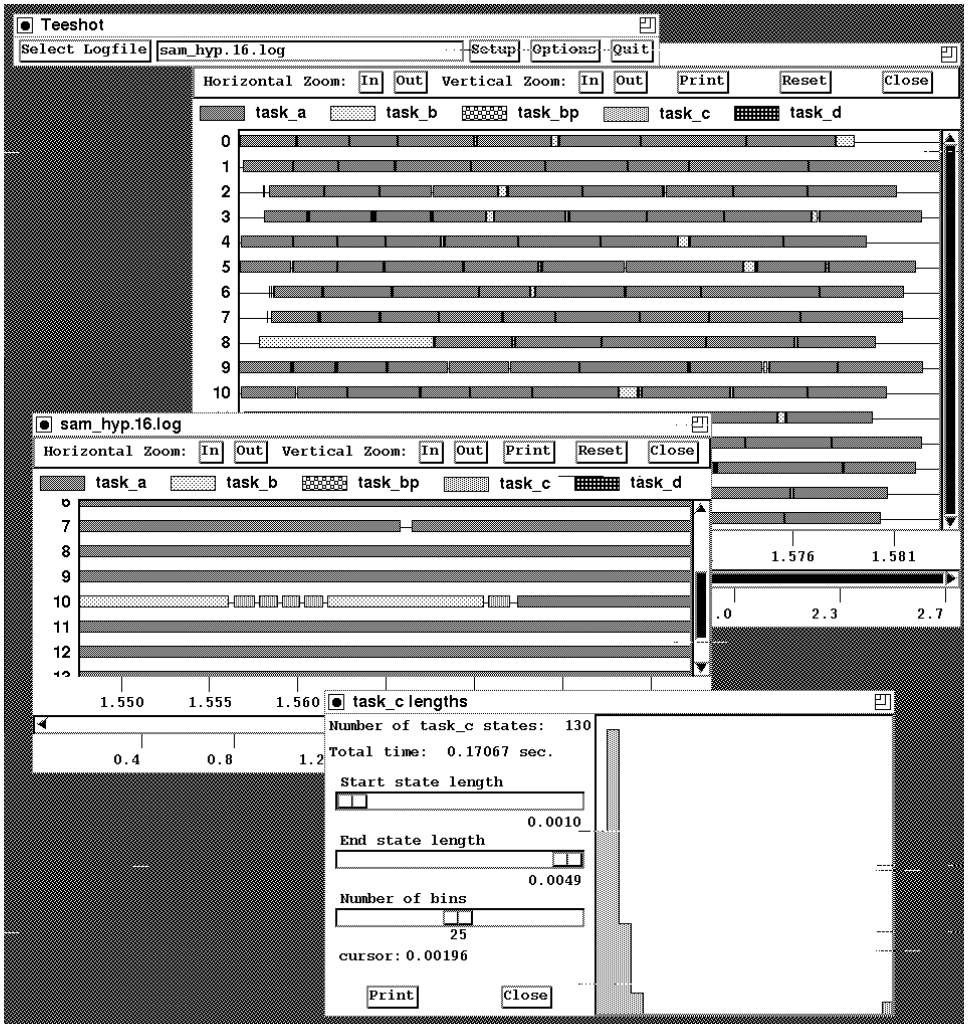

Figure 1: A screendump from `upshot`

process states, like one of the paraGraph [5]. The view can be zoomed in or out, horizontally or vertically, centered on any point in the display chosen with the mouse. In Figure 1, the middle window has resulted from zooming in on the upper window at a chosen point to show more detail. The window at the bottom of the screen show a histogram of state durations, with several adjustable parameters.

5.4 Jumpshot

Another tool that is new and is now available is called `Jumpshot`, which has evolved from Upshot and Nupshot. `Jumpshot` is a graphical visualization tool for interpreting binary clog tracefiles which displays them onto a canvas object, such as the one depicted in Figure 2. It is written in `Java` and a number of useful features hava been added.

1. The arrow display graphically portrays communication between processes which issue send or receive commands. By default, the arrows are turned on. However, once Jumpshot has displayed your canvas object, the arrows can be turned off.

2. The view can be horizontally zoomed in or out, centered on any point chosen by placing the mouse at the desired point and then issuing a `CONTROL Z` or `SHIFT Z`. The zoom factor can also be changed by clicking on the `Configure Zoom` button. Clicking the `Refresh` button will set your canvas object back to its original state.

3. A time marker can be set where your mouse is positioned by issuing a `CONTROL T` or `SHIFT T`. To set the time marker back to its original state, click on the `Options` button which will bring up the `Canvas Options` window. In this window, click on the `Reset Elapsed Time` button.

4. The `States` button will bring up the `Process States` window which will indicate the color of each possible state. Within this window, each state can be turned off or on. For the action to take effect, you must click the OK button. If you click on a state in this window, you will bring up a histogram of the state durations for that state. A variety of controls are available in that window for manipulating the histogram display.

5. The bottom scrollbar can be used to move forward or backward in time.

6. The `Print` button will bring up the `Print Dialog` window where you can print your canvas object along with choosing printer-related options.

7. Clicking and holding down any mouse button on a specific state will bring up the `State Info` window which will display the name of the state, when the state occurred, the length of the state in seconds, and the process number.

On the canvas object, there is an upper and lower scroll bar. The upper scroll bar is visible due to a bug in Java Version 1.1 JDK and should never be used. If the display gets distorted due to the use of upper scroll bar, you should:

1. Press the `Refresh Button`

2. Move lower scrollbar

3. Resize Frame

4. Zoom in or out

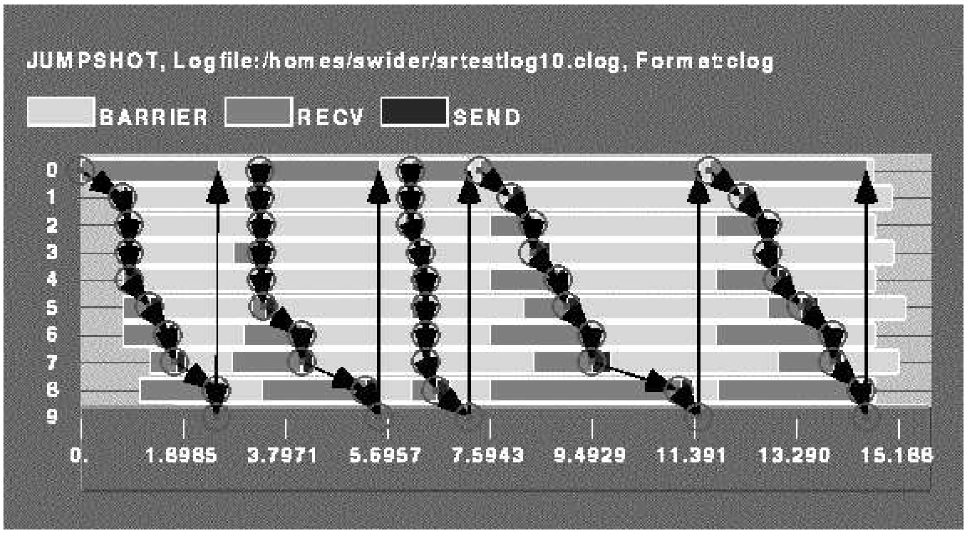

Figure 2: Jumpshot Display

5.5 Accessing the profiling libraries

If the MPE libraries have been built, it is very easy to access the profiling libraries. The easiest way is the use the `mpicc` and `mpif77` commands. If you are using the makefile templates and `mpireconfig` instead, then using the profiling libraries is also simple. The sample makefiles contain the makefile variable `PROFLIB`; by making with different values of this symbol, different profiling effects can be accomplished. In the following examples, we list the libraries that must be *added* to the list of libraries *before* the '`-lmpi`' library.

-ltmpi -lpmpi Trace 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.

-llmpi -lpmpi -lm Generate an upshot-style log file of all MPI calls. The name of the output file is `executablename_profile.log`. For example, if the program is `sendrecv`, the generated log file is `sendrecv_profile.log`.

-lampi -lmpe -lm -lX11 -lpmpi Produce a real-time animation of the program. This requires the MPE graphics, and uses X11 Window System operations. You may need to provide a specific path for the X11 libraries (instead of `-lX11`).

In Fortran, it is necessary to include the library '`-lfmpi`' ahead of the profiling libraries. This allows C routines to be used for implementing the profiling libraries for use by both C and Fortran programs. For example, to generate log files in a Fortran program, the library list is '`-lfmpi -llmpi -lpmpi -lm`'.

5.6 Automatic generation of profiling libraries

For each of these libraries, the process of building the library was very similar. First, profiling versions of `MPI_Init` and `MPI_Finalize` must be written. The profiling versions of the other MPI routines are similar in style. The code in each looks like

```
int MPI_Xxx( . . . )
{
       do something for profiling library
    retcode = PMPI_Xxx( . . . );
       do something else for profiling library
    return retcode;
}
```

We generate these routines by writing the "do something" parts only once, in schematic form, and then wrapping them around the `PMPI_` calls automatically. It is thus easy to generate profiling libraries. See the `README` file in `mpich/mpe/profiling/wrappergen` or Appendix A.

Examples of how to write wrapper templates are located in the `mpe/profiling/lib` subdirectory. There you will find the source code (the `.w` files) for creating the three profiling libraries described above. An example `Makefile` for trying these out is located in the `mpe/profiling/examples` directory.

5.7 Tools for Profiling Library Management

The sample profiling wrappers for MPICH are distributed as wrapper definition code. The wrapper definition code is run through the `wrappergen` utility to generate C code (see Section 5.6. Any number of wrapper definitions can be used together, so any level of profiling wrapper nesting is possible when using wrappergen.

A few sample wrapper definitions are provided with MPICH:

timing Use `MPI_Wtime()` to keep track of the total number of calls to each MPI function, and the time spent within that function. This simply checks the timer before and after the function call. It does not subtract time spent in calls to other functions.

logging Create logfile of all pt2pt function calls.

vismess Pop up an X window that gives a simple visualization of all messages that are passed.

allprof All of the above. This shows how several profiling libraries may be combined.

Note: These wrappers do not use any mpich-specific features besides the MPE graphics and logging used by 'vismess' and 'logging', respectively. They should work on any MPI implementation.

You can incorporate them manually into your application, which involves three changes to the building of your application:

- Generate the source code for the desired wrapper(s) with wrappergen. This can be a one-time task.
- Compile the code for the wrapper(s). Be sure to supply the needed compile-line parameters. 'vismess' and 'logging' require the MPE library ('`-lmpe`'), and the 'vismess' wrapper definition requires MPE_GRAPHICS.
- Link the compiled wrapper code, the profiling version of the mpi library, and any other necessary libraries ('vismess' requires X) into your application. The required order is:

```
$(CLINKER)   <application object files...> \
<wrapper object code> \
<other necessary libraries (-lmpe)> \
<profiling mpi library (-lpmpi)> \
<standard mpi library (-lmpi)>
```

To simplify it, some sample makefile sections have been created in '`mpich/mpe/profiling/lib`':

```
Makefile.timing - timing wrappers
Makefile.logging - logging wrappers
Makefile.vismess - animated messages wrappers
Makefile.allprof - timing, logging, and vismess
```

To use these Makefile fragments:

1. (optional) Add `$(PROF_OBJ)` to your application's dependency list:

   ```
   myapp:  myapp.o $(PROF_OBJ)
   ```

2. Add `$(PROF_FLG)` to your compile line `CFLAGS`:

   ```
   CFLAGS = -O $(PROF_FLG)
   ```

3. Add `$(PROF_LIB)` to your link line, after your application's object code, but before the main MPI library:

   ```
   $(CLINKER) myapp.o -L$(MPIR_HOME)/lib/$(ARCH)/$(COMM) $(PROF_LIB) -lmpi
   ```

4. (optional) Add `$(PROF_CLN)` to your clean target:

   ```
   rm -f *.o *~ myapp $(PROF_CLN)
   ```

5. Include the desired Makefile fragment in your makefile:

```
include $(MPIR_HOME)/mpe/profiling/lib/Makefile.logging
```

(or

```
#include $(MPIR_HOME)/mpe/profiling/lib/Makefile.logging
```

if you are using the wildly incompatible BSD 4.4-derived make)

5.8 Examining event logs with `upshot`

The original `upshot` was written in C and distributed several years ago [6]. To go with MPICH, it has been completely rewritten by Ed Karrels to provide more robustness and flexibility. The current version resides in the directory `mpe/profiling/upshot`, but gets installed in `mpich/bin`. A newer version, currently under development but functional, providing less functionality but greater speed on large log files, is in `mpe/profiling/nupshot`.

6 Debugging MPI programs with built-in tools

Debugging of parallel programs is notoriously difficult, and we do not have a magical solution to this problem. Nonetheless, we have built into MPICH a few features that may be of use in debugging MPI programs.

6.1 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`. As part of the `mpe` library, we include two other error handlers to facilitate the use of `dbx` in debugging MPI programs.

```
MPE_Errors_call_dbx_in_xterm
MPE_Signals_call_debugger
```

These error handlers are located in the `mpe` directory. A configure option (`-mpedbg`) includes these error handlers into the regular MPI libraries, and allows the command-line argument `-mpedbg` to make `MPE_Errors_call_dbx_in_xterm` the default error handler (instead of `MPI_ERRORS_ARE_FATAL`).

6.2 Command-line arguments for `mpirun`

`mpirun` provides some help in starting programs with a debugger.

```
mpirun -dbx -np 2 program
```

starts `program` on two machines, with the local one running under the `dbx` debugger. The option `-gdb` selects the `gdb` debugger instead. The option `-xxgdb` allows you to use the `xxgdb` (X Window GUI interface to `gdb`).

6.3 MPI arguments for the application program

These are currently undocumented, and some require configure options to have been specified (like `-mpipktsize` and `-chmemdebug`). The `-mpiversion` option is useful for finding out how your installation of MPICH was configured and exactly what version it is.

-mpedbg If an error occurs, start `xterm`s attached to the process that generated the error. Requires the MPICH be configured with `-mpedbg` and works on only some workstations systems.

-mpiversion Print out the version and configuration arguements for the MPICH implementation being used.

These arguments are provided to the program, not to `mpirun`. That is,

```
mpirun -np 2 a.out -mpiversion
```

6.4 p4 Arguments for the `ch_p4 Device`

6.4.1 p4 Debugging

If your configuration of MPICH used `-device=ch_p4`, then some of the `p4` debugging capabilities are available to you. The most useful of these are the command line arguments to the application program. Thus

```
mpirun -np 10 myprog -p4dbg 20 -p4rdbg 20
```

results in program tracing information at a level of 20 being written to `stdout` during execution. For more information about what is printed at what levels, see the `p4` Users' Guide [1].

If one specifies `-p4norem` on the command line, `mpirun` will not actually start the processes. The master process prints a message suggesting how the user can do it. The point of this option is to enable the user to start the remote processes under his favorite debugger, for instance. The option only makes sense when processes are being started remotely, such as on a workstation network. Note that this is an argument to the *program*, not to `mpirun`. For example, to run `myprog` this way, use

```
mpirun -np 4 myprog -p4norem
```

6.4.2 Setting the Working Directory for the p4 Device

By default, the working directory for processes running remotely with `ch_p4` device is the same as that of the executable. To specify the working directly, use `-p4wdir` as follows:

```
mpirun -np 4 myprog -p4wdir myrundir
```

6.5 Command-line arguments for the application program

Arguments on the command line that follow the application program name and are not directed to the MPICH system (don't begin with `-mpi` or `-p4`) are passed through to all processes of the application program. For example, if you execute

```
mpirun -echo -np 4 myprog -mpiversion -p4dbg 10 x y z
```

then `-echo -np 4` is interpreted by `mpirun` (echo actions of `mpirun` and run four processes), `-mpiversion` is interpreted by MPICH (each process prints configuration information), `-p4dbg 10` is interpreted by the p4 device if your version was configured with `-device=ch_p4` (sets p4 debugging level to 10), and `x y z` are passed through to the application program. In addition, `MPI_Init` strips out non-application arguments, so that after the call to `MPI_Init` in your C program, the argument vector `argv` contains only

```
myprog x y z
```

and your program can process its own command-line arguments in the normal way.

6.6 Starting jobs with a debugger

The `-dbx` option to `mpirun` causes processes to be run under the control of the `dbx` debugger. This depends on cooperation between `dbx` and MPICH and does not always work; if it does not, you will know immediately. If it does work, it is often the simplest way to debug MPI programs. Similiarly, the argument `-gdb` makes use of the GNU debugger.

For example, enter

```
mpirun -dbx or mpirun -gdb a.out
```

In some cases, you will have to wait until the program completes and then type `run` to run the program again. Also, `mpirun` relies on the `-sr` argument to `dbx` (this tells `dbx` to read initial commands from a file). If your `dbx` does not support that feature, `mpirun` will fail to start your program under the debugger.

6.7 Starting the debugger when an error occurs

Enter

```
mpirun ... a.out -mpedbg
```

(requires `mpich` built with `-mpedbg` option; do `-mpiversion` and look for `-mpedbg` option).

6.8 Attaching debugger to a running program

On workstation clusters, you can often attach a debugger to a running process. For example, the debugger `dbx` often accepts a process id (pid) which you can get by using the `ps` command. The form is either

```
dbx a.out pid
```

or

```
dbx -pid pid a.out
```

One can also attach the TotalView debugger to a running program (See Section 7.3 below).

6.9 Related tools

The *Scalable Unix Tools* (SUT) is a collection for managing workstation networks as a MPP. These include programs for looking at all of the processes in a cluster and performing operations on them (such as attaching the debugger to every process you own that is running a particular program). This is not part of MPI but can be very useful in working with workstation clusters. These tools are not available yet, but will be released soon.

6.10 Contents of the library files

The directory containing the MPI library file ('`libmpich.a`') contains a few additional files. These are summarized here

libmpich.a MPI library (`MPI_Xxxx`)

libpmpich.a Profiling version (`PMPI_Xxxx`)

libmpe.a MPE graphics, logging, and other extensions (`PMPE_Xxxx`)

libmpe_nompi.a MPE graphics without MPI

mpe_prof.o Sample profiling library (C)

mpe_proff.o Sample profiling library (Fortran)

7 Debugging MPI programs with TotalView

TotalView (©) is a powerful, commercial-grade, portable debugger for parallel and multi-threaded programs, available from Dolphin Interconnect Solutions (`http://www.dolphinics.com/toolworks`). TotalView understands multiple MPI implementations, including MPICH. By "understand" is meant that if you have TotalView installed on your system, it is easy to start your MPICH program under the control of TotalView, even if you are running on multiple machines, manage your processes both collectively and individually through TotalView's convenient GUI, and even examine internal MPICH data structures to look at message queues. The general operation model of TotalView will be familiar to users of command-line-based debuggers such as `gdb` or `dbx`.

7.1 Preparing MPICH for TotalView debugging

See the *Installation Guide* for instructions on configuring MPICH so that TotalView can display message queues.

7.2 Starting an MPICH program under TotalView control

To start a parallel program under TotalView control, simply add '`-tv`' to your `mpirun` arguments:

```
mpirun -tv -np 4 cpi
```

TotalView will come up and you can start the program by typing '`G`'. A window will come up asking whether you want to stop processes as they execute `MPI_Init`. You may find it more convenient to say "no" and instead to set your own breakpoint after `MPI_Init` (See Section 7.4.) This way when the process stops it will be on a line in your program instead of somewhere inside `MPI_Init`.

7.3 Attaching to a running program

TotalView can attach to a running MPI program, which is particularly useful if you suspect that your code has deadlocked. To do this start TotalView with no arguments, and then press '`N`' in the root window. This will bring up a list of the processes that you can attach to. When you dive through the initial MPICH process in this window TotalView will also acquire all of the other MPICH processes (even if they are not local). (See the TotalView manual for more details of this process.)

7.4 Debugging with TotalView

You can set breakpoints by clicking in the left margin on a line number. Most of the TotalView GUI is self-explanatory. You select things with the left mouse button, bring up an action menu with the middle button, and "dive" into functions, variables, structures, processes, etc., with the right button. Pressing `cntl-?` in any TotalView window brings up help relevant to that window. In the initial TotalView window it brings up general help. The full documentation (The TotalView *User's Guide*) is available at `http://www.dolphinics.com/tw/tvdemo37.htm`.

You switch from viewing one process to the next with the arrow buttons at the top-right corner of the main window, or by explicitly selecting (left button) a process in the root window to re-focus an existing window onto that process, or by diving (right button) through a process in the root window to open a new window for the selected process. All the keyboard shortcuts for commands are listed in the menu that is attached to the middle button. The commands are mostly the familiar ones. The special one for MPI is the '`m`' command, which displays message queues associated with the process.

Note also that if you use the MPI-2 function `MPI_Comm_set_name` on a communicator, TotalView will display this name whenever showing information about the communicator, making it easier to understand which communicator is which.

7.5 Summary

TotalView is a very powerful, flexible debugger for parallel and multithreaded programs. It has many capabilities beyond those described here, which are just enough to get you started. For more details, see the *TotalView User's Guide*, and particularly the section on MPICH.

8 Other MPI Documentation

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: U. of T. report [2]
  - As postscript for ftp: at `ftp.mcs.anl.gov` in `pub/mpi/mpi-report.ps`.
  - As hypertext on the World Wide Web: `http://www.mpi-forum.org`
  - As a journal article: in the Fall 1994 issue of the Journal of Supercomputing Applications [8]
- MPI Forum discussions
  - The MPI Forum email discussions and both current and earlier versions of the Standard are available from `netlib`.
- Books:
  - *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, by Gropp, Lusk, and Skjellum [4].

- *MPI: The Complete Reference*, by Snir, et al. [9].

- Newsgroup:
  - `comp.parallel.mpi`
- Mailing lists:
  - `mpi-comm@cs.utk.edu`: The MPI Forum discussion list.
  - `mpi-impl@mcs.anl.gov`: The implementors' discussion list.
  - `mpi-bugs@mcs.anl.gov`: The address to report problems with `mpich` to.
- Implementations available by `ftp`:
  - MPICH is available by anonymous `ftp` from `ftp.mcs.anl.gov` in the directory `pub/mpi/mpich`, file `mpich.tar.Z`.
  - LAM is available by anonymous `ftp` from `http://www.mpi.nd.edu/lam/`.
- Test code repository (new):
  - `ftp://ftp.mcs.anl.gov/pub/mpi/mpi-test`

9 In Case of Trouble

This section describes some commonly encountered problems and their solutions. It also describes machine-dependent considerations. 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., 1.1.1)
- The output of running your program with the `-mpiversion` argument (e.g., `mpirun -np 1 a.out -mpiversion`)
- The output of

  ```
  uname -a
  ```

  for your system. If you are on an SGI system, also

  ```
  hinv
  ```

- If the problem is with a script like configure or mpirun, run the script with the `-echo` argument (e.g., `mpirun -echo -np 4 a.out`).
- If you are using a network of workstations, also send the output of `bin/tstmachines` or `util/tstmachines` [2].

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.

[2] `tstmachines` is not supported in the `globus` device

9.1 Problems compiling or linking Fortran programs

9.1.1 General

1. **Q:** When linking the test program, the following message is generated:

   ```
   f77 -g -o secondf secondf.o -L/usr/local/mpich/lib/sun4/ch_p4 -lmpi
   invalid option -L/usr/local/mpich/lib/sun4/ch_p4
   ld: -lmpi: No such file or directory
   ```

   A: This `f77` program does not accept the `-L` command to set the library search path. Some systems provide a shell script for `f77` that is very limited in its abilities. To work around this, use the full library path instead of the `-L` option:

   ```
   f77 -g -o secondf secondf.o /usr/local/mpich/lib/sun4/ch_p4/libmpi.a
   ```

2. **Q:** When linking Fortran programs, I get undefined symbols such as

   ```
   f77  -c secondf.f
   secondf.f:
    MAIN main:
   f77  -o secondf secondf.o -L/home/mpich/lib/solaris/ch_shmem -lmpi
   Undefined                       first referenced
    symbol                             in file
   getdomainname
   /home/mpich/lib/solaris/ch_shmem/libmpi .a(shmempriv.o)
   ld: fatal: Symbol referencing errors. No output written to secondf
   ```

   There is no problem with C programs.

   A: This means that your C compiler is providing libraries for you that your Fortran compiler is not providing. Find the option for the C compiler and for the Fortran compilers that indicate which library files are being used (alternately, you may find an option such as `-dryrun` that shows what commands are being used by the compiler). Build a simple C and Fortran program and compare the libraries used (usually on the `ld` command line). Try the ones that are present for the C compiler and missing for the Fortran compiler.

9.2 Problems Linking C Programs

9.2.1 General

1. **Q:** When linking programs, I get messages about `__builtin_saveregs` being undefined.

   A: You may have a system on which C and Fortran routines are incompatible (for example, using `gcc` and the vendor's Fortran compiler). If you do not plan to use Fortran, the easiest fix is to rebuild with the `-nof77` option to configure.

You should also look into making your C compiler compatible with your Fortran compiler. One possibility is use `f2c` to convert Fortran to C, then use the C compiler to compile everything. If you take this route, remember that *every* Fortran routine has to be compiled using `f2c` and the C compiler.

9.2.2 Sun Solaris

1. **Q:** When linking on Solaris, I get an error like this:

```
cc -g -o testtypes testtypes.o -L/usr/local/mpich/lib/solaris/ch_p4 -lmpi
       -lsocket -lnsl -lthread
ld: warning: symbol '_defaultstkcache' has differing sizes:
       (file /usr/lib/libthread.so value=0x20; file /usr/lib/libaio.so value=0x8);
       /usr/lib/libthread.so definition taken
```

A: This is a bug in Solaris 2.3 that is fixed in Solaris 2.4. There may be a patch for Solaris 2.3; contact Sun for more information.

9.2.3 HPUX

1. **Q:** When linking on HPUX, I get an error like this:

```
cc -o pgm pgm.o -L/usr/local/mpich/lib/hpux/ch_p4 -lmpi  -lm
/bin/ld: Unsatisfied symbols:
sigrelse (code)
sigset (code)
sighold (code)
*** Error code 1
```

A: You need to add the link option `-lV3`. The p4 device uses the System V signals on the HP; these are provided in the '`V3`' library.

9.3 Problems starting programs

9.3.1 General

1. **Q:** When trying to start a program with

```
mpirun -np 2 cpi
```

either I get an error message or the program hangs.

A: On Intel Paragons and IBM SP1 and SP2, there are many mutually exclusive ways to run parallel programs; each site can pick the approach(es) that it allows. The script `mpirun` tries one of the more common methods, but may make the wrong choice. Use the `-v` or `-t` option to `mpirun` to see how it is trying to run the program, and then compare this with the site-specific instructions for using your system. You may need to adapt the code in `mpirun` to meet your needs.

2. **Q:** When trying to run a program with, e.g., `mpirun -np 4 cpi`, I get

   ```
   usage : mpirun [options] <executable> [<dstnodes>] [-- <args>]
   ```

   or

   ```
   mpirun [options] <schema>
   ```

   A: You have a command named `mpirun` in your path ahead of the MPICH version. Execute the command

   ```
   which mpirun
   ```

   to see which command named `mpirun` was actually found. The fix is to either change the order of directories in your path to put the MPICH version of `mpirun` first, or to define an alias for `mpirun` that uses an absolute path. For example, in the csh shell, you might do

   ```
   alias mpirun /usr/local/mpich/bin/mpirun
   ```

3. **Q:** When trying to start a large number of processes on a workstation network, I get the message

   ```
   p4_error: latest msg from perror: Too many open files
   ```

   A: There is a limitation on the number of open file descriptors. On some systems you can increase this limit yourself; on others you must have help from your system administrator. You could experiment with the secure server, but it is not a complete solution. We are working now on a more scalable startup mechanism for the next release.

4. **Q:** When I issue the command:

   ```
   mpirun -dbx -np 1 foo
   ```

   `dbx` does start up but this message appears:

   ```
   dbx version 3.19 Nov  3 1994 19:59:46
   Unexpected argument ignored: -sr
   /scr/MPI/me/PId8704 is not an executable
   ```

   A: Your version of `dbx` does not support the `-sr` argument; this is needed to give `dbx` the initial commands to execute. You will not be able to use `mpirun` with the `-dbx` argument. Try using `-gdb` or `-xxgdb` instead of `-dbx` if you have the GNU debugger. Another excellent debugging option is to use TotalView.

5. **Q:** When attempting to run cpilog I get the following message:

   ```
   ld.so.1: cpilog: fatal: libX11.so.4: can't open file: errno 2
   ```

A: The X11 version that configure found isn't properly installed. This is a common problem with Sun/Solaris systems. One possibility is that your Solaris machines are running slightly different versions. You can try forcing static linking (-Bstatic on SunOS).

Consider adding these lines to your '`.login`' (assuming C shell):

```
setenv OPENWINHOME /usr/openwin
setenv LD_LIBRARY_PATH /opt/SUNWspro/lib:/usr/openwin/lib
```

(you may want to check with your system administrator first to make sure that the paths are correct for your system). Make sure that you add them *before* any line like

```
if ($?USER == 0 || $?prompt == 0) exit
```

6. **Q:** My program fails when it tries to write to a file.

   A: If you opened the file *before* calling `MPI_INIT`, the behavior of MPI (not just MPICH) is undefined. On the `ch_p4` version, only process zero (in `MPI_COMM_WORLD`) will have the file open; the other processes will not have opened the file. Move the operations that open files and interact with the outside world to after `MPI_INIT` (and before `MPI_FINALIZE`).

7. **Q:** Programs seem to take forever to start.

   A: This can be caused by any of several problems. On systems with dynamically-linked executables, this can be caused by problems with the file system suddenly getting requests from many processors for the dynamically-linked parts of the executable (this has been measured as a problem with some DFS implementations). You can try statically linking your application.

   On workstation networks, long startup times can be due to the time used to start remote processes; see the discussion on the secure server.

9.3.2 Workstation Networks

1. **Q:** When I use `mpirun`, I get the message `Permission denied`.

   A: If you see something like this

```
% mpirun -np 2 cpi
Permission denied.
```

   when using the `ch_p4` or `chameleon` device, it probably means that you do not have permission to use `rsh` to start processes. The script `tstmachines` can be used to test this. For example, if the architecture type (the `-arch` argument to configure) is `sun4`, then try

```
tstmachines sun4
```

If this fails, then you may need a '`.rhosts`' or '`/etc/hosts.equiv`' file (you may need to see your system administrator) or you may need to use the p4 server (see Section 3.1.3). Another possible problem is the choice of the remote shell program; some systems have several. Check with your systems administrator about which version of `rsh` or `remsh` you should be using.

If your system allows a '`.rhosts`' file, do the following:

(a) Create a file '`.rhosts`' in your home directory

(b) Change the protection on it to user read/write only: `chmod og-rwx .rhosts`.

(c) Add one line to the '`.rhosts`' file for each processor that you want to use. The format is

```
host username
```

For example, if your username is `doe` and you want to user machines `a.our.org` and `b.our.org`, your '`.rhosts`' file should contain

```
a.our.org doe
b.our.org doe
```

Note the use of fully qualified host names (some systems require this).

On networks where the use of .rhosts files is not allowed, (such as the one in MCS at Argonne), you should use the p4 server to run on machines that are not trusted by the machine that you are initiating the job from.

Finally, you may need to use a non-standard `rsh` command within MPICH. MPICH must be reconfigured with `-rsh=command_name`, and perhaps also with `-rshnol` if the remote shell command does not support the `-l` argument. Systems using Kerberos and/or AFS may need this. See the section in the *Installation Guide* on using the secure shell `ssh`, for example.

An alternate source of the "Permission denied." message is that you have used the su command to change your effective user id. On some systems the `ch_p4` device will not work in this situation. Log in normally and try again.

2. **Q:** When I use `mpirun`, I get the message `Try again`.

   A: If you see something like this

```
% mpirun -np 2 cpi
Try again.
```

   it means that you were unable to start a remote job with the remote shell command on some machine, even though you would normally be able to. This may mean that the destination machine is very busy, out of memory, or out of processes. The man page for `rshd` may give you more information.

   The only fix for this is to have your system administrator look into the machine that is generating this message.

3. **Q:** When running the workstation version (`-device=ch_p4`), I get error messages of the form

```
stty: TCGETS: Operation not supported on socket
```

or

```
stty: tcgetattr: Permission denied
```

or

```
stty: : Can't assign requested address
```

A: This means that one your login startup scripts (i.e., '`.login`' and '`.cshrc`' or '`.profile`') contains an unguarded use of the `stty` or `tset` program. For C shell users, one typical fix is to check for the variables `TERM` or `PROMPT` to be initialized. For example,

```
if ($?TERM) then
    eval `tset -s -e^\? -k^U -Q -I $TERM`
endif
```

Another solution is to see if it is appropriate to add

```
if ($?USER == 0 || $?prompt == 0) exit
```

near the top of your '`.cshrc`' file (but *after* any code that sets up the runtime environment, such as library paths (e.g., `LD_LIBRARY_PATH`).

4. **Q:** When using `mpirun` I get strange output like

```
arch: No such file or directory
```

A: This is usually a problem in your '`.cshrc`' file. Try the shell command

```
which hostname
```

If you see the same strange output, then your problem is in your '`.cshrc`' file.

5. **Q:** When I try to run my program, I get

```
p0_4652:  p4_error: open error on procgroup file (procgroup): 0
```

A: This indicates that the `mpirun` program did not create the expected input to run the program. The most likely reason is that the mpirun command is trying to run a program build with device `ch_p4` (workstation networks) as shared memory or some special system.

Try the following:

Run the program using mpirun and the `-t` argument:

```
mpirun -t -np 1 foo
```

This should show what mpirun would do (`-t` is for testing). Or you can use the `-echo` argument to see exactly what mpirun is doing:

```
mpirun -echo -np 1 foo
```

In general, you should select the mpirun in 'lib/<architecture>/<device>' directory over the one in the 'bin' directory.

6. **Q:** When trying to run a program I get this message:

   ```
   icy%  mpirun -np 2 cpi -mpiversion
   icy: icy: No such file or directory
   ```

   A: Your problem is that '/usr/lib/rsh' is not the remote shell program. Try the following:

   ```
   which rsh
   ls /usr/*/rsh
   ```

   You probably have '/usr/lib' in your path ahead of '/usr/ucb' or '/usr/bin'. This picks the 'restricted' shell instead of the 'remote' shell. The easiest fix is to just remove '/usr/lib' from your path (few people need it); alternately, you can move it to after the directory that contains the 'remote' shell rsh.

   Another choice would be to add a link in a directory earlier in the search path to the remote shell. For example, I have '/home/gropp/bin/sun4' early in my search path; I could use

   ```
   cd /home/gropp/bin/sun4
   ln -s /usr/bin/rsh  rsh
   ```

   there (assuming '/usr/bin/rsh' is the remote shell).

7. **Q:** When trying to run a program I get this message:

   ```
   trying normal rsh
   ```

   A: You are using a version of the remote shell program that does not support the -l argument. Reconfigure with -rshnol and rebuild MPICH. You may suffer some loss of functionality if you try to run on systems where you have different user names. You might also try using ssh.

8. **Q:** When I run my program, I get messages like

   ```
   | ld.so: warning: /usr/lib/libc.so.1.8 has older revision than expected 9
   ```

   A: You are trying to run on another machine with an out-dated version of the basic C library. For some reason, some manufacturers do not make the shared libraries compatible between minor (or even maintenance) releases of their software. You need to have you system administrator bring the machines to the same software level.

   One temporary fix that you can use is to add the link-time option to force static linking instead of dynamic linking. For some Sun workstations, the option is -Bstatic.

9. **Q:** Programs never get started. Even `tstmachines` hangs.

   A:

   Check first that `rsh` works at all. For example, if you have workstations `w1` and `w2`, and you are running on `w1`, try

   ```
   rsh w2 true
   ```

   This should complete quickly. If it does not, try

   ```
   rsh w1 true
   ```

   (that is, use `rsh` to run `true` on the system that you are running on). If you get `permission denied`, see the help on that. If you get

   ```
   krcmd: No ticket file (tf_util)
   rsh: warning, using standard rsh: can't provide Kerberos auth data.
   ```

   then your system has a faulty installation of `rsh`. Some FreeBSD systems have been observed with this problem. Have your system administrators correct the problem (often one of an inconsistent set of `rsh/rshd` programs).

10. **Q:** When running the workstation version (`-device=ch_p4`), I get error messages of the form

    ```
    more slaves than message queues
    ```

    A: This means that you are trying to run MPICH in one mode when it was configured for another. In particular, you are specifying in your p4 procgroup file that several processes are to shared memory on a particular machine by either putting a number greater than 0 on the first line (where it signifies number of local processes besides the original one), or a number greater than 1 on any of the succeeding lines (where it indicates the total number of processes sharing memory on that machine). You should either change your procgroup file to specify only one process on line, or reconfigure MPICH with

    ```
    configure -device=ch_p4 -comm=shared
    ```

    which will reconfigure the p4 device so that multiple processes can share memory on each host. The reason this is not the default is that with this configuration you will see busy waiting on each workstation, as the device goes back and forth between selecting on a socket and checking the internal shared-memory queue.

11. **Q:** My programs seem to hang in `MPI_Init`.

    A: There are a number of ways that this can happen:

    (a) One of the workstations you selected to run on is dead (try '`tstmachines`'[3]).

[3] '`tstmachines`' is not supported in the `globus` device

(b) You linked with the FSU pthreads package; this has been reported to cause problems, particularly with the system `select` call that is part of Unix and is used by MPICH.

Another is if you use the library '`-ldxml`' (extended math library) on Digital Alpha systems. This has been observed to case `MPI_Init` to hang. No workaround is known at this time; contact Digital for a fix if you need to use MPI and '`-ldxml`' together.

The root of this problem is that the `ch_p4` device uses `SIG_USR1`, and so any library that also uses this signal can interfere with the operation of MPICH if it is using `ch_p4`.

12. **Q:** My program (using device `ch_p4`) fails with

```
p0_2005:  p4_error: fork_p4: fork failed: -1
          p4_error: latest msg from perror: Error 0
```

A: The executable size of your program may be too large. When a `ch_p4` or `ch_tcp` device program starts, it may create a copy of itself to handle certain communication tasks. Because of the way in which the code is organized, this (at least temporarily) is a full copy of your original program and occupies the same amount of space. Thus, if your program is over half as large as the maximum space available, you wil get this error. On SGI systems, you can use the command `size` to get the size of the executable and `swap -l` to get the available space. Note that `size` gives you the size in bytes and `swap -l` gives you the size in 512-byte blocks. Other systems may offer similar commands.

A similar problem can happen on IBM SPx using the `ch_mpl` device; the cause is the same but it originates within the IBM MPL library.

13. **Q:** Sometimes, I get the error

```
Exec format error. Wrong Architecture.
```

A: You are probably using NFS (Network File System). NFS can fail to keep files updated in a timely way; this problem can be caused by creating an executable on one machine and then attempting to use it from another. Usually, NFS catches up with the existence of the new file within a few minutes. You can also try using the `sync` command. `mpirun` in fact tries to run the `sync` command, but on many systems, `sync` is only advisory and will not guarentee that the file system has been made consistent.

14. **Q:** There seem to be two copies of my program running on each node. This doubles the memory requirement of my application. Is this normal?

A: Yes, this is normal. In the `ch_p4` implementation, the second process is used to dynamically establish connections to other processes. With Version 1.1.1 of MPICH, this functionality is placed in a separate thread on many architectures, and this second process will not be seen.

15. **Q:** `MPI_Abort` sometimes doesn't work. Why?

A: Currently (Version 1.1.1) a process detects that another process has aborted only when it tries to receive a message, and the aborting process is one that it has communicated with in the past. Thus it is porssible for a process busy with computation not to notice that one of its peers has issued an `MPI_Abort`, although for many common communication patterns this does not present a problem. This will be fixed in a future release.

9.3.3 Intel Paragon

1. **Q:** How do I run jobs with `mpirun` under NQS on my Paragon?

   A: Give `mpirun` the argument `-paragontype nqs`.

9.3.4 IBM RS6000

1. **Q:** When trying to run on an IBM RS6000 with the `ch_p4` device, I got

   ```
   % mpirun -np 2 cpi
   Could not load program /home/me/mpich/examples/basic/cpi
   Could not load library libC.a[shr.o]
   Error was: No such file or directory
   ```

   A: This means that MPICH was built with the `xlC` compiler but that some of the machines in your '`util/machines/machines.rs6000`' file do not have `xlC` installed. Either install `xlC` or rebuild MPICH to use another compiler (either `xlc` or `gcc`; `gcc` has the advantage of never having any licensing restrictions).

2. **Q:** When trying to run on an IBM RS6000 with the `ch_p4` device, I got

   ```
   % mpirun -np 2 cpi
   Could not load program /home/me/mpich/examples/basic/cpi
   Could not load library libC.a[shr.o]
   Error was: No such file or directory
   ```

   A: This means that MPICH was built with the `xlC` compiler but that some of the machines in your '`util/machines/machines.rs6000`' file do not have `xlC` installed. Either install `xlC` or rebuild MPICH to use another compiler (either `xlc` or `gcc`; `gcc` has the advantage of never having any licensing restrictions).

9.3.5 IBM SP

1. **Q:** When starting my program on an IBM SP, I get this:

   ```
   $ mpirun -np 2 hello
   ERROR: 0031-124  Couldn't allocate nodes for parallel execution.  Exiting ...
   ERROR: 0031-603  Resource Manager allocation for task: 0, node:
   me1.myuniv
   ```

```
.edu, rc = JM_PARTIONCREATIONFAILURE
ERROR: 0031-635  Non-zero status -1 returned from pm_mgr_init
```

A: This means that either `mpirun` is trying to start jobs on your SP in a way different than your installation supports or that there has been a failure in the IBM software that manages the parallel jobs (all of these error messages are from the IBM `poe` command that `mpirun` uses to start the MPI job). Contact your system administrator for help in fixing this situation. You system administrator can use

```
dsh -av "ps aux | egrep -i 'poe|pmd|jmd'"
```

from the control workstation to search for stray IBM POE jobs that can cause this behavior. The files `/tmp/jmd_err` on the individual nodes may also contain useful diagnostic information.

2. **Q:** When trying to run on an IBM SP, I get the message from `mpirun`:

```
  ERROR: 0031-214  pmd: chdir </a/user/gamma/home/mpich/examples/basic>
  ERROR: 0031-214  pmd: chdir </a/user/gamma/homempich/examples/basic>
```

   A: These are messages from tbe IBM system, not from `mpirun`. They may be caused by an incompatibility between POE, the automounter (especially AMD) and the shell, especially if you are using a shell other than `ksh`. There is no good solution; IBM often recommends changing your shell to `ksh`!

3. **Q:** When I tried to run my program on an IBM SP, I got

```
ERROR : Cannot locate message catalog (pepoe.cat) using current NLSPATH
INFO  : If NLSPATH is set correctly and catalog exists, check LANG or
LC_MESSAGES variables
(C) Opening of "pepoe.cat" message catalog failed
```

   (and other variations that mention NLSPATH and "message catalog").

   A: This is a problem in your system; contact your support staff. Have them look at (a) value of NLSPATH, (b) links from '`/usr/lib/nls/msg/prime`' to the appropriate language directory. The messages are not from MPICH; they are from the IBM POE/MPL code the the MPICH implementation is using.

4. **Q:** When trying to run on an IBM SP, I get this message:

```
ERROR: 0031-124  Less than 2 nodes available from pool 0
```

   A: This means that the IBM POE/MPL system could not allocate the requested nodes when you tried to run your program; most likely, someone else was using the system. You can try to use the environment variables `MP_RETRY` and `MP_RETRYCOUNT` to cause the job to wait until the nodes become available. Use `man poe` to get more information.

5. **Q:** When running on an IBM SP, my job generates the message

```
Message number 0031-254 not found in Message Catalog.
```

and then dies.

A: If your user name is eight characters long, you may be experiencing a bug in the IBM POE environment. The only fix at the time this was written was to use an account whose user name was seven characters or less. Ask your IBM representative about PMR 4017X (poe with userids of length eight fails) and the associated APAR IX56566.

9.4 Programs fail at startup

9.4.1 General

1. **Q:** With some systems, you might see

   ```
   /lib/dld.sl: Bind-on-reference call failed
   /lib/dld.sl: Invalid argument
   ```

   (This example is from HP-UX).

   ```
   ld.so: libc.so.2: not found
   ```

   (This example is from SunOS 4.1; similar things happen on other systems).

   A: The problem here is that your program is using shared libraries, and the libraries are not available on some of the machines that you are running on. To fix this, relink your program without the shared libraries. To do this, add the appropriate command-line options to the link step. For example, for the HP system that produced the errors above, the fix is to use `-Wl,-Bimmediate` to the link step. For SunOS, the appropriate option is `-Bstatic`.

9.4.2 Workstation Networks

1. **Q:** I can run programs using a small number of processes, but once I ask for more than 4–8 processes, I do not get output from all of my processes, and the programs never finish.

   A: We have seen this problem with installations using AFS. The remote shell program, `rsh`, supplied with some AFS systems seems to limit the number of jobs that can use standard output. This seems to prevent some of the processes from exiting as well, causing the job to hang. There are four possible fixes:

   (a) Use a different `rsh` command. You can probably do this by putting the directory containing the non-AFS version first in your `PATH`. This option may not be available to you, depending on your system. At one site, the non-AFS version was in '`/bin/rsh`'.

   (b) Use the secure server (`serv_p4`). See the discussion in the Users Guide.

(c) Redirect all standard output to a file. The MPE routine `MPE_IO_Stdout_to_file` may be used to do this.

(d) Get a fixed `rsh` command. The likely source of the problem is an incorrect usage of the `select` system call in the `rsh` command. If the code is doing something like

```
int mask;
mask |= 1 << fd;
select( fd+1, &mask, ... );
```

instead of

```
fd_set mask;
FD_SET(fd,&mask);
select( fd+1, &mask, ... );
```

then the code is incorrect (the `select` call changed to allow more than 32 file descriptors many years ago, and the `rsh` program (or programmer!) hasn't changed with the times).

A fourth possiblity is to get an AFS version of `rsh` that fixes this bug. As we are not running AFS ourselves, we do not know whether such a fix is available.

2. **Q:** Not all processes start.

   A: This can happen when using the `ch_p4` device and a system that has extremely small limits on the number of remote shells you can have. Some systems using "Kerberos" (a network security package) allow only three or four remote shells; on these systems, the size of `MPI_COMM_WORLD` will be limited to the same number (plus one if you are using the local host).

   The only way around this is to try the secure server; this is documented in the MPICH installation guide. Note that you will have to start the servers "by hand" since the `chp4_servs` script uses remote shell to start the servers.

9.5 Programs fail after starting

9.5.1 General

1. **Q:** I use `MPI_Allreduce`, and I get different answers depending on the number of processes I'm using.

   A: The MPI collective routines may make use of associativity to achieve better parallelism. For example, an

   ```
   MPI_Allreduce( &in, &out, MPI_DOUBLE, 1, ... );
   ```

   might compute

   $$(((((((a+b)+c)+d)+e)+f)+g)+h)$$

   or it might compute

   $$((a+b)+(c+d))+((e+f)+(g+h)),$$

where $a, b, \ldots$ are the values of `in` on each of eight processes. These expressions are equivalent for integers, reals, and other familar objects from mathematics but are *not* equivalent for fixed precision datatypes used in computers. The association that MPI uses will depend on the number of processes, thus, you may not get exactly the same result when you use different numbers of processes. Note that you are not getting a wrong result, just a different one (most programs assume the arithmetic operations are associative).

2. **Q:** I get the message

   ```
   No more memory for storing unexpected messages
   ```

   when running my program.

   A: MPICH has been configured to "aggressively" deliver messages. This is appropriate for certain types of parallel programs, and can deliver higher performance. However, it can cause applications to run out of memory when messages are delivered faster than they are processed. The MPICH implementation does attempt to control such memory usage, but there are still a few more steps to take in the MPICH implementation. As a work-around, you can introduce synchronous sends or barriers into your code. The need for these will be eliminated in a future MPICH release.

3. **Q:** My Fortran program fails with a BUS error.

   A: The C compiler that MPICH was built with and the Fortran compiler that you are using have different alignment rules for things like `DOUBLE PRECISION`. For example, the GNU C compiler `gcc` may assume that all `double`s are aligned on eight-byte boundaries, but the Fortran language requires only that `DOUBLE PRECISION` align with `INTEGER`s, which may be four-byte aligned.

   There is no good fix. Consider rebuilding MPICH with a C compiler that supports weaker data alignment rules. Some Fortran compilers will allow you to force eight-byte alignment for `DOUBLE PRECISION` (for example, `-dalign` or `-f` on some Sun Fortran compilers); note though that this may break some correct Fortran programs that exploit Fortran's storage association rules.

   Some versions of `gcc` may support `-munaligned-doubles`; MPICH should be rebuilt with this option if you are using `gcc`, version 2.7 or later.

4. **Q:** I'm using `fork` to create a new process, or I'm creating a new thread, and my code fails.

   A: The MPICH implementation is not thread safe and does not support either fork or the creation of new processes. Note that the MPI specification is intended to be thread safe, but implementations are not required to be so. At this writing, few implementations are thread-safe, primarily because this reduces the performance of the MPI implementation (you at least need to check to see if you need a thread lock, actually getting and releaseing the lock is even more expensive).

9.5.2 HPUX

1. **Q:** My Fortran programs seem to fail with `SIGSEGV` when running on HP workstations.
   A: Try compiling and linking the Fortran programs with the option `+T`. This *may*

be necessary to make the Fortran environment correctly handle interrupts used by MPICH to create connections to other processes.

9.6 Trouble with Input and Output

9.6.1 General

1. **Q:** I want output from `printf` to appear immediately.

   A: This is really a feature of your C and/or Fortran runtime system. For C, consider

   ```
   setbuf( stdout, (char *)0 );
   ```

9.6.2 IBM SP

1. **Q:** I have code that prompts the user and then reads from standard input. On IBM SPx systems, the prompt does not appear until *after* the user answers the prompt!

   A: This is a feature of the IBM POE system. There is a POE routine, `mpc_flush(1)`, that you can use to flush the output. Read the man page on this routine; it is synchronizing over the entire job and cannot be used unless all processes in `MPI_COMM_WORLD` call it. Alternately, you can always end output with the newline character (\); this will cause the output to be flushed but will also put the user's input on the next line.

9.6.3 Workstation Networks

1. **Q:** I want standard output (`stdout`) from each process to go to a different file.

   A: MPICH has no built-in way to do this. In fact, it prides itself on gathering the stdouts for you. You can do one of the following:

   (a) Use Unix built-in commands for redirecting `stdout` from inside your program (`dup2`, etc.). The MPE routine `MPE_IO_Stdout_to_file`, in '`mpe/mpe_io.c`', shows one way to do this. Note that in Fortran, the approach of using `dup2` will work only if the Fortran `PRINT` writes to `stdout`. This is common but by no means universal.

   (b) Write explicitly to files instead of to `stdout` (use `fprintf` instead of `printf`, etc.). You can create the file name from the process's rank. This is the most portable way.

9.7 Upshot and Nupshot

The `upshot` and `nupshot` programs require specific versions of the `tcl` and `tk` languages. This section describes only problems that may occur once these tools have been successfully built.

9.7.1 General

1. **Q:** When I try to run `upshot` or `nupshot`, I get

   ```
   No display name and no $DISPLAY environment variables
   ```

   A: Your problem is with your X environment. Upshot is an X program. If your workstation name is "'`foobar.kscg.gov.tw`'", then before running any X program, you need to do

   ```
     setenv DISPLAY foobar.kscg.gov.tw:0
   ```

   If you are running on some other system and displaying on foobar, you might need to do

   ```
     xhost +othermachine
   ```

   on foobar, or even

   ```
     xhost +
   ```

   which gives all other machines permission to write on foobar's display.

   If you do not have an X display (you are logged in from a Windows machine without an X capability) then you cannot use `upshot`.

2. **Q:** When trying to run `upshot`, I get

   ```
   upshot: Command not found.
   ```

   A: First, check that `upshot` is in your path. You can use the command

   ```
   which upshot
   ```

   to do this.

   If it is in your path, the problem may be that the name of the `wish` interpreter is too long for your Unix system. Look at the first line of the '`upshot`' file. It should be something like

   ```
   #! /usr/local/bin/wish -f
   ```

   If it is something like

   ```
   #! /usr/local/tcl7.4-tk4.2/bin/wish -f
   ```

   this may be too long of a name (some Unix systems restrict this first line to a mere 32 characters). To fix this, you'll need to put a link to '`wish`' somewhere where the name will be short enough. Alternately, you can start `upshot` with

   ```
      /usr/local/tcl7.4-tk4.2/bin/wish -f /usr/local/mpich/bin/upshot
   ```

9.7.2 HP-UX

1. **Q:** When trying to run `upshot` under HP-UX, I get error messages like

   ```
   set: Variable name must begin with a letter.
   ```

   or

   ```
   upshot: syntax error at line 35: '(' unexpected
   ```

   A: Your version of HP-UX limits the shell names for very short strings. `Upshot` is a program that is executed by the `wish` shell, and for some reason HP-UX is both refusing to execute in this shell and then trying to execute the `upshot` program using your current shell (e.g., '`sh`' or '`csh`'), instead of issuing a sensible error message about the command name being too long. There are two possible fixes:

   (a) Add a link with a much shorter name, for example

   ```
   ln -s /usr/local/tk3.6/bin/wish /usr/local/bin/wish
   ```

   Then edit the `upshot` script to use this shorter name instead. This may require root access, depending on where you put the link.

   (b) Create a regular shell program containing the lines

   ```
   #! /bin/sh
   /usr/local/tk3.6/bin/wish -f /usr/local/mpi/bin/upshot
   ```

   (with the appropriate names for both the '`wish`' and '`upshot`' executables).

   Also, file a bug report with HP. At the very least, the error message here is wrong; also, there is no reason to restrict general shell choices (as opposed to login shells).

Appendices

A Automatic generation of profiling libraries

The profiling wrapper generator (wrappergen) has been designed to complement the MPI profiling interface. It allows the user to write any number of 'meta' wrappers which can be applied to any number of MPI functions. Wrappers can be in separate files, and can nest properly, so that more than one layer of profiling may exist on indiividual functions.

Wrappergen needs three sources of input:

1. A list of functions for which to generate wrappers.
2. Declarations for the functions that are to be profiled. For speed and parsing simplicity, a special format has been used. See the file 'proto'.
3. Wrapper definitions.

The list of functions is simply a file of whitespace-separated function names. If omitted, any forallfn or fnall macros will expand for every function in the declaration file.

A.1 Writing wrapper definitions

Wrapper definitions themselves consist of C code with special macros. Each macro is surrounded by the {{ }} escape sequence. The following macros are recognized by wrappergen:

```
{{fileno}}
```

An integral index representing which wrapper file the macro came from. This is useful when declaring file-global variables to prevent name collisions. It is suggested that all identifiers declared outside functions end with _{{fileno}}. For example:

```
static double overhead_time_{{fileno}};
```

might expand to:

```
static double overhead_time_0;
```

(end of example).

```
{{forallfn <function name escape> <function A> <function B> ... }}
   ...
{{endforallfn}}
```

The code between {{forallfn}} and {{endforallfn}} is copied once for every function profiled, except for the functions listed, replacing the escape string specified by <function name escape> with the name of each function. For example:

```
{{forallfn fn_name}}static int {{fn_name}}_ncalls_{{fileno}};
{{endforallfn}}
```

might expand to:

```
static int MPI_Send_ncalls_1;
static int MPI_Recv_ncalls_1;
static int MPI_Bcast_ncalls_1;
```

(end of example)

```
{{foreachfn <function name escape> <function A> <function B> ... }}
  ...
{{endforeachfn}}
```

{{foreachfn}} is the same as {{forallfn}} except that wrappers are written only the functions named explicitly. For example:

```
{{forallfn fn_name mpi_send mpi_recv}}
   static int {{fn_name}}_ncalls_{{fileno}};
{{endforallfn}}
```

might expand to:

```
static int MPI_Send_ncalls_2;
static int MPI_Recv_ncalls_2;
```

(end of example)

```
{{fnall <function name escape> <function A> <function B> ... }}
  ...
  {{callfn}}
  ...
{{endfnall}}
```

{{fnall}} defines a wrapper to be used on all functions except the functions named. Wrappergen will expand into a full function definition in traditional C format. The {{callfn}} macro tells wrappergen where to insert the call to the function that is being profiled. There must be exactly one instance of the {{callfn}} macro in each wrapper definition. The macro specified by `<function name escape>` will be replaced by the name of each function.

Within a wrapper definition, extra macros are recognized.

`{{vardecl <type> <arg> <arg> ... }}`

Use `vardecl` to declare variables within a wrapper definition. If nested macros request variables through vardecl with the same names, wrappergen will create unique names by adding consecutive integers to the end of the requested name (var, var1, var2, ...) until a unique name is created. It is unwise to declare variables manually in a wrapper definition, as variable names may clash with other wrappers, and the variable declarations may occur later in the code than statements from other wrappers, which is illegal in classical and ANSI C.

`{{<varname>}}`

If a variable is declared through `vardecl`, the requested name for that variable (which may be different from the uniquified form that will appear in the final code) becomes a temporary macro that will expand to the uniquified form. For example,

```
{{vardecl int i d}}
```

may expand to:

```
int i, d3;
```

(end of example)

`{{<argname>}}`

Suggested but not neccessary, a macro consisting of the name of one of the arguments to the function being profiled will be expanded to the name of the corresponding argument. This macro option serves little purpose other than asserting that the function being profilied does indeed have an argument with the given name.

`{{<argnum>}}`

Arguments to the function being profiled may also be referenced by number, starting with 0 and increasing.

`{{returnVal}}`

`ReturnVal` expands to the variable that is used to hold the return value of the function being profiled.

`{{callfn}}`

`callfn` expands to the call of the function being profiled. With nested wrapper definitions, this also represents the point at which to insert the code for any inner nested functions. The nesting order is determined by the order in which the wrappers are encountered by wrappergen. For example, if the two files '`prof1.w`' and '`prof2.w`' each contain two wrappers for `MPI_Send`, the profiling code produced when using both files will be of the form:

```
int MPI_Send( args...)
arg declarations...
{
   /*pre-callfn code from wrapper 1 from prof1.w */
   /*pre-callfn code from wrapper 2 from prof1.w */
   /*pre-callfn code from wrapper 1 from prof2.w */
   /*pre-callfn code from wrapper 2 from prof2.w */

   returnVal = MPI_Send( args... );
```

```
        /*post-callfn code from wrapper 2 from prof2.w */
        /*post-callfn code from wrapper 1 from prof2.w */
        /*post-callfn code from wrapper 2 from prof1.w */
        /*post-callfn code from wrapper 1 from prof1.w */

        return returnVal;
     }
```

```
{{fn <function name escape>  <function A> <function B> ... }}
  ...
  {{callfn}}
  ...
{{endfnall}}
```

fn is identical to fnall except that it only generates wrappers for functions named explicitly. For example:

```
{{fn this_fn MPI_Send}}
  {{vardecl int i}}
  {{callfn}}
  printf( "Call to {{this_fn}}.\n" );
  printf( "{{i}} was not used.\n" );
  printf( "The first argument to {{this_fn}} is {{0}}\n" );
{{endfn}}
```

will expand to:

```
int  MPI_Send( buf, count, datatype, dest, tag, comm )
void * buf;
int count;
MPI_Datatype datatype;
int dest;
int tag;
MPI_Comm comm;
{
  int  returnVal;
  int i;
  returnVal = PMPI_Send( buf, count, datatype, dest, tag, comm );
  printf( "Call to MPI_Send.\n" );
  printf( "i was not used.\n" );
  printf( "The first argument to MPI_Send is buf\n" );
  return returnVal;
}
```

A sample wrapper file is in 'sample.w' and the corresponding output file is in 'sample.out'.

B Options for mpirun

The options for mpirun[4], as shown by mpirun -help, are

```
mpirun [mpirun_options...] <progname> [options...]

  mpirun_options:
    -arch <architecture>
            specify the architecture (must have matching machines.<arch>
            file in ${MPIR_HOME}/bin/machines) if using the execer
    -h      This help
    -machine <machine name>
            use startup procedure for <machine name>
    -machinefile <machine-file name>
            Take the list of possible machines to run on from the
            file <machine-file name>
    -np <np>
            specify the number of processors to run on
    -nolocal
            don't run on the local machine (only works for
            p4 and ch_p4 jobs)
    -stdin filename
            Use filename as the standard input for the program.  This
            is needed for programs that must be run as batch jobs, such
            as some IBM SP systems and Intel Paragons using NQS (see
            -paragontype below).
    -t      Testing - do not actually run, just print what would be
            executed
    -v      Verbose - throw in some comments
    -dbx    Start the first process under dbx where possible
    -gdb    Start the first process under gdb where possible
    -xxgdb  Start the first process under xxgdb where possible
             (on the Meiko, selecting either -dbx or -gdb starts prun
             under totalview instead)

    Options for the globus device:

    With the exception of -h, these are the only mpirun options supported
    by the globus device.

    -machinefile <machine-file name>
            Take the list of possible machines to run on from the
            file <machine-file name>
    -np <np>
            specify the number of processors to run on
```

[4] Not all the options are supported by the globus device. See the section "Options for the globus device" in this appendix.

```
-leave_pg
        Don't delete the globus RSL file after running
-globusargs dumprsl | {stage, nostage}
        dumprsl - display the RSL string that would have been
                  used to submit the job.  using this option
                  does not run the job.
        {stage, nostage} - stage takes the executable named on
                           the mpirun command and transfers it
                           to all the executable machines
                           (~/.gass_cache) and removes it
                           after executing.
                           this only works when all the executable
                           machines are binary compatible.
                           the default value is nostage.
        when specifying more than one value to -globusargs, you
        must surrond them all with double quotation marks, e.g.,
        -globusargs "dumprsl stage"
-globusrsl <globus-rsl-file name>
        <globus-rsl-file name> must contain a Globus RSL
        string.  When using this option all other mpirun options
        are ignored and -leave_pg is implied.


Special Options for Batch Environments:

-mvhome Move the executable to the home directory.  This
        is needed when all file systems are not cross-mounted
        Currently only used by anlspx
-mvback files
        Move the indicated files back to the current directory.
        Needed only when using -mvhome; has no effect otherwise.
-maxtime min
        Maximum job run time in minutes.  Currently used only
        by anlspx.  Default value is $max_time minutes.
-nopoll Do not use a polling-mode communication.
        Available only on IBM SPx.
-mem value
       This is the per node memory request (in Mbytes).  Needed for some
       CM-5s. ( Default $max_mem. )

-cpu time
       This is the the hard cpu limit used for some CM-5s in
       minutes. (Default $maxtime minutes.)

Special Options for IBM SP2:

-cac name
        CAC for ANL scheduler.  Currently used only by anlspx.
```

```
        If not provided will choose some valid CAC.

Special Options for Intel Paragon:

-paragontype name
       Selects one of default, mkpart, NQS, depending on how you want
       to submit jobs to a Paragon.

-paragonname name
       Remote shells to name to run the job (using the -sz method) on
       a Paragon.

-paragonpn name
       Name of partition to run on in a Paragon (using the -pn name
       command-line argument)
```

On exit, `mpirun` returns a status of zero unless `mpirun` detected a problem, in which case it returns a non-zero status (currently, all are one, but this may change in the future).

Multiple architectures may be handled by giving multiple `-arch`[5] and `-np` arguments. For example, to run a program on 2 sun4s and 3 rs6000s, with the local machine being a sun4, use

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program
```

This assumes that program will run on both architectures. If different executables are needed, the string '%a' will be replaced with the arch name. For example, if the programs are `program.sun4` and `program.rs6000`, then the command is

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 program.%a
```

If instead the executables are in different directories; for example, '`/tmp/me/sun4`' and '`/tmp/me/rs6000`', then the command is

```
mpirun -arch sun4 -np 2 -arch rs6000 -np 3 /tmp/me/%a/program
```

It is important to specify the architecture with `-arch` *before* specifying the number of processors. Also, the *first* `arch` command must refer to the processor on which the job will be started. Specifically, if `-nolocal` is NOT specified, then the first `-arch` must refer to the processor from which mpirun is running.

C MPI, Globus, and GUSTO

In the section we describe how to run MPI programs using the MPICH Globus device in the GUSTO distributed computing environment.

[5] `-arch` is not supported by the `globus` device.

C.1 Using the globus device on GUSTO

Three features of the GUSTO environment make the use of the globus device particularly easy:

- The public-key-based implementation of Globus security services are used; this allows you to log on just once and then access resources at a variety of GUSTO sites. (However, you must first set up your Globus security environment: see below).
- Globus process creation servers called Globus Resource Allocation Managers (GRAMs) or simply *resource managers* already exist on the various GUSTO resources.
- You have access to a Lightweight Directory Access Protocol (LDAP) database called the *Metacomputing Directory Service (MDS)* where information describing Globus installations is stored. We shall see how this is used below.

Perform the following experiment to verify that your machine has the Globus client software installed and configured to use GUSTO. Use the following `ldapsearch` command to query MDS for the names of all GUSTO resource managers:[6],

```
% <globusinstalldir>/bin/ldapsearch "mn=*" | grep ^mn
```

which should result in something similar to this

```
mn=bolas.isi.edu-fork@globus.org, ou=ISI, o=University of Southern California, o=Globus, c=US
mn=bolas.isi.edu-fork@globus.org
mn=fr1n12.mhpcc.edu-fork@globus.org, o=Maui High Performance Computing Center, o=Globus, c=US
mn=fr1n12.mhpcc.edu-fork@globus.org
mn=fr37n13.mhpcc.edu-fork@globus.org, o=Maui High Performance Computing Center, o=Globus, c=US
mn=fr37n13.mhpcc.edu-fork@globus.org
mn=sif.ncsa.uiuc.edu-lsf@globus.org, o=NCSA, o=Globus, c=US
mn=sif.ncsa.uiuc.edu-lsf@globus.org
mn=pitcairn.mcs.anl.gov-fork@globus.org, ou=MCS, o=Argonne National Laboratory, o=Globus, c=US
mn=pitcairn.mcs.anl.gov-fork@globus.org
```

C.1.1 Setting up security for GUSTO

Because GUSTO uses public key security, you must spend a few minutes setting up your security environment before you can start using the globus device on GUSTO. This entails obtaining a private key from the GUSTO certificate authority (CA) and setting up some local configuration information. This process is somewhat complex but only has to be performed once. See `http://www.globus.org/security/tutorial.html` for a more detailed discussion of these topics.

1. Set your `GLOBUS_DIR` environment variable to point to the top level Globus directory:

   ```
   % setenv GLOBUS_DIR <globusinstalldir>
   ```

2. Augment your path to include the bin directories for both Globus and the SSL libraries, which must also be installed to use the globus device on GUSTO (see http://www.globus.org for details). These are

[6] Complete instructions regarding how to use `ldapsearch` to query MDS can be found at http://www.globus.org.

```
$GLOBUS_DIR/bin
<sslinstalldir>/bin
```

3. Make a directory in which to store your certificate and key:

```
% mkdir ~/cert
% cd ~/cert
```

4. Run the SSL certreq program to generate your certificate request and private key:

```
% certreq
```

Something like this will appear upon your screen. It may take a few seconds to finish generating the private key.

```
Using configuration from
/soft/pub/packages/SSLeay-0.8.1/etc/ssleay.globus.cnf
Generating a 1024 bit RSA private key
................................+++++
......................+++++
writing new private key to 'newkey.pem'
Enter PEM pass phrase:
```

At this point, you should enter in a pass phrase. A pass phrase is basically a password, except that it can be longer (64 characters) and can include spaces. It will immediately ask you to re-enter your password, for verification. The subsequent screen output will be:

```
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
```

You can press return to answer any question with the default value (within the brackets []). Otherwise, enter the appropriate information. The next few questions are as follows:

```
Main Organization [Globus]:
Home Globus Site [Argonne National Laboratory]:
Organizational Unit Name (eg, section) [MCS]:
name (eg, Globus id without the @globus.org) []:
```

For name, enter your old "globusid", which tended to be your username or last name, (e.g., smartin, wsmith, foster).

The next questions pertain to optional information not currently being used by the authentication program. You may simply hit return to all further questions. The screen output will be:

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Private key is in newkey.pem
Request is in newreq.pem
E-Mail the newreq.pem file to Globus CA:
ca@globus.org
```

At this point, you are done with the certreq script. It will return you to the UNIX prompt, having created 2 new files in your directory, `newkey.pem` and `newreq.pem`.

5. Mail the newreq.pem file to the GUSTO CA:

   `% mail ca@globus.org < newreq.pem`

   You should receive a response from the CA with your signed certificate.

6. Store the certificate received from the GUSTO CA (including the `----BEGIN CERTIFICATE----` and `----END CERTIFICATE----` delimiters) in the file `~/cert/newcert.pem`.

7. Set protection modes on the files in `~/cert`:

   ```
   % chmod 444 newcert.pem
   % chmod 400 newkey.pem
   ```

   These protections are essential if you are to insure the security and integrity of your private key and certificate. Basically they ensure that the certificate is world readable but unalterable, while the private key file is readable only by you, the user, and also unalterable.

8. Use the `setenv` command to set the environment variables used to locate your certificate file, private key, and trusted certificates:

   ```
   % cd
   % setenv X509_CERT_DIR $GLOBUS_DIR/share
   % setenv X509_USER_CERT cert/newcert.pem
   % setenv X509_USER_KEY cert/newkey.pem
   % setenv X509_USER_PROXY cert/newproxy.pem
   ```

   The first of these commands indicates the directory containing the trusted certificate; the second the filename of the file containing your certificate; the third the filename of the file containing your private key; and the fourth the filename in which your temporary proxy certificate and key should be stored.

9. Enable access to the resources that you wish to use. Before you can use any GUSTO resource, you must (a) have an account on those resources, and (b) have any entry in an access control list called a `globusmap` file associated with the resource. You must email the Globus administrator of each resource in question.

C.1.2 Authenticating yourself to GUSTO

Now that you have set up your Globus security environment, you are ready to run some MPI programs. As a first step, we authenticate ourselves ("log on") to GUSTO by running the command `cinit`. This command creates a *temporary credential*, which allows you to use GUSTO resources for a certain amount of time. This credential is stored in a file, the name of which must be recorded in the environment variable `X509_USER_PROXY`. Hence, we might type:

```
% setenv X509_USER_PROXY /tmp/my_temporary_cert
% cinit -out $X509_USER_PROXY
```

The temporary credential is valid for a default period of time, typically 12 hours. You can change this period by using the `-hours` flag on the `cinit` command, for example:

```
% setenv X509_USER_PROXY /tmp/my_temporary_cert
% cinit -hours 6 -out $X509_USER_PROXY
```

You may acquire a new security credential at any time. If you acquire a new credential before a previous one expires, the new credential simply overwrites the old one. **Attempting to start an MPI application in the presence of an expired security credential will result in failure.**

C.1.3 Using `mpirun` on GUSTO

Before typing your first `mpirun` command you must identify the computers on which you wish to run your application. This is done by listing the *manager names* associated with these computers in a `machines` file on your Globus client. For example, the following `machines` file

```
ico16.mcs.anl.gov-easymcs@globus.org 10
sp023e.sp.uh.edu-loadleveler@globus.org 5
```

names managers associated with two IBM SPs, one at Argonne and the other at the University of Houston. Assuming the `machines` file and executable `myapp` are in your current directory, you can then start your application as follows:

```
% mpirun -np 15 -globusargs stage myapp
```

This command loads `myapp` on 10 nodes on the Argonne SP, transfers a copy of `myapp` to the Houston SP (`-globusargs stage`), and loads `myapp` on 5 nodes there. (The number of nodes to create on each machine are specified by the counts appearing at the end of each line in the `machines` file.) Issues of authentication and the submission of appropriate requests to the Argonne and Houston SP schedulers are handled automatically. All 15 nodes behave as a single MPI application, i.e., a single `MPI_COMM_WORLD` with nodes ranked 0 through 14. Standard output and standard error are routed back to the originating node. Hence, the behavior is identical to that of an MPI program running on a single 15-node parallel computer.

Determining resource manager names. This example assumes that you know the resource manager names for the computers on which you want to run. To learn the manager name for a particular machine[7], e.g., `pitcairn.mcs.anl.gov`, use the `ldapsearch` command introduced earlier:

```
yukon% ldapsearch "hn=pitcairn.mcs.anl.gov" | grep ^mn
mn=pitcairn.mcs.anl.gov-fork@globus.org, ou=MCS, o=Argonne National Laboratory,
o=Globus, c=US
mn=pitcairn.mcs.anl.gov-fork@globus.org
```

which returns two different names for the one GRAM running on `pitcairn.mcs.anl.gov`. The first, longer name is called the *distinguished name*; we are only interested in the second, shorter name, `pitcairn.mcs.anl.gov-fork@globus.org`.

To find all the manager names at a particular location, e.g., Argonne National Laboratory,

```
ldapsearch "mn=*" mn | grep ^mn | grep "anl.gov"
mn=pitcairn.mcs.anl.gov-fork@globus.org, ou=MCS, o=Argonne National Laboratory,
o=Globus, c=US
mn=pitcairn.mcs.anl.gov-fork@globus.org
mn=ico16.mcs.anl.gov-fork@globus.org, ou=MCS, o=Argonne National Laboratory, o=G
lobus, c=US
mn=ico16.mcs.anl.gov-fork@globus.org
mn=ico16.mcs.anl.gov-easymcs@globus.org, ou=MCS, o=Argonne National Laboratory,
o=Globus, c=US
mn=ico16.mcs.anl.gov-easymcs@globus.org
```

Specifying node counts. The optional numbers appearing at the end of each line in a `machines` file (default value = 1) are used to determine the maximum number of nodes to create on each machine. Hence, given the `machines` file above, the command `mpirun -np 8` would start just 8 nodes on the Argonne SP. `mpirun` "wraps around" the `machines` file, so `-np 18` would create 10 nodes on the Argonne SP, 5 nodes at Houston, and a further 3 nodes at Argonne. These three groups of nodes comprise three distinct "subjobs." This distinction is important because communication within a subjob can be very different from communication between subjobs, particularly on MPPs such as the IBM SP. Communication

[7] Although a single machine typically has one GRAM, multiple GRAMs may run on a single machine.

between subjobs is always done using TCP/IP. Communication within a subjob is done using the fastest protocol available, for example IBM's MPL.

Staging. In the above example, we used the `-globusargs stage` option to request that our application be staged to the computers on which we wanted to run. Staging works here because the computers in question are binary compatible. If even one of the platforms listed in the `machines` file is not binary compatible then you may not use the `-globusargs stage` option. Instead, you must either stage executables manually, prior to running the program, or use the more flexible staging commands described in C.1.4 below.

If you wanted to run an application on a cluster of binary compatible workstations (one process on each) that all share the same filesystem, then staging is not required. In this case you write a `machines` file listing the Globus servers in your cluster, e.g.,

```
pitcairn.mcs.anl.gov-fork@globus.org
tuva.mcs.anl.gov-fork@globus.org
```

and omit the `-globusargs stage` option from the `mpirun` command:

```
% mpirun -np 2 myapp
```

Locating the `machines` file. The `mpirun` command determines which `machines` file to use as follows:

1. If a `-machinefile <machinefilename>` argument is specified on the `mpirun` command, it uses that; otherwise,
2. it looks for a file `machines` in the directory in which you typed `mpirun`; and finally,
3. it looks for a `machines` file in `<mpidir>/lib/<arch>/globus` where `<mpidir>` is the directory where you built MPICH and `<arch>` is the architecture MPICH was built on, e.g., `solaris`, `IRIX64`, ...

If none of these information are provided, then `mpirun` fails.

C.1.4 Advanced features of the globus device

As noted above, the `-globusargs stage` command does not support staging when machines are not binary compatible. In this situation, we must currently use something called a *Resource Specification Language (RSL)* request to specify the executable filename for each machine. This technique is very flexible, but rather complex; work is currently underway to simplify the manner in which these issues are addressed.

The easiest way to learn how to write your own RSL request is to study the one generated for you by mpirun. Consider the example where we wanted to run an application on a cluster of workstations. Recall our `machines` file looked like this:

```
pitcairn.mcs.anl.gov-fork@globus.org
tuva.mcs.anl.gov-fork@globus.org
```

To view the RSL request generated in this situation, *without* actually launching the program, we type the following `mpirun` command:

```
% mpirun -globusargs dumprsl -np 2 myapp 123 456
```

which produces the following output:

```
+
( &(resourceManagerContact="pitcairn.mcs.anl.gov:8711:pitcairn.mcs.anl.gov-f
ork@globus.org")
   (count=1)
   (label="subjob 0")
   (arguments=" 123 456")
   (directory=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus)
   (executable=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus/myapp)
)
( &(resourceManagerContact="tuva.mcs.anl.gov:8711:tuva.mcs.anl.gov-fork@glob
us.org")
   (count=1)
   (label="subjob 1")
   (arguments=" 123 456")
   (directory=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus)
   (executable=/homes/karonis/MPI/mpich.yukon/mpich/lib/IRIX64/globus/myapp)
)
```

This RSL specification specifies such things as the "contact string" for the GRAMs that we will be using, the number of nodes that we want to create, a unique label for each subjob, our home directory, and the name of the executable that we will be using.

We can write a custom RSL request by simply modifying the request obtained above. For example, if we want to use a different executable on the two different machines, we can simply change the executable name. Once we have made this change, we use the `-globusrsl` option to supply this modified request to `mpirun`:

```
% mpirun -globusrsl <myrslrequestfile>
```

where `<myrslrequestfile>` is the filename of your RSL reqeust. You do not specify any other arguments to `mpirun` (e.g., `-np`, executable, command line arguments, etc.), and even the `machines` file is ignored, as all required information is contained in the RSL request.

RSL is a flexible language capable of doing much more than has been presented here. For example, it can be used to stage executables and to set environment variables on remote computers before starting execution. A full description of the language can be found at http://www.globus.org.

Acknowledgments

The work described in this report has benefited from conversations with and use by a large number of people. We also thank those that have helped in the implementation of MPICH, particularly Patrick Bridges and Edward Karrels. Particular thanks goes to Nathan Doss and Anthony Skjellum for valuable help in the implementation and development of MPICH. More recently, Debbie Swider has joined the MPICH team. The Globus device was implemented by Nick Karonis of Argonne.. The C++ bindings were implemented by Andrew Lumsdaine and Jeff Squyres of the University of Notre Dame. The ROMIO MPI-2 parallel I/O subsystem was implemented by Rajeev Thakur of Argonne.

References

[1] Ralph Butler and Ewing Lusk. User's guide to the `p4` parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, 1992.

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

[3] William Gropp and Ewing Lusk. Installation guide for `mpich`, a portable implementation of MPI. Technical Report ANL-96/5, Argonne National Laboratory, 1996. The updated version is at `ftp://ftp.mcs.anl.gov/pub/mpi/install.ps`.

[4] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1994.

[5] M. T. Heath. Recent developments and case studies in performance visualization using ParaGraph. In G. Haring and G. Kotsis, editors, *Performance Measurement and Visualization of Parallel Systems*, pages 175–200, Amsterdam, The Netherlands, 1993. Elsevier Science Publishers.

[6] Virginia Herrarte and Ewing Lusk. Studying parallel program behavior with `upshot`. Technical Report ANL–91/15, Argonne National Laboratory, Argonne, IL 60439, 1991.

[7] Edward Karrels and Ewing Lusk. Performance analysis of MPI programs. In Jack Dongarra and Bernard Tourancheau, editors, *Proceedings of the Workshop on Environments and Tools For Parallel Scientific Computing*. SIAM Publications, 1994.

[8] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994.

[9] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, 1995.