Minutes of MPI meeting
July 26–28
Chicago IL

William Gropp

July 28, 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. 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*. Please send any corrections to `gropp@mcs.anl.gov`.

2 Wednesday

Rusty Lusk opens meeting with a discussion of the agenda.

Bill Saphir begins discussion of dynamic process management with a review of the goals of process management in MPI.

2.1 Resource aquisition

Resources are *discovered* (what is available) and *allocated* (reserved for the program). The routine `MPI_RESOURCE_DISCOVER` can discover features of the resources. The concept of "recommended" number of processes raised the issue of how it is defined. Many thought that it should be the number of CPUs; others wanted it to be undefined (for example, weighted by the speed of the CPU). Hans-Christian suggests an attribute-like mechanism for acquiring information about resources; this has the advantage of being extensible.

Disucssion of merging and splitting resource objects. Much concern about race conditions between discover and allocation. For example, discover in a network of workstations, when done by several separate jobs, might return the same resources as available to separate jobs, when the jobs expected exclusive access to the resources. One possible solution is that discover gives you information only about resources that are already allocated. This is the case for MPPs, where discover returns the part of the MPP that the MPI job is already running on. In a PVM-like NOW, it might give the members of the virtual machine (but not processors that could be added to the virtual machine).

Brief discussion of client-server and the reason for the introduction of `MPI_Connection`.

Break

2.2 One-sided communication

Paul Pierce introduces his proposal for get and put operations.

`MPI_Target` specifies the location in memory that can be the target of a put or get. A datatype is attached to the target to allow for operation in heterogeneous environment. There is concern over whether `MPI_BYTE` is appropriate at the datatype when the user wants to use the same area with many different datatypes.

Lloyd Lewins raises the issue of whether the sharable memory is any memory or is some special memory that needs to be allocated separately. Bill Gropp mentions that this is the case on workstations (mmap and System V shared memory) and systems without hardware cache coherence. One suggestion is `MPI_TARGET_MALLOC` which would be like `malloc` on systems with cache-coherent fully shared memory but might be something else on other systems. The call is collective and may or may not be synchronizing. In the case of mmap or system V shared memory, it will be; where it can be implemented as `malloc`, it may not be synchronizing. A FREE routine is also needed. Steven Fleishman asks about `MPI_TARGET_MALLOC` on NUMA systems, where you might want control over where the memory is allocated. One suggestion is a root in the call.

Another issue on `MPI_TARGET` is whether there should be a way to restrict access to either gets or puts but not both (for example, a read-only region would be get-only).

No one supported the `MPI_SPUT`.

The issue of a flag that can be set when data is delivered/accessed was discussed. Alternatives include incrementing a value.

Synchronizing memory (through a fence across a communicator) was also discussed. This fence would cause all puts to complete before the fence completes. The obvious fence routine is `MPI_BARRIER`; note that the `MPI_TARGET` routine produces a communicator; this would allow `MPI_BARRIER` to know that it needed to perform the fence function.

Lloyd Lewins raises the issue of atomicity (for example, accessing single bytes when the basic coherent unit is a word/cache-line/etc). Bill Gropp points out that even if `MPI_TARGET` specifies a datatype that is "large enough", local accesses (without put/get) can still affect partial words. This can constrain some possible fast implementations.

Steve Huss-Lederman brings up the issue of ordering, with, for example, overlapping puts. Does `put(a);put(b)` cause the destination to get just `b`, or just one of `a` or `b`, or some arbitrary combination.

Evening subcommittee meetings.

3 Thursday

Rusty Lusk opens meeting.

3.1 Collective extensions

Steve Huss-Lederman describes the work of the collective operation routines and some of the external interfaces. The issue is whether threads are part of the design. Possibilities are

1. Say nothing about threads; add `MPI_IBCAST` etc.
2. Design to be thread safe. The issue here is what happens with nonblocking collective operations when used with threads. In other words, what are the *semantics* of the calls in a multi-threaded environment. The disadvantage of this is that there is a lot of hard work needed.
3. No `Ixxx` versions; everything is done with threads (probably POSIX). The advantage is no new `Ixxx` functions; also is forward looking. Disadvantage is that there is no standard threads (POSIX is too weak; does not guarentee non-blocking nature of all operations). There are also many tough questions about the semantics.

As example of the threads issues, it was noted that in a single process, multiple non-blocking collective operations can be distinguished by an internally maintained tag, with the requirement that the user call the routines "in the same order." However, in a multi-threaded environment, there is no notion of "same order" if several threads can use the same communicator. Bill Gropp mentions that threads could be constrained to use different communicators, but that this has its own problems, particularly in the numbers of communicators required and in the ability to send a message to an unspecified thread with point-to-point calls.

For using `Ixxx`: *25*. For using threads and no `Ixxx`: *2*.

This was followed by some discussion about how much effort should be spent on developing threadsafe semantics; this was encouraged by a unanimous straw vote.

3.2 Subcommittee meetings

The rest of Thursday was devoted to subcommittee meetings. Concurrent sessions; morning Realtime, External, and Resource. Early afternoon: Dynamic and Collective. Late afternoon: Miscellaneous and One-sided.

4 Friday

Rusty Lusk opens with the day's agenda: meeting schedules, chapter schedules, and reports from subcommittees.

Meeting dates are

September 6–8 in Chicago
October 23–25 in Denver
MPI BOF at Supercomputing'95
January 24–26, 1996 in Vienna

Information will be distributed later.

5 Subcommittee presentations

5.1 Miscellaneous

Rusty Lusk presents the miscellaneous committee discussions. The areas that will be discussed include

1. Standardize mpirun
2. Repairs to MPI 1.1
3. Datatype improvements (UB and LB, type contig)
4. Interlanguage handle translations
5. Interlanguage communication
6. Error handling
7. Making arguments robust (null status)
8. Interplatform operability

5.2 Realtime

Tony Skjellum presents the realtime committee discussions. Committee reached concensus that it is reasonable to add some realtime extensions to MPI. Such things include priority messages and channels; the expectation is that it will take a while to develop.

5.3 Dynamic

Al Geist presents the dynamic process management committee discussions. The sucommittee developed a number of new ideas. These are still under discussion.

Definitions: A resource object is an id of a set of resources that the owner can start processes on. A process id is a `(group,rank)`.

Properties: ownership

- not collective
- initially `(MPI_COMM_WORLD,0)` owns pre-existing resources
- any Task can have ownership through Iallocate
- ownership passed to children at SPAWN

Resource managment functions include discover, split, merge, and allocate. (We are considering blocking allocate first, but expect to do nonblocking eventually)

Resource inquiry functions include nslots, description, and processes. nslots represents the number of processes that can/should run on a resource.

Process management now has `process_init( a.out, argv, n, resource, inherit_resource, group )`; this allows a process to inherit resource. `start_attach( group, comm, &intercomm )` starts an MPI job and creates a intercommunicator with the parent. `start_abandon( group )` starts a job that does not communicate with the parent (e.g., a non-MPI process). `MPI_Init` sets `MPI_COMM_WORLD` and `MPI_COMM_PARENT`.

Renamed "client/server" as "Independant start".

`register(name, &handle)` gets a handle for a name. `accept(handle, root, comm, &intercomm )` accepts a connection to an external MPI process. `contact( name, comm, &intercomm )` connects to the MPI process that has done an accept on a handle returned from a register of name. Of course, we need a `free_handle( &handle )` to free the handle created by `register`.

5.4 One-sided communication

Paul Pierce presents the discussions of the one-sided committee.

One addition is the addition of an `MPI_TARGET_MALLOC`.

1. Snir proposal PUT/GET
   - Flexible datatypes
   - Flags
   - SYNC
2. Scatter/gather

3. others

One issue is how the target can determine that a get/put has occurred. Paul Pierce has suggested adding counters: `MPI_PUT_COUNTER( ..., counter_index, add_value )`, `MPI_SET_COUNTER( value, &request )` followed by an `MPI_Wait( &request, &status )`. There is a way to initialize counters.

There are many questions about how much behavior should be mandated, particularly with respect to ordering of operations.

Pratap Pattnaik made a short presentation arguing that the one-sided routines should have simple properties for maximum efficiency. For example, no ordering requirements. Another example is `MPI_TARGET`; IBM prefers `MPI_TARGET` to specify only ranges in memory, not datatypes. Another way to express this is that these routines should not be one-sided versions of send/receive.

5.5 External

Steve Huss-Lederman presents the discussions of the A problem with generalized requests had to do with how the user-functions were called. A solution to this problem involves *registering* user requests.

- Register
  - may finish before register
  - can one deregister?
- system knows when complete
- user funcations can return errors
- user funcations atomic: still need to coordinate with collective, etc.
- cancel
  - series of request cancles
  - call-back function?

Steven Fleishmann asks about polling implementations of this, or whether an implementation will have to make some guarentees about servicing these requests. This is an open question.

Another suggestion, made by Nathan Doss, is generalized caching (attributes on other MPI objects). There are problems, particularly with `MPI_Status`, since `MPI_Status` is user-allocated object that is never freed with an MPI routine. Rusty Lusk asks whether this will be applied to all opaque MPI datatypes, even ones that may be introduced in the future.

Proposals for decoding datatypes include a simple representation of the type map, and an ASCII representation.

There is a proposal for globally unique id for communicators for use in profilers and debuggers. Tony Skjellum raises the issue that this may require less scalable implementations of contexts. The problem becomes worse with dynamic processes.

Other needs for profilers and debuggers. Primarily a call for information from those who need the information.

5.6 Collective

Tony Skjellum presents the discussions of the collective subcommittee.

The current proposal includes nonblocking routines, a multicast, a re-examination of the sendbuf/recvbuf arguments to the MPI-1 collective routines, and collective extensions to intercommunicators. For the intercommunicator extensions, there may be new functions rather than re-using the intracommunicator functions.

6 Closing

Steve Huss-Lederman closes with the traditional -(mpi-2) presentation.

7 Attendees

Name	Institution	email
Greg Burns	OSC	gdburns@tbag.osc.edu
Joel Clark	Intel	joel@ssd.intel.com
Nathan Doss	MSU	doss@erc.msstate.edu
Stephen Fleischman	Convex	sfleish@convex.com
Al Geist	ORNL	gst@ornl.gov
Robert George	ARL	george@cs.nps.navy.mil
David Greenberg	Sandia	dsgreen@cs.sandia.gov
Bill Gropp	ANL	gropp@mcs.anl.gov
Kei Harada	Hitachi	haradake@soft.hitichi.co.jp
Tom Henderson	NOAA/FSL	hender@fsl.noaa.gov
Hans-Christian Hoppe	PALLAS	hch@pallas.de
Steve Huss-Lederman	CCS	lederman@super.org
Joefon Jann	IBM	joefon@watson.ibm.com
Arkady Kanevsky	MITRE	arkady@mitre.org
Susan Kraus	NEC	kraus@nstc.necsyl.com
Lloyd Lewins	Hughes Aircraft	llewins@msmail4.hac.com
Andrew Lumsdaine	Notre Dame	lumsdaine.1@nd.edu
Rusty Lusk	ANL	lusk@mcs.anl.gov
Jarek Nieplocha	PNL	j+nieplocha@pnl.gov
Pratap Pattnaik	IBM	pratap@watson.ibm.com
Paul Pierce	Intel	prp@ssd.intel.com
Heidi Poxon	Cray Research	hlg@cray.com
James Prvyne		
Nobutoshi Sagawa	Hitachi	sagawa@crl.hitachi.co.jp
Arindam Saha	MSU	saha@erc.msstate.edu
Eric Salo	SGI	salo@sgi.com
William Saphir	NAS (NASA Ames)	wcs@nas.nasa.gov
Eric Sharakan	TMC	sharaan@think.com
Lance Shuler	Sandia	shuler@cs.sandia.gov
Tony Skjellum	MSU	tony@cs.msstate.edu
A Gordon Smith	EPCC	smith@epcc.ed.ac.uk
David Taylor	TMC	taylor@think.com