A Test Implementation
of the MPI Draft Message-Passing Standard

by

William Gropp and Ewing Lusk

Mathematics and Computer Science Division

December 1992

This work was supported in part by the Office of Scientific Computing, U.S. Department of Energy.
## Contents

Abstract 1

1 Introduction 1
   1.1 Structure of the Send and Receive Library 1
   1.2 C and Fortran 2
   1.3 Additions 2
   1.4 Omissions 3
   1.5 Restrictions 3

2 Comments on the Draft Standard 3

3 Starting and Stopping Processes 4

4 Very Simple User Interface 4

5 Examples 4

6 Availability 5

7 Basic Routines for Point-to-Point Messages 5
   7.1 General Communication Control 6
   7.2 Contiguous Communications 8
      7.2.1 General Routines 8
      7.2.2 Blocking Routines 13
      7.2.3 Nonblocking Routines 18
   7.3 Constant Stride Communications 22
      7.3.1 General Routines 22
      7.3.2 Blocking Routines 27
      7.3.3 Nonblocking Routines 32
   7.4 General Scatter/Gather 37
      7.4.1 General Routines 37
      7.4.2 Blocking Routines 42
      7.4.3 Nonblocking Routines 46
   7.5 Process Groups 50
   7.6 Communication Contexts 57
   7.7 Program Management 58
   7.8 Utility Routines 59

Index 62
A Test Implementation of the MPI Draft Message-Passing Standard

by

William Gropp and Ewing Lusk

Abstract

Message passing is a common method for programming parallel computers. The lack of a standard has significantly impeded the development of portable software and libraries for these machines. Recently, an ad-hoc committee was formed to develop a standard for message-passing software for parallel computers. This group first met in April 1992 at a workshop sponsored in part by the Center for Research on Parallel Computation (CRPC). Four of the attendees at that meeting produced a draft standard, henceforth referred to as the MPI (Message-Passing Interface) draft standard. After review by a larger group, and significant changes in the document, a meeting was held in November to discuss the MPI draft standard. This document is a result of those discussions; it describes a running implementation of most of the proposed standard, plus additional routines that were suggested by the discussions at the November meeting.

1 Introduction

This document describes a test implementation of the Message-Passing Interface (MPI) draft standard. Supplying an implementation along with the standard itself provides several benefits. It allows the draft standard to be tested for expressivity and implementability. It brings to light potential inconsistencies and omissions in the draft standard as the draft standard develops. It allows experiments that measure possible restrictions on performance imposed by the draft standard.

Subject to the small number of restrictions and omissions cited below, it is an implementation of all of the draft standard. It also includes other routines that might be considered for inclusion in the draft standard as it develops.

1.1 Structure of the Send and Receive Library

This implementation provides a relatively large number of simple operations that are small and therefore easy to describe precisely. Larger operations can then be implemented and defined in terms of these operations. The basic send and receive operations are broken down as follows:

\[
\begin{bmatrix}
c \\ s \\ g \\ b \\ n \\ h \\ r
\end{bmatrix}
\begin{bmatrix}
\text{send} \\ \text{recv}
\end{bmatrix}
\begin{bmatrix}
\rightarrow \\ \rightarrow
\end{bmatrix},
\]

where the first letter specifies the layout of the data, the second specifies the extent to which the calling process synchronizes with the local message-passing subsystem and with the remote one, the next four letters specify a send or receive operation, and the final letter(s) specify optional additional functionality.

- Data layout
  - c (contiguous) The bytes are to be sent from or received into a contiguous region of memory, described by starting address and length.
  - s (stride) The data to be sent consists of data items of the same type and size, separated by a constant distance (stride) in memory, described by a starting address, length of a single item, and number of items.
g (gather) The data to be sent consists of data items of varying lengths and addresses, described by an array of addresses and lengths.

- Synchronization
  n (nonblocking) The operation returns control to the user immediately, to facilitate overlapping computation and communication.
  b (blocking) the operation does not return until the message area is available for reuse.
  s (synchronous) The operation does not return until the message has been received by the destination process (in the case of a send) or the acknowledgment has been sent (in the case of a receive).

- Send or Receive
  send (send) Transfer a message from the calling process to the specified one.
  recv (receive) Transfer a message into memory if specified conditions are met.

- Heterogeneity
  w (default) This is the default.
  h (heterogeneous) The message will be processed in such a way that differing data representations on different machines will be taken into account. On the [c] and [s] routines this requires a data type parameter. On the [g] operations, it requires data type information in the vector describing the data location.

- Alternate Protocol
  w (default) This is the default.
  rr (receiver ready) The operation will take advantage of any underlying protocol that is available when the receiver of a message is known to have issued the receive before the corresponding send is executed.

The routines specified in the current draft standard, which use a mode argument to specify the synchronization level, can easily be defined (and implemented) in terms of these operations, and we have done so. This organization makes it easy to experiment with and understand capabilities not currently in the draft, such as the rr and h options.

1.2 C and Fortran

The implementation described here is for the C language. Enough of the Fortran versions of these routines have been provided to write and run a simple program. These are the routines MPI.csend, MPI.crecv, a few inquiry routines (e.g., MPI.getid), and MPI.main.

1.3 Additions

In order to write actual programs, it was necessary to add a few routines for program management. We have added the routines MPI.main and MPI.stopall for this purpose.

We also have suggested a small set of routines (see Section 4) that are at a higher level than the draft standard, in order to meet the needs of users who wish to express a message-passing algorithm, but who do not need to know about the more complex performance issues this implementation and the draft standard itself tries to address, such as overlapping communication and computation with n or reducing latency with rr.

We also added an error value for "unknown mode."
1.4 Omissions

We have not implemented the time and date routines since there seemed to be a consensus that they need not be part of this standard. We can add them if they are desired. We did not implement the \texttt{MPI.pack}, etc., routines because they now seem to be subsumed by the [s] and [g] versions of \texttt{send} and \texttt{receive}. The man pages do not yet include the "Description" section; this can in most cases be taken from the draft standard.

1.5 Restrictions

This is a very preliminary, fast implementation, designed to allow experimentation with at least some of the ideas in the draft standard. In the interest of getting it out very quickly, we have taken some shortcuts. We intend to remove these as time goes by. Currently (November 25, 1992) the following restrictions apply to the implementation of the draft standard:

- There is only one process group. Since the draft standard does not specify that there be more than one, this implementation is (draft) standard-conforming. However, it is not currently possible to run a program that uses more than one group. (You are likely to get the "too many groups" error.)
- There is only one communication context. Again, this conforms to the letter but not the spirit of the draft standard.
- Neither selection on source nor selection on type range (that is, the negative type values) is implemented. This restriction allows us to use existing (vendor-supplied) message-passing implementations on a variety of machines.

2 Comments on the Draft Standard

One reason for doing a prototype implementation is to identify potential problems with the specification of the standard. In this section, we detail some of the problems that we have detected.

- The routines that take mode for "blocking," "nonblocking," or "synchronous" return a value whose meaning depends on the value of mode.
- The routines to get information on "the last message" are tricky to specify precisely, because the notion of "the last message" is imprecise. In particular, "the last message" means the last message received, probed, or otherwise looked at. This means that an \texttt{MPI.probe} will change the values that these routines will return. It is also unclear how process groups and communications contexts affect the meaning of "the last message."
- The standard specifies a 32 bit type field. With 64 bit systems on the horizon, this seems shortsighted.
- Because there are no minimum number of communications contexts or process groups specified, an implementation can conform to the standard by providing a single communication context and a single process group. This is in fact what our implementation does.
- Were process groups to be implemented, the interpretation of the destination field (\texttt{dest}) and the return value from \texttt{MPI.infos} is unclear. Is it the process id? Is it the rank of the process in the current group? What is the meaning of receive-from-any-processor as a selection in a receive routine?
- There is no way for the user to control the ranking of processes in a process group.
There is no way to discover the length of a message before receiving it into a user buffer. This prohibits using dynamic memory allocation (either by \texttt{malloc} in C or by explicit allocation of work areas in Fortran 77) to manage messages that are of unknown length at compile time.

As the examples demonstrate, there is no attractive way to determine such simple things as the process id or the number of processes.

Error handling is unattractive. As written, the user must check return codes. While we agree that users (particularly software library writers) need this option, not all users will be diligent about checking the return codes. One of the examples below emphasizes this.

3 Starting and Stopping Processes

In this section we give the specifications for routines needed to support the creation and destruction of the processes that will be communicating. We have added a routine \texttt{MPI.stopall} that causes all processes in an application to exit.

It has been our experience that a major source of portability problems is in how a parallel program is started up and initializes its environment. We have added \texttt{MPI.main} as a standardized way to accomplish this. This replaces \texttt{main} in C and \texttt{PROGRAM} in Fortran 77. In addition, it may be useful to provide a subroutine-level interface for initializing the MPI package.

4 Very Simple User Interface

A very simple interface can be defined that consists of the routines

\begin{verbatim}
MPI_numprocs = MPI.infog(0, 0, 0)
MPI_mypid = MPI.getid(0)
MPI_send = MPI.csend
MPI_recv = MPI.crecv
\end{verbatim}

as well as \texttt{MPI.main} for defining a program.

5 Examples

We present here two programs that send a message around a ring of processors. The Fortran version of this program is

\begin{verbatim}
integer function MPI.main()
integer buf, size, type, np, right, left
integer actlen

size = 4
type = 3
np = MPI.infog(0, 0, 0)
right = mod(MPI.getid(0) + 1, np)
left = mod(MPI.getid(0) - 1 + np, np)

if (MPI.getid(0).eq. 0) then
  buf = 1
  actlen = MPI.csend( 'blocking', buf, right, type, size )
  actlen = MPI.crecv( 'blocking', buf, left, type, size )
else
\end{verbatim}
The C version of this program is

```c
#include "../include/mpi.h"

int MPI_main( argc, argv )
int argc;
char **argv;
{
  int buf, siz = sizeof(int), type = 3, np, right, left;

  np = MPI_infog( 0, 0, (int *)0 ); /* Number of processes */
  right = (MPI_getid(0) + 1) % np;
  left = (MPI_getid(0) - 1 + np) % np;

  if (MPI_getid(0) == 0) {
    buf = 1;
    MPI_csend( "blocking", &buf, right, type, siz );
    MPI_crecv( "blocking", &buf, left, type, siz );
  } else {
    MPI_crecv( "blocking", &buf, left, type, siz );
    MPI_csend( "blocking", &buf, right, type, siz );
  }
  return 0;
}
```

6 Availability

This implementation is available by anonymous ftp from info.mcs.anl.gov. In the directory pub/mpi the file mpi.man.ps.Z is this document. The file mpi.tar.Z contains a compressed tar file of this implementation. The implementation is built on top of the Chameleon system; the file chameleon.tar.Z is all that is needed to build Chameleon for a variety of parallel computers (including groups of workstations). To use MPI on a system of workstations, either p4 or pvm are needed. Both are available from netlib (netlib@ornl.gov); we have included a recent version of p4 in the file p4-1.2.tar.Z in the /pub/mpi directory for convenience.

7 Basic Routines for Point-to-Point Messages

In this section we describe the routines that form the lowest level of the implementation. They are designed to be consistent with the upper-level routines.
7.1 General Communication Control

**MPI_cancel**

Cancel a previously initiated nonblocking send or receive

**Input Parameter**

msgid Message id returned by a call to a nonblocking send or receive

**Synopsis**

```c
int MPI_cancel( msgid )
int msgid;
```

**Location**

mpi.c

**MPI_infos**

Determine the source process of a pending receive

**Synopsis**

```c
int MPI_infos()
```

**Returns**

The source of the just received message.

**Location**

mpi.h

**MPI_infot**

Determine the type of the last receive

**Synopsis**

```c
int MPI_infot()
```
Returns

The source of the just received message.

Location

mpi.h

---

**MPI probe**

**MPI probe** — Check for pending messages

**Input Parameters**

*source* the PID of the process sending the message
*type* the message type

**Returns**

Length of pending message if available, else -1.

**Synopsis**

```c
int MPI_probe( source, type )
int source, type;
```

Location

mpi.c

---

**MPI stats**

**MPI stats** — Check the status of a nonblocking send or receive

**Input Parameter**

*msg_id* Message id returned by a call to a nonblocking send or receive

**Returns**

Length of available message if it is pending, else -1.

**Synopsis**

```c
int MPI_stats( msg_id )
int msg_id;
```
Location

mpi.c

MPI_wait

MPI_wait - Block until a nonblocking send or receive operation has completed

Input Parameter

msg_id  id returned by a nonblocking send or receive routine (of any type)

Returns

Number of bytes sent or received, or -1 on error.

Synopsis

int MPI_wait( msg_id )
int msg_id;

Location

mpi.c

7.2 Contiguous Communications

7.2.1 General Routines

MPI_crev

MPI_crev - Draft standard contiguous receive

Input Parameters

mode  one of "blocking," "nonblocking," or "synchronous"
buf  buffer to receive into
source  sending processor
type  message type
maxlen  maximum length in bytes of message

Returns

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc.,
or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.
Synopsis

```c
int MPI_crecv( mode, buf, source, type, maxlen )
char *mode;
void *buf;
int source, type, maxlen;
```

Location

mpic.c

---

**MPI_crecv** -- Friendly contiguous heterogeneous receive

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **buf**: buffer to receive into
- **source**: sending processor
- **type**: message type
- **maxlen**: maximum length in bytes of message
- **datatype**: type of data

**Returns**

- If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.
- Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

```c
int MPI_crecvh( mode, buf, source, type, maxlen, datatype )
char *mode;
void *buf;
int source, type, maxlen, datatype;
```

Location

mpic.c

---

**MPI_crecvh** -- Friendly contiguous heterogeneous receive for ready receivers

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
buf buffer to receive into
source sending processor
type message type
maxlen maximum length in bytes of message
datatype type of data

Returns

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

int MPI_crecvrr( mode, buf, source, type, maxlen, datatype )
char *mode;
void *buf;
int source, type, maxlen, datatype;

Location

mpic.c

MPI_crecvrr — Friendly contiguous receive for ready receivers

Input Parameters

mode one of "blocking," "nonblocking," or "synchronous"
buf buffer to receive into
source sending processor
type message type
maxlen maximum length in bytes of message

Returns

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

int MPI_crecvrr( mode, buf, source, type, maxlen )
char *mode;
void *buf;
int source, type, maxlen;
**MPI_csend** — Draft standard contiguous send

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **buf**: buffer to send
- **dest**: destination processor
- **type**: message type
- **len**: length in bytes of message

**Returns**

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error. Otherwise, returns the actual length of the message in bytes, or -1 on error.

**Synopsis**

```c
int MPI_csend( mode, buf, dest, type, len )
char *mode;
void *buf;
int dest, type, len;
```

**Location**

`mpic.c`

---

**MPI_csendh** — Friendly contiguous heterogeneous send

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **buf**: buffer to send
- **dest**: destination processor
- **type**: message type
- **len**: length in bytes of message
- **datatype**: type of data
Returns

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

```
int MPI_csendh( mode, buf, dest, type, len, datatype )
char *mode;
void *buf;
int dest, type, len, datatype;
```

Location

mpic.c

---

MPI_csendhrr — Friendly contiguous heterogeneous send for ready receivers

Input Parameters

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **buf**: buffer to send
- **dest**: destination processor
- **type**: message type
- **len**: length in bytes of message
- **datatype**: type of data

Returns

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

```
int MPI_csendhrr( mode, buf, dest, type, len, datatype )
char *mode;
void *buf;
int dest, type, len, datatype;
```

Location

mpic.c
**MPI_csendrr** - Friendly contiguous send for ready receivers

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **buf**: buffer to send
- **dest**: destination processor
- **type**: message type
- **len**: length in bytes of message

**Returns**

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

**Synopsis**

```c
define MPI_csendrr( mode, buf, dest, type, len )
    char *mode;
    void *buf;
    int dest, type, len;
```

**Location**

mpic.c

**7.2.2 Blocking Routines**

**MPI_cbrrecv**

**MPI_cbrrecv** - Blocking contiguous receive

**Input Parameters**

- **buf**: buffer to receive into
- **source**: sending processor
- **type**: message type
- **maxlen**: maximum length in bytes of message

**Returns**

Number of bytes actually received.
Synopsis

```c
int MPI_cbrecv( buf, source, type, maxlen )
void *buf;
int source, type, maxlen;
```

Location
mpic.c

---

**Synopsis**

`MPI_cbrecv()`: Blocking contiguous heterogeneous receive

**Input Parameters**

- `buf`: buffer to receive into
- `source`: sending processor
- `type`: message type
- `maxlen`: maximum length in bytes of message
- `datatype`: type of data

**Returns**

Number of bytes actually received.

---

**Synopsis**

`MPI_cbrecvhi()`: Blocking contiguous heterogeneous receive for ready receivers

**Input Parameters**

- `buf`: buffer to receive into
- `source`: sending processor
- `type`: message type
- `maxlen`: maximum length in bytes of message
- `datatype`: type of data

Location
mpic.c
Returns

Number of bytes actually received.

Synopsis

```c
int MPI_cbrecvhrr( buf, source, type, maxlen, datatype )
void *buf;
int source, type, maxlen, datatype;
```

Location

`mpic.c`

---

**MPI_chrecvrr** — Blocking contiguous receive for ready receivers

**Input Parameters**

- `buf` buffer to receive into
- `source` sending processor
- `type` message type
- `maxlen` maximum length in bytes of message

**Returns**

Number of bytes actually received.

Synopsis

```c
int MPI_cbrecvhrr( buf, source, type, maxlen )
void *buf;
int source, type, maxlen;
```

Location

`mpic.c`

---

**MPI_chsend** — Blocking contiguous send

**Input Parameters**

- `buf` buffer to send
- `dest` destination
Returns

Number of bytes sent, or -1 for an error.

Synopsis

```c
int MPI_cbsend( buf, dest, type, len )
void *buf;
int dest, type, len;
```

Location

`mpic.c`

---

**MPI_cbsendh**

`MPI_cbsendh` — Blocking contiguous heterogeneous send

**Input Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>buffer to send</td>
</tr>
<tr>
<td>dest</td>
<td>destination</td>
</tr>
<tr>
<td>type</td>
<td>message type</td>
</tr>
<tr>
<td>len</td>
<td>length in bytes of message</td>
</tr>
<tr>
<td>datatype</td>
<td>type of data</td>
</tr>
</tbody>
</table>

Returns

Number of bytes sent, or -1 for an error.

Synopsis

```c
int MPI_cbsendh( buf, dest, type, len, datatype )
void *buf;
int dest, type, len, datatype;
```

Location

`mpic.c`

---

**MPI_cbsendhrr**

`MPI_cbsendhrr` — Blocking contiguous heterogeneous send for ready receivers

16
Input Parameters

buf  buffer to send
dest destination
type  message type
len   length in bytes of message
datatype type of data

Returns

Number of bytes sent, or -1 for an error.

Synopsis

int MPI_cbsendhrr( buf, dest, type, len, datatype )
void *buf;
int dest, type, len, datatype;

Location

mpic.c

MPI_cbsendrr

MPI_cbsendrr -- Blocking contiguous send for ready receivers

Input Parameters

buf  buffer to send
dest destination
type  message type
len   length in bytes of message

Returns

Number of bytes sent, or -1 for an error.

Synopsis

int MPI_cbsendrr( buf, dest, type, len )
void *buf;
int dest, type, len;

Location

mpic.c
7.2.3 Nonblocking Routines

MPI_cnrrecv

MPI_cnrrecv — Nonblocking contiguous receive

Input Parameters

buf buffer to receive into
source sending processor
type message type
maxlen maximum length in bytes of message

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

int MPI_cnrrecv( buf, source, type, maxlen )
void *buf;
int source, type, maxlen;

Location

mpic.c

MPI_cnrrecvh

MPI_cnrrecvh — Nonblocking contiguous heterogeneous receive

Input Parameters

buf buffer to receive into
source sending processor
type message type
maxlen maximum length in bytes of message
datatype type of data

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

int MPI_cnrrecvh( buf, source, type, maxlen, datatype )
void *buf;
int source, type, maxlen, datatype;
Location
mpic.c

MPI_{recvhrr

MPI_{recvhrr -- Nonblocking contiguous heterogeneous receive for ready receivers

Input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>buffer to receive into</td>
</tr>
<tr>
<td>source</td>
<td>sending processor</td>
</tr>
<tr>
<td>type</td>
<td>message type</td>
</tr>
<tr>
<td>maxlen</td>
<td>maximum length in bytes of message</td>
</tr>
<tr>
<td>datatype</td>
<td>type of data</td>
</tr>
</tbody>
</table>

Returns

Integer id of receive to be used in MPI_{wait, etc., or -1 on error.

Synopsis

```c
int MPI_{recvhrr(buf, source, type, maxlen, datatype)
void *buf;
int source, type, maxlen, datatype;
```

Location
mpic.c

MPI_{recvrr

MPI_{recvrr -- Nonblocking contiguous receive for ready receivers

Input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>buffer to receive into</td>
</tr>
<tr>
<td>source</td>
<td>sending processor</td>
</tr>
<tr>
<td>type</td>
<td>message type</td>
</tr>
<tr>
<td>maxlen</td>
<td>maximum length in bytes of message</td>
</tr>
</tbody>
</table>

Returns

Integer id of receive to be used in MPI_{wait, etc., or -1 on error.
Synopsis

    int MPI_cnrecvrr( buf, source, type, maxlen )
    void *buf;
    int source, type, maxlen;

Location

    mpic.c

---

**MPI cnsend**

**MPI cnsend** — Nonblocking contiguous send

**Input Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>buffer to send</td>
</tr>
<tr>
<td>dest</td>
<td>destination</td>
</tr>
<tr>
<td>type</td>
<td>message type</td>
</tr>
<tr>
<td>len</td>
<td>length in bytes of message</td>
</tr>
</tbody>
</table>

**Returns**

    Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

    int MPI_cnsend( buf, dest, type, len )
    void *buf;
    int dest, type, len;

Location

    mpic.c

---

**MPI cnsendhl**

**MPI cnsendhl** — Nonblocking contiguous heterogeneous send

**Input Parameters**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>buffer to send</td>
</tr>
<tr>
<td>dest</td>
<td>destination</td>
</tr>
<tr>
<td>type</td>
<td>message type</td>
</tr>
<tr>
<td>len</td>
<td>length in bytes of message</td>
</tr>
<tr>
<td>datatype</td>
<td>type of data</td>
</tr>
</tbody>
</table>
Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

int MPI_cnsendh(buf, dest, type, len, datatype)
void *buf;
int dest, type, len, datatype;

Location

mpic.c

MPI_cnsendhrr

MPI_cnsendhrr Nonblocking contiguous heterogeneous send for ready receivers

Input Parameters

buf buffer to send
dest destination
type message type
len length in bytes of message
datatype type of data

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

int MPI_cnsendhrr(buf, dest, type, len, datatype)
void *buf;
int dest, type, len, datatype;

Location

mpic.c

MPI_cnsendhr

MPI_cnsendhr Nonblocking contiguous send for ready receivers

Input Parameters

buf buffer to receive into
source  sending processor

Returns

Returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

int MPI_cnsendrr( buf, dest, type, len )
void *buf;
int dest, type, len;

Location

mpic.c

7.3 Constant Stride Communications

7.3.1 General Routines

Synopsis

int MPI_srecv( mode, buf, source, type, lenblk, stride, nblks )
char *mode;
void *buf;
int source, type, lenblk, stride, nblks;
**Location**

mpis.c

---

**MPI_srecvh**

---

**MPI_srecvh** — Friendly heterogeneous receive into buffer with constant stride

**Input Parameters**

- `mode`: one of “blocking,” “nonblocking,” or “synchronous”
- `buf`: buffer to receive into
- `source`: sending processor
- `type`: message type
- `lenblk`: size in bytes of each data block
- `stride`: number of bytes between the start of each data block
- `nblks`: maximum number of data blocks
- `datatype`: type of data

**Returns**

If the mode is “nonblocking,” returns the integer id of receive to be used in `MPI_wait`, etc., or -1 on error.

Otherwise, returns the actual length of the message in bytes, or -1 on error.

**Synopsis**

```c
int MPI_srecvh( mode, buf, source, type, lenblk, stride, nblks, datatype )
cchar *mode;
void *buf;
int source, type, lenblk, stride, nblks, datatype;
```

**Location**

mpis.c

---

**MPI_srecvhr**

---

**MPI_srecvhr** — Friendly heterogeneous receive into a buffer with constant stride for ready receivers

**Input Parameters**

- `mode`: one of “blocking,” “nonblocking,” or “synchronous”
- `buf`: buffer to receive into
- `source`: sending processor
- `type`: message type
- `lenblk`: size in bytes of each data block
- `stride`: number of bytes between the start of each data block
Returns

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

int MPI_srecvhrr( mode, buf, source, type, lenblk, stride, nblks, datatype )
char *mode;
void *buf;
int source, type, lenblk, stride, nblks, datatype;

Location

mpis.c

---

MPI_srecvrr | MPI_srecvrr

MPI_srecvrr Friendly receive into buffer with constant stride for ready receivers

Input Parameters

mode one of "blocking," "nonblocking," or "synchronous"
buf buffer to receive into
source sending processor
type message type
lenblk size in bytes of each data block
stride number of bytes between the start of each data block
nblks maximum number of data blocks

Returns

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

int MPI_srecvrr( mode, buf, source, type, lenblk, stride, nblks )
char *mode;
void *buf;
int source, type, lenblk, stride, nblks;

Location

mpis.c
MPI_ssend — Draft standard send with constant stride

Input Parameters

mode one of “blocking,” “nonblocking,” or “synchronous”
buf buffer to send
dest destination processor
type message type
lenblk size in bytes of each data block
stride number of bytes between the start of each data block
nblks number of data blocks

Returns

If the mode is “nonblocking,” returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

int MPI_ssend( mode, buf, dest, type, lenblk, stride, nblks )
char *mode;
void *buf;
int dest, type, lenblk, stride, nblks;

Location

mpis.c

MPI_ssendh — Friendly heterogeneous send with constant stride

Input Parameters

mode one of “blocking,” “nonblocking,” or “synchronous”
buf buffer to send
dest destination processor
type message type
lenblk size in bytes of each data block
stride number of bytes between the start of each data block
nblks number of data blocks
datatype type of data
Returns

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

int MPI_ssendh( mode, buf, dest, type, lenblk, stride, nblks, datatype )
char *mode;
void *buf;
int dest, type, lenblk, stride, nblks, datatype;

Location

mpis.c

MPI_ssendhrr — Friendly heterogeneous send with constant stride for ready receivers

Input Parameters

mode one of "blocking," "nonblocking," or "synchronous"
buf buffer to send
dest destination processor
type message type
lenblk size in bytes of each data block
stride number of bytes between the start of each data block
nblks number of data blocks
datatype type of data

Returns

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

int MPI_ssendhrr( mode, buf, dest, type, lenblk, stride, nblks, datatype )
char *mode;
void *buf;
int dest, type, lenblk, stride, nblks, datatype;

Location

mpis.c
**MPI_ssendrr** -- Friendly send with constant stride for ready receivers

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **buf**: buffer to send
- **dest**: destination processor
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: number of data blocks

**Returns**

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.

Otherwise, returns the actual length of the message in bytes, or -1 on error.

**Synopsis**

```c
int MPI_ssendrr( mode, buf, dest, type, lenblk,stride,nblks );
```

**Location**

`mpis.c`

### 7.3.2 Blocking Routines

**MPI_sbrrev** -- Blocking receive into buffer with constant stride

**Input Parameters**

- **buf**: buffer to receive into
- **source**: sending processor
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: maximum number of data blocks
Returns

Number of bytes actually received.

Synopsis

```c
int MPI_sendreq( buf, source, type, lenblk, stride, nblks )
void *buf;
int source, type, lenblk, stride, nblks;
```

Location

mpis.c

---

**MPI_sendreq**

MPI_sendreq - Blocking heterogeneous receive into buffer with constant stride

**Input Parameters**

- **buf** buffer to receive into
- **source** sending processor
- **type** message type
- **lenblk** size in bytes of each data block
- **stride** number of bytes between the start of each data block
- **nblks** maximum number of data blocks
- **datatype** type of data

**Returns**

Number of bytes actually received.

Synopsis

```c
int MPI_sendreq( buf, source, type, lenblk, stride, nblks, datatype )
void *buf;
int source, type, lenblk, stride, nblks, datatype;
```

Location

mpis.c

---

**MPI_sendreq**

MPI_sendreq - Blocking heterogeneous receive into buffer with constant stride for ready receivers
Input Parameters

buf         buffer to receive into
source      sending processor
type        message type
lenblk      size in bytes of each data block
stride      number of bytes between the start of each data block
nblks       maximum number of data blocks
datatype    type of data

Returns

Number of bytes actually received.

Synopsis

int MPI_sbrecvrr( buf, source, type, lenblk, stride, nblks, datatype )
void *buf;
int source, type, lenblk, stride, nblks, datatype;

Location

mpis.c

MPI_sbrecvrr — Blocking receive into buffer with constant stride for ready receivers

Input Parameters

buf         buffer to receive into
source      sending processor
type        message type
lenblk      size in bytes of each data block
stride      number of bytes between the start of each data block
nblks       maximum number of data blocks

Returns

Number of bytes actually received.

Synopsis

int MPI_sbrecvrr( buf, source, type, lenblk, stride, nblks )
void *buf;
int source, type, lenblk, stride, nblks;

Location

mpis.c
### MPI_sbsend

**Blocking send with constant stride**

**Input Parameters**

- **buf**: buffer to send
- **dest**: destination
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: number of data blocks

**Returns**

Number of bytes sent, or -1 for an error.

**Synopsis**

```c
int MPI_sbsend( buf, dest, type, lenblk, stride, nblks )
```

**Location**

`mpis.c`

---

### MPI_sbsendh

**Blocking heterogeneous send with constant stride**

**Input Parameters**

- **buf**: buffer to send
- **dest**: destination
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: number of data blocks
- **datatype**: type of data

**Returns**

Number of bytes sent, or -1 for an error.
Synopsis

```c
int MPI_sbsendh( buf, dest, type, lenblk, stride, nblks, datatype )
void *buf;
int dest, type, lenblk, stride, nblks, datatype;
```

Location

`mpis.c`

---

**MPI_sbsendhrr**

MPI_sbsendhrr - Blocking heterogeneous send with constant stride for ready receivers

### Input Parameters

- **buf**: buffer to send
- **dest**: destination
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: number of data blocks
- **datatype**: type of data

### Returns

Number of bytes sent, or -1 for an error.

---

Synopsis

```c
int MPI_sbsendhrr( buf, dest, type, lenblk, stride, nblks, datatype )
void *buf;
int dest, type, lenblk, stride, nblks, datatype;
```

Location

`mpis.c`

---

**MPI_sbsendrr**

MPI_sbsendrr - Blocking send with constant stride for ready receivers

### Input Parameters

- **buf**: buffer to send
- **dest**: destination
- **type**: message type
- **lenblk**: size in bytes of each data block

31
stride  number of bytes between the start of each data block
nblks  number of data blocks

Returns

Number of bytes sent, or -1 for an error.

Synopsis

```c
int MPI_sbsendrr( buf, dest, type, lenblk, stride, nblks )
void *buf;
int dest, type, lenblk, stride, nblks;
```

Location

mpis.c

7.3.3 Nonblocking Routines

<table>
<thead>
<tr>
<th>MPI_snsrecv</th>
<th>MPI_snsrecv</th>
</tr>
</thead>
</table>

MPI_snsrecv -- Nonblocking receive into buffer with constant stride

Input Parameters

- **buf**  buffer to receive into
- **source**  sending processor
- **type**  message type
- **lenblk**  size in bytes of each data block
- **stride**  number of bytes between the start of each data block
- **nblks**  maximum number of data blocks

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snsrecv( buf, source, type, lenblk, stride, nblks )
void *buf;
int source, type, lenblk, stride, nblks;
```

Location

mpis.c
MPI_snrecvh -- Nonblocking heterogeneous receive into buffer with constant stride

Input Parameters

- **buf**: buffer to receive into
- **source**: sending processor
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: maximum number of data blocks
- **datatype**: type of data

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snrecvh( buf, source, type, lenblk, stride, nblks, datatype );
```

Location

`mpis.c`

MPI_snrecvhrr

MPI_snrecvhrr -- Nonblocking heterogeneous receive into buffer with constant stride for ready receivers

Input Parameters

- **buf**: buffer to receive into
- **source**: sending processor
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: maximum number of data blocks
- **datatype**: type of data

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.
Synopsis

```c
int MPI_snrecvrr( buf, source, type, lenblk, stride, nblks, datatype )
void *buf;
int source, type, lenblk, stride, nblks, datatype;
```

Location

`mpis.c`

---

**MPI_snrecvrr**

MPI_snrecvrr -- Nonblocking receive into buffer with constant stride for ready receivers

**Input Parameters**

- **buf**: buffer to receive into
- **source**: sending processor
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: maximum number of data blocks

**Returns**

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snrecvrr( buf, source, type, lenblk, stride, nblks )
void *buf;
int source, type, lenblk, stride, nblks;
```

Location

`mpis.c`

---

**MPI_ssend**

MPI_ssend -- Nonblocking send with constant stride

**Input Parameters**

- **buf**: buffer to send
- **dest**: destination
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
number of data blocks

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snsend( buf, dest, type, lenblk, stride, nblks )
void *buf;
int dest, type, lenblk, stride, nblks;
```

Location

`mpis.c`

---

Input Parameters

- **buf**: Buffer to send
- **dest**: Destination
- **type**: Message type
- **lenblk**: Size in bytes of each data block
- **stride**: Number of bytes between the start of each data block
- **nblks**: Number of data blocks
- **datatype**: Type of data

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snsendh( buf, dest, type, lenblk, stride, nblks, datatype )
void *buf;
int dest, type, lenblk, stride, nblks, datatype;
```

Location

`mpis.c`

---

Input Parameters

MPI_snsendh -- Nonblocking heterogeneous send with constant stride

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snsendh( buf, dest, type, lenblk, stride, nblks, datatype )
void *buf;
int dest, type, lenblk, stride, nblks, datatype;
```

Location

`mpis.c`

---

MPI_snsendhrr

MPI_snsendhrr -- Nonblocking heterogeneous send with constant stride for ready receivers
Input Parameters

- **buf**: buffer to send
- **dest**: destination
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: number of data blocks
- **datatype**: type of data

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snsendhrr( buf, dest, type, lenblk, stride, nblks, datatype )
void *buf;
int dest, type, lenblk, stride, nblks, datatype;
```

Location

`mpis.c`

---

MPI_snsendrr -- Nonblocking send with constant stride for ready receivers

Input Parameters

- **buf**: buffer to receive into
- **source**: sending processor
- **type**: message type
- **lenblk**: size in bytes of each data block
- **stride**: number of bytes between the start of each data block
- **nblks**: number of data blocks

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_snsendrr( buf, dest, type, lenblk, stride, nblks )
void *buf;
int dest, type, lenblk, stride, nblks;
```

Location

`mpis.c`
7.4 General Scatter/Gather

These routines use structures named MPI_DATAVEC and MPI_HDATAVEC (for heterogeneous communication). The definitions of these are in mpi.h. MPI_DATAVEC is the same as struct iovec; this is a structure that contains a pointer to data and the size of that data in bytes. MPI_HDATAVEC adds a value that is the type of the data, in MPI format.

7.4.1 General Routines

**MPI_grecv**

MPI_grecv -- Draft standard receive into buffer with arbitrary scatter

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **source**: sending processor
- **type**: message type
- **desc**: description of data to be scattered, in MPI_DATAVEC format
- **bcnt**: number of blocks

**Returns**

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.

Otherwise, returns the actual length of the message in bytes, or -1 on error.

**Synopsis**

```c
int MPI_grecv( mode, source, type, desc, bcnt )
char  *mode;
int    source, type, bcnt;
MPI_DATAVEC *desc;
```

**Location**

mpig.c

**MPI_greevh**

MPI_greevh Friendly heterogeneous receive into buffer with arbitrary scatter

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **source**: sending processor
- **type**: message type
- **desc**: description of data to be scattered, in MPI_DATAVEC format
Synopsis

```c
int MPI_grech( mode, source, type, desc, bcnt )
char *mode;
int source, type, bcnt;
MPI_HDATAVEC *desc;
```

Returns

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Location

`mpig.c`

---

MPI_grechrr --- Friendly heterogeneous receive into a buffer with arbitrary scatter for ready receivers

Input Parameters

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **source**: sending processor
- **type**: message type
- **desc**: description of data to be scattered, in MPI_DATAVEC format
- **bcnt**: number of blocks

Returns

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

```c
int MPI_grechrr( mode, source, type, desc, bcnt )
char *mode;
int source, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

`mpig.c`
**MPI_grecvrr** — Friendly receive into buffer with arbitrary scatter for ready receivers

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **source**: sending processor
- **type**: message type
- **desc**: description of data to be scattered, in MPI_DATAVEC format
- **bcnt**: number of blocks

**Returns**

If the mode is "nonblocking," returns the integer id of receive to be used in MPI_wait, etc., or -1 on error.

Otherwise, returns the actual length of the message in bytes, or -1 on error.

**Synopsis**

```c
int MPI_grecvrr( mode, source, type, desc, bcnt )
char *mode;
int source, type, bcnt;
MPI_DATAVEC *desc;
```

**Location**

`mpig.c`

---

**MPI_gsend** — Draft standard send with arbitrary gather

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **dest**: destination processor
- **type**: message type
- **desc**: description of data to be gathered, in MPI_DATAVEC format
- **bcnt**: number of blocks

**Returns**

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.

Otherwise, returns the actual length of the message in bytes, or -1 on error.
Synopsis

```c
int MPI_gsend( mode, dest, type, desc, bcnt )
char   *mode;
int    dest, type, bcnt;
MPI_DATAVEC *desc;
```

Location

`mpig.c`

---

**MPI_gsendh**

MPI_gsendh — Friendly heterogeneous send with arbitrary gather

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **dest**: destination processor
- **type**: message type
- **desc**: description of data to be gathered, in MPI_DATAVEC format
- **bcnt**: number of blocks

**Returns**

- If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
- Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

```c
int MPI_gsendh( mode, dest, type, desc, bcnt )
char   *mode;
int    dest, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

`mpig.c`

---

**MPI_gsendhrr**

MPI_gsendhrr — Friendly heterogeneous send with arbitrary scatter for ready receivers

**Input Parameters**

- **mode**: one of "blocking," "nonblocking," or "synchronous"
- **dest**: destination processor
Returns

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

```c
int MPI_gsendrr( mode, dest, type, desc, bcnt )
```

```
char *mode;
int dest, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

mpig.c

MPI_gsendrr

MPI_gsendrr — Friendly send with arbitrary gather for ready receivers

Input Parameters

mode one of "blocking," "nonblocking," or "synchronous"
derst destination processor
type message type
desc description of data to be gathered, in MPI_DATAVEC format
bcnt number of blocks

Returns

If the mode is "nonblocking," returns the integer id of send to be used in MPI_wait, etc., or -1 on error.
Otherwise, returns the actual length of the message in bytes, or -1 on error.

Synopsis

```c
int MPI_gsendrr( mode, dest, type, desc, bcnt )
```

```
char *mode;
int dest, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

mpig.c
7.4.2 Blocking Routines

**MPI_gbrccv**

**MPI_gbrccv** — Blocking receive into buffer with arbitrary scatter

**Input Parameters**

source : sending processor

type : message type

desc : description of data to be scattered, in MPI_DATAVEC format

bcnt : number of blocks

**Returns**

Number of bytes actually received.

**Synopsis**

```c
int MPI_gbrccv( source, type, desc, bcnt );
```

**Location**

`mpig.c`

---

**MPI_gbrccvh**

**MPI_gbrccvh** — Blocking heterogeneous receive into buffer with arbitrary scatter

**Input Parameters**

source : sending processor

type : message type

desc : description of data to be scattered, in MPI_HDATAVEC format

bcnt : number of blocks

**Returns**

Number of bytes actually received.

**Synopsis**

```c
int MPI_gbrccvh( source, type, desc, bcnt );
```
MPI_gbrecvhrr -- Blocking heterogeneous receive into buffer with arbitrary scatter for ready receivers

Input Parameters

source: sending processor

type: message type

desc: description of data to be scattered, in MPI_DATAVEC format

cnt: number of blocks

Returns

Number of bytes actually received.

Synopsis

```c
int MPI_gbrecvhrr( source, type, desc, bcnt )
```

Location

`mpig.c`

MPI_gbrecvrr -- Blocking receive into buffer with arbitrary scatter for ready receivers

Input Parameters

source: sending processor

type: message type

desc: description of data to be scattered, in MPI_DATAVEC format

cnt: number of blocks

Returns

Number of bytes actually received.
Synopsis

```c
int MPI_gbsend( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_DATAVEC *desc;
```

Location

`mpig.c`

---

**MPI_gbsend** — Blocking send with arbitrary gather

### Input Parameters

- `dest` destination
- `type` message type
- `desc` description of data to be gathered, in MPI_DATAVEC format
- `bcnt` number of blocks

### Returns

Number of bytes sent, or -1 for an error.

---

### Synopsis

```c
int MPI_gbsend( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_DATAVEC *desc;
```

### Location

`mpig.c`

---

**MPI_gbsendh** — Blocking send with arbitrary gather

### Input Parameters

- `dest` destination
- `type` message type
- `lenblk` size in bytes of each data block
- `desc` description of data to be gathered, in MPI_DATAVEC format
- `bcnt` number of blocks
Returns

Number of bytes sent, or -1 for an error.

Synopsis

```c
int MPI_gbsendh( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

`mpig.c`

MPI_gbsendhrr

MPI_gbsendhrr -- Blocking heterogeneous send with arbitrary gather for ready receivers

Input Parameters

<table>
<thead>
<tr>
<th>dest</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>message type</td>
</tr>
<tr>
<td>desc</td>
<td>description of data to be gathered, in MPI_DATAVEC format</td>
</tr>
<tr>
<td>bcnt</td>
<td>number of blocks</td>
</tr>
</tbody>
</table>

Returns

Number of bytes sent, or -1 for an error.

Synopsis

```c
int MPI_gbsendhrr( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

`mpig.c`

MPI_gbsendrr

MPI_gbsendrr -- Blocking send with arbitrary gather for ready receivers

Input Parameters

<table>
<thead>
<tr>
<th>dest</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>message type</td>
</tr>
</tbody>
</table>
Returns

Number of bytes sent, or -1 for an error.

Synopsis

```c
int MPI_gbsendr( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_DATAVEC *desc;
```

Location

`mpig.c`

7.4.3 Nonblocking Routines

**MPI_gncv**

Nonblocking receive into buffer with arbitrary scatter

Input Parameters

- `source`: sending processor
- `type`: message type
- `desc`: description of data to be scattered, in MPI_DATAVEC format
- `bcnt`: number of blocks

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_gncv( source, type, desc, bcnt )
int source, type, bcnt;
MPI_DATAVEC *desc;
```

Location

`mpig.c`

**MPI_gncv**

Nonblocking heterogeneous receive into buffer with arbitrary scatter
Input Parameters

- **source**: sending processor
- **type**: message type
- **desc**: description of data to be scattered, in MPI_DATAVEC format
- **bcnt**: number of blocks

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_gnrecvh( source, type, desc, bcnt )
int     source, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

`mpig.c`

---

**MPI_gnrecvhrr**

`MPI_gnrecvhrr` -- Nonblocking heterogeneous receive into buffer with arbitrary scatter for ready receivers

Input Parameters

- **source**: sending processor
- **type**: message type
- **desc**: description of data to be scattered, in MPI_DATAVEC format
- **bcnt**: number of blocks

Returns

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

Synopsis

```c
int MPI_gnrecvhrr( source, type, desc, bcnt )
int     source, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

`mpig.c`
**MPI_gnrecvrr** --- Nonblocking receive into buffer with arbitrary scatter for ready receivers

**Input Parameters**

- **source** sending processor
- **type** message type
- **desc** description of data to be scattered, in MPI_DATAVEC format
- **bcnt** number of blocks

**Returns**

Integer id of receive to be used in MPI_wait, etc., or -1 on error.

**Synopsis**

```c
int MPI_gnrecvrr( source, type, desc, bcnt )
int source, type, bcnt;
MPI_DATAVEC *desc;
```

**Location**

mpig.c

---

**MPI_gnsend** --- Nonblocking send with arbitrary gather

**Input Parameters**

- **dest** destination
- **type** message type
- **desc** description of data to be gathered, in MPI_DATAVEC format
- **bcnt** number of blocks

**Returns**

Integer id of send to be used in MPI_wait, etc., or -1 on error.

**Synopsis**

```c
int MPI_gnsend( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_DATAVEC *desc;
```
MPI_gnsendh

Nonblocking heterogeneous send with arbitrary gather

Input Parameters

dest  destination

Type  message type

desc  description of data to be gathered, in MPI_DATAVEC format

bcnt  number of blocks

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.

Synopsis

int MPI_gnsendh(dest, type, desc, bcnt)

Location

mpig.c

MPI_gnsendhrr

Nonblocking heterogeneous send with arbitrary gather for ready receivers

Input Parameters

dest  destination

Type  message type

Desc  description of data to be gathered, in MPI_DATAVEC format

bcnt  number of blocks

Returns

Integer id of send to be used in MPI_wait, etc., or -1 on error.
Synopsis

```c
int MPI_gnsendrr( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_HDATAVEC *desc;
```

Location

`mpig.c`

**MPI_gnsendrr** — Nonblocking send with arbitrary gather for ready receivers

**Input Parameters**

- **source**: sending processor
- **type**: message type
- **desc**: description of data to be gathered, in MPI_DATAVEC format
- **bcnt**: number of blocks

**Returns**

Integer id of send to be used in MPI_wait, etc., or -1 on error.

**Synopsis**

```c
int MPI_gnsendrr( dest, type, desc, bcnt )
int dest, type, bcnt;
MPI_DATAVEC *desc;
```

Location

`mpig.c`

**7.5 Process Groups**

**MPI_child** — Get information about the children of a specified group

**Input Parameters**

- **gid**: process group id
- **maxlis**: size of clist
Output Parameter
   clist    array containing process group id

Returns
   Number of children in the group, or -1 on error.

Synopsis
   int MPI_child( gid, maxlis, clist )
   int gid, maxlis, *clist;

Location
   group.c

---

MPI_copyg

MPI_copyg — Create a root group from an existing group

Input Parameter
   gid    process group id

Returns
   The gid of the new root group, or -1 on error.

Synopsis
   int MPI_copyg( gid )
   int gid;

Location
   group.c

---

MPI_defrg

MPI_defrg — Define a root group

Input Parameters
   nprocs    number of processes in the group
   plist     array of PID that are members of the group
Returns
The gid of the new root group, or -1 on error.

Synopsis
int MPI_defrg( nprocs, plist )
int nprocs, *plist;

Location
group.c

MPI_defrg

MPI_defrg -- Discard the descendants of a specified group

Input Parameter
gid group id

Returns
0 on success, or -1 on error.

Synopsis
int MPI_free( gid )
int gid;

Location
group.c

MPI_free

MPI_free -- Discard the specified group and all of its descendants

Input Parameter
gid group id

Returns
0 on success, or -1 on error.
Synopsis

    int MPI_freeg( gid )
    int gid;

Location

group.c

---

**MPI_getid**

MPI_getid - Determine the group context PID of the calling process for a specified group id number.

**Input Parameter**

gid group id

**Synopsis**

    int MPI_getid( gid )
    int gid;

**Returns**

Relative process number of the calling process in the specified group.

**Location**

mpi.h

---

**MPI_infog**

MPI_infog -- Determine the number of processes in a group and return the PID numbers of the group members

**Input Parameters**

gid group id
maxlis size of plist
plist integer array to hold the members of the group

**Returns**

The number of members in the group, or -1 on error.
Synopsis

    int MPI_infog(gid, maxlis, plist)
    int gid, maxlis, *plist;

Location

    mpi.c

---

MPI_parent

MPI_parent — Determine the group id number of the parent of a specified group

Input Parameter

    gid        group whose parent is to be found

Returns

    The group id of the parent, or -1 on error.

Synopsis

    int MPI_parent( gid )
    int gid;

Location

    group.c

---

MPI_partg

MPI_partg — Partition a group into subgroups

Input Parameters

    gid        group to be partitioned
    key        key whose value determines the partitioning

Returns

    The gid of the subgroup to which the calling process belongs, otherwise -1.

Synopsis

    int MPI_partg( gid, key )
    int gid, key;


MPI_popg

MPI_popg - Reestablish the process group context

Returns
The process group id that is reestablished as the root, otherwise -1.

Synopsis
int MPI_popg()

Location
group.c

MPI_pushg

MPI_pushg - Establish the process group context

Input Parameter
gid The group context to establish

Returns
The number of processes in the group gid, or -1 on error.

Synopsis
int MPI_pushg( gid )
int gid;

Location
group.c

MPI_rootg

MPI_rootg - Get information about root groups
Input Parameter

maxlis  size of rlist

Output Parameter

rlist  array of the process group id numbers

Returns

Number of defined root groups, or -1 on error.

Synopsis

int MPI_rootg( maxlis, rlist )
int maxlis, *rlist;

Location

group.c

MPI_siblg

MPI_siblg

MPI_siblg  --  Get information about the siblings of a group

Input Parameters

gid  process group id
maxlis  size of slist

Output Parameter

slist  array of the process group ids

Returns

Number of siblings, or -1 on error.

Synopsis

int MPI_siblg( gid, maxlis, slist )
int gid, maxlis, slist;

Location

group.c
7.6 Communication Contexts

MPI_infoc -- Get information about valid communication contexts

Input parameter

- maxlis  maximum size of ilist

Output parameter

- ilist  array containing communication context ID numbers

Synopsis

```c
int MPI_infoc( maxlis, ilist )
int maxlis, *ilist;
```

Location

`mpi.c`

MPI_newc -- Create a new communication context

Returns

The id number of a new communication context, or -1 on error.

Synopsis

```c
int MPI_newc()
```

Location

`mpi.c`

MPI_popc -- Reestablish former communication context

Returns

The id number of a new communication context, or -1 on error.
Synopsis

int MPI_popc()

Location

mpi.c

---

**MPI_pushc**

MPI_pushc — Establish a new communication context

**Input Parameter**

ccid the ID number of communication context to establish

**Returns**

0 on success or -1 on error.

**Synopsis**

int MPI_pushc( ccid )
int ccid;

**Location**

mpi.c

---

### 7.7 Program Management

**MPI_main**

MPI_main — Name of main program for MPI applications

**Note**

Use "MPI_main" instead of "main" in C programs, and instead of "PROGRAM <name>" in Fortran programs.

**Synopsis**

(C)

int MPI_main( argc, argv )
int argc;
char **argv;
Synopsis

(Fortran)
integer function MPI_main()

Location
mpi.c

MPI_stopall

MPI_stopall  stop all processes

Input Parameter

rc       return code to pass back to calling environment

Synopsis

void MPI_stopall(rc)

Location
mpi.c

7.8 Utility Routines

MPI_error

MPI_error — Determine the error status

Returns

Integer giving the error status for the preceding call to an MPI routine.

Synopsis

int MPI_error()

Location
mpi.c
MPI_etext -- Return the text corresponding to an error value

Input Parameter
  ierrno    Value returned by MPI_error

Synopsis
  char *MPI_etext( ierrno )
  int ierrno;

Location
  mpi.c

MPI_machine -- Get machine name, version, and related information

Returns
  A character string giving the name, etc., of the machine.

Synopsis
  char *MPI_machine( )

Location
  mpi.c

MPI_infoln -- Get information on the machine configuration

Input Parameter
  maxlis    the size of ilist

Output Parameter
  ilist     integer array containing information about the system.
ilist[0] - number of physical processors in the machine
ilist[1] - total number of processors in the machine

Synopsis

int MPI_inform( maxlis, ilist )
int maxlis, *ilist;

Location

mpi.c

MPI_sync

MPI_sync

MPI_sync -- Synchronize all processes

Synopsis

int sync

Location

mpi.h
Index

MPI_cancel, 6
MPI_brecv, 13
MPI_brecv(), 14
MPI_brecv(), 14
MPI_brecv(), 15
MPI bsend, 15
MPI_bsend(), 16
MPI_bsend(), 16
MPI_bsendr(), 17
MPI_child, 50
MPI_recei, 18
MPI_recv(), 18
MPI_recei(), 19
MPI_recv(), 19
MPI_send, 20
MPI_send(), 20
MPI_send(), 21
MPI_send(), 21
MPI_recv(), 51
MPI_recv(), 9
MPI_recv(), 9
MPI_recv(), 10
MPI_send, 11
MPI_send(), 11
MPI_send(), 12
MPI_send(), 13
MPI_defr(), 51
MPI_error, 59
MPI_text, 60
MPI_free, 52
MPI_free(), 52
MPI_grecv, 42
MPI_grecv(), 42
MPI_grecv(), 43
MPI_grecv(), 43
MPI_gsend, 44
MPI_gsend(), 44
MPI_gsend(), 45
MPI_gsend(), 45
MPI_ggetid, 53
MPI_grecv, 46
MPI_grecv(), 46
MPI_grecv(), 47
MPI_grecv(), 48
MPI_gsend, 48
MPI_gsend(), 49
MPI_gsend(), 49
MPI_gsendr(), 50
MPI_gsendr(), 50
MPI_grecv, 37
MPI_grecv(), 37
MPI_grecvh(), 38
MPI_grecvrr, 39
MPI_gsend, 39
MPI_gsend(), 40
MPI_gsend(), 40
MPI_gsendr(), 41
MPI_info, 57
MPI_info, 53
MPI_info(), 60
MPI_info, 6
MPI_info(), 6
MPI_info(), 60
MPI_main, 58
MPI_newc, 57
MPI_parent, 54
MPI_partg, 54
MPI_popc, 57
MPI_popc, 55
MPI_probe, 7
MPI_pushc, 58
MPI_pushc, 55
MPI_rootg, 55
MPI_srecv, 27
MPI_srecv(), 28
MPI_srecv(), 28
MPI_srecv(), 29
MPI_ssend, 30
MPI_ssend(), 30
MPI_ssend(), 31
MPI_ssendr(), 31
MPI_ssendr(), 31
MPI_sibig, 56
MPI_srecv, 32
MPI_srecv(), 33
MPI_srecv(), 33
MPI_srecv(), 34
MPI_ssend, 34
MPI_ssend(), 35
MPI_ssend(), 35
MPI_ssend(), 36
MPI_sreccv, 22
MPI_sreccv(), 23
MPI_sreccv(), 23
MPI_srecv, 24
MPI_ssend, 25
MPI_ssend(), 25
MPI_ssend(), 26
MPI_ssendr(), 27
MPI_stats, 7
MPI_stopall, 59
MPI_sync, 61
MPI_wait, 8

62
Distribution for ANL-92/47

Internal:

J. M. Beumer (100)
F. Y. Fradin
W. D. Gropp (10)
E. L. Lusk (10)
G. W. Pieper
R. L. Stevens
C. L. Wilkinson

ANL Patent Department
TIS File

External:

DOE-OSTI, for distribution per UC-405 (54)
ANL-E Library
ANL-W Library
Manager, Chicago Field Office, DOE
Mathematics and Computer Science Division Review Committee:

W. W. Bledsoe, University of Texas, Austin
B. L. Buzbee, National Center for Atmospheric Research
J. G. Glimm, State University of New York at Stony Brook
M. T. Heath, University of Illinois at Urbana
E. F. Infante, University of Minnesota
D. O’Leary, University of Maryland
R. E. O’Malley, Rensselaer Polytechnic Institute
M. H. Schultz, Yale University
H. Berryman, Yale University
J. Cavallini, Department of Energy - Energy Research
J. Choi, Oak Ridge National Laboratory
J. Dongarra, University of Tennessee
J. Flower, ParaSoft Corporation
D. Frye, IBM Corporation
A. Geist, Oak Ridge National Laboratory
I. Glendinning, University of Southampton, U.K.
A. Greenberg, Thinking Machines Corp.
L. Hart, National Oceanic and Atmospheric Administration
R. Hempel, GMD, Germany
T. Henderson, National Oceanic and Atmospheric Administration
T. Hey, University of Southampton, U.K.
M. Hill, University of Southampton, U.K.
C. Ho, IBM Almaden Research Center
F. Howes, Department of Energy - Energy Research
S. Huss-Lederman
J. Kapenga, Western Michigan University
K. Kennedy, Rice University
R. Knighten, Intel Scientific Computers
N. MacDonald, University of Edinburgh, U.K.
O. McBryan, University of Colorado, Boulder
C. Mosher, ARCO Exploration and Production Technology
S. Otto, Oregon Graduate Institute
C. Pancake, Oregon State University
R. Panda, IBM Corporation
M. Peters, IBM UK Scientific Centre
P. Pierce, Intel Scientific Computers
P. Rigsbee, Cray Research, Inc.
A. Skjellum, Mississippi State University
M. Snir, IBM T. J. Watson Research Center
V. Sunderam, Emory University
M. Surridge, University of Southampton, U.K.
L. Tucker, Thinking Machines Corp.
R. van de Geijn, University of Texas, Austin
D. Walker, Oak Ridge National Laboratory
L. Ward, IBM Corporation
S. Zenith, Kuck and Associates, Inc.