Errata Sheet for MPI Specification

Steve Otto

October 27, 1994

This document lists errors in the May 5th version of the MPI specification. Some of these have been fixed (at the galley stage) in the version to be published by MIT Press in the *International Journal of Supercomputer Applications*, and are noted as such.

Errors

The following are known errors or changes that we have made to the specification.

1. (Fixed in IJSA version) Removal of complex data types from reduce operations. Changes:
   - page 113, line 36, remove "Complex."
   - page 115, line 42, remove the entire line containing MPI_2COMPLEX.
   - page 209, line 12, remove the line containing MPI_2COMPLEX.
2. (Fixed in IJSA version) `MPI_Status` passed as pointer everywhere (an issue for the C binding only). Changes:
   - page 21, line 16, should say: `MPI_Status *status`
   - page 73, line 33, should say: `MPI_Status *status`
   - page 52, line 6, should say: `MPI_Status *status`
   - page 211, line 24, should say: `MPI_Status *status`
   - page 212, line 26, should say: `MPI_Status *status`
   - page 213, line 35, should say: `MPI_Status *status`
3. (Fixed in IJSA version) Pointers to storage area should be of type `MPI_Aint`, rather than `int`. Changes:
   - page 68, line 27, should say: `MPI_Aint *extent`
   - page 68, line 39, should say: `MPI_Aint *size`
   - page 70, line 15, should say: `MPI_Aint *displacement`
   - page 70, line 26, should say: `MPI_Aint *displacement`
   - page 213, line 24, should say: `MPI_Aint *extent`
   - page 213, line 25, should say: `MPI_Aint *size`

- page 213, line 29, should say: `MPI_Aint *displacement`
- page 213, line 30, should say: `MPI_Aint *displacement`

4. (Fixed in IJSA version)

   - page 83, lines 36, 42, 48, change "outcount" to "outsize"
   - page 84, lines 1, 4, change "OUTCOUNT" to "OUTSIZE"
   - page 221, lines 8, 11, change "OUTCOUNT" to "OUTSIZE"

5. (Fixed in IJSA version) In section 3.2.2, the sentence fragment:

   > MPI_DOUBLE_COMPLEX for double precision complex in Fortran declared to be of type DOUBLE PRECISION;

   should be

   > MPI_DOUBLE_COMPLEX for double precision complex in Fortran declared to be of type DOUBLE COMPLEX;

6. The wording of the description of `MPI_GROUP_RANGE_EXCL` is confusing but I haven't decided what to change it to, yet. The function prototypes are correct, I believe.

7. The example on page 148. The example would be more illustrious if

   ```
   MPI_Reduce(...,MPI_SUM, 1, commslave) -> MPI_Reduce(...,MPI_SUM, 0, commslave)
   ```

   The communicator `commslave` is composed of the group `grprem` which is `MPI_GROUP_WORLD` without process 0. As such, `grprem` is numbered from 0 to (Numprocs - 2) where Numprocs is the size of `MPI_GROUP_WORLD`. Making the reduce go to process 0 makes the renumbering clear. It also makes it work for only 2 nodes - the current code fails since `grprem` would only have one node/process.

8. (Fixed in IJSA version)

   - page 150, line 16, should say: `MPI_Comm comm`

9. Page 151. This example seems to imply that MPI_Comm_rank returns MPI_UNDEFINED in ma and mb if the calling process is not in comm_a or comm_b, respectively. 142 says nothing about this. If this was the intent (that is would return MPI_UNDEFINED) then the text about MPI_COMM_RANK should say this. This comment might also apply to MPI_GROUP_RANK on 135 if we want it to act the same. Their being the same is implied in the comments in the text.

10. Page 159. It would be a little cleaner to add:

    ```
    MPI_Comm_free(&myComm);
    ```

    This is done for Example 2 on page 160.

11. Page 162 Instead of (MPI_Comm) 0 I think it should be MPI_COMM_NULL. 145:8 says MPI_Comm_split returns MPI_COMM_NULL when it has funny cases. I don't recall reading 0 and NULL were the same for this case.

12. Page 162.

    ```
    MPI_Attr_keyval_create -> MPI_Keyval_create
    ```

    as per page 166. Same comment applies to page 170.

13. Page 165.

    ```
    MPI_Intercomm_create(local_leader, local_comm, buffer[0],new_world, ..) ->
    MPI_Intercomm_create(local_comm, local_leader, new_world, buffer[0], ..)
    ```

    as defined on 157.

14. Page 167 It seems to me that the Fortran definition needs IERROR. 167 says that it returns MPI_SUCCESS if it works. Only the C version seems to have a way to return this. The same comment applies to: 167 for DELETE_FUNCTION

15. Page 167 Do we want MPI_NULL_FN, MPI_DUP_FN in Appendix A? It is a function but some are listed

16. Page 167 MPI_KEYVAL_INVALID seems to be missing from Appendix A.

17. Page 169, line 38, A null Rational should be removed.

18. Page 170 MPI_Abort takes a comm and error code, not string as defined Page 197.

19. Page 171

    ```
    MPI_Comm comm -> MPI_Comm *comm.
    ```

    See definition on 167. And, For the same reason,

    ```
    int keyval -> int *keyval
    ```

    And

    ```
    gop_stuff_copier(..., gop_stuff, extra) ->
    gop_stuff_copier(..., extra, gop_stuff_in, gop_stuff_out, flag)
    ```

    per definition on page 166. And

    ```
    MPI_Comm comm -> MPI_Comm *comm
    ```

    And

`int keyval -> int *keyval.`

Also need to add `*flag` for above change

`gop_stuff_type *gop_stuff -> gop_stuff_type *gop_stuff_in, **gop_stuff_out`

And, I think we need to add the line:

`*gop_stuff_out = gop_stuff_in;`

Right now, the routine does not appear to copy anything; only increments the reference count. And, need to make

`gop_stuff -> gop_stuff_in`

if you make the other changes.

20. Page 215, line 31:

    `int MPI_Comm_compare( MPI_Comm comm1, comm2, int *result )`

    should be

    `int MPI_Comm_compare( MPI_Comm comm1, MPI_Comm comm2, int *result )`

21. Page 142, line 39. The sentence should read:

    MPI_IDENT results if and only if comm1 and comm2 are handles to the same object (identical groups and same contexts) or both are null.

22.
    - page 47 Example 3.14, line 35

      `CALL MPI_IRECV(a(i,1), n, MPI_REAL, 0, tag,`

      should be replaced with

      `CALL MPI_IRECV(a(i,1), n, MPI_REAL, i, tag,`

    - page 47 Example 3.14, line 41

      `CALL MPI_IRECV(a(1,index), n, MPI_REAL, 0, tag,`

      should be replaced with

      `CALL MPI_IRECV(a(1, index), n, MPI_REAL, index, tag,`

    - page 48, Example 3.15, line 11

      `CALL MPI_IRECV(a(i,1), n, MPI_REAL, 0, tag,`

      should be replaced with

      `CALL MPI_IRECV(a(i,1), n, MPI_REAL, i, tag,`

- page 48, Example 3.15, line 19

  ```
  CALL MPI_IRECV(a(1, index_list(i)), n, MPI_REAL, 0, tag,
            comm, request_list(i), ierr)
  ```

  should be replaced with

  ```
  CALL MPI_IRECV(a(1, index_list(i)), n, MPI_REAL, index_list(i), tag,
            comm, request_list(index_list(i)), ierr)
  ```

Unresolved Issues

The following are some issues that are still being discussed and that haven't yet reached consensus.

1. An issue related to error number 2: Should the arguments POSITION and OUTCOUNT in `MPI_Pack` be of type `int` (as they are now) or of type `MPI_Aint`? That is, do we want to allow for a buffer with more than 2 Gbytes?

2. In lines 26 and 27, page 88 of the May 5 MPI draft, shouldn't `&lbuf` just be `lbuf` ?

3. There are some inconsistencies in the description of the use of MPI_NULL_FN when used with MPI_KEYVAL_CREATE. In particular, when describing the "copy_function", we say

   > MPI_NULL_FN is a function that does nothing other than returning flag = 0.

   This is inconsistent with a null function (by definition, a null function should do nothing) and will fail when used as the "delete_function", where we say

   > It may be specified as the null function pointer in C or as MPI_NULL_FN from either C or FORTRAN, in which case no delete callback occurs for keyval.

   There are two obvious fixes: 1) Replace MPI_NULL_FN with MPI_NULL_COPY_FN (for correctness) and MPI_NULL_DELETE_FN (for consistency) in the discussion of MPI_KEYVAL_CREATE. 2) Change the wording under "copy_function" to

   > MPI_NULL_FN is a function that does nothing; when used as the copy function, it has the same effect as a function that does nothing other than returning flag = 0.

   I prefer (1) because it is clearer as to what is going on and (naturally) makes the implementation simpler. Implementing (2) is straightforward but unpleasant.

4. Related to previous item, we have (From Hubertus Franke):

   I extract this out of the MPI-F include files.

   ```
   #define MPI_DUP_FN   mpi_dup_fn
   /* note the draft provides MPI_NULL_FN, however this violates
    * the prototype differences in usage for copy and delete function
   ```

```
 * In agreement with the public domain libary developers we
 * distinguish these functions
   #define MPI_NULL_FN
 */
#define MPI_NULL_COPY_FN    ...
#define MPI_NULL_DELETE_FN  ...
```

I've discussed this with Nathan Doss and I believe this has been incorporated into the public domain version as well.

5. There is a long, still unresolved discussion centering around PACK/UNPACK. At the end of it, (at least the last message I see), Marc Snir says:

   Conclusion:

   - a) A message that was created by one or several sequences of related PACK calls is sent with type MPI_PACKED.
   - b) A message that is going to be unpacked by one or several sequences of related UNPACK calls is received with type MPI_PACKED.
   - c) A Message created by one sequence of related PACK calls, and sent with type MPI_PACKED is identical to a message sent with a regular send.

   I don't understand why you came to the conclusion it is wrong to send the concatenation of several packing units with type MPI_PACKED.

   (end of Marc). Where do we stand here?

6. MPI_BOTTOM in Fortran.

   There is a bug (or unintended effect) in the definition of MPI_BOTTOM for Fortran. Specifically, MPI_BOTTOM can be used as a buffer address; in C the appropriate value will often be (void *)0. But in Fortran, there are no pointer types; in most implementations, all values are passed by reference. Now consider the following two code fragments:

```
C:
void *p;

p = (void *)MPI_BOTTOM;
MPI_Send( p, ... );

Fortran:
integer p

p = MPI_BOTTOM;
MPI_Send( p, ... );
```

   The C case is clearly allowed; the Fortran case must not be.

The only way that I can see to provide MPI_BOTTOM to Fortran is to make it a special constant that is like a reserved word - assigning it to a variable is erroneous (that is, there is a special location that is known as MPI_BOTTOM; the routines can test for that special location). This may seem obvious, but it is easily overlooked. We certainly did while writing the model implementation. I believe that this merits some comment in the standard, since to Fortran, MPI_BOTTOM is not a "constant" as defined by that standard.

7. The requirement for an Environmental Enquiry that allows to find whether two nodes use the same data representation is reasonable, irrespective of the specific use that Rolf has in mind, for Parmacs emulation. More generally, users may want enquiry functions to find the type of node on which a process runs. This would help in situations where the user is willing to run on any available machine on a network, but may want to use different code on different machine types.

   It is not reasonable to assume that the MPI forum will develop its own nomenclature for data representations (size of each basic language data type, big vs little endian, floating point format, character codes, etc.). First because other standards try to handle this issue and, second, because a complete characterization of the data representation used by a process (which depends on the underlying machine architecture and on the compiler) is likely to be quite lengthy.

   We agreed that vendors can attach implementation specific predefined attributes that are associated with MPI_COMM_WORLD. I would suggest the following:

   Associate a predefined attribute key MPI_PROC_TYPE with MPI_COMM_WORLD. The value of this attribute encodes the "type" of the executing process, i.e. the type of data representation used by this process. The meaning of the values of this attribute is implementation dependent, except that if two processes return the same value, then they use the same data representation (i.e., data can be moved untyped between these processes).

   Implementers of MPI for homogeneous SPMD systems just need to always return the same value (0?) with this attribute. Implementers of MPI on heterogeneous systems may need some work.

   If this suggestion sounds reasonable, then the various MPI implementers may accept it now as "common practice", and we can, at a latter time, add this to the standard.