Minutes of MPI meeting
April 24–26
Chicago IL

William Gropp

April 26, 1995

1 Introduction

This is an unedited set of minutes taken during this MPI meeting. This contains both a summary of some of the discussions and official, binding votes of the MPI Forum. Official votes are set in bold type; for example **1-2-3**. Straw votes, which are non-binding and may be voted on by anyone present, are shown as *1-2-3*. In some cases votes among several alternatives are held by voting for each of the alternatives; in that case, only one number, the votes for, is presented. Please send any corrections to `gropp@mcs.anl.gov`.

2 Monday

Rusty Lusk opens meeting with a discussion of the agenda.

17 valid voters.

Marc Snir opens discussion of MPI-1 items that need discussion. Distributed modified version of the point-to-point chapter. Presented for second vote. These are the items listed:

1. `MPI_BSEND_OVERHEAD`
2. Changes to the various forms of wait/test when given an empty list of requests.

Operation	Success	Failure	Empty list
WAIT	good status	blocks	null status
TEST	flag = T, good status	flag = F	flag = T; null status
WAITANY	good index	blocks	index = MPI_UNDEFINED
TESTANY	flag = T; good index	flag = F	index = MPI_UNDEFINED
WAITSOME	outcount > 0	blocks	outcount = MPI_UNDEFINED
TESTSOME	outcount > 0	outcount = 0	outcount = MPI_UNDEFINED
WAITALL	good statuses	blocks	null statuses
TESTALL	flag = T; good statuses	flag = F	flag = T; null statuses

3. Change definition of extent

4. Deleted MPI_TYPE_COUNT

5. Use of MPI_Aint for extents.

The following applies to other parts of the documents (these were raised at the last meeting):

1. State that MPI calls do not update handles (unless freed or committed), even though they are passed by reference.

2. MPI_TAG_UB must have the same value on all processes.

3. MPI_Error_string can be applied to class numbers as well as codes.

4. User may not modify environment attributes (i.e., MPI_TAG_UB, MPI_IO, MPI_HOST).

5. No MPI_MINLOC and MPI_MAXLOC for MPI_COMPLEX and MPI_DOUBLE_COMPLEX.

6. Null handle is an invalid argument in MPI calls except for those that create objects.

7. Examples are not part of the standard.

Several items required more detailed proposals. These are:

1. MPI Constants. Paul Pierce presents a discussion of static constants (can be used at compile time) and dynamic (can only be used between MPI_INIT and MPI_FINALIZE and are unchanging in value between those calls).

   Proposals for clarifying MPI constants include

   (a) All constants are static *9-2-9*

   (b) No changes

(c) Specify sizes, error codes, MPI_UNDEFINED, MPI_xxx_NULL, and C ints and Fortran status object constants as static; let the others be dynamic

(d) Make MPI_Datatypes static in addition to the above. *19-0-0*

It is noted that MPI_BOTTOM in Fortran is a peculiar object and may be in a class by itself.

General agreement that at least the set in (3) be static. Bill Gropp points out that the easiest implementation requires at least the introduction of new names in the 'mpi.h' file (e.g., struct MPI_INTERNAL_COMM MPI_COMM_WORLD_AREA). Paul Pierce will prepare two proposals (all and just datatypes).

2. Data representation type Han Christian Hoppe presented case for MPI_DATA_REPRESENTATION, an attribute for MPI_COMM_WORLD such that, the value of MPI_DATA_REPRESENTATION is equal if and only if the data representation of all MPI datatypes are the same. (The value may be different on each process; this allows processes to do pairwise comparison.)

   The rationale for this is in porting legacy message-passing applications that assume a uniform data representation (and trying to maintain performance).

   Steve Lederman suggests that the "only if" be removed: if the data representations are the same, the representation of all MPI datatypes are the same.

   Eric Salo suggests that this be an attribute of MPI_COMM_SELF. Bill Gropp seconds. Lloyd Lewins suggests a function that can be used to get the same information.

   Marc Snir suggests a more precise definition, such as that messages sent with MPI_BYTE can be correctly received, be used to define the meaning of data representation.

   Marc Snir suggests that only whether the communicator is heterogeneous; Bill Gropp remarks that the differences in MPI_PACK/MPI_UNPACK (no source and destination) and the send/receive routines means that you need both (is a pair of processes heterogeneous and is the communicator heterogeneous).

   Greg Burns asks whether MPI_UNDEFINED is allowed; everyone seems to agree.

   Steve Lederman-Huss moves that we take a straw vote on M. Snir's proposal (a boolean attribute MPI_IS_HOMOGENEOUS) on all communicators. The value is true if no data conversion is required, false otherwise. There is discussion of whether this is a strong or weak (weak means giving false even when true was valid). Note that the name is still being chosen. *6-4-10*

3. Is clock synchronized (discussed below)
4. Get error from status in multiple (discussed below)
5. void ** in MPI_ATTR_GET and MPI_BUFFER_DETACH. (Will try experiment on word-addressed machine).
6. Null function for copy and delete attributes (Bill Gropp to provide for Tuesday).
7. Standard error classes No consensus on this among the implementors.
8. Progress rule Greg Burns is working on a proposal but it needs more work. He suggests a subcommittee to thrash out the issues.
9. MPI_CART_SHIFT arguments: are origins zero or one in Fortran for direction of shift? Choices include
   (a) $0 \ldots n-1$ **15-1-1**
   (b) $1 \ldots n$
   (c) $0 \ldots n-1$ in C, $1 \ldots n$ in Fortran

   Case 2 has no defenders. Case 1 passes by wide margin.
10. color needs to non-negative in MPI_COMM_SPLIT. (because MPI_UNDEFINED is a valid color; MPI_UNDEFINED needs to be negative for other uses, so making color non-negative makes MPI_COMM_SPLIT behave properly. **12-0-3**

Discussion of who owns the document; R. Lusk points out the S. Otto has ownership of MPI-1 and Steve Lederman-Huss has ownership of MPI-2 document.

P. Pierce suggests deferring votes until Wednesday to give everyone a chance to read the suggestions.

Break for dinner (there are still additional items above to discuss).

3 Tuesday

Begin at 9:00 AM . Finish discussions on MPI 1.

Peter Pacheco begins with discussion about IO in MPI. Concern is access to stdin and stdout. Proposes that MPI implementations provide stdin on at least one process and stdout on all processes. In email discussions, concern raised about stdin and stdout, with printf and scanf being used instead. Several people point out that this is a quality of implementation issue, others that this is a C or Fortran issue. No vote.

Lloyd Lewins discusses reporting errors from multiple completion routines (e.g., MPI_WAITALL). He proposes putting that into the MPI_STATUS (with a specific field name). An alternative is to make the error code accessible through a routine call. One problem is that there are now multiple places for an error return. For example, should the function return MPI_ERR_IN_STATUS? Marc Snir proposes that the value in status be undefined unless there is an error. More precisely, the status error condition is unchanged if there is no error; this allows the user to set status.MPI_ERROR to MPI_SUCCESS. Bill Saphir prefers always making the value correct. Eric Salo notes that their implementation keeps an error code in the internal data structures. Snir's proposal is clarified that in the multiple completion case, the status fields of *all* status members are set if *any* error occurs (so that the user can look for the errors). Bill Gropp points out that routines such as MPI_TESTSOME must treat a failure as a completion so that there is some place to put the error, and that the standard should be clarified to make this valid behavior.

For a error field in status: *18-0-4*

A discussion of the overhead of setting the extra field in terms of additional memory operations and cache misses; rough guess is 3%.

For the error field in status only valid if the function returns other than MPI_SUCCESS; otherwise all status are valid. *13*

Alternative, the error fields is always valid. *8*

Next, there was a discussion of what is the function error return if there is an error and a single status (do you get the error or just MPI_ERR_IN_STATUS?). Additional discussion to make clear that MPI_ERR_IN_STATUS must only be used when some status will have an error (other than MPI_SUCCESS) in status.MPI_ERROR.

Bill Gropp asks what happens to an error handler other than errors-return? What error code is given to the routine? Marc Snir points out that the standard says that the error routine is called with the error code that will be returned. In the case where there are multiple errors returned in the status array, the user error-handler will be called once with MPI_ERR_IN_STATUS (but see Wednesday, when a different interpretation was taken).

Lloyd Lewins will bring a proposal.

Bill Gropp discusses the results of checking the replacement of `void **` with `void *` on several systems, including one with word-oriented pointers (all systems handled the test as expected).

Also presented is exact text for replacing MPI_NULL_FN with MPI_NULL_COPY_FN and MPI_NULL_DELETE_FN in MPI_KEYVAL_CREATE. **17-0-1**

Marc Snir will bring a specific proposal about `void *` and the description to users.

Ian Foster raises an example of datatypes with MPI_SCATTER where he could not create a datatype that would do the scatter with a single MPI_SCATTER call. The committee suspects that this can be done with MPI_STRUCT and explicit extents. Ian promises to send a detailed example to the list.

Break

Rusty Lusk begins discussion of dynamic process creation and management. Starts with a general discussion of the issues.

Scope:

1. Task farming
2. Dynamically changing requirements
3. Network applications (I-WAY)
4. Client/server
5. Ability to recover from faults (fault-tolerance)

Functionality

1. Porting from PVM
2. Interaction with resource management and process management separately
   (a) Schedulers on MPP's
   (b) I-WAY and other network applications
   (c) Schedulers on NOWs
3. Client/Server
4. Management of non-MPI processes (Writing environment in MPI)
5. Changing into MPI processes, specifying alternate communication paths
6. Blocking, non-blocking calls

Basic Objects to be used

1. Processes, tasks
2. Processors, nodes
3. SMP's

Marc Snir suggests separating into

1. resource management (how to specify, what answers you can get, can you receive resources asynchronously, etc.)
2. process management (how to specify, start, connect stdin/stdout, set the environment, handle errors)
3. communication system management (startup, failures, etc.)

Tom Henderson asks about how programs are initiated (e.g., the syntax of `mpirun`). This was viewed as a good thing to discuss; Rusty Lusk mentioned that some audiences seem to need to be told the external interface (e.g. mpirun) in order to understand MPI.

Rusty Lusk then began discussing the Gropp/Lusk proposal and Bill Saphir's comments. Comments that B. Saphir's comments add some new objects to separate the use of requests from the object that the request operates on. This is primarily process management.

On resource management, Tom Henderson asks if process topology is included. Rusty Lusk comments that network applications may have bandwidth requirement; Henderson mentions that some MPPs need particular layouts (e.g., compute and I/O areas).

Tony Skjellum asks that we limit the discussion to those features needed to simplify porting PVM jobs to MPI; R. Lusk comments that his management needs a more general solution for I-WAY.

Marc Snir suggests that we follow the same procedure as used for point-to-point: start from nonblocking and build blocking and easier to use interfaces by putting together the lower-level operations.

Tony Skjellum mentions that MPI 1 was standardizing practice while the dynamic process management proposals are standardizing research. R. Lusk responds that we had some of this discussion at the previous meeting. W. Saphir comments that most of the functionality above is present in some form in PVM, so in some sense, there is existing practice. Greg Burns expresses concern that may not have time to create a draft before SC'95. More discussion about whether we should start with a top down (e.g., define `MPI_SPAWN`) or bottom up (e.g., define separate interfaces to allocate resources, create and manage processes, etc.).

Steve Lederman-Huss asks how important it is to get a solid proposal for SC'95. R. Lusk responds that the four meetings should be enough to put together a solid proposal. More discussion about the scope of the proposal with much emphasis on delivery by SC'95.

Bill Saphir suggests all proposals include a subset with some `MPI_SPAWN`. Paul Pierce mentions that even at the high level, many of the issues (e.g., resource management) are visible.

Ian Foster asks about the "basic objects" and whether a node is a process (or processor). W. Saphir says that the answer is usually yes. Additional discussion about whether nodes might timeshare with other nodes. Marc Snir mentions gang-scheduling, where timesharing is done by entire gangs, not individual processes. Ian Foster expresses concern that the resource unit is a processor or process, mentioning that for many programmers, the natural form is processor.

Additional discussion on what the individual subcommittees should deliver. For example, the resource management part should deliver an opaque object that can be used in starting some number of processes; the total number may be limited by the particular resource request (and will need to be determinable from the opaque resource object). It is mentioned that resources like "12 processors sharing the same high-speed switch" or a particular partition of processes imply that some resources have to be acquired collectively.

Rusty Lusk summarizes the morning's discussion. Paul Pierce suggests two working groups: a resource management and a process and communication working group.

Lunch

Steve Lederman-Huss begins discussion of external interfaces.

The most pressing is generalized requests.

Encourage input from the community; known areas of interest are access to message queues, I/O, decode datatypes, others.

For example, if the user wants to create a non-blocking user broadcast. Then `Request_type_create()` to create, `Request_init` to create a particular instances and initialize it, a start to begin the operation, and some test and/or wait routines. The only new functions are `Request_type_create` and `Request_init`. There is much discussion about the use of callback functions in `Request_init`.

Marc Snir mentions that one desirable feature is for the request to operate in such a way that whenever any communication in the request occurs, the request's handler might be invoked (to handle the case of a user-defined request, such as a nonblocking broadcast) to ensure continued progress of the user-defined request.

Bill Saphir asks what `status` means for user-defined requests.

More questions on when the request's handler is invoked. Marc Snir says it is invoked when either the request is started or any of its communications complete. Bill Gropp points out that this requires some sort of association of communication request with user-defined request, and that one such mechanism, an `hrecv`, might remove the need for user-defined requests. Much additional discussion about the interaction of these requests and things like the progress rule.

Suresh Damodaran-Kamal presents message queue access. He would like the functionality to

1. read from message queue
2. block delivery of messages
3. deliver selective messages

Possible solutions include a function call for each of the above or a routine to deflect all messages send to a specific MPI process to the calling process.

Rusty Lusk asks about the Ptools MQM (Message Queue Manager) interface. Tom Henderson asks if this would require a separate library or will it affect performance of all jobs.

Additional discussion of debugger features, including deadlock detection. Being able to look at the send and receive queues is one way for users to detect and understand deadlock and other problems.

William Saphir describes two routines, `MPI_Describe_pending_messages(level)`; output is implementation-dependant. `level 0` is most concise and limited; higher levels give additional detail. This is used from within a debugger that lets you call routines. Another version puts the output into a string.

General opinion seems to be that this is a good idea, but that details need to be worked out.

Scheduling of the next meeting (June 5-7 and July 26-28). Adding September 6-8 also at OHare Hilton. Tom Henderson offers to setup a meeting in Boulder (airport is in Denver).

Break

Begin with a "last call" for MPI-1 topics.

Marc Snir introduces an alternative to usage of `MPI_UB`. Add the routine `MPI_TYPE_SET_EXTENT` which specifically sets the upper bound, overriding any explicitly set upper bounds in the type signature. A related issue is when can an MPI implementation on a homogeneous system convert a datatype representing a C struct use a memcpy even though the struct has gaps in it (e.g., `struct int, char, int`).

Tony Skjellum suggests that the peer communicator on `MPI_INTERCOMM_CREATE` be allowed to be an intercommunicator as well as an intracommunicator. Note that the `remote_leader` refers to the remote-group in the peer-communicator. *10-0-7*

Bill Gropp raises the issue of discovering if the clocks are synchronized, for example, if an MPP provides a uniform clock, this is very valuable for profiling libraries. Conversely, if it is not available, attempts to adjust the individual clock reduce the accuracy of the clocks. Many discussions about what synchronization means; an alternative is to provide a parameter that indicates the amount of insynchrony. *12-0-8*

Tony Skjellum gives an overview of the multicast proposals. Among the issues

1. Progress — If one process does not do a receive, then a scalable algorithm (say a tree) could fail at the process that does not do the receive.
2. Ordering — If messages are sent with the same message tag from a process that does a regular send and a multicast, we can not guarantee that the message sent first arrives first.

Rusty Lusk then introduced the MPI-2 draft outline:

- Introduction
- Terms and Conventions

- External Interfaces
  - Generic Requests
  - Access to internal Data Structures
- One-Sided Operations
- Dynamic Process Management
  - Resource management
  - process management
  - communication management
- Collective Extensions
  - Multicast
  - Operations on Intercommunicators
  - Topology Extensions
  - Non-blocking collective
- Bindings
  - C++
  - Fortran 90

It is noted that this is a very provisional outline primarily to get things going.

Brief discussion of interoperability of different MPI implementations (no progress for now).

Another issue is Fortran/C interoperability (for example, `MPI_COMM_WORLD` may not be represented the same way in C and Fortran).

4 Wednesday

R. Lusk opens meeting. Remaining topics are closure on MPI-1 issues.

18 voting organizations.

1. Paul Pierce brings MPI Constant proposal (distributed at meeting)

   > Named constants may be used after `MPI_INIT` and before `MPI_FINALIZE` between which the value of the constants and the semantics of the underlying objects shall not change. [Some] named constants shall be statically defined such that they can be used as initializers.

> [These are constants of type int of `MPI_Datatype` in the C binding, the constants associated with the status object in the Fortran binding, and constants with "NULL" in their names.
>
> Other constants of type `MPI_Comm`, `MPI_Group`, `MPI_Op`, `MPI_Errhandler`, and the constant `MPI_BOTTOM` may not be used as initializers.]
>
> Annex A lists all constants [and whether they are statically defined].

The weak form includes the material enclosed within brackets.

It is noted that the stronger form will still need to exclude `MPI_BOTTOM` in Fortran.

Vote for Strong form **14-3-1**.

Vote to amend MPI-1 with the strong form **15-1-2**.

Nathan Doss presents new version of `MPI_INTERCOMM_CREATE` that allows the peer communicator to be any communicator. **13-0-5**

Lloyd Lewins presents changes in error reporting for multiple completion. Distributed at meeting.

Discussion about what error codes would be passed to error handlers, and whether the handler would be called once or multiple times. Discussion leaned towards calling the error handler for each occurance, not just once for `MPI_ERR_IN_STATUS`.

One choice is whether `status.MPI_ERROR` is always set or only when `MPI_ERR_IN_STATUS` is the return value of the function.

- Vote for `status.MPI_ERROR` always set: **3**
- Vote for `status.MPI_ERROR` only set if `MPI_ERR_IN_STATUS` is set:**11**
- Abstains-**3**

Vote for this change to MPI-1: **17-0-1**

Marc Snir presents the following clarifications (distributed a page of small corrections): (the text below is a shortened version for the purpose of identifying the votes)

1. MPI calls do note change handles
2. Null handles are not valid (unless explicitly allowed)
3. `MPI_BSEND_OVERHEAD`
4. Test/wait with empty lists
5. extent is $ub - lb$ with no padding

6. Remove MPI_TYPE_COUNT

7. color must be non-negative in MPI_COMM_SPLIT

8. direction in MPI_CART_SHIFT starts at zero

9. Attributes MPI_TAG_UB, MPI_IO, and MPI_HOST do not change during execution and cannot be set

10. MPI_TAG_UB has the same value at all processes in MPI_COMM_WORLD.

11. Value of MPI_IO: if all processes can do IO, then MPI_ANY_SOURCE is the value of the MPI_IO attribute. (note that page has MPI_UNDEFINED where it should have MPI_PROC_NULL; corrected before vote.)

12. Error classes are a subset of error codes.

16-1-0

Some discussion of adding MPI_TYPE_ELEMENTS to return the number of leaf elements of a datatype (sort of a replacement for MPI_TYPE_COUNT). Tabled for MPI 2.

Change C binding of MPI_Attr_get and MPI_Buffer_detach to use `void *` instead of `void **`. There is still concern that, while the change is correct C and requires less explicit casting by the user, the bindings are no longer self-documenting and users may be more likely to make an error (e.g., passing a pointer instead of the address of the pointer). **12-2-3**

Bill Gropp brings proposal for MPI_WTIME_IS_GLOBAL, an attribute about whether MPI_WTIME is a global clock.

The attribute has value 1 if the **10-1-4**

Steve Lederman-Huss closes with the traditional mpi– presentation.