CMPSCI 691W Parallel and Concurrent Programming **Spring 2006**

Lecture 12: April 3

Lecturer: Emery Berger *Scribe: Alex Epshteyn*

12.1 Message Passing

12.1.1 Why distribute?

- SMPs have lots of memory but just one bus, which is a bottleneck
- whenever the task doesn't fit into each processor's cache, performance degrades

12.1.2 Distributed Memory Computation

- no shared global memory; local memory at each processor
- communication over network
- have to explicitly partition shared data

12.1.3 Message Passing (MP)

Pros:

- can communicate just the minimum of data needed
- each node utilizes its own bus

Cons:

- unnatural to program
- very hard to debug

12.1.4 Portability

- in the past each vendor provided a different MP architecture, which became obsolete in a few years
- code could not be shared between different institutions
- different cluster architectures
  - Supercomputers
  - Communicating SMPs
  - Beowulf clusters - cheap boxes wired togetheron the same network

12.1.5 Message Passing Interface (MPI)

- library approach for platform independence
- bindings for popular languages
- hardware vendors provide own implementation

Drawback:

- performance of the same program can vary greatly on different platforms, since cluster architectures may be optimized for differing kinds of communication

12.1.6 MPI Execution Model

- spawn the same program on each processor
- the program has to find out which node it is to determine what to do

12.1.7 MPI Communication

1. point-to-point: MPI_Send, MPI_Recv
2. broadcast: MPI_Bcast
3. "reflection":
   (a) MPI_Comm_size - returns the number of processors participating
   (b) MPI_Comm_rank - "which processor am I?"
4. misc:
   (a) MPI_Init - initializes communication
   (b) MPI_Finalize - closes communication

Exercise: Write a program which passes a message in a ring from node 0 to 1 to 2 ... to N

Use MPI communication calls:

MPI_Recv(data_ptr, count MPI_INT, prevId, o, MPI_COMM_WORLD);

MPI_Send(data_ptr, count MPI_INT, nextId, o, MPI_COMM_WORLD);