Minutes of MPI meeting
March 6–8
Chicago IL

William Gropp

March 8, 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. Offical 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 meeting with a discussion the schedule of the next meetings and the readings of the chapters, followed with a discussion of the agenda.

William Saphir starts with a discussion of the dynamic process section. The various `spawn` routines have been specified completely (all arguments defined completely). The first argument now contains the executable and argument list, complete with a way to specify arguments that contain embedded blanks. Much discussion about this but no constructive suggestions (Recall that the `argv[]`-like form was rejected in previous meetings).

The `info` argument is a string that contains all other information, and is opaque to MPI. Rusty Lusk notes that if in fact the `info` has a filename that points to very extensive information, the command line might be supplied by the file, and hence the command-line argument should be `null` in that case. Laurie Costello asked whether the `info` argument should be a `char *` or `void *`.

Marc Snir points out that the "boolean" `flags` are not actually booleans; Bill Gropp adds that the text is describing `flags` as a bit string, not a boolean.

A question was raised about the interpretation of the `array_of_errorcodes` in the case where the (opaque) `info` argument specifies a particular collection *and ordering* of processes. That is, how does the user determine which process, described in `info`, does `array_of_errorcodes[6]` refer to? Bill Gropp pointed out that where this is important, `MPI_SPAWN_MULTIPLE` may be used. Richard Frost asked if we need an array of errorcodes for `MPI_SPAWN`. Bill Saphir points out that `MPI_SPAWN` returns an error code itself; if the return is `MPI_SUCCESS`, the user can ignore `array_of_errorcodes`.

We moved on to `MPI_SPAWN_MULTIPLE` and `MPI_SPAWN_INDEPENDENT`. Spirited discussion about using `MPI_SPAWN_INDEPENDENT` for spawning MPI jobs that do not interact with the parent. For example, for using MPI to manage unrelated MPI programs.

Bill Saphir then disucsses `MPI_UNIVERSE_SIZE`. The original version was just an integer; Rolf suggested something like `MPI_UNIVERSE_SIZE( in info, out size )`, where `info` is the same `info` used with the spawn routines, along with an error code (class?) of `MPI_ERR_SIZE_UNKNOWN`. This lets the user get information about a heterogeneous environment (including one that is dynamically defined by constructing an `info` string that accesses new resources during execution). Much discussion of this and its relationship to the resource manager discussions.

Break.

We continued the discussion of dynamic processes. It was noted that the text says that the wait/test routines are collective; this needs to be changed.

The signal routines are then described. Lloyd Lewins asked if in `MPI_SIGNAL_GROUP` the signal can be delivered to some but *not* all of the processes (e.g., it is a POSIX signal and in a heterogeneous environment, some processes don't support POSIX). Should `MPI_SIGNAL_GROUP` return an array of error values instead of a single value?

Should there be more just `MPI_SIGNAL_KILL`? *15-3-7*

On to `MPI_NOTIFY`. One proposal allows a different kinds of events, the other provides only for notification on process exits (or becomes unreachable). Marc Snir notes that the need for `MPI_NOTIFY` relates primarily to `MPI_SPAWN_INDEPENDENT` processes (note particularly the use of a `MPI_Group` instead of an `MPI_Comm` argument to `MPI_Notify`). Marc also notes that an attempt to use a communicator that contains a process that died (and caused a notify) after receiving the notify will still (strictly should, since the program is now erroneous) fail; that is, receiving the notify does not "fixup" the communicator (I hope I've got Marc's comment right).

Notify on general events: *16*; on process exit only *2*, abstain *8*.

An inconsistency between `MPI_NOTIFY` and `MPI_SIGNAL` was noted in that there is no notify routine for a single member of a group as there is for signal.

- leave as is *12*
- change to make same *8*

- abstain *6*

Lloyd Lewins asked if the `MPI_SIGNAL` is synchronous in the MPI sense (e.g., like `MPI_Ssend`). That is, does `MPI_SIGNAL` returns *only* when the signal has been delivered. Marc Snir points out that most MPI routines complete locally (except for things like `MPI_Ssend`) and this routine should be like the others. General agreement.

Much concern about whether the functionality of notify/signal will *mandate* demons on some systems. No convincing proof was offered.

Bill Saphir then moved on to the `MPI_ACCEPT/MPI_CONNECT` routines. The issues include

1. What is the name?
2. Who manages name (user or system)?
3. How do client/server get the same name?
4. What is the scope of name (per job, per application, per user, global)?
5. Is a nameserver (object that can transfer name from server to client) allowed/required/excluded?
6. When is name valid to connect to? Is it persistent?

Break for dinner.

3 Thursday

Marc Snir opens the meeting on page 38 with a discussion of the access to communication buffers that are modified outside of MPI. For example,

```
Process 0                     Process 1
flag = 1                      while(get(flag) == 0)
```

is not allowed. Note that in a case where the remote memory is implemented either with message-passing or with weak memory coherency, this restriction is needed by implementations.

A related issue is the need (in some cases) for users to declare variables `volatile` or take their address (`&a`) (as Dick Truemann reminded us) so that a compiler will know whether the contents of the variable can be changed outside the knowledge of the compiler.

Marc moves on to `MPI_ACCUMULATE`. The current draft allows user-defined functions for the accumulate; the problem with this is that only the calling process has the user-defined function.

Lloyd Lewins suggests an alternative to require the source to perform the operation. This would still allow an optimization for the predefined operations (e.g., `MPI_SUM`).

No objection raised to dropping user-defined functions.

The next issue is non-blocking accumulate, which Lloyd Lewins suggests dropping. Jarek N. defends the need for `MPI_IACCUMULATE`. This discussion moved directly into read-modify-write. Lloyd Lewins is concerned that an implementation of nonblocking read-modify-write may cause a lock that is held for a long time (until the wait); Marc Snir responds that the implementation could simply complete the read-modify-write before returning.

Marc Snir points out that compare and swap is not associative, and the current functions like `MPI_REDUCE` that take user-defined operations "essentially" require associate operations. Bill Gropp points out that user-defined functions already have a flag that indicates whether it is commutative; an alternative approach is to allow an additional flag that indicates that the function in non-associative. This would meet a user request for "specified order" collective computation (both for some numerically sensitive algorithms and for bit-wise comparision with sequential computations), and would make compare-and-swap an example of such a function, rather than a special case. Bill Gropp volunteered to write a specific proposal.

Is RMW is an implicit fence? Jarek N. points out that `MPI_FENCE` may be expensive. Some possibilities for `MPI_FENCE` are

1. global per caller
2. global per caller and window
3. per caller and target
4. per caller and target and window
5. per operation

Much discussion about what Fence actually means; this is particularly important for distributed memory systems where an implementation may be quite expensive. Should there be a synchronous put (like the `MPI_Ssend`)? General consensus that this requires more thought and written out proposals.

We moved on to counters; currently there is one counter per window. Is the request in the `MPI_SET_COUNTER_THRESHOLD` persistent? Marc Snir notes that this is a general problem in the draft, since other functions (e.g., dynamic processes) return requests, not persistent requests. Questions about `MPI_SET_COUNTER_THRESHOLD` include: are multiple thresholds and their requests allowed? Should there be a blocking version (wait for counter)? If there are multiple requests, how are they delivered (since the threshold is subtracted from the counter, what if there are two requests with the same threshold: is the threshold subtracted once or twice)?

Locks for windows. Once major issue is how this would be implemented on shared memory system. There were no supporters.

The global datatypes were deleted because an alternate mechanism (through a defined external representation of datatypes) is presented in the Miscellany section).

We moved on to a discussion of atomicity. Lloyd Lewins points out that while read-modify-write can be used to implement locks, it is not an efficient approach on some shared memory implementations (in particular, not scalable in memory accesses, since a user is likely going to spin-loop on read-modify-write calls). A proposal for a separate MPI_LOCK will be provided.

The issue of whether the put operations are ordered or not came up again. It was pointed out that this is what the MPI_FENCE is for, even though the exact specification of MPI_FENCE is still being worked on. Several implementation scenarios were presented where both the implementation and user would desire unordered puts. For example,

```
PUT( 1000000 bytes )
PUT( 1 bytes )
```

The large put might use a separate memory move engine, while the short put might use a short message.

This brought back the call for a "synchronous put" which does not complete until the data is moved. There was much objection about the use of the name "synchronous"; Bill Gropp pointed out that what was really wanted was an "ordered put" (MPI_OPUT).

Break.

Break into two groups for dynamic and real time, with external and binding after lunch and miscellaneous at the end of the day.

Bill Saphir started with a discussion of the accept/connect functions in the dynamic process chapter by reviewing the various proposals.

Discussion of the published, advice, and internal names. An issue is how much detail (for things like authentication, quality of service, etc) is mandated. It is pointed out that the proposals are (attempting) to isolate the way in which "names" are comunicated between 'client' and 'server' so as to not require some kind of name server or broker. One issue is whether a name can be used by multiple servers (allowing any server to respond to a client) or if a single server can choose to use the same 'port' for several methods. One approach is to say that this is essentially like the info string in MPI_SPAWN; that is, it is opaque to MPI and is constructed using some separate API (for example, using DCE calls to access a name broker). This API could provide additional interfaces for authentication, quality of service, etc., just as an API for the info string can provide complex resource control.

Marc Snir presented a proposal that separates published names from methods, allowing a server to accept by method. That is, to separate a particular server from a service (which could be served by one of many servers).

Lunch.

Marc Snir opens with a discussion of handlers, in particular, the change to making the requests persistent. One problem is with

```
MPI_Irecv( ...., &req );
MPI_Post_handler( ..., req );
```

Note that this has a potential race condition if the request is complete before the `MPI_POST_HANDLER` is called. Since the handler is called when the request is complete, there is a question about when the request is actually complete (when the handler starts? ends?).

Can the user do a wait/test on the request above? For example,

```
MPI_Irecv( ..., &req );
MPI_Post_handler( ..., req );
MPI_Wait( &req, &status );
```

Bill Gropp proposes an alternate model based on reference counts on the request; both `MPI_IRECV` etc. and `MPI_POST_HANDLER` increment the reference count, wait/test/free and the action of the handler decrement the reference count. This allows the user to do either the irecv/handler/wait above or

```
MPI_Irecv( ..., &req );
MPI_Post_handler( ..., req );
MPI_Request_free( &req );
```

A big question is how these function in a multithreaded environment; for example, what happens if one thread is already waiting on a request when the handler fires?

We moved on to `MPI_HANDLER_LOCK` and `UNLOCK`. Problems with these include that these are asymetric and coarse grain. Lloyd Lewins pointed out a number of problems with the current definition. General agreement that this needs more work.

Steve Huss-Lederman started generalized request, with Bill Gropp explaining the need to require that `MPI_REQUEST_MARK_COMPLETED` be called from within a handler; this allows using generalized requests for complex MPI communications (like the nonblocking allgather example) but may not be sufficient for other, unrelated MPI operations (e.g., using a generalized request for a nonblocking I/O operation).

We moved to getting information about MPI opaque objects, starting with requests. There was concern about requiring information that an implementation may not otherwise need to save. Some discussion of what 'unique' means for communicator id. In particular, is it enough to have unique within overlapping groups but not unique between disjoint groups. Marc Snir notes that you can accomplish the same functionality (for profiling) by trapping communicator creation calls. Steve called for a straw vote for having a comm id function and a way determine if the id is unique *10-10-2*.

Bill Gropp and Marc Snir point out that the routines that get information from a request (e.g., `MPI_REQUEST_TAG`) may not be available; in the case of a send, there may be no information in the request to return; in the case of a receive, there may be no data other than what is available through the `MPI_Status`. Persistent requests are another issue.

When discussing the true extent of datatypes, the question came up whether `MPI_TYPE_LB` gives the true LB or the user-set (with an `MPI_LB` element in a `MPI_TYPE_STRUCT`). Refered to Miscellany for clarification.

Next, caching on MPI handles was discussed.

Finally, we returned to datatype decoding. A problem is that `CHARACTER*(*)` datatype in Fortran is limited in size (one implementation limits this to 255 characters) and a datatype representation may be quite long. Much discussion. Bill Gropp suggests adding a new basic MPI datatype, `MPI_DATATYPE`, which could be used for send and receive. Marc Snir points out that by using `MPI_PACK/UNPACK` with this, you can accomplish the encode/decode for things like saving the datatype description to a file. Bill Gropp volunteers to provide this proposal.

Meetings are April 24–26, June 5–7, July 17–19, September 4–6, and October 9–11, at the Ramada near the Chicago O'Hare Airport.

Break.

Rusty Lusk starts Miscellany. Lloyd Lewins discusses the datatype improvements. These have to do with compiler rules for structure alignment.

`MPI_TYPE_SIMPLE_STRUCT` would use the default alignment behavior of the compiler. A concern was expressed over whether this would be prone to error because of compiler switches (or different choices of compilers) that would work with the same MPI library (often the case now, since the only user-visible structure is `MPI_Status`, which is likely just 4 `ints`).

A similar discussion was held on `MPI_TYPE_CONTIGUOUS_STRUCT`. Because this routine does not assume a default alignment, it is not as subject to the potential problems of changes in compiler alignment rules.

Marc Snir then presents interlanguage interoperability. There was a discussion of whether there are routines to convert MPI handles to and from a cannonical representation or specific functions for Fortran to and from C (the case of current practical interest). There seemed to be support for the direct Fortran to/from C conversion.

We moved to a discussion of `MPI_BOTTOM`. The three approaches are roughly

1. `MPI_BOTTOM` is the same in C and Fortran. May be hard to implement if a third language is added.

2. Addresses are the same, but `MPI_BOTTOM` may differ. This may add complexity to the Fortran implementation.

3. Leave as is (Fortran and C datatype not interoperable).

On to the addressing problem with Fortran, where an address may not fit in a Fortran `INTEGER`. A Fortran 90 solution uses `INTEGER(kind=MPI_ADDRESS_SIZE)` with the MPI implementation providing `MPI_ADDRESS_SIZE`. A similar approach can be used with Fortran 77.

Laurie Costello then discussed the handling of `CHARACTER` arguments for `<choice>` arguments. It was pointed out that `MPI_SENDRECV` takes two choice arguments with no requirement that the datatypes be related.

1. Add `MPI_CADDRESS` and type `MPI_TYPE_STRUCT` with `MPI_BOTTOM` (ugly but simple). *8*

2. Allow implementations to not support `CHARACTER` everywhere. *2*

3. Add new functions for all routines that may take character arguments. *2*

4. No change *5*

5. Abstain *5*

Rusty Lusk then reviewed some miscellaneous changes, starting with allowing the use of a null status. This is no change to the MPI 1 standard, just a nonconflicting extension.

Next, we discussed the nondestructive test `MPI_GET_STATUS`.

And proceeded onto MPI 1 clarifications. The clarification to `MPI_INTERCOMM_CREATE` was accepted (without vote). So was the `MPI_ERR_KEYVAL`. The discussion of `MPI_Type_size` concluded with acceptance of Bill Gropp's proposal to leave it unchanged but add an advice to users that the binding is as intended. The final topic is a `MPI_VERSION`; the suggestion is that there be an `MPI_VERSION` and `MPI_SUBVERSION` to simplify using the C preprocessor to deal with small differences in MPI (for example, the changes in the attribute routines).

Eric Salo asks if invalid but irrelevant arguments are erroneous or not (this came from a user question). The actual case the user encountered was an invalid tag value when sending to `MPI_PROC_NULL`. This needs a written proposal.

Leslie Hart asked about the impact of errors that are 'not so bad'. Leslie Hart and Rusty Lusk will write up a proposal.

4 Friday

Rusty Lusk opens with a review of the agenda.

A presentation of the discussions of the realtime subcommittee followed. MPI/Realtime (MPI/RT) has decided not to determine but to provide tools to implement the following paradigms:

- priority based
- time driven
- event driven
- ad hoc (real fast)

MPI/RT will not replace the native runtime system or scheduler, but will provide a portable API to these systems. It will also be a subset of MPI-2. Embedded MPI may not have the full functionality of MPI/RT. Signals in addition to KILL are required. Timeouts will be provided on both sender and receiver sides. Can be added unobtrusively with `MPI_WAIT`. A number of other needs (such as predicitability and quality of service) were presented. The subcommittee will develop a representative suite of real world applications for proof-of-concept.

Jeff Squyres presented the work of the bindings committee, which concentrated on the C++ binding. Two proposals (One from Notre Dame and one from Mississippi state) were discussed. The issues discussed included

- Copy semantics of MPI objects (concluded shallow)
- No overrriding of operators, except assignment (possibly also `==` and `!=`).
- Destructors: Should they automatically call `MPI_*_free`? No, because
  1. Not consistent with C
  2. MPI style - user responsible for alloc/dealloc
  3. An object going out-of-scope can cause unexpected behavior (e.g., `MPI_Comm_free` might be collective)
- Should there be C++ bindings for all of MPI-2: yes

Plan to merge the two bindings, discuss by email, with reliminary reading at the next meeting.

On Fortran 90, there was no new text proposed. Overloading functions possible but not clear. An alternative approach is to use an MPI module, which will fix a number of problems. More work remains to be done.

Steve Huss-Lederman first reviewed the process of putting the document together (Steve is the editor). He then reviewed the discussion of the MPI external chapter (minutes above).

Bill Saphir returns to the dynamic process chapter. General agreement to re-arrange the chapter to separate the `MPI_SPAWN_INDEPENDENT` etc. (non MPI processes) from the MPI processes. Next, we discussed changing the binding of `info` from `char *` to `void *`. For `char *` over `void *` *5-14-7*.

We went on with the command line processing. The vote was: some quote mechanism *20*, no quote *1*, argv *4*.

Break.

What should be the Fortran equivalent of `NULL` in cases where we want to allow the user to, for example, not pass an array of error codes. To do something: *6-8-9*.

Next, how is the `flags` argument to `SPAWN` interpreted? One suggestion is to replace `flags` with a value that is the minimum number of processes; a value of `1` (or zero!) has the effect of `MPI_SPAWN_SOFT`. Much discussion about whether 'soft' should be in the `info` argument.

A vote on what to do with this came out as: Soft flag *7*, min/max *11*, neither *1*, abstain *8*.

Next, what are the order of the `error_codes` on output from the spawn commands? In particular, what if the `info` argument specifies an order?

Should MPI specify the order of error codes (yes/no/abstain): *17-3-5*.

Next, should `MPI_SPAWN_INDEPENDENT` remain? If so, should it be able to create an independent MPI job? Concern was expressed about whether this should be done by spawning a single instance of `mpirun` or can `MPI_SPAWN_INDEPENDENT` start an MPI job with their own `MPI_COMM_WORLD` and without contacting their parent?

Greg Burns points out that the real issue is how do we create MPI processes that do not communicate with the parent?

Deleting `MPI_SPAWN_INDEPENDENT` *7*; keep *14*; abstain *4*.

- As is: *3*
- Change: *19*
    1. Can spawn, but no intercomm *17-2-5*
    2. Can spawn non-MPI processes (task farming) *21-1*
    3. Should there be a flag *5* or separate functions *16*?
- Abstain: *3*

Steve Huss-Lederman closes with the traditional MPI -2 presentation.

5 Attendees

Name	Institution	email
Ron Brightwell	Sandia Nat. Labs.	bright@cs.sandia.gov
Greg Burns	OSC	gdburns@osc.edu
Joel Clark	Intel	joel@ssd.intel.com
Laurie Costello	Cray	lmc@cray.com
Raja Daoud	HP (Convex)	raja@rsn.hp.com
Terry Dontje	TMC	tdd@think.com
Stephen Fleischman	HP (Convex)	sfleish@convex.hp.com
Richard Frost@SDSC@frost@sdsc.edu		
Al Geist	ORNL	gst@ornl.gov
Robert George	MSU	robert@erc.msstate.edu
David Greenberg	Sandia	dsgreen@cs.sandia.gov
Bill Gropp	ANL	gropp@mcs.anl.gov
Leslie Hart	NOAA/FSL	hart@fsl.noaa.gov
Shane Herbert	MSG	shane@erc.msstate.edu
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
Mario Lauria		
Lloyd Lewins	Hughes Aircraft	llewins@msmail4.hac.com
Rusty Lusk	ANL	lusk@mcs.anl.gov
Jarek Niplocha	PNL	j_nieplocha@pnl.gov
Rolf Rabenseifner	University Stuttgart	rabenseifner@rus.univstuttgart.de
Arindam Saha	Mississippi State U.	dee.msstate.edu
Eric Salo	SGI	salo@sgi.com
William Saphir	NAS (NASA Ames)	wcs@nas.nasa.gov
Lance Shuler	Sandia Nat. Labs	shuler@cs.sandia.gov
Marc Snir	IBM	snir@watson.ibm.com
Jeffrey Squires	Notre Dame	squyres@cse.nd.edu
Steven Taylor	Caltech	steve@scp.caltech.edu
MaryDell Tholburn	LANL	marydell@lanl.gov
Dick Treumann	IBM PPS	treumann@kgn.ibm.com
Simon Tsang	Intel	simont@ssd.intel.com
Jerrell Watts	Caltech	jwatts@scp.caltech.edu
Parkson Wong	NASA/Ames	parkson@nas.nasa.gov