Chapter 1

Errata for MPI-1.1

This document was processed on October 12, 1998.

The known corrections to MPI-1.1 are listed in this document. All page and line numbers are for the official version of the MPI-1.1 document available from the MPI Forum home page at `http://www.mpi-forum.org`. Information on reporting mistakes in the MPI documents is also located on the MPI Forum home page.

- Page "i" (first page after title page), line 7 reads
  Version 1.0: June, 1994.
  but should read
  Version 1.0: May, 1994.

- Page "i" (first page after title page), line 16 reads
  June 12, 1995
  but should read
  May 5, 1994

- Page 11, line 36 reads
  MPI_ADDRESS
  but should read
  MPI_ADDRESS_TYPE

- Page 19, lines 1–2 reads
  for (64 bit) C integers declared to be of type longlong int
  but should read
  for C integers declared to be of type long long

- Page 40, line 48 should have the following text added:

  Advice to users. To prevent problems with the argument copying and register optimization done by Fortran compilers, please note the hints in subsections

"Problems Due to Data Copying and Sequence Association," and "A Problem with Register Optimization" in Section 10.2.2 of the MPI-2 Standard, pages 286 and 289. (*End of advice to users.*)

- Page 41, lines 16–18 reads
  A **empty** status is a status which is set to return tag = MPI_ANY_TAG, source = MPI_ANY_SOURCE, and is also internally configured so that calls to MPI_GET_COUNT and MPI_GET_ELEMENTS return count = 0.
  but should read
  A **empty** status is a status which is set to return tag = MPI_ANY_TAG, source = MPI_ANY_SOURCE, error = MPI_SUCCESS, and is also internally configured so that calls to MPI_GET_COUNT and MPI_GET_ELEMENTS return count = 0 and MPI_TEST_CANCELLED returns false.

- Page 52, lines 46–48 read

```
100          CALL MPI_RECV(i, 1, MPI_INTEGER, 0, 0, status, ierr)
        ELSE
200          CALL MPI_RECV(x, 1, MPI_REAL, 1, 0, status, ierr)
```

  but should read

```
100          CALL MPI_RECV(i, 1, MPI_INTEGER, 0, 0, comm, status, ierr)
        ELSE
200          CALL MPI_RECV(x, 1, MPI_REAL, 1, 0, comm, status, ierr)
```

- Page 53, lines 18–23 read

```
100              CALL MPI_RECV(i, 1, MPI_INTEGER, MPI_ANY_SOURCE,
                                  0, status, ierr)
            ELSE
200              CALL MPI_RECV(x, 1, MPI_REAL, MPI_ANY_SOURCE,
                                  0, status, ierr)
```

  but should read

```
100              CALL MPI_RECV(i, 1, MPI_INTEGER, MPI_ANY_SOURCE,
                                  0, comm, status, ierr)
            ELSE
200              CALL MPI_RECV(x, 1, MPI_REAL, MPI_ANY_SOURCE,
                                  0, comm, status, ierr)
```

- Page 59, line 3 should have the following text added:

> *Advice to users.* To prevent problems with the argument copying and register optimization done by Fortran compilers, please note the hints in subsections "Problems Due to Data Copying and Sequence Association," and "A Problem with Register Optimization" in Section 10.2.2 of the MPI-2 Standard, pages 286 and 289. (*End of advice to users.*)

- Page 59, lines 42–45 read

```
int  MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype,
             int dest, int sendtag, void *recvbuf, int recvcount,
             MPI_Datatype recvtype, int source, MPI_Datatype recvtag,
             MPI_Comm comm, MPI_Status *status)
```

  but should read

```
int  MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype,
             int dest, int sendtag, void *recvbuf, int recvcount,
             MPI_Datatype recvtype, int source, int recvtag,
             MPI_Comm comm, MPI_Status *status)
```

- Page 60, line 3 reads

```
SOURCE, RECV TAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR
```

  but should read

```
SOURCE, RECVTAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR
```

- Page 70, line 16 should have the following text added:

> *Advice to users.* To prevent problems with the argument copying and register optimization done by Fortran compilers, please note the hints in subsections "Problems Due to Data Copying and Sequence Association," and "A Problem with Register Optimization" in Section 10.2.2 of the MPI-2 Standard, pages 286 and 289. (*End of advice to users.*)

- Page 71, line 10 reads
  and do not affect the the content of a message
  but should read
  and do not affect the content of a message

- Page 74, lines 39–45 read
  A datatype may specify overlapping entries. The use of such a datatype in a receive operation is erroneous. (This is erroneous even if the actual message received is short enough not to write any entry more than once.)

  A datatype may specify overlapping entries. If such a datatype is used in a receive operation, that is, if some part of the receive buffer is written more than once by the receive operation, then the call is erroneous.

  The first part was an MPI-1.1 addition. The second part overlaps with it. The old text will be removed so it now reads

A datatype may specify overlapping entries. The use of such a datatype in a receive operation is erroneous. (This is erroneous even if the actual message received is short enough not to write any entry more than once.)

- Page 75, line 24 should have the following text added:
  The **datatype** argument should match the argument provided by the receive call that set the **status** variable.

- Page 77, lines 41-2 read

```
        CALL MPI_TYPE_HVECTOR( 9, 1, 100*100*sizeofreal, twoslice, 1,
                               threeslice, ierr)
```

  but should read

```
        CALL MPI_TYPE_HVECTOR( 9, 1, 100*100*sizeofreal, twoslice,
                               threeslice, ierr)
```

- Page 85, line 36 reads
  "specified by **outbuf** and **outcount**"
  but should read
  "specified by **outbuf** and **outsize**."

- Page 90, line 3 reads
  MPI_Pack_size(count, MPI_CHAR, &k2);
  but should read
  MPI_Pack_size(count, MPI_CHAR, comm, &k2);

- Page 90, line 10 reads
  MPI_Pack(chr, count, MPI_CHAR, &lbuf, k, &position, comm);
  but should read
  MPI_Pack(chr, count, MPI_CHAR, lbuf, k, &position, comm);

- Page 97, line 41 reads

$$\mathtt{MPI\_Recv}(\mathtt{recvbuf} + \mathtt{disp}[\mathtt{i}] \cdot \mathtt{extent}(\mathtt{recvtype}), \mathtt{recvcounts}[\mathtt{i}], \mathtt{recvtype}, \mathtt{i}, ...).$$

  but should read

$$\mathtt{MPI\_Recv}(\mathtt{recvbuf} + \mathtt{displs}[\mathtt{i}] \cdot \mathtt{extent}(\mathtt{recvtype}), \mathtt{recvcounts}[\mathtt{i}], \mathtt{recvtype}, \mathtt{i}, ...).$$

- Page 109, lines 26–27 and page 110, lines 28–29 reads
  The `j`th block of data sent from each process is received by every process and placed in the `j`th block of the buffer recvbuf.
  but should read
  The block of data sent from the `j`th process is received by every process and placed in the `j`th block of the buffer recvbuf.

- Page 117, lines 22–23 reads
  MPI provides seven such predefined datatypes.
  but should read
  MPI provides nine such predefined datatypes.

- Page 121, line 1 reads

```
FUNCTION USER_FUNCTION( INVEC(*), INOUTVEC(*), LEN, TYPE)
```

  but should read

```
SUBROUTINE USER_FUNCTION(INVEC, INOUTVEC, LEN, TYPE)
```

- Page 122, lines 35–36 read
  MPI_OP_FREE(op)

IN	op	operation (handle)

  but should read
  MPI_OP_FREE(op)

INOUT	op	operation (handle)

- Page 125, line 1 reads
  CALL MPI_ALLREDUCE(sum, c, n, MPI_REAL, MPI_SUM, 0, comm, ierr)
  but should read
  CALL MPI_ALLREDUCE(sum, c, n, MPI_REAL, MPI_SUM, comm, ierr)

- Page 141, lines 27–27 read

IN	ranges	an array of integer triplets, of the form (first rank, last rank, stride) indicating ranks in group of processes to be included in newgroup

but should read

IN	ranges	a one-dimensional array of integer triplets, of the form (first rank, last rank, stride) indicating ranks in group of processes to be included in newgroup

- Page 142, line 10 reads

IN	n	number of elements in array ranks (integer)

but should read

IN	n	number of triplets in array ranks (integer)

- Page 153, line 39 reads

```
if(comm_a != MPI_COMM_NULL)
```

but should read

```
if(comm_b != MPI_COMM_NULL)
```

- Page 172, line 26 reads

```
      if ( ! MPI_keyval_create( gop_stuff_copier,
```

but should read

```
      if ( ! MPI_Keyval_create( gop_stuff_copier,
```

- Page 194, lines 30–31 reads
  to the greatest possible, extent,
  but should read
  to the greatest possible extent,

- Page 182, line 18 reads

OUT	status	topology type of communicator comm (choice)

  but should read

OUT	status	topology type of communicator comm (state)

- Page 184, lines 31–32 read

OUT	coords	integer array (of size ndims) containing the cartesian coordinates of specified process (integer)

  but should read

OUT	coords	integer array (of size ndims) containing the cartesian coordinates of specified process (array of integers)

- Page 194, line 48 reads
  MPI_ERRHANDLER_CREATE(FUNCTION, HANDLER, IERROR)
  but should read
  MPI_ERRHANDLER_CREATE(FUNCTION, ERRHANDLER, IERROR)

- Page 195, line 15 should have the following text added:
  In the Fortran language, the user routine should be of the form:

```
SUBROUTINE HANDLER_FUNCTION(COMM, ERROR_CODE, .....)
INTEGER COMM, ERROR_CODE
```

Advice to users. Users are discouraged from using a Fortran HANDLER_FUNCTION since the routine expects a variable number of arguments. Some Fortran systems may allow this but some may fail to give the correct result or compile/link this code. Thus, it will not, in general, be possible to create portable code with a Fortran HANDLER_FUNCTION. (*End of advice to users.*)

- Page 196, lines 1–2 reads
  MPI_ERRHANDLER_FREE(errhandler)

IN	errhandler	MPI error handler (handle)

  but should read
  MPI_ERRHANDLER_FREE(errhandler)

INOUT	errhandler	MPI error handler (handle)

- Page 197, line 25 should have added:

  An MPI error class is a valid MPI error code. Specifically, the values defined for MPI error classes are valid MPI error codes.

- Page 200, lines 27-29 read
  MPI implementations are required to define the behavior of MPI_ABORT at least for a comm of MPI_COMM_WORLD. MPI implementations may ignore the comm argument and act as if the comm was MPI_COMM_WORLD.
  but should read
  It may not be possible for an MPI implementation to abort only the processes represented by comm if this is a subset of the processes. In this case, the MPI implementation should attempt to abort all the connected processes but should not abort any unconnected processes. If no processes were spawned, accepted or connected then this has the effect of aborting all the processes associated with MPI_COMM_WORLD.

- Page 201, line 28 reads
  ...of different language bindings is is done ....
  but should read
  ...of different language bindings is done ....

- Page 203, line 1 reads
  `MPI_PCONTROL(level)`

  but should read
  `MPI_PCONTROL(LEVEL)`

- Page 210, line 44 reads
  MPI_PENDING

but should read
MPI_ERR_PENDING

- Page 211, line 44 reads
  MPI_DOUBLE_COMPLEX
  but should be moved to Page 212, line 22 since it is an optional Fortran datatype.
- Page 212, add new lines of text at line 22 and line 25 to read:
  etc.
  Thus, the text will now read:

```
/* optional datatypes (Fortran) */
MPI_INTEGER1
MPI_INTEGER2
MPI_INTEGER4
MPI_REAL2
MPI_REAL4
MPI_REAL8
etc.

/* optional datatypes (C) */
MPI_LONG_LONG_INT
etc.
```

- Page 213, line 28. The following text should be added:

```
/* Predefined functions in C and Fortran */
MPI_NULL_COPY_FN
MPI_NULL_DELETE_FN
MPI_DUP_FN
```

- Page 213, line 41. Add the line

```
MPI_Errhandler
```

- Page 214, line 9 reads

```
FUNCTION USER_FUNCTION( INVEC(*), INOUTVEC(*), LEN, TYPE)
```

  but should read

```
SUBROUTINE USER_FUNCTION( INVEC, INOUTVEC, LEN, TYPE)
```

- Page 214, lines 14 and 15 read

```
 PROCEDURE COPY_FUNCTION(OLDCOMM, KEYVAL, EXTRA_STATE,
               ATTRIBUTE_VAL_IN, ATTRIBUTE_VAL_OUT, FLAG, IERR)
```

but should read

```
 SUBROUTINE COPY_FUNCTION(OLDCOMM, KEYVAL, EXTRA_STATE,
               ATTRIBUTE_VAL_IN, ATTRIBUTE_VAL_OUT, FLAG, IERR)
```

- Page 214, line 21 reads

```
PROCEDURE DELETE_FUNCTION(COMM, KEYVAL, ATTRIBUTE_VAL, EXTRA_STATE, IERR)
```

but should read

```
SUBROUTINE DELETE_FUNCTION(COMM, KEYVAL, ATTRIBUTE_VAL, EXTRA_STATE, IERR)
```

- Page 214, line 23 should have the following text added:
  The handler-function for error handlers should be declared like this:

```
SUBROUTINE HANDLER_FUNCTION(COMM, ERROR_CODE, .....)
INTEGER COMM, ERROR_CODE
```

- Page 216, lines 4–7 read

```
int  MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype,
              int dest, int sendtag, void *recvbuf, int recvcount,
              MPI_Datatype recvtype, int source, MPI_Datatype recvtag,
              MPI_Comm comm, MPI_Status *status)
```

but should read

```
int  MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype,
              int dest, int sendtag, void *recvbuf, int recvcount,
              MPI_Datatype recvtype, int source, int recvtag,
              MPI_Comm comm, MPI_Status *status)
```

- Page 220, lines 19–20 reads
  int double MPI_Wtime(void)
  int double MPI_Wtick(void)
  but should read
  double MPI_Wtime(void)
  double MPI_Wtick(void)

- Page 222, line 34 reads
  INTEGER REQUEST, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR
  but should read
  INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

- Page 222, line 38 reads
  INTEGER REQUEST, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR
  but should read
  INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

- Page 227, lines 19–20 reads
  MPI_INTERCOMM_MERGE(INTERCOMM, HIGH, INTRACOMM, IERROR)
  INTEGER INTERCOMM, INTRACOMM, IERROR
  but should read
  MPI_INTERCOMM_MERGE(INTERCOMM, HIGH, NEWINTRACOMM, IERROR)
  INTEGER INTERCOMM, NEWINTRACOMM, IERROR

- Page 228, line 46 reads
  MPI_ERRHANDLER_CREATE(FUNCTION, HANDLER, IERROR)
  but should read
  MPI_ERRHANDLER_CREATE(FUNCTION, ERRHANDLER, IERROR)

- Page 229, line 33 reads
  `MPI_PCONTROL(level)`

  but should read
  `MPI_PCONTROL(LEVEL)`