Minutes of MPI meeting
April 24–26
Chicago IL

William Gropp

May 20, 1996

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* (yes/no/abstain). In non-binary choices, *6* indicates the number of votes for a proposal. Please send any corrections to `gropp@mcs.anl.gov`.

2 Wednesday

Rusty Lusk opens the meeting by having everyone introduce themselves and then covers the agenda.

William Saphir begins the discussion of dynamic process management. Beginning with `MPI_SPAWN`, the opaque object `info` and whether it should be declared `char *` or `void *`. Jim Cownie points out that while `void *` is more natural for C, `CHARACTER` is more natural in Fortran. Rusty Lusk notes that some Fortrans allow relatively short `CHARACTER` values; Bill Gropp notes that the Fortran has no minimum requirement, and the minimum of major implementations is a mere 255 characters. Jim Cownie recaps; yet another discussion of portability starts. Jim's position is that the binding is implementation defined, since the meaning of the argument is implementation defined. The one exception is the NULL argument (again a problem only for Fortran; C may use `(void *)0`). Keep the binding `char *`: *14-5-10*.

Al Geist raises the issue of how `MPI_Spawn` decides that it has failed; for example, it has started less than `minprocs` processes but has started some. After some discussion of time outs, Marc Snir points out that some of the spawned processes may have side effects before calling `MPI_Init`; if some process fails before calling MPI_Init and the MPI_Spawn tries to kill the processes (the spawn has failed), some side effects (such as creating or even deleting files) may already have occurred. [Also note that while we are assuming that Spawn doesn't return until after all processes involved in the spawn have called MPI_Init, implementations can always assume correct programs and return more quickly if they can construct the required intercommunicator.] Joel Clark asks what happens when minprocs are not available.

Stay with minprocs/maxprocs: *14-9-9*.

Marc Snir raised several issues about the command line arguments. Bill Saphir will draw up clarifications.

Jim Cownie points out % is meaningful in sprintf etc., and could cause problems for some programmers. The general feeling was that there is no good character and % is no worse than others.

Moving on to MPI_UNIVERSE_SIZE and MPI_GET_SPAWNABLE_SIZE, much of the discussion revolved around exactly what these mean. One comment was that this really doesn't give anything over the min/maxprocs values in the spawn commands (as Richard Frost pointed out, the user could ask for the maximum number of processes that could be effectively used and take the number available). Should there be some routines for this? *11-12-10*.

Constant *14*

Function *7*

Different *2*

Abstain *9*

Marc Snir points out that even for universe size, the size may change dynamically (processes complete, the environment changes, etc.). We need to more carefully define what universe size means.

On to the spawn independent routines. Propose to merge the routines and provide a flag. A straw vote to keep as is (two functions) *7-22-6*.

Moving on to Ispawn, Saphir notes that the requests created by the ispawn cannot be cancelled. Bill Gropp notes that a very common use of ispawn is likely to be to request additional resources in a task-farming example; when the job is done, any pending requests for resources are cancelled. In a cluster implementation interacting with a queuing system, cancelling the ispawn requires only telling the queuing system to dequeue the request (or kill the started processes). After much discussion, we deferred this to the collective discussions.

In the process utilities section, the behavior of MPI_PROCESS_SIGNAL when the signal is not deliverable (for example, the destination system does not support that signal) was specified (drop silently); Bill Gropp points out that the behavior should be left to the implementation (it is an erroneous user program). Tony Skjellum raises the issue of thread safety; the signal routines are not thread-safe (or rather, do not have thread-safe effect). General consensus to merge process signal and group signal routines.

Break.

Marc Snir begins the discussion of the one-sided operations by going right to the issue of how much like shared-memory the one-sided operations are; this has generated the majority of email on the one-sided operations. For example, does

```
Process 0                                   Process 1
Put( a, 1 )                                 while ( a == 0 );
```

work? Does

```
begin_flag++                     while (1) {
store data                         get(end_flag)
end_flag++                         get(data)
                                   get(begin_flag)
                                   if (end_flag == begin_flag) break;
                                   }
```

work? In hardware terms, these have to do with cache coherence and the ordering of memory updates. The concern is that some hardware is not cache-coherent and may not be able to efficiently implement some of the one-sided operations.

If remote memory access is *not* (necessarily) cache coherent, then the possibilities include

- Any MPI call
- Select MPI calls
  - barrier, others
  - explicit window expose/hide

Marc Snir notes that the second example won't work on shared memory systems because of the lack of sequential consistency in the store/get operations. Rolf R. presented a chart of some models. Much discussion of the problems of different shared memory models. Call for a straw vote for whether we have a coherent model of put/get Specifically,

- No synchronization required between put/get and load/store *2*
- Some synchronization required between put/get and load/store *20*
- Abstentions *10*

Raja Daoud points out that the second example still requires something to force storage ordering, such as explicit machine instructions to force storage ordering.

Additional discussions of the interpretation of the datatype, particularly for heterogeneous put/get.

Should the `target_count` be kept *11-11-11* in put/iput/get/iget. Alternative (suggested by Eric Salo): separate function when `origin_count == target_count` and `origin_type == target_type` *4-15-12*.

Steve Huss-Lederman notes that a clarification of MPI 1 is needed to match the restrictions on `MPI_(I)Accumulate` about the MPI predefined operations applying only to the predefined datatypes or datatypes containing only one predefined datatype. Steve and Bill Gropp agree to write a clarification; Marc Snir also notes that MPI 1 does not specifically require the same operation in an `MPI_Reduce` call and that this also needs clarification; Steve and Bill agree to do this as well.

On to fences. A proposal to split fence into begin/end fence, where endfence waits for the completion of puts begun after the beginfence. Issues include nesting and multithreading.

- Keep fence as in text *8*
- Replace with begin/end fence *7*

Close for dinner.

3 Thursday

Rusty Lusk opens the meeting with a reminder that we need to be taking votes, not continuing to seek out new proposals.

Future meetings are June 5–7, July 17–19, September 4–6, October 9–11, November Supercomputing, and a meeting in February for "cleanup" (clarifications and corrections). Binding votes are scheduled as (recall that there are two binding votes for each section; note that these are planned and may change).

April Dynamic, One-sided

June Dynamic(2), One-sided(2), External

July External(2), Binding, Miscellany, Collective

Sept Collective(2), Binding(2), Miscellany(2), Real Time

Oct Real Time(2)

Marc Snir discusses the issue of I/O and the MPI-I/O effort. Specifically that the MPI Forum begin working on an I/O specification, not that it adopt the MPI-I/O effort (rather, noting that MPI I/O is an example of current practice). Much discussion on this. Rusty Luck calls for a vote on whether the MPI Forum should start at the next meeting a parallel I/O effort: *26-7-4*.

Split into two subcommittees: dynamic and external. Dynamic looked at peer-to-peer communication (connect/attach). This section has been completely rewritten.

(Minutes taker departed for external discussion, with emphasis on external representations of datatypes.)

Lunch

Split into two subcommittees: one-sided and bindings.

Much discussion of the issues of what calls are valid inside a handler, leading to a view that MPI provide a handler interface based on threads, which provides an MPI "convenience" interface to using threads. Concern about the performance costs of threads/handlers was expressed, with many noting that, even though users don't always appreciate this, having thread friendly/handler MPI will (often) decrease performance of operations like send/receive (particularly latency). Many noted that in the "any function may be used in a handler" case, the *users* code may need to be signal-safe. Many scenarios that would either deadlock or cause frequent (usually unnecessary) lock/unlocking were described.

- Full handlers only for "thread friendly" MPI *13*
- restricted handlers *0*
- nothing *5*
- Full function handler (equivalent to requiring threads) *1*
- Full function, eventually executing handler (e.g., may be a polling implementation) *5*

Break

Split into Miscellaneous and collective subcommittees.

4 Friday

Notes provided by Rusty Lusk.

Jeff Squyres presented the C++ binding.

Much discussion ensued focussing on

1. Use of "const", and whether this should be propagated back into the C binding. Conclusion was that this was a *bad* idea, since it would be a change to the MPI 1.1 standard. (Even though it should not affect any user code, merely make life harder for implementors).

2. Asymmetries in the binding of functions which use a single object (and have become member functions) vs similar functions operating on arrays of objects (which haven't). Conclusion was that we'd have to live with the

asymmetry in this level of binding, since otherwise the reasonable places to do store allocation move. (And we need many more classes, one for each `array_of` that we need).

3. The rules for converting the C bindings into C++ bindings, and in particular which object to make the functions methods of when the original function operates on multiple objects. (No strong conclusion reached, since this is clearly a matter of taste). There was general acceptance, however, that laying down a set of rules for this conversion was a *good* idea.

A decision was made not to go to a formal vote on the chapter yet, since much re-wording is required to turn it from its current state as a proposal into an imperative form suitable for the standard. This is expected to be done prior to the next meeting.

The current binding reflects only MPI 1.1, chapter chairs for MPI 2 are requested to provide the C++ bindings of functions in their chapters for inclusion in the binding.

Marc Snir presented the One Sided chapter for its first official vote.

There were 22 voting organizations present.

Voting proceeded by section.

Introduction : no vote taken, as it is not substantive.

Section 4.2: `RMA_Init`, `RMA_Malloc` etc Vote: **17:1:3** *PASSED*

Section 4.3: Put and Get calls

Proposed Amendment: Put/Get should not have a `target_count` argument, since this can be deduced from the data type and source count arguments.

Those who prefer the amendment (i.e. no `target_count` argument believe that it gives the user extra work, for no benefit. Those against argue that the user likely knows it, that having the user give the information can save the system working it out, and that the argument should be there from symmetry with a send/receive, in which both counts are necessarily given)

Vote: **4:15:3** *Amendment FAILED*

Proposed Amendment: Delete the non-blocking RMA calls.

Argument for the amendment is that RMA calls have a relaxed completion semantic in any case (completion of a put does *not* guarantee that the data has been stored in the target memory, merely that it is safe to overwrite the local buffer from which the put was occurring), and therefore having immediate versions of the calls adds little except more functions and implementation complexity. Argument against is from symmetry with message passing calls where we have both blocking and immediate versions, and that these functions may be expensive in some implementations.

Vote: **1:17:4** *Amendment FAILED*

Whole of section 4.3: Vote: **23:0:0** *PASSED* (Possible mis-count by 1, but this is clearly irrelevant to the outcome...)

Tony Skjellum then suggested the addition of persistent versions of the RMA operations. These would bind all the arguments including the buffer addresses at each end. Since there is no proposal a straw vote was held to see if such a proposal is worth making. Straw Vote: Large majority in favor of seeing such a proposal.

Tony will therefore prepare something for next time.

Section 4.4 Accumulate functions.

A friendly amendment was accepted to add detailed phrasing to rule out the possibility that any storage in the target data type can be written more than once by a single initiator.

Vote: **19:1:0** *PASSED*

Section 4.5 Fence

Much discussion on how exclusive to make the fence.

A straw vote (beauty contest) was held on three possible proposals

1. Proposal as written. (Fence fences all relevant RMA operations). *14*

2. Introduce bracketing `fence_begin` and `fence_end` calls, the `fence_end` only forces completion of calls since the `fence_begin`. (Issues of nesting are not clear here.) *6*

3. Option 2, and also delete the implicit fence operation from `MPI_Barrier`. *0*

A `fence_begin/fence_end` proposal may be included for consideration next time.

4.5 as in document Vote: **19:0:3** *PASSED*

4.6 Semantics

Discussion and vote delayed pending detailed proposal for expose/deliver (as per Rabenseifner).

4.7 Extensions

4.7.1 Counters Vote: **11:6:3** *PASSED*

Next proposal will add counter arguments to all RMA operations.

4.7.2 Data type transfer operations

Postponed for discussion in MISC chapter.

4.7.3 Access to window arguments Vote: **17:0:2** *PASSED*

4.7.4 Read Modify Write operations

Amendment to reduce the functionality significantly by removing datatype argument (only int will be permitted), count argument (only one is permitted), op restricted to limited set (to be determined).

The objective here is to make this into a fast semaphore operation, and discourage its use for anything else. A straw vote was taken on the amendment: Straw Vote: *15:1:5* *PASSED*

Vote on function as amended: Vote: **21:1:0**

No votes were taken on 4.8 (Communication handlers), in the light of previous decisions effectively to remove them from non-threaded environments.

Steve H-L presented MPI -2.

It has reduced the number of functions to 1: `MPI_DO_WHAT_I_WANT(...)` other un-minuted humor was also presented.

5 Attendees

(Partial list)

Name	Institution	email
Ron Brightwell	Sandia Nat. Labs.	bright@cs.sandia.gov
Dennis Cottel	NRAD	
Joel Clark	Intel	joel@ssd.intel.com
Jim Cownie	BBN	
Raja Daoud	HP (Convex)	raja@rsn.hp.com
Terry Dontje	TMC	tdd@think.com
Richard Frost@SDSC@frost@sdsc.edu		
Al Geist	ORNL	gst@ornl.gov
Bill Gropp	ANL	gropp@mcs.anl.gov
Leslie Hart	NOAA/FSL	hart@fsl.noaa.gov
Hans-Christian Hoppe	PALLAS	hch@pallas.de
Steve Huss-Lederman	CCS	lederman@cs.wisc.edu
Arkady Kanevsky	MITRE	arkady@mitre.org
Koichi Konishi	NEC	konishi@research.nj.nec.com
Susan Kraus	NEC	kraus@nstc.necsyl.com
Andrew Lumsdaine	Notre Dame	lumsdaine.1@nd.edu
Rusty Lusk	ANL	lusk@mcs.anl.gov
Nicholas Nevin	OSC	
Rolf Rabenseifner	University Stuttgart	rabenseifner@rus.uni-stuttgart.de
Eric Salo	SGI	salo@sgi.com
William Saphir	NAS (NASA Ames)	wcs@nas.nasa.gov
Marc Snir	IBM	snir@watson.ibm.com
Jeffrey Squires	Notre Dame	squyres@cse.nd.edu
MaryDell Tholburn	LANL	marydell@lanl.gov
Dick Treumann	IBM PPS	treumann@kgn.ibm.com
Parkson Wong	NASA/Ames	parkson@nas.nasa.gov