MPI-2 Features

MPI-2 Features Overview

- Parallel I/O specified
- Dynamic process creation
- Extended collective operations
- One-sided communications
- Bindings for C++ (big disappointment)
- Bindings for Fortran 2000+ (smaller disappointment)
- Thread information access

MPI-2 and Threads

- Threads can be used for parallelism and concurrency (typically, waiting for events)
- MPI allows thread safety, does not require it
  - MPI-2 supplies inquiry functions for level of thread safety in the MPI implementation
  - Only MPI_Probe( ) involves testing for "current state" information in MPI-1 (*Probe waits until a message matching source, tag, and comm arrives)*

MPI-2 and Threads

- MPI_Init_thread(int *argc, char **argv, int *provided)
- Additional argument provided requests, returns level
  - MP_THREAD_SINGLE: only one user thread
  - MP_THREAD_FUNNELED: multiple threads allowed, but only one user thread makes MPI calls
  - MP_THREAD_SERIALIZED: mutiple threads allowed, but only one at a time makes MPI calls
  - MP_THREAD_MULTIPLE: all allowed

MPI-2 and Threads

- Thread that calls MPI_Init_thread is main thread
  - Only one allowed to call MPI_Finalize
- Value of requested thread level need not be same on all processes calling MPI_Init_thread
- Thread creation, management is not part of MPI
  - left to existing thread packages
  - using pthreads, OpenMP as shown

MPI-2 Externals

- External interface specs for debuggers, other tools to attach to MPI (or to layer MPI-2 on top of MPI-1)
  - MPI_Type_get_envelope/contents for decoding an MPI derived datatype
  - Generalized requests: user can define new nonblocking messaging functions, then use MPI_Test, MPI_Wait, etc.
- Explicit handling of C/Fortran interoperability
  - E.g., type MPI_Fint in C

MPI-2 Odds and Ends

- Can use collective communications with *intercommunicators* as well as *intracommunicators*
  - MPI_AllReduce, MPI_Bcast most common
- Intracommunicator collective calls can have an in-place option, allowing send and receive buffers to be same.
  - Recall, aliasing explicitly not allowed in MPI !
- Attribute caching for more than just communicators
  - datatypes, windows (used for RMA)

MPI-2 Odds and Ends

- One-sided communications (remote memory ops)
  - Looks like *shmem* programs; one process can essentially send w/o other explicitly receiving
  - Based on idea of window: portion of a process's address space it offers for remote memory ops by other MPI processes
  - Use put, get, accumulate to store to, load from, add into another process's window
  - Not implemented by some MPI implementations; buggy, unoptimized

MPI2: Process Creation

- MPI-1 is static: no processes after startup
- PVM was built around same time as MPI
  - parallel virtual machine
  - provides some distributed OS facilities
  - processes can be created, destroyed dynamically
  - can access information via *mailboxes*
  - based at Oak Ridge National Lab, still actively used
  - part of most Linux distributions (like OpenMPI)

MPI2: Process Management

- MPI-1 is strictly an *interface*, avoids OS jobs like resource management
- MPI-2 process mgmt operations are collective among all processes (ones creating, and ones being created)
- Resulting sets are represented by an intercomm
- Two fundamental operations
  - spawning new tasks
  - connecting to them

MPI2: Process Creation

- MPI_COMM_SPAWN starts MPI processes and establishes communication with them, returning an intercommunicator.
- MPI_COMM_SPAWN_MULTIPLE starts several different binaries (or same binary with different arguments).
- MPI_Info( ) is a hook to allow communication with underlying mechanism for distributed OS.

MPI2: Process Creation

- Attribute MPI_UNIVERSE_SIZE of MPI_COMM_WORLD gives total number of processes can usefully be started in all.
- UNIVERSE - WORLD is how many processes might usefully be started in addition to those already running.
- Note: implementation can return a value that is not anywhere near useful.

New Process Communication

- MPI-2 can establish communication between two sets of MPI processes that *do not* share a communicator
- Collective but asymmetric process: one group is server, another is client.
- How does client group discover server group?
  - MPI_Open_port + MPI_Comm_accept, or
  - MPI_Publish_name + MPI_Lookup_name

MPI C++ bindings

- MPI was intended partly be multilanguage
  - Call a C library from Fortran, vice-versa
  - C++ did not exist at time (but Fortran 90 did)
  - Lead to "lowest common denominator"
- What object should a base C++ class for MPI represent?
- No direct way to pass objects, just pack/unpack their data members

MPI C++ bindings

- Thin layer on top of MPI-C, wrapping mpi.h in an *extern "C" {...}*
- Names of member functions are "consistent with" the underlying MPI functions.
- Uses namespaces

C++ Bindings

```
namespace MPI {
  class Comm                                {...};
  class Intracomm : public Comm             {...};
  class Graphcomm : public Intracomm        {...};
  class Cartcomm  : public Intracomm        {...};
  class Intercomm : public Comm             {...};
  class Datatype                            {...};
  class Errhandler                          {...};
  class Exception                           {...};
  class Group                               {...};
  class Op                                  {...};
  class Request                             {...};
  class Prequest  : public Request          {...};
  class Status                              {...};
  class File                                {...};
  class Grequest  : public Request          {...};
  class Info                                {...};
  class Win                                 {...};   }
```

roundrobin with C++-bindings

```
#include <mpi++.h>  ...   // Note I use comm_world here but should use different name
  MPI::Init(argc, argv);
  int rank = MPI::COMM_WORLD.Get_rank();
  int size  = MPI::COMM_WORLD.Get_size();
  int to     = (rank + 1) % p;
  int from = (p + rank - 1) % p;
  if (rank = = p - 1)
          MPI::COMM_WORLD.Send(&msg, 1, MPI::INT, to, 4);
  MPI::COMM_WORLD.Recv(&msg, 1, MPI::INT, from, MPI::ANY_TAG);
  MPI::COMM_WORLD.Send(&msg, 1, MPI::INT, to, 4);
  if (rank == 0) {
       MPI::COMM_WORLD.Recv(&msg, 1, MPI::INT, from, MPI::ANY_TAG);
       cout << "Node " << rank << " received " << msg << endl; }
  MPI::Finalize();   ...}
```

OOMPI

- Library specification providing higher level OO
- A *port* is an object encapsulating *src* and *dest* of a message (a genuine connection model)
- Has info about underlying MPI communicator and the rank of destination
- i^{th} port of a communicator is array-indexed

```
OOPMPI_Intra_comm rowcomm;
rowcomm[m].Send(i);      // send i from port m
rowcomm[n].Receive(j);  // recv j from port n
```

OOMPI Messages

- OOMPI_Message object supports basic datatypes
- OOMPI_Array_message for arrays
- User-defined types, packing similar to C versions
- Avoid explicit message construction when possible

```
OOPMPI_Port port;
double v[100]; int tag;
...
port.Send(v, 100, tag)
```

OOMPI Messages

- OOMPI_Port::Send(OOMPI_Message buffer, int tag = OOPMPI_NO_TAG);
- OOMPI_Port::Send(OOMPI_Array_message buffer, int count int tag = OOPMPI_NO_TAG);
- Allows users to avoid explicit tagging if desired
- Stream interface also allowed (but inefficient)

```
OOPMPI_Port port;
...
port << i << j;
port >> i >> j;
```

roundrobin in OOPMPI

```
OOMPI_COMM_WORLD.Init(argc, argv);
int rank = OOMPI_COMM_WORLD.Rank( );
int p = OOMPI_COMM_WORLD.Size( );
OOMPI_Port to      = OOMPI_COMM_WORLD[(rank+1)%p];
OOMPI_Port from  = OOMPI_COMM_WORLD[(p+rank-1)%p];

if (rank == p-1)  to << msg;
from >> msg;
to << msg;
if (rank == 0) from >> msg;

OOMPI_COMM_WORLD.Finalize( );
```