

COMP4300: MPI-2 and Beyond

Alistair Rendell

13.1 MPI

MPI-1 (May 94)

- Fixed processor model
- Point-point communications
- Collective operations
- Communicators for safe library writing
- Utility routines

MPI-2 (July 97)

- Dynamic process management
- One-sided communications
- Cooperative I/O
- (Other small things!) C++ binding, Fortran 90, extended collective, misc.

Much more complicated, and much slower vendor uptake

13.2 Dynamic Process Management

- MPI-1 static or fixed number of processors
 - Could not add or delete processors
 - (You could have a fixed pool of processors and only use some of them, but cost of having idle processes may be large - implementation dependent)
- Some applications favour dynamic spawning
 - Run time assessment of environment
 - Serial applications with parallel modules
 - Scavenger applications
- Caution
 - Task initiation is expensive and should be used with careful thought

13.3 MPI-2 Process Management

Options:

- Parents spawn children
- Existing MPI applications can connect
- Formerly independent sub applications can tear down communications and become independent again

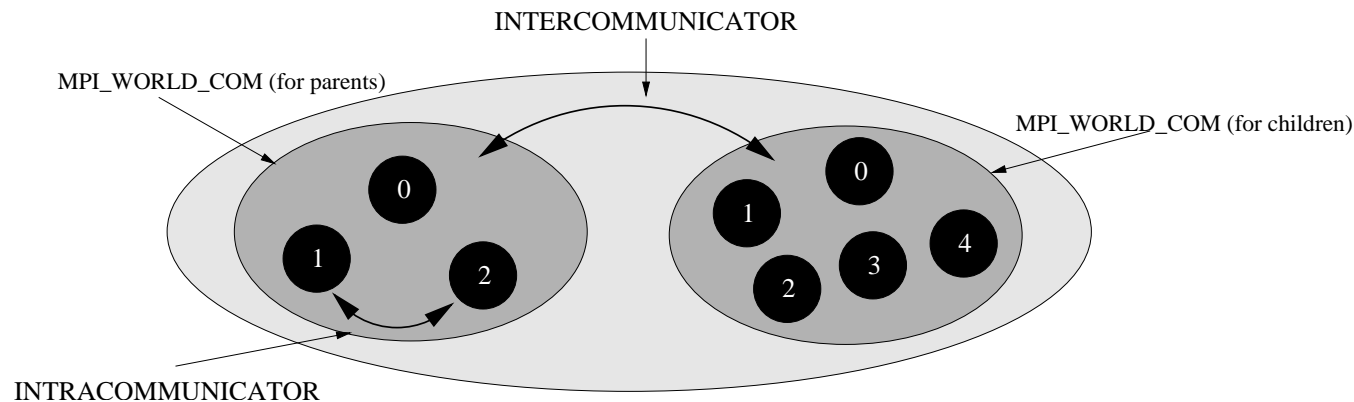
Task Spawning:

```
MPI_Comm_spawn(command, argv, nprocs, info, root,  
                parent_intracomm, intercomm, errcodes)
```

- Collective over parents
- Things to worry about (info parameter): host, architecture, work directory, path etc
- intercomm and errcodes are returned values

13.4 Communicators

- MPI processes identified by (group, rank) pair
- Communicators are either
 - *Intragroup* communicators
 - *Intergroup* communicators
- Parents and children each have their own MPI_COMM_WORLD
- MPI_Send/Recv etc have destination and inter/intracommunicator
- Possible to merge processes or free parents from children(!)
MPI_Intercomm_merge() and MPI_Comm_free()



13.5 One-Sided Communications

In traditional message passing

- One process sends the other receives (cooperative data transfer)
 - An implicit synchronization - although it may be delayed by asynchronous message passing

One-Sided Communication

- Paradigm strongly driven by Cray SHMEM library (T3D/T3E systems), although MPI-2 model is a bit unusual!
- One process specifies all communication parameters
 - Data transfer and synchronization are separate
- Typical operations are put, get, accumulate:
MPI_Put()
MPI_Get()
MPI_Accumulate()

13.6 MPI-2 Remote Memory Access (RMA)

- Processes assign a portion (or window) of their address space that they explicitly expose to RMA operations
- Two types of targets
 - **Active target RMA:** requires all processors that created the window to call `MPI_Win_fence` before any RMA operation is guaranteed to have completed
 - Communication is one sided in that no process is required to post a receive
 - Communication is cooperative in that all processes must synchronize before any of them are guaranteed to have got/put their data
 - **Passive target RMA:** only requirement is that originating process places `MPI_Win_lock` and `MPI_Win_unlock` before and after data transfer
 - Transfer guaranteed to have completed on return from `MPI_Win_unlock`
 - This is what I (and Cray SHMEM) call one-sided communications!
- Potential for one process to get and a second process to put to the same location on a 3rd process give arbitrary results
 - Avoid this using locks or mutexes

13.7 MPI-2 File Operations

Positioning

- Explicit offset
- Shared pointer
- Individual pointers

Synchronization

- Blocking
- Non-blocking (asynchronous)

Coordination

- Collective
- Non-collective

Filetypes

- Filetype is a datatype made up of elementary types (etypes)
- Files can be tiled, such that process rank I writes every N^{th} ($N = \text{size}$) block of the file

13.8 MPI-IO Use

- Every process writes its own data to a separate file
 - This is what we have now, ie just using language specific I/O
- Processes cooperatively write a large matrix to a file
 - Create a filetype to tile the file
 - Use individual pointers
 - Use collective operations to allow data shuffling
- Processes append data to a common file
 - Example would be having a common log file (eg what prun does)
 - No tiling of file
 - Non collective operations
 - Common shared file pointer

13.9 MPI-2 Summary

- MPI-2 added a lot of new functionality
- Implementation on a given machine may be incomplete or incur a performance penalty, eg no dynamic task spawning
- Uptake of new features in MPI-2 has been much much slower than for MPI-1

13.10 Beyond MPI-2

- After many years of nothing, there has been recent activity!
- MPI 2.1 was ratified in Sept 2008, only modest changes
- MPI 2.2 is scheduled to be ratified this year
- MPI 3.0 is a more major upgrade with discussions in progress and intention to release in 2010. Discussion topics include
 - Fault Tolerance/Dynamic Process Control
 - Generalized Requests (and their progress)
 - Improved One-sided Communications
 - Non-Blocking Collectives
 - Hybrid programming
 - Active messages