Message Passing Toolkit: MPI Programmer's Manual

SR–2197 1.2

Document Number 007–3687–001

New Features

Message Passing Toolkit: MPI Programmer's Manual SR–2197 1.2

This rewrite of *Message Passing Toolkit: MPI Programmer's Manual*, publication SR–2197, supports the 1.2 release of the Cray Message Passing Toolkit and the Message Passing Toolkit for IRIX (MPT), and documents the integration of MPI with Silicon Graphics Array Services on UNICOS and UNICOS/mk systems.

Record of Revision

Version	*Description*
1.0	January 1996 Original Printing. This manual documents the Message Passing Toolkit implementation of the Message Passing Interface (MPI).
1.1	August 1996 This revision supports the Message Passing Toolkit (MPT) 1.1 release.
1.2	January 1998 This revision supports the Message Passing Toolkit (MPT) 1.2 release for UNICOS, UNICOS/mk, and IRIX systems.

Contents

Page

Preface

This publication documents the Cray Message Passing Toolkit and Message Passing Toolkit for IRIX (MPT) 1.2 implementation of the Message Passing Interface (MPI) supported on the following platforms:

- Cray PVP systems running UNICOS release 9.0.2.7 or later or UNICOS release 10.0 or later. The MPT 1.2 release requires a bugfix package to be installed on UNICOS systems running release 9.0.2.7 or 10.0. The bugfix package, `MPT12_OS_FIXES`, is available through the `getfix` utility. It is also available from the anonymous FTP site `ftp.cray.com` in directory `/pub/mpt/fixes/MPT12_OS_FIXES`.

  MPT 1.2 is not supported on systems running UNICOS release 9.3 or 9.3.0.1.

- CRAY T3E systems running UNICOS/mk release 1.5 or later

- Silicon Graphics MIPS based systems running IRIX release 6.2 or later

  IRIX 6.2 systems running MPI require the kernel rollup patch 1650 or later.

  IRIX 6.3 systems running MPI require the kernel rollup patch 2328 or later.

IRIX systems running MPI applications must also be running Array Services software version 3.0 or later. MPI consists of a library, a profiling library, and commands that support MPI. The MPT 1.2 release is a software package that supports parallel programming across a network of computer systems through a technique known as *message passing*.

Related Publications

The following documents contain additional information that might be helpful:

- *Message Passing Toolkit: PVM Programmer's Manual*, publication SR–2196
- *Application Programmer's Library Reference Manual*, publication SR–2165
- *Installing Programming Environment Products*, publication SG–5191

All of these are Cray Research publications and can be ordered from Cray Research. For ordering information, see "Ordering Publications."

Other Sources

Material about MPI is available from a variety of other sources. Some of these, particularly World Wide Web pages, include pointers to other resources. Following is a grouped list of these sources:

The MPI standard:

- As a technical report: University of Tennessee report (reference [24] from *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, by Gropp, Lusk, and Skjellum)
- As online PostScript or hypertext on the World Wide Web:

  `http://www.mpi-forum.org/`
- As a journal article in the fall issue of the *International Journal of Supercomputer Applications*, volume 8, number 3/4, 1994
- As text through the IRIS InSight library (for customers with access to this tool)

Books:

- *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, by Gropp, Lusk, and Skjellum, publication TPD–0011

Newsgroup:

- `comp.parallel.mpi`

Ordering Publications

Silicon Graphics maintains publications information at the following URLs:

`http://techpubs.sgi.com/library`

`http://techpubs.sgi.com/infosearch`

These websites contain information that allows you to browse documents online, order documents, and send feedback to Silicon Graphics.

Cray Research also has documents available online at the following URL:

`http://www.cray.com/swpubs`

The *User Publications Catalog*, publication CP–0099, describes the availability and content of all Cray Research hardware and software documents for customers.

Silicon Graphics and Cray Research customers who subscribe to the Cray Inform (CRInform) program can access this information on the CRInform system.

To order a Cray Research or Silicon Graphics document, either call the Distribution Center in Mendota Heights, Minnesota, at +1–612–683–5907, or send a facsimile of your request to fax number +1–612–452–0141. Cray Research employees may send electronic mail to `orderdsk` (UNIX system users).

Customers who subscribe to the CRInform program can order software release packages electronically by using the `Order Cray Software` option.

Customers outside of the United States and Canada should contact their local service organization for ordering and documentation information.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers:

1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs
4	Devices (special files)
4P	Protocols
5	File formats
7	Miscellaneous topics
7D	DWB-related information

	8 Administrator commands
	Some internal routines (for example, the `_assign_asgcmd_info`() routine) do not have man pages associated with them.
variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

In this manual, references to Cray PVP systems include the following machines:

- CRAY C90 series
- CRAY C90D series
- CRAY EL series (including CRAY Y-MP EL systems)
- CRAY J90 series
- CRAY Y-MP E series
- CRAY Y-MP M90 series
- CRAY T90 series

Silicon Graphics systems include all MIPS based systems running IRIX 6.2 or later.

The following operating system terms are used throughout this document.

Term	Definition
UNICOS	Operating system for all configurations of Cray PVP systems
UNICOS/mk	Operating system for all configurations of CRAY T3E systems
UNICOS MAX	Operating system for all configurations of CRAY T3D systems

IRIX	Operating system for all configurations of MIPS based systems

The default shell in the UNICOS and UNICOS/mk operating systems, referred to in Cray Research documentation as the *standard shell*, is a version of the Korn shell that conforms to the following standards:

- Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) Standard 1003.2–1992
- X/Open Portability Guide, Issue 4 (XPG4)

The UNICOS and UNICOS/mk operating systems also support the optional use of the C shell.

Cray UNICOS version 10.0 is an X/Open Base 95 branded product.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. You can contact us in any of the following ways:

- Send us electronic mail at the following address:

  `publications@cray.com`

- Contact your customer service representative and ask that an SPR or PV be filed. If filing an SPR, use `PUBLICATIONS` for the group name, `PUBS` for the command, and `NO-LICENSE` for the release name.
- Call our Software Publications Group in Eagan, Minnesota, through the Customer Service Call Center, using either of the following numbers:

  1–800–950–2729 (toll free from the United States and Canada)

  +1–612–683–5600

- Send a facsimile of your comments to the attention of “Software Publications Group” in Eagan, Minnesota, at fax number +1–612–683–5599.

We value your comments and will respond to them promptly.

Overview [1]

The Cray Message Passing Toolkit and Message Passing Toolkit for IRIX (MPT) is a software package that supports parallel programming across a network of computer systems through a technique known as *message passing*. This style of parallel programming is an explicit method in which the application requests that data be sent from one task to another or between groups of tasks. MPT also provides support for shared parallel programming within a computer system through a technique known as *data passing*.

The MPT 1.2 package contains the following components and the appropriate accompanying documentation:

- Parallel Virtual Machine (PVM)
- Message Passing Interface (MPI)
- Logically shared, distributed memory (SHMEM) data-passing routines

The Message Passing Interface (MPI) is a standard specification for a message-passing interface, allowing portable message-passing programs in Fortran and C languages.

This chapter provides an overview of the MPI software that is included in the toolkit, a description of the basic MPI components, and a list of general steps for developing an MPI program. Subsequent chapters address the following topics:

- Building MPI applications
- Using `mpirun` to execute applications
- Setting environment variables
- Debugging programs on IRIX systems
- Launching programs with NQE

1.1 MPI Overview

MPI is a standard specification for a message-passing interface, allowing portable message-passing programs in Fortran and C languages. MPI was created by the Message Passing Interface Forum (MPIF). MPIF is not sanctioned or supported by any official standards organization. Its goal was to develop a widely used standard for writing message-passing programs. Silicon Graphics and Cray Research support implementations of MPI that are released as part of the Message Passing Toolkit. These implementations are available on IRIX, UNICOS, and UNICOS/mk systems. The MPI standard is available from the IRIS InSight library (for customers who have access to that tool), and is documented online at the following address:

`http://www.mcs.anl.gov/mpi`

1.2 MPI Components

On UNICOS and UNICOS/mk systems, the MPI library is provided as a statically linked set of objects (a file with a name that ends in `.a`). On IRIX systems, the MPI library is provided as a dynamic shared object (DSO) (a file with a name that ends in `.so`). The basic components that are necessary for using MPI are the `libmpi.a` library (or `libmpi.so` for IRIX systems), the include files, and the `mpirun`(1) command.

For UNICOS/mk systems, in addition to the `libmpi.a` library, the `libpmpi.a` library is provided for profiling support. For UNICOS and IRIX systems, profiling support is included in the `libmpi.a` and `libmpi.so` libraries, respectively. Profiling support replaces all `MPI_` *Xxx* prototypes and function names with `PMPI_`*Xxx* entry points.

1.3 MPI Program Development

To develop a program that uses MPI, you must perform the following steps:

Procedure 1: Steps for MPI program development

1. Add MPI function calls to your application for MPI initiation, communications, and synchronization. For descriptions of these functions, see the online man pages or *Using MPI: Portable Parallel Programming with the Message-Passing Interface* or the MPI standard specification.

2. Build programs for the systems that you will use, as described in Chapter 2, page 5.

3. Execute your program by using one of the following methods:

   - `mpirun`(1) (available on all systems (see Chapter 3, page 11))

   - XMPI visualization tool (available on IRIX systems (see Chapter 5, page 25))

- Execute directly (available on UNICOS/mk systems (see Section 3.3.3, page 17))

Note: On IRIX systems, for information on how to execute MPI programs across more than one host or how to execute MPI programs that consist of more than one executable file, see Section 2.3, page 10.

Building MPI Applications [2]

This chapter provides procedures for building MPI applications on UNICOS, UNICOS/mk, and IRIX systems.

2.1 Building Applications on UNICOS Systems

On UNICOS systems, the MPI library supports either a shared memory or a Transmission Control Protocol (TCP) communications model.

The following section provides information about the shared memory model, and information about compiling and linking shared memory MPI programs.

Note: Software included in the 1.2 release of the Message Passing Toolkit is designed to be used with the Cray Programming Environment. When building an application that uses the shared memory version of MPI, you must be using the Programming Environment 3.0 release or later. Before you can access the Programming Environment, the `PrgEnv` module must be loaded. For more information on using modules, see *Installing Programming Environment Products*, publication SG–5191, or, if the Programming Environment has already been installed on your system, see the online ASCII file `/opt/ctl/doc/README`.

Section 2.1.3, page 9, provides information that applies to MPI programs that use TCP for communication.

2.1.1 Building Applications That Use Shared Memory MPI on UNICOS Systems

To build an executable file that makes use of shared memory, perform the following steps:

1. Convert all global and static data to `TASKCOMMON` data.

   In a multitasking environment, all members of the multitasking group can access all global or static data because they share one user address space. An MPI process is described as a separate address space. To emulate an MPI process in a multitasking environment, all global or static data that can be modified during the course of execution of a program must be treated as data local to each task. This is done by placing the data in `TASKCOMMON` blocks. `TASKCOMMON` storage is a mechanism that is used in multitasked programs to provide a separate copy of data for each member of the multitasking group. `TASKCOMMON` data is still globally accessible across functions within a multitasked program, but it is private to each specific task. Fortran examples of global or static data that must be placed in `TASKCOMMON` include data that resides in `COMMON` blocks and data that appears in `DATA` or `SAVE` statements. In C, you must place all data that is declared static (either locally or globally) or data declared at a global level (outside of any function) in `TASKCOMMON`.

Because changing your program so that all global and static data is private is tedious and makes a program less portable, Cray Research provides support in the form of compile-time command line options to do the conversions. You can convert most global and static data to `TASKCOMMON` data automatically by using the following command-line options:

- For C programs:

  ```
  cc -h taskprivate
  ```

- For Fortran programs:

  ```
  f90 -a taskcommon
  ```

When you are placing data in `TASKCOMMON`, there may be cases in which the compiler cannot do the conversion because of insufficient information. The compiler notes these cases by issuing a warning during compilation. For such cases, you must convert the data by hand. Most of the time, these cases are related to initialization that involves Fortran `DATA` or `SAVE` statements or C initialized static variables, and you might need to change only how or when the data is initialized for it to be successfully placed in `TASKCOMMON`.

The following is an example of a case that the compiler cannot handle:

```
int a;
int b = &a
```

If variable `a` resides in `TASKCOMMON`, its address will not be known until run time; therefore, the compiling system cannot initialize it. In this case, the initialization must be handled within the user program.

2. Use the `cc`(1) or `f90`(1) commands to build your shared memory MPI program, as in the following examples:

   C programs:

   ```
   cc -htaskprivate -D_MULTIP_ -L$MPTDIR/lib/multi file.c
   ```

For C programs, the `-D` and `-L` options are needed to access the reentrant version of `libc`. This version is required to provide safe access to `libc` routines in a multitasking environment. When the `mpt` module is loaded, the module software sets `$MPTDIR` automatically and points to the default MPT software library. (For information on using modules, see *Installing Programming Environment Products*, publication SG–5191.) To make compiling in C easier, the environment variable `$LIBCM` is also set automatically when the `mpt` module is loaded. You can use `$LIBCM` with the `cc`(1) command to request the reentrant version of `libc`. `$LIBCM` is set to the following value:

```
-D_MULTIP_ -L$MPTDIR/lib/multi
```

The following example uses `$LIBCM`:

```
cc -htaskprivate $LIBCM file.c
```

Fortran programs:

```
f90 -ataskcommon file.f
```

3. Select private I/O if private Fortran file unit numbers are desired.

   In a multitasking environment, Fortran unit numbers are, by default, shared by all members of the multitasking group. This behavior forces all files to be shared among MPI processes. The user can request that files be private to each MPI process by specifying the private I/O option on the `assign`(1) command. The examples in Table 1, page 8, request private I/O.

Table 1. `assign` examples

Example	Description
`assign -P private u:10`	Specifies that unit 10 should be private to any MPI process that opens it.
`assign -P private p:%`	Specifies that all named Fortran units should be private to any MPI process that opens them. This includes all units connected to regular files and excludes units such as 5 and 6, which are connected to `stdin`, `stdout`, or `stderr` by default.

`assign -P global u:0`
`assign -P global u:5`
`assign -P global u:6`
`assign -P global u:100`
`assign -P global u:101`
`assign -P global u:102`
	This set of `assign` commands can be used in conjunction with `assign -P private g:all` to retain units connected by default to `stdin`, `stdout`, and `stderr` as global units. A unit connected to these standard files cannot be a private unit.

For more information on private I/O functionality on Cray PVP systems, see the `assign`(1) man page.

4. Run the application. To start or run an application that uses the shared memory version of MPI, you must use the `-nt` option on the `mpirun`(1) command (for example, `mpirun -nt 4 compute`).

You should also consider using Autotasking instead of message passing whenever your application is run on a UNICOS system. The communications overhead for Autotasking is orders of magnitude less than that for sockets, even on the same system, so it might be better to have only one fully autotasked MPI process on the UNICOS system. In many cases, you might be able to achieve this simply by invoking the appropriate compiler options and sending a larger file of input data to the MPI process on the UNICOS system.

2.1.2 Shared Memory MPI Limitations

Emulating a shared memory environment with the use of Cray Research multitasking software might cause unexpected program behavior. The goal is to preserve the original behavior as much as possible. However, it is not efficient or productive to completely preserve the original MPI behavior in a multitasked environment. The intent is to document possible changes in behavior. For example, changes in behavior might occur with the use of signals; therefore, it is not recommended that signals be used with the shared memory version of MPI.

The shared memory implementation of MPI supports the running of only 32 MPI processes within a multitasking group. Because MPI processes must share CPU resources, running with more than the number of physical CPUs available on the UNICOS system will begin to degrade performance.

2.1.3 Building Files on UNICOS Systems When Using TCP

On UNICOS systems, after you have added the MPI function calls described in Procedure 1, step 1, page 2, a simple MPI program can be linked as follows:

```
cc -o compute compute.o
```

This command links the `compute.o` object code to produce the `compute` executable file.

If you are using more than one host, the executable files should be installed on the Cray Research systems that you will be using. By default, MPI uses the path name of the user-initiated executable file on all systems. You can override this file by using a process group file.

In some installations, certain hosts are connected in multiple ways. For example, an Ethernet connection may be supplemented by a high-speed FDDI ring. Usually, alternate host names are used to identify the high-speed connection. You must put these alternate names in your machine file.

If your hosts are connected in multiple ways, you must not use `local` in your machine file to identify the local host, but must use the name of the local host instead. For example, if hosts `host1` and `host2` have Asynchronous Transfer Mode (ATM) connected to `host1-atm` and `host2-atm`, respectively, the correct machine file is as follows:

```
host1-atm
host2-atm
```

2.2 Building Applications on UNICOS/mk Systems

On UNICOS/mk systems, after you have added MPI function calls to your program, as described in Procedure 1, step 1, page 2, you can compile and link an MPI program, as in the following examples:

```
cc -o compute compute.c
```

or

```
f90 -o compute -X4 compute.f
```

If you have loaded the `mpt` module, the directory that contains the MPI include files is automatically searched, and the MPI library is automatically loaded.

2.3 Building Applications on IRIX Systems

On IRIX systems, after you have added MPI function calls to your program, as described in Procedure 1, step 1, page 2, you can compile and link the program, as in the following examples:

To use the 64-bit MPI library:

```
cc -64 compute.c -lmpi
```

To use the 32-bit MPI library:

```
cc -n32 compute.c -lmpi
```

If you are using modules and have loaded the `mpt` module file, the directory that contains the MPI include files is automatically searched and the MPI library is automatically loaded; therefore, you do not need to specify the `-lmpi` option, and you can compile as in the following example:

```
cc -64 compute.c
```

Using mpirun to Execute Applications [3]

The mpirun(1) command is the primary job launcher for the MPT implementations of MPI. The mpirun command must be used whenever a user wishes to run an MPI application on IRIX or UNICOS systems (on IRIX systems, XMPI can be used in place of mpirun). On IRIX or UNICOS systems, you can run an application on the local host only (the host from which you issued mpirun) or distribute it to run on any number of hosts that you specify. Use of the mpirun command is optional for UNICOS/mk systems and currently supports only the -np option. Note that several MPI implementations available today use a job launcher called mpirun, and because this command is not part of the MPI standard, each implementation's mpirun command differs in both syntax and functionality.

3.1 Syntax of the mpirun Command

The format of the mpirun command for UNICOS and IRIX is as follows:

```
mpirun [global_options ] entry [: entry ... ]
```

The *global_options* operand applies to all MPI executable files on all specified hosts. The following global options are supported:

Option	Description
-a[rray] *array_name*	Specifies the array to use when launching an MPI application. By default, Array Services uses the default array specified in the Array Services configuration file, arrayd.conf.
-d[ir] *path_name*	Specifies the working directory for all hosts. In addition to normal path names, the following special values are recognized:

.	Translates into the absolute path name of the user's current working directory on the local host. This is the default.
~	Specifies the use of the value of $HOME as it is defined on each

machine. In general, this value can be different on each machine.

`-f[ile]` *file_name*	Specifies a text file that contains `mpirun` arguments.
`-h[elp]`	Displays a list of options supported by the `mpirun` command.
`-p[refix]` *prefix_string*	Specifies a string to prepend to each line of output from `stderr` and `stdout` for each MPI process. Some strings have special meaning and are translated as follows:

- `%g` translates into the global rank of the process producing the output. (This is equivalent to the rank of the process in `MPI_COMM_WORLD`.)
- `%G` translates into the number of processes in `MPI_COMM_WORLD`.
- `%h` translates into the rank of the host on which the process is running, relative to the `mpirun`(1) command line.
- `%H` translates into the total number of hosts in the job.
- `%l` translates into the rank of the process relative to other processes running on the same host.
- %L translates into the total number of processes running on the host.
- %@ translates into the name of the host on which the process is running.

For examples of the use of these strings, first consider the following code fragment:

```
main(int argc, char **argv)
{
   MPI_Init(&argc, &argv);

   printf("Hello world\n");
   MPI_Finalize();
}
```

Depending on how this code is run, the results of running the `mpirun` command will be similar to those in the following examples:

```
mpirun -np 2 a.out
Hello world
Hello world
```

```
mpirun -prefix ">" -np 2 a.out
>Hello world
>Hello world
```

```
mpirun -prefix "%g" 2 a.out
0Hello world
1Hello world
```

```
mpirun -prefix "[%g] " 2 a.out
[0] Hello world
[1] Hello world
```

```
mpirun -prefix "<process %g out of %G> " 4 a.out
<process 1 out of 4> Hello world
<process 0 out of 4> Hello world
<process 3 out of 4> Hello world
<process 2 out of 4> Hello world
```

```
mpirun -prefix "%@: " hosta,hostb 1 a.out
hosta: Hello world
hostb: Hello world
```

```
mpirun -prefix "%@ (%l out of %L) %g: " hosta 2, hostb 3 a.out
hosta (0 out of 2) 0: Hello world
hosta (1 out of 2) 1: Hello world
hostb (0 out of 3) 2: Hello world
hostb (1 out of 3) 3: Hello world
hostb (2 out of 3) 4: Hello world
```

```
mpirun -prefix "%@ (%h out of %H): " hosta,hostb,hostc 2 a.out
hosta (0 out of 3): Hello world
hostb (1 out of 3): Hello world
hostc (2 out of 3): Hello world
hosta (0 out of 3): Hello world
hostc (2 out of 3): Hello world
hostb (1 out of 3): Hello world
```

`-v[erbose]`	Displays comments on what `mpirun` is doing when launching the MPI application.

The *entry* operand describes a host on which to run a program, and the local options for that host. You can list any number of entries on the `mpirun` command line.

In the common case (same program, multiple data (SPMD)), in which the same program runs with identical arguments on each host, usually only one entry needs to be specified.

Each entry has the following components:

- One or more host names (not needed if you run on the local host)
- Number of processes to start on each host
- Name of an executable program
- Arguments to the executable program (optional)

An entry has the following format:

host_list local_options program program_arguments

The *host_list* operand is either a single host (machine name) or a comma-separated list of hosts on which to run an MPI program.

The *local_options* operand contains information that applies to a specific host list. The following local options are supported:

Option	Description
`-f[ile]` *file_name*	Specifies a text file that contains `mpirun` arguments (same as *global_options*.) For more details, see Section 3.2.
`-np` *np*	Specifies the number of processes on which to run. (UNICOS/mk systems support only this option.)
`-nt` *nt*	On UNICOS systems, specifies the number of tasks on which to run in a multitasking or shared memory environment. On IRIX systems, this option behaves the same as `-np`.

The *program program_arguments* operand specifies the name of the program that you are running and its accompanying options.

3.2 Using a File for `mpirun` Arguments (UNICOS Or IRIX)

Because the full specification of a complex job can be lengthy, you can enter `mpirun` arguments in a file and use the `-f` option to specify the file on the `mpirun` command line, as in the following example:

```
mpirun -f my_arguments
```

The arguments file is a text file that contains argument segments. White space is ignored in the arguments file, so you can include spaces and newline characters for readability. An arguments file can also contain additional `-f` options.

3.3 Launching Programs on the Local Host Only

For testing and debugging, it is often useful to run an MPI program on the local host only without distributing it to other systems. To run the application locally, enter `mpirun` with the `-np` or `-nt` argument. Your entry must include the number of processes to run and the name of the MPI executable file.

The following command starts three instances of the application `mtest`, to which is passed an arguments list (arguments are optional).

```
mpirun -np 3 mtest 1000 "arg2"
```

You are not required to use a different host in each entry that you specify on the `mpirun`(1) command. You can launch a job that has two executable files on the same host. On a UNICOS system, the following example uses a combination of shared memory and TCP. On an IRIX system, both executable files use shared memory:

```
mpirun host_a -np 6 a.out : host_a -nt 4 b.out
```

3.3.1 Using `mpirun`(1) to Run Programs in Shared Memory Mode (UNICOS Or IRIX)

For running programs in MPI shared memory mode on a single host, the format of the `mpirun`(1) command is as follows:

`mpirun -nt`[*nt*] *progname*

The `-nt` option specifies the number of tasks for shared memory MPI, and can be used on UNICOS systems only if you have compiled and linked your program as described in Section 2.1.1, page 5. A single UNIX process is run with multiple tasks representing MPI processes. The *progname* operand specifies the name of the program that you are running and its accompanying options.

The `-nt` option to `mpirun` is supported on IRIX systems for consistency across platforms. However, since the default mode of execution on a single IRIX system is to use shared memory, the option behaves the same as if you specified the `-np` option to `mpirun`. The following example runs ten instances of `a.out` in shared memory mode on `host_a`:

```
 mpirun -nt 10 a.out
```

3.3.2 Using the `mpirun`(1) Command on UNICOS/mk Systems

The `mpirun`(1) command has been provided for consistency of use among IRIX, UNICOS, and UNICOS/mk systems. Use of this command is optional, however, on UNICOS/mk systems. If your program was built for a specific number of PEs, the number of PEs specified on the `mpirun`(1) command line must match the number that was built into the program. If it does not, `mpirun`(1) issues an error message.

The following example shows how to invoke the `mpirun`(1) command on a program that was built for four PEs:

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

3.3.3 Executing UNICOS/mk Programs Directly

Instead of using the mpirun(1) command, you can choose to launch your MPI programs on UNICOS/mk systems directly. If your UNICOS/mk program was built for a specific number of PEs, you can execute it directly, as follows:

```
./a.out
```

If your program was built as a *malleable* executable file (the number of PEs was not fixed at build time, and the -Xm option was used instead), you can execute it with the mpprun(1) command. The following example runs a program on a partition with four PEs:

```
mpprun -n 4 a.out
```

3.4 Launching a Distributed Program (UNICOS Or IRIX)

You can use mpirun(1) to launch a program that consists of any number of executable files and processes and distribute it to any number of hosts. A host is usually a single Origin, CRAY J90, or CRAY T3E system, or can be any accessible computer running Array Services software. Array Services software runs on IRIX and UNICOS systems and must be running to launch MPI programs. For available nodes on systems running Array Services software, see the /usr/lib/array/arrayd.conf file.

You can list multiple entries on the mpirun command line. Each entry contains an MPI executable file and a combination of hosts and process counts for running it. This gives you the ability to start different executable files on the same or different hosts as part of the same MPI application.

The following examples show various ways to launch an application that consists of multiple MPI executable files on multiple hosts.

The following example runs ten instances of the a.out file on host_a:

```
mpirun host_a -np 10 a.out
```

When specifying multiple hosts, the -np or -nt option can be omitted with the number of processes listed directly. On UNICOS systems, if you omit the -np or -nt option, mpirun assumes -np and defaults to TCP for communication. The following example launches ten instances of fred on three hosts. fred has two input arguments.

```
mpirun host_a, host_b, host_c 10 fred arg1 arg2
```

The following example launches an MPI application on different hosts with different numbers of processes and executable files, using an array called `test`:

```
mpirun -array test host_a 6 a.out : host_b 26 b.out
```

The following example launches an MPI application on different hosts out of the same directory on both hosts:

```
mpirun -d /tmp/mydir host_a 6 a.out : host_b 26 b.out
```

Setting Environment Variables [4]

This chapter describes the variables that specify the environment under which your MPI programs will run. Environment variables have predefined values. You can change some variables to achieve particular performance objectives; others are required values for standard-compliant programs.

4.1 Setting MPI Environment Variables on UNICOS and IRIX Systems

This section provides a table of MPI environment variables you can set for IRIX systems only, and a table of environment variables you can set for both UNICOS and IRIX systems. For environment variables for UNICOS/mk systems, see Section 4.2, page 22.

Table 2. MPI environment variables for IRIX systems only

Variable	Description	Default
MPI_BUFS_PER_HOST	Determines the number of shared message buffers (16 KB each) that MPI is to allocate for each host. These buffers are used to send long messages.	16 pages (each page is 16 KB)
MPI_BYPASS_DEVS	Sets the order for opening HIPPI adapters. The list of devices does not need to be space-delimited (0123 is also valid). An array node usually has at least one HIPPI adapter, the interface to the HIPPI network. The HIPPI bypass is a lower software layer that interfaces directly to this adapter. The bypass sends MPI control and data messages that are 16 Kbytes or shorter.	0 1 2 3

Variable	Description	Default
	When you know that a system has multiple HIPPI adapters, you can use the `MPI_BYPASS_ DEVS` variable to specify the adapter that a program opens first. This variable can be used to ensure that multiple MPI programs distribute their traffic across the available adapters. If you prefer not to use the HIPPI bypass, you can turn it off by setting the `MPI_BYPASS_OFF` variable. When a HIPPI adapter reaches its maximum capacity of four MPI programs, it is not available to additional MPI programs. If all HIPPI adapters are busy, MPI sends internode messages by using TCP over the adapter instead of the bypass.	
`MPI_BYPASS_OFF`	Disables the HIPPI bypass.	Not enabled
`MPI_DSM_OFF`	Turns off nonuniform memory access (NUMA) optimization in the MPI library.	Not enabled
`MPI_DSM_MUSTRUN`	Specifies the CPUs on which processes are to run. For jobs running on IRIX systems, you can set the `MPI_DSM_VERBOSE` variable to request that the `mpirun` command print information about where processes are executing.	Not enabled
`MPI_DSM_PPM`	Sets the number of MPI processes that can be run on each node of an IRIX system.	2
`MPI_DSM_VERBOSE`	Instructs `mpirun` to print information about process placement for jobs running on NUMA systems.	Not enabled
`MPI_MSGS_PER_HOST`	Sets the number of message headers to allocate for MPI messages on each MPI host. Space for messages that are destined for a process on a different host is allocated as shared memory on the host on which the sending processes are located. MPI locks these pages in memory. Use the `MPI_MSGS_PER_HOST` variable to allocate buffer space for interhost messages. ⚠ **Caution:** If you set the memory pool for interhost packets to a large value, you can cause allocation of so much locked memory that total system performance is degraded.	128

Table 3. MPI environment variables for UNICOS and IRIX systems

Variable	Description	Default
`MPI_ARRAY`	Sets an alternative array name to be used for communicating with Array Services when a job is being launched.	The default name set in the `arrayd.conf` file
`MPI_BUFS_PER_PROC`	Determines the number of private message buffers (16 KB each) that MPI is to allocate for each process. These buffers are used to send long messages.	16 pages (each page is 16 KB)
`MPI_CHECK_ARGS`	Enables checking of MPI function arguments. Segmentation faults might occur if bad arguments are passed to MPI, so this is useful for debugging purposes. Using argument checking adds several microseconds to latency.	Not enabled
`MPI_COMM_MAX`	Sets the maximum number of communicators that can be used in an MPI program. Use this variable to increase internal default limits. (May be required by standard-compliant programs.)	256
`MPI_DIR`	Sets the working directory on a host. When an `mpirun` command is issued, the Array Services daemon on the local or distributed node responds by creating a user session and starting the required MPI processes. The user ID for the session is that of the user who invokes `mpirun`, so this user must be listed in the `.rhosts` file on the responding nodes. By default, the working directory for the session is the user's `$HOME` directory on each node. You can direct all nodes to a different directory (an NFS directory that is available to all nodes, for example) by setting the `MPI_DIR` variable to a different directory.	`$HOME` on the node. If using `-np` or `-nt`, the default is the current directory.
`MPI_GROUP_MAX`	Sets the maximum number of groups that can be used in an MPI program. Use this variable to increase internal default limits. (May be required by standard-compliant programs.)	256

Variable	Description	Default
`MPI_MSGS_PER_PROC`	Sets the maximum number of buffers to be allocated from sending process space for outbound messages going to the same host. (May be required by standard-compliant programs.) MPI allocates buffer space for local messages based on the message destination. Space for messages that are destined for local processes is allocated as additional process space for the sending process.	128
`MPI_REQUEST_MAX`	Sets the maximum number of simultaneous nonblocking sends and receives that can be active at one time. Use this variable to increase internal default limits. (May be required by standard-compliant programs.)	1024
`MPI_TYPE_DEPTH`	Sets the maximum number of nesting levels for derived datatypes. (May be required by standard-compliant programs.) The `MPI_TYPE_DEPTH` variable limits the maximum depth of derived datatypes that an application can create. MPI logs error messages if the limit specified by `MPI_TYPE_DEPTH` is exceeded.	8 levels
`MPI_TYPE_MAX`	Sets the maximum number of derived data types that can be used in an MPI program. Use this variable to increase internal default limits. (May be required by standard-compliant programs.)	1024

4.2 Setting MPI Environment Variables on UNICOS/mk Systems

This section provides a table of MPI environment variables you can set for UNICOS/mk systems.

Table 4. Environment variables for UNICOS/mk systems

Variable	Description	Default
`MPI_SM_POOL`	Specifies shared memory queue. When MPI is started, it allocates a pool of shared memory for use in message passing. This pool represents space used to buffer message headers and small messages while the receiving PE is doing computations or I/O. The default of 1024 bytes is the number of bytes that can be pending.	1024 bytes
`MPI_SM_TRANSFER`	Specifies number of queue slots. Specifies the number of slots in the shared memory queue that can be occupied by a send operation at the receiver. A slot consists of four UNICOS/mk words. By default, a single send operation can occupy 128 slots (or buffer 512 words) while the receiving PE is doing computations or I/O.	128
`MPI_BUFFER_MAX`	Specifies maximum buffer size. Specifies a maximum message size, in bytes, that will be buffered for MPI standard, buffered, or ready send communication modes.	No limit
`MPI_BUFFER_TOTAL`	Specifies total buffer memory. Specifies a limit to the amount of memory the MPI implementation can use to buffer messages for MPI standard, buffered, or ready send communication modes.	No limit

4.3 Internal Message Buffering in MPI (IRIX Systems Only)

An MPI implementation can copy data that is being sent to another process into an internal temporary buffer so that the MPI library can return from the MPI function, giving execution control back to the user. However, according to the MPI standard, you should not assume any message buffering between processes because the MPI standard does not mandate a buffering strategy. Some implementations choose to buffer user data internally, while other implementations block in the MPI routine until the data can be sent. These different buffering strategies have performance and convenience implications.

Most MPI implementations do use buffering for performance reasons and some programs depend on it. Table 5, page 24, illustrates a simple sequence of MPI operations that cannot work unless messages are buffered. If sent messages were not buffered, each process would hang in the initial `MPI_Send` call, waiting for an `MPI_Recv` call to take the message. Because most MPI implementations do buffer messages to some degree, often a program such as this will not hang. The `MPI_Send` calls return after putting the messages into buffer space, and the `MPI_Recv` calls get the messages. Nevertheless, program logic such as this is not valid by the MPI standard. The Silicon Graphics implementation of MPI for IRIX systems buffers messages of all sizes. For buffering purposes, this implementation recognizes short message lengths (64 bytes or shorter) and long message lengths (longer than 64 bytes).

Table 5. Outline of improper dependence on buffering

Process 1	Process 2
MPI_Send(2,....)	MPI_Send(1,....)
MPI_Recv(2,....)	MPI_Recv(1,....)

Debugging Programs on IRIX Systems [5]

To debug MPI 3.1 programs on IRIX systems, you can use the XMPI graphical interface or either of the Silicon Graphics debuggers, CVD (ProDev WorkShop) and DBX. For information on using XMPI, see Section 5.3, page 27.

You cannot simply start an MPI program from the debuggers because MPI 3.1 uses Array Services to start an MPI application and the application is a collection of IRIX processes. Instead, you must start the program with `mpirun` or `xmpi` commands as you normally do, then attach to the running program with the debugger.

This startup method is inconvenient, because you cannot initially start an MPI application in a stopped state. If your program is long running, this may not be a problem because there will be time to obtain the process IDs and attach the debugger to them. If your program is short, or you need to debug it soon after MPI initialization, code a call to a sleep routine to allow time to attach to the program.

The debugging procedures in this chapter assume that you have added a sleep routine to the beginning of your program.

Procedure 2: Using the ProDev WorkShop debugger, CVD

Use the following procedure to debug an MPI program with the ProDev WorkShop debugger, CVD:

1. Modify the MPI application to call a sleep routine.

   To modify the MPI process, begin with a preliminary call to `SLEEP` before calling the MPI initialization routine. The following example illustrates a modified Fortran program:

   ```
   CALL SLEEP(60)
   CALL MPI_Init ()
   ```

2. Start your MPI program after building it.

3. Obtain process IDs.

   Be sure to obtain the processes with your program name, not the `mpirun` process, from the output of the following command:

   `ps -u` *userid*

4. Attach to your application by using the CVD debugger. Attaching by using CVD, as follows, places you in the sleep library routine in a stopped state:

   ```
   cvd -pid pid
   ```

   Note: To find your program source, use the `cvd up` command to move up the stack several times.

5. Instruct the CVD debugger to automatically attach to child processes of the program as they are created (forked) by doing the following:

   a. Select `Admin ---> Multiprocess View`.

   b. `Select Config ---> Preferences`.

   c. Select `Attach to forked processes` and `Copy traps to forked processes`.

6. Use the CVD debugger to set a breakpoint, if you wish.

7. Instruct CVD to automatically attach to your application's processes by selecting `Continue`. After your selection, processes are listed in the `Multiprocess View`. From this view, you can start CVD windows for other processes to debug different MPI ranks.

   The initial attached process remains running in `MPI_Init` for the duration of your program. For example, if you specify four processes on the `mpirun` command line, five processes will be listed in the `Multiprocess View`.

Caution: Do not stop the `MPI_init` process. Stopping `MPI_init` can halt MPI communications.

Procedure 3: Using the DBX debugger

Use the following procedure to debug an MPI program by using the DBX debugger:

1. Follow steps 1 through 3 in Procedure 2.

2. Attach to your running program by using the following `dbx` command:

   ```
   dbx -p pid
   ```

3. Debug child processes as they are created.

Use the following command to tell DBX that you want to debug child processes as MPI creates them:

```
set $promptonfork = 2
```

4. Continue your program by entering the `continue` command.
5. As each process is created, set a breakpoint in it manually, then continue it. The DBX debugger stops each child process separately. For more information on debugging multiprocess programs with DBX, enter `help mp` and `help hint_mp_debug` on the `dbx` command line.

5.1 Core File Contents

If an MPI program aborts, the `core` file contains only the aborting process. The remaining processes in the application continue to run. You can inspect the `core` file using either cvd or dbx, but with either debugger, you will see only information for the failing process.

5.2 Running Distributed Applications

When an application is distributed across multiple nodes in an array, you must run multiple copies of cvd to debug the process on each node. See Array Services documentation for more information.

Use the following `cvd` command to attach to a particular process on a specific node:

`cvd -host` *hostname* `-pid` *pid*

The Array Services commands `ainfo`, `array`, and `aview` also provide information on MPI processes.

5.3 Using XMPI

The XMPI graphical interface starts MPI applications and allows you to view message passing events that occur during program execution. It can also save a file of these events if you want a permanent record. XMPI must be running on any node on which it is used.

If you understand the concepts of MPI, you can probably master XMPI by starting it, experimenting with its menus, and browsing its help displays. But the information in this section acquaints you with XMPI menus and dialogs to make your exploration of XMPI more focused and efficient. It also explains how to use menus and dialog boxes to do several commonly performed tasks.

5.3.1 Starting XMPI

To start XMPI, type the `xmpi` command in a shell window. After your entry, the XMPI main window is displayed as follows:

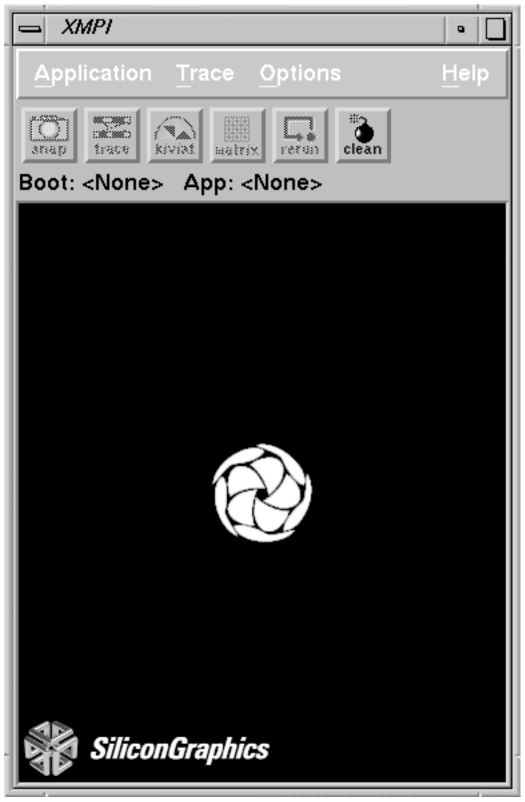


Figure 1. XMPI main window

The XMPI menu bar offers the following menus:

- The `Application` menu contains options for running MPI applications and managing other aspects of program execution. Section 5.3.2, page 29, describes each of the options on this menu and contains procedures for running MPI applications with XMPI.
- The `Trace` menu contains options for viewing and managing trace files. Section 5.3.3, page 37, describes the options on this menu and contains procedures for viewing traces.
- The `Options` menu contains implementation-specific functions that are not relevant to the Silicon Graphics version of XMPI.
- The `Help` menu provides a list of topics that you can select to obtain information about XMPI features and functions. Section 5.3.4, page 41, explains how to access additional XMPI documentation.

The buttons below the menu bar duplicate selections on the `Application` menu. When you are viewing a trace, it is often easier to use these buttons than the menu selections.

5.3.2 The `Application` Menu

The `Application` menu, shown in Figure 2, contains options for running and managing MPI applications. Notice that this menu contains two selections for running MPI applications: `Build&Run...` and `Browse&Run....` Choose the one that best suits your needs.

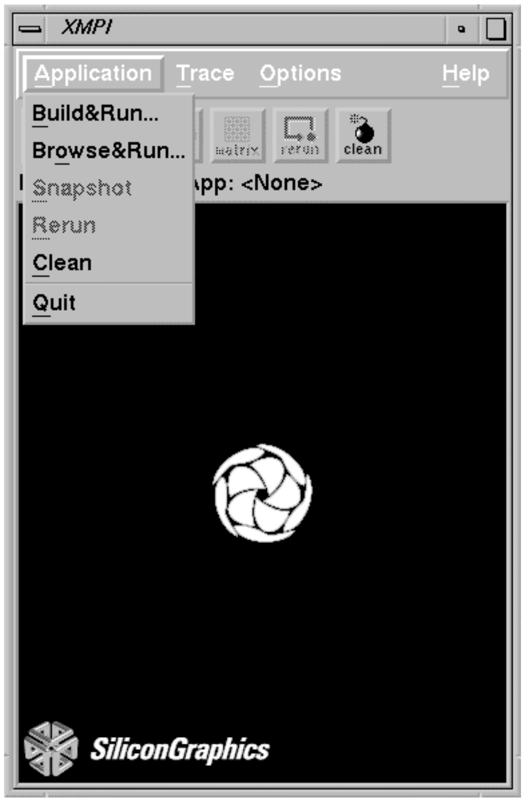


Figure 2. `Application` menu

Use the `Application` menu options for the following tasks:

- Use the `Build&Run...` option to run a job on distributed hosts or to define new execution parameters for the job. With this option, you can save the execution parameters to a file and use the parameters when you run the application later.
- Use the `Browse&Run...` option to run an application when a file of its execution parameters was previously created with the `Build&Run...` option.
- Use the `Snapshot` option on a running program to record information on trace events at a given instant in time.

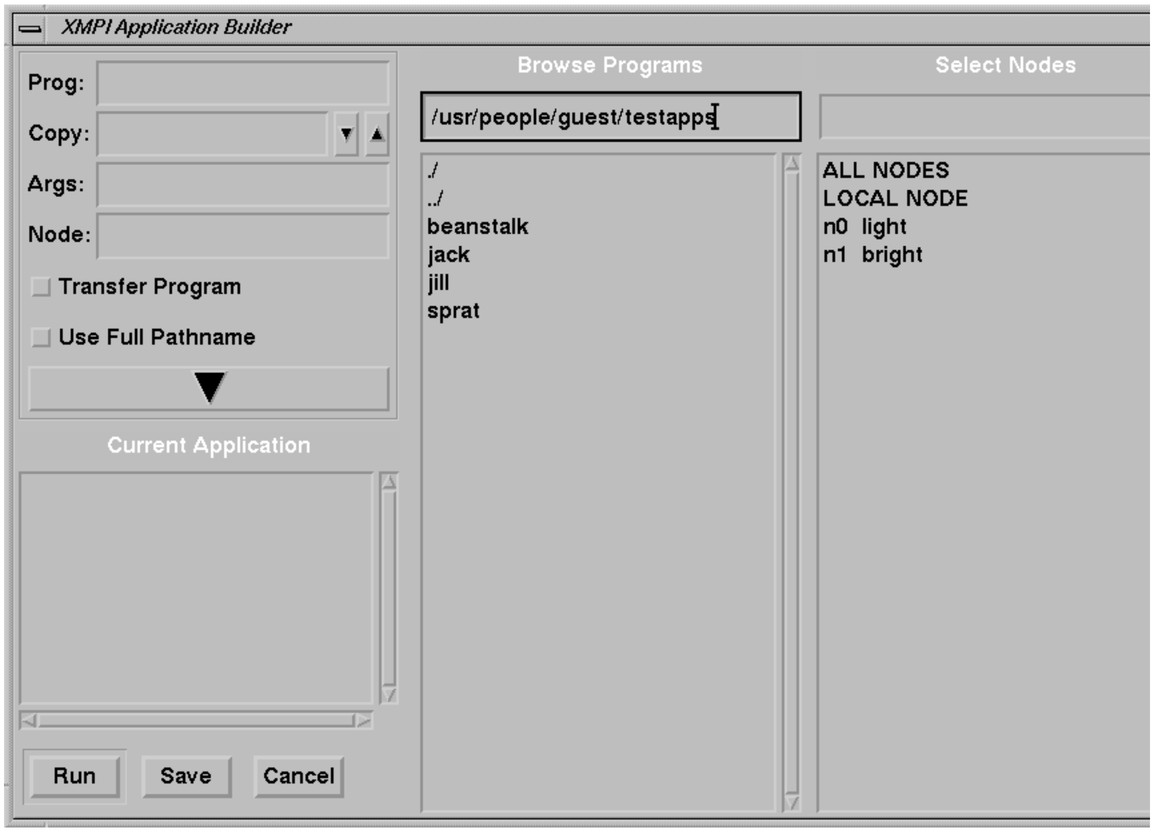
- Use the `Rerun` option to clean any remnants left from a previous job and run it again.
- Use the `Clean` option to cancel a running MPI application and remove status information about its processes from collection tables.

After an application is launched, you can use the `Snapshot` option to collect information on process events at any instant in time. Use the `Clean` option to stop a process without stopping XMPI.

Procedure 4: Using the `Build&Run...` option

Use the following procedure to define parameters for MPI programs and launch them from XMPI:

1. Select the `Build&Run...` option from the `Application` menu. When you select this option, the `Application Builder` dialog box is displayed (shown in Figure 3, page 32). This dialog box accepts MPI program parameters, queues the programs, and launches them.



/usr/people/guest/testapps

```
./
../
beanstalk
jack
jill
sprat
```

```
ALL NODES
LOCAL NODE
n0  light
n1  bright
```

Figure 3. `Application Builder` dialog box

2. Specify program parameters in the left column of the dialog box.

   Enter the name of the MPI application, the number of copies that you wish to run, program arguments, and the node on which the application is to run. (You can also select the program and node from the `Browse Programs` and `Select Nodes` lists at the right.)

   To change the directory or node parameters, go on to Procedure 4, step 3, page 33. If directory and node parameters are satisfactory, go directly to Procedure 4, step 4, page 33.

   Note: The `Transfer Program` and `Use Full Pathname` buttons are not implemented for this release.

3. Change the program directory and MPI node, as necessary.

   By default, XMPI assumes that you want to run a program located in the directory in which you entered the `xmpi` command. If the `Browse Programs` column does not contain the program that you want to run, enter a different directory name at the top of this column, then select the program that you want from the new list that is displayed after your entry. If the program that you want to run is on a different node, use the `Select Nodes` column.

   The `Select Nodes` column contains a list of the available hosts on which you can run an MPI application. You can select an individual node, all nodes, or the just the local node from the available nodes list. You can also add a node to the list by entering its name in the entry field at the top of the column. The node that you add must be listed in the following file:

   ```
   /usr/lib/array/arrayd.conf
   ```

4. Save the execution parameters, as necessary.

   To store the execution parameters that you entered in a file, click the `Save` button. When you select this button, you are prompted to enter the name of the parameter file.

5. To queue the job, click the `commit` button (the widget containing a down arrow).

   The `commit` button queues the job for execution. The job remains in the queue until you select the `Run` button.

6. Queue additional jobs, as necessary.

   To specify various combinations of nodes and program parameters, and to queue the additional jobs, repeat steps 1 through 5.

   To remove a job from the queue, select the job by double-clicking its name in the queue and then press the `Delete` key.

   If you change your mind and decide not to run any applications, click the `Cancel` button to close the `Application Builder` dialog box without taking action on your entries.

7. To start the application, click the `Run` button.

   The XMPI banner is replaced by a display of the running application, shown in the following figure:

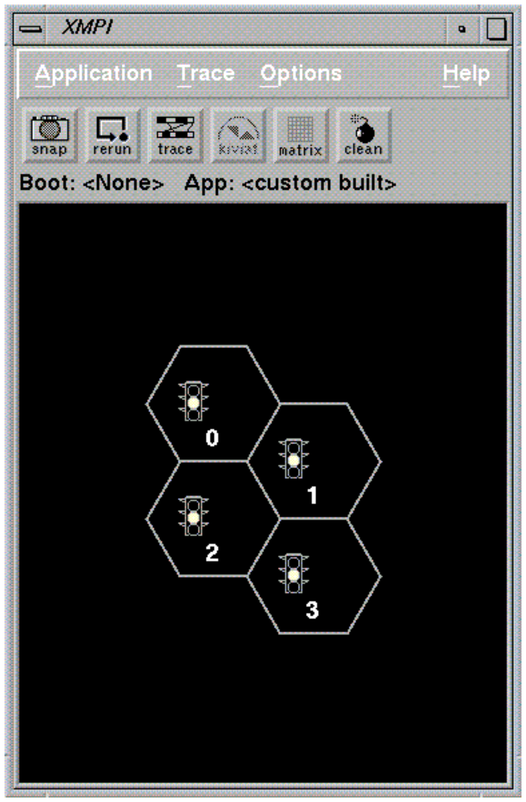


Figure 4. XMPI display of a running application

Each process in the job is represented by a hexagonal icon. The color of the traffic light in the icon indicates the following process status:

- Green indicates that the process is running but it is not executing an MPI routine.
- Yellow indicates that an MPI routine call is in progress and no blocking is occurring.
- Red indicates that an executing MPI routine is blocking.

You can display a record of events for any process by clicking its icon. The individual record for each process is shown in the following figure:

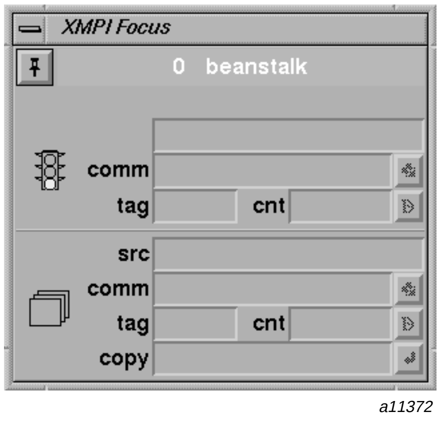


Figure 5. Individual record of events for an MPI process

8. Perform the following functions as necessary:

- To refresh the information on the record and in the process icons, click the `snap` button on the main window (or choose the `Snapshot` option from the `Application` menu).
- To clear the information and remove records from the window, click the `clean` button.
- To clear the information and restart the application, click the `rerun` button.

Procedure 5: Using the `Browse&Run...` option

When you select `Browse&Run...`, the `Application Browser` dialog box (shown in Figure 6, page 36) is displayed. Notice from this figure that this dialog box has no entry fields for execution parameters. With this option, execution parameters are read from a file that was previously created by selecting `Build&Run...` (see Procedure 4, step 4, page 33).

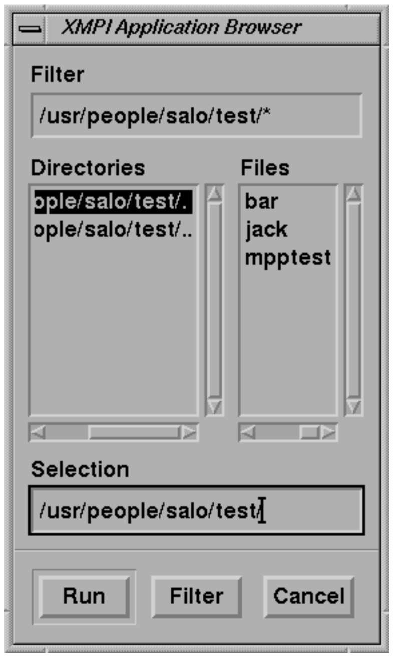


Figure 6. `Application Browser` dialog box

To run an MPI application with the `Browse&Run...` option, use the following procedure:

1. Choose a directory and program name from the `Directories` and `Files` lists in the `Application Browser` dialog box.

2. Change the directory, as necessary.

   XMPI assumes that the program that you want to run is in the directory in which you entered the `xmpi` command. If the `Directories` and `Files` lists do not contain the directory that you want, enter a directory name in the `Filter` field at the top of the dialog box. Then select the program from the new ones displayed in the `Files` list.

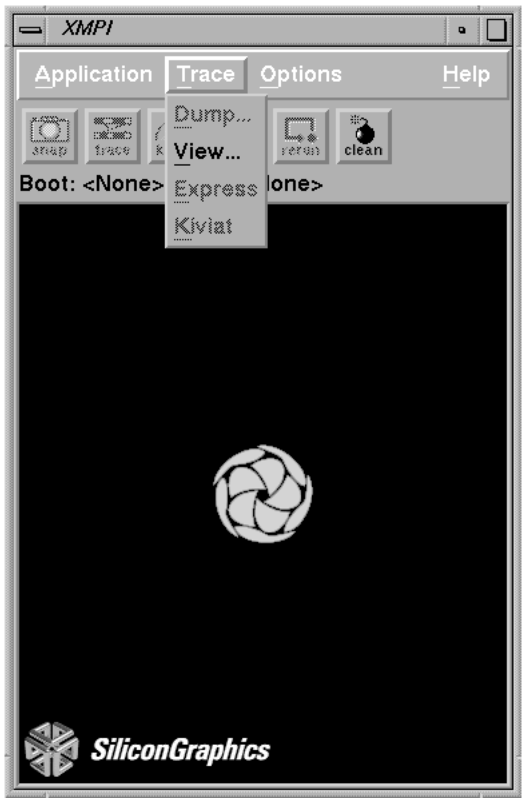
3. Click the `Run` button to start the application.

   After your selection, the XMPI banner is replaced by a display of the running application. See Figure 4, page 34, and the discussion that follows it for information on monitoring the application after it starts.

5.3.3 The Trace Menu

Whenever you launch an MPI application with XMPI, it automatically gathers trace information from the MPI daemons on the host nodes. The `Trace` menu (shown in Figure 7, page 37) contains options for viewing information collected from these daemons. Use the following `Trace` menu options to perform these tasks:

- `Dump` saves trace information in a file.
- `View` displays information that was previously saved in a trace file.
- `Express` displays trace information while the application is executing.
- `Kiviat` displays a trace file in a Kiviat diagram.



a11374

Figure 7. XMPI `Trace` menu

Procedure 6: Creating and viewing trace information

Use the following procedure to create and view a trace file:

1. Run the MPI application.

   To launch the MPI program, use the `Build&Run...` or the `Browse&Run...` option from the `Application` menu. (For more information, see Procedure 4, page 31, or Procedure 5, page 35.)

2. Dump the trace data to a file.

   To create a file containing the trace information, choose the `Dump` option from the `Trace` menu. When you choose this option, a dialog box is displayed so that you can specify the name of the trace file. If you do not include a directory in your specification, the trace file will be stored in the directory in which you entered the `xmpi` command.

3. Choose the trace file that you want to display.

   After a trace file is created, you can view it by choosing the `View` option on the `Trace` menu. When you choose `View`, the `Trace Selection` dialog box (shown in Figure 8, page 39) is displayed so that you can specify the file to display.

   If you want to display a trace file other than the one that was just created, use the `Directory` and `Files` lists to select a different trace file.

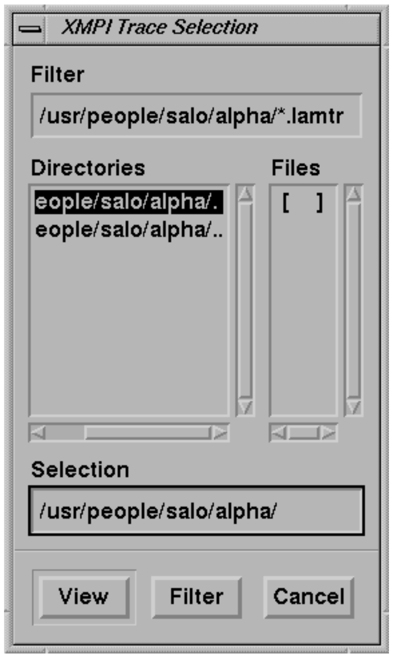


Figure 8. Dialog box for displaying trace files

4. View the trace file.

   Click the `View` button to see a time line of the trace. Figure 9, page 40, illustrates a portion of a time line on two processes.

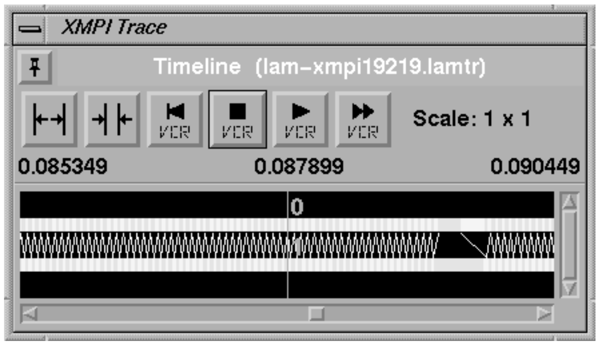


Figure 9. XMPI display of trace information

The horizontal bars reflect the state of each host over time. Arrows between bars represent messages that are being passed between hosts. To control the level of detail that is shown, use the `expand` and `compress` buttons (icons showing arrows pointing out and arrows pointing in above the display). To control the time-lapse display, use the VCR buttons; they start and stop, fast-forward, and rewind the trace.

> **Note:** To zoom in on an area of the time line to see it in greater detail, position the cursor over the area of the time line that you want to expand. Press the right mouse button and sweep over the area of interest to frame it. Each time you create a new frame, the contents of the frame are magnified.

Procedure 7: Viewing a trace as an application runs

Use the following procedure to view a trace file while an application is running:

1. Run the MPI application.

   To launch the MPI program, use the `Build&Run...` or the `Browse&Run...` option from the `Application` menu. (For more information, see Procedure 4, page 31, or Procedure 5, page 35.)

2. Choose the `Express` option on the `Trace` menu.

   When you choose the `Express` option, a time line of the trace is displayed. For a description of the display, see Figure 9, page 40, and the information following the figure.

Note: To view a trace while an application is running, use the `Trace` button on the main window instead of the `Express` option.

5.3.4 `Help` Menu

The `Help` menu (shown in Figure 10, page 42) contains the primary documentation for XMPI. The `About XMPI` option gives the XMPI version number and copyright information. This option also contains a World Wide Web location at which you can find help on XMPI.

When you select the `Help` option from the `Help` menu, your Web browser launches and connects you to the Web site containing XMPI help. To access this Web page, the XMPI Web site must be accessible from the host from which you are using XMPI.

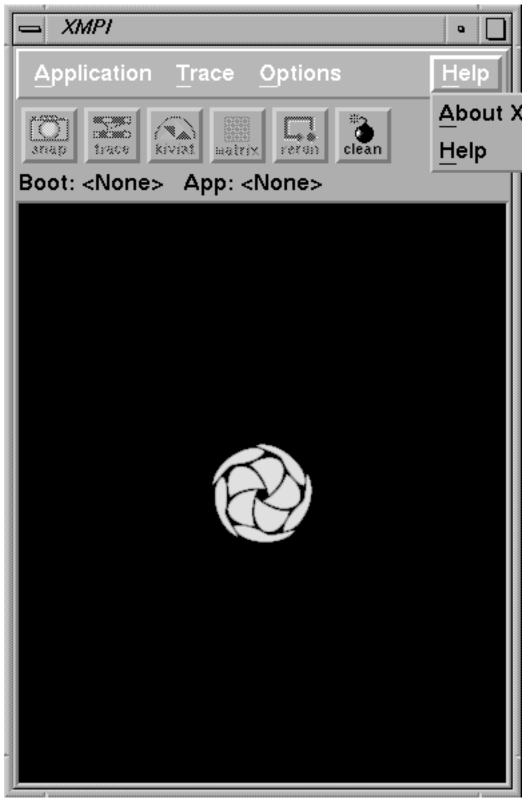


Figure 10. XMPI `Help` menu

Launching Programs with NQE [6]

After an MPI program is debugged and ready to run in a production environment, it is often useful to submit it to a queue to be scheduled for execution. The Network Queuing Environment (NQE) provides this capability. NQE selects a node appropriate for the resources that an MPI job needs, routes the job to a node, and schedules it to run.

This chapter explains how to use the NQE graphical interface on IRIX systems to submit an MPI program for execution. For information on using NQE to submit UNICOS or UNICOS/mk programs, see the *NQE User's Guide*, publication SG–2148.

6.1 Starting NQE

Before you begin, set your `DISPLAY` variable so that the NQE screens appear on your workstation. Then enter the `nqe` command, as shown in the following example:

```
setenv DISPLAY myworkstation:0
<nqe
```

Figure 11 shows the NQE button bar, which appears after your entry.

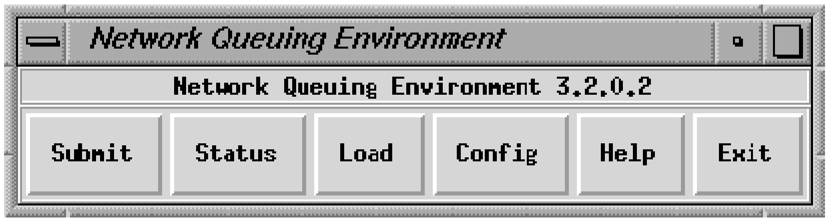
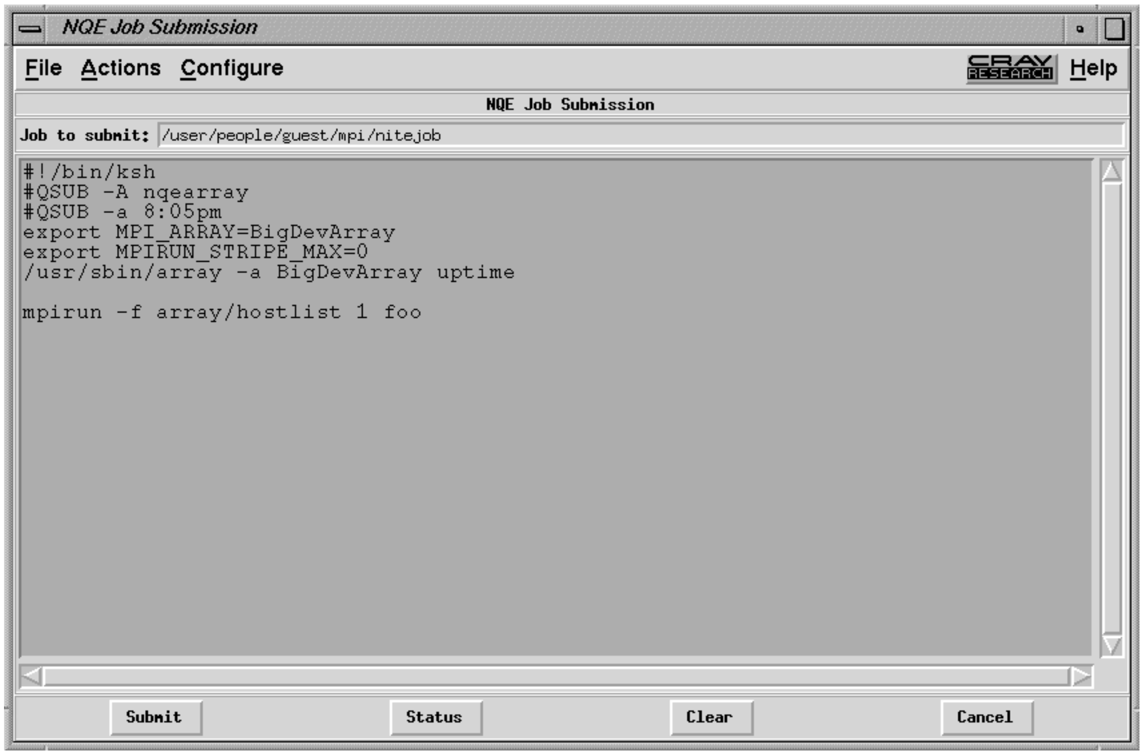

Network Queuing Environment
Network Queuing Environment 3.2.0.2
Submit
Status
Load
Config
Help
Exit
a11378

Figure 11. NQE button bar

6.2 Submitting a Job with NQE

To submit a job, click the `Submit` button on the `NQE Job Submission` window. Figure 12 shows the `NQE Job Submission` window with a sample job script ready to be submitted.



```
#!/bin/ksh
#QSUB -A nqearray
#QSUB -a 8:05pm
export MPI_ARRAY=BigDevArray
export MPIRUN_STRIPE_MAX=0
/usr/sbin/array -a BigDevArray uptime

mpirun -f array/hostlist 1 foo
```

Figure 12. NQE `NQE Job Submission` window

Notice in this figure that the difference between an NQE job request and a shell script lies in the use of the `#QSUB` identifiers. In this example, the directive `#QSUB -A nqearray` tells NQE to run this job under the `nqearray` project account. The directive `#QSUB -a 8:05pm` tells NQE to wait until 8:05 p.m. to start the job.

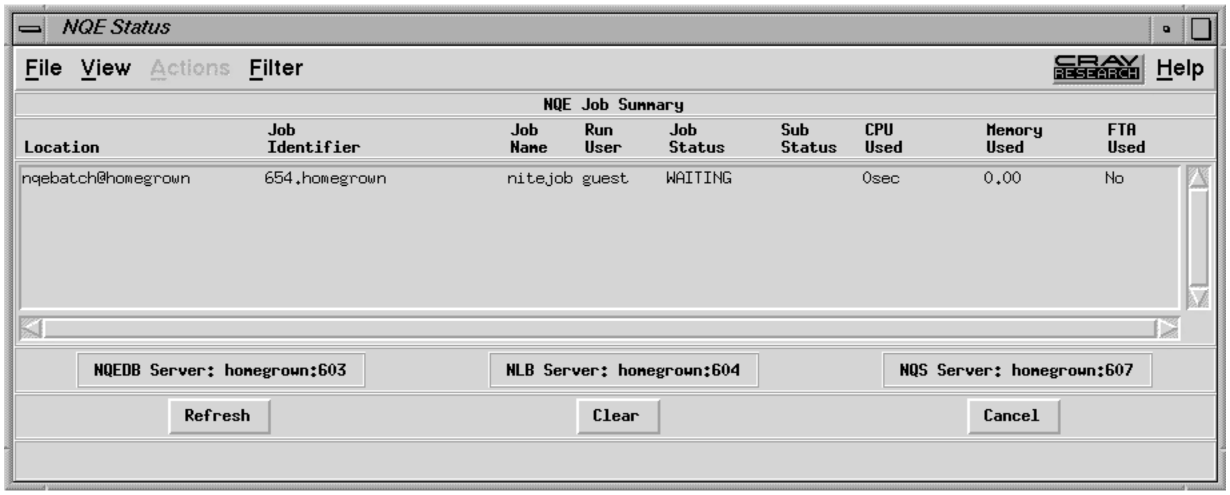
Also notice in Figure 12 that the MPI program is already compiled and distributed to the proper hosts. The file `array/hostlist` has the list of parameters for this job, as you can see in the output from the following `cat` command:

```
% cat array/hostlist
homegrown, disarray, dataarray
```

6.3 Checking Job Status with NQE

To see the status of jobs running under NQE, click the `Status` button to display the `NQE Status` window.

Figure 13 shows an example of the `NQE Status` window. Notice in this figure that the MPI job is queued and waiting to run.



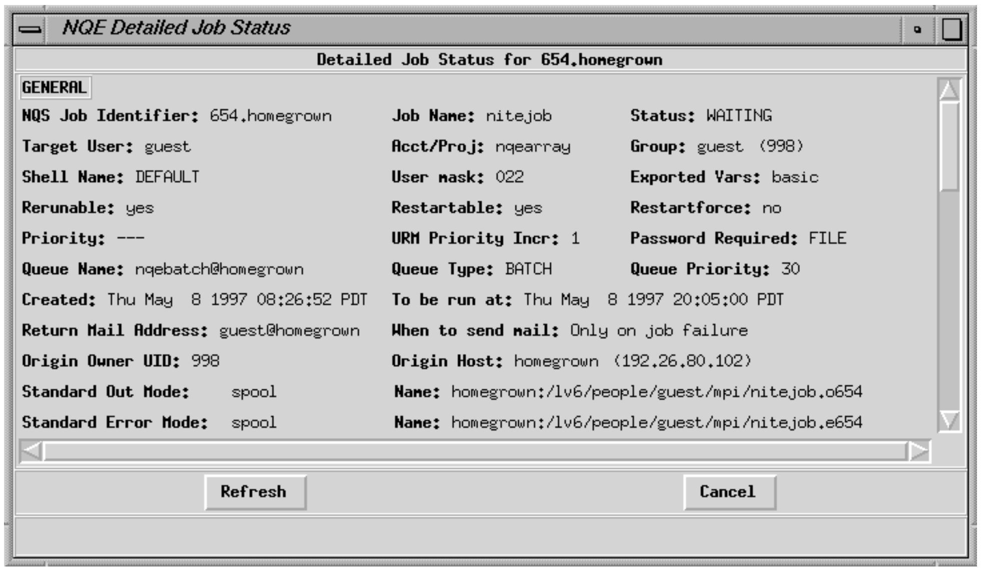
Location	Job Identifier	Job Name	Run User	Job Status	Sub Status	CPU Used	Memory Used	FTA Used
nqebatch@homegrown	654.homegrown	nitejob	guest	WAITING		0sec	0.00	No

NQEDB Server: homegrown:603 NLB Server: homegrown:604 NQS Server: homegrown:607

Figure 13. `NQE Status` window

1. After the job starts, use XMPI as you do for any other MPI job.
2. To verify the scheduled starting time for the job, position the mouse cursor on the line that shows the job and double-click it.

   This displays the `NQE Detailed Job Status` window, shown in Figure 14. Notice that the job was created at 8:26 PDT and is to run at 20:05 PDT.



```
Detailed Job Status for 654.homegrown
GENERAL
NQS Job Identifier: 654.homegrown      Job Name: nitejob        Status: WAITING
Target User: guest                     Acct/Proj: nqearray      Group: guest (998)
Shell Name: DEFAULT                    User mask: 022           Exported Vars: basic
Rerunable: yes                         Restartable: yes         Restartforce: no
Priority: ---                          URM Priority Incr: 1     Password Required: FILE
Queue Name: nqebatch@homegrown         Queue Type: BATCH        Queue Priority: 30
Created: Thu May  8 1997 08:26:52 PDT  To be run at: Thu May  8 1997 20:05:00 PDT
Return Mail Address: guest@homegrown   When to send mail: Only on job failure
Origin Owner UID: 998                  Origin Host: homegrown (192.26.80.102)
Standard Out Mode:    spool            Name: homegrown:/lv6/people/guest/mpi/nitejob.o654
Standard Error Mode:  spool            Name: homegrown:/lv6/people/guest/mpi/nitejob.e654
```

a11381

Figure 14. `NQE Detailed Job Status` window

6.4 Getting More Information

For more information on using NQE, see the following NQE publications:

- *Introducing NQE*, publication IN–2153
- *NQE Release Overview*, publication RO–5237
- *NQE Installation*, publication SG–5236
- *NQE Administration*, publication SG–2150
- *NQE User's Guide*, publication SG–2148

The preceding publications are also available in the Cray Research Online Software Publications Library at the following URL:

```
http://www.cray.com/products/software/publications
```

PostScript files of NQE publications are available through the Cray Research Online Publications Software Library. To download a publication, select `Summary` next to the book title you want on the `Titles` Web page, which is located at the following URL:

```
http://www.cray.com/products/software/publications/dynaweb/docs/titles.html#N
```

The main Cray Research Online Software Library Web page is at the following URL:

```
http://www.cray.com/products/software/publications/
```

For general information about NQE, see the following URLs:

```
http://www.cray.com/products/software/nqe
http://www.cray.com
```

(search for NQE)

Index

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-3687-001.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-932-0801
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California 94043-1389