Slide 1

Administrivia

- (None.)

Slide 2

MPI — the Message Passing Interface

- Idea was to come up with a single standard (concepts and library) for message-passing programs, then allow many implementations. Similar to language standards (C, C++, etc.). Good for portability.
- MPI Forum — international consortium — began work in 1992. MPI 1.1 and MPI 2.0 standards defined. Huge! 1.1 specification is 500+ pages.
- Reference implementation — MPICH (Argonne National Lab). Another popular and free implementation (installed here) — LAM/MPI (Local Area Multicomputer).

Slide 3

What's an MPI Program Like?

- "SPMD" (Single Program, Multiple Data) model — many processes, all running the same source code, but each with its own memory space and each with a different ID. Could take different paths through the code depending on ID.
- Source code in C/C++/Fortran, with calls to MPI library functions.
- How programs get started isn't specified by the standard! (for historical/political reasons — some early target platforms were very restrictive, would not have supported what academic-CS types wanted).
- (Compare and contrast all of the above with OpenMP.)

Slide 4

What's in the MPI Library?

- Setup and bookkeeping — initialization, cleanup, environment query, etc.
- Data management — pack/unpack, derived data types.
- Point-to-point communication — several varieties, differing mostly in how much synchronization.
- Collective operations — e.g., broadcast.

Slide 5

MPI "Communicators"

- (One more thing to define before we can write simple code.)
- MPI allows grouping processes; group plus associated context called a "communicator". Makes it easier to write "safe" parallel libraries.
- Predefined communicator `MPI_COMM_WORLD` includes all processes. Programmers can create additional ones.

Slide 6

Simple Examples / Compiling and Executing

- Look at sample program `hello.c`. (All sample programs from class should be on the Web, linked from course "sample programs" page, with short instructions on how to use MPI.)
- We'll use the LAM/MPI that comes with FC5. There should be `man` pages for all commands and functions.
- Compile with `mpicc`.
- Before running, must "boot" (`lamboot` command) — start MPI background processes on all machines to be used.
- Execute with `mpirun`.
- Shut down with `lamhalt`. (Otherwise background processes continues to run.)

Slide 7

Simple (Blocking) Point-to-Point Communication in MPI

- Send with `MPI_Send` — returns as soon as data has been copied to system buffer, buffer in program can be reused.
- Receive with `MPI_Recv` — waits until message has been received.
- Can use "tags" to distinguish between kinds of messages. Can receive selectively or not (`MPI_ANY_TAG`). Received tag is in returned `MPI_Status` variable (e.g., `status.MPI_TAG`).
- Can receive from specific sender or from any sender. (`MPI_ANY_SOURCE`). Sender is in returned `MPI_Status` variable (e.g., `status.MPI_SOURCE`).
- For `MPI_Recv`, "length" parameter specifies buffer length. Use `MPI_Get_count` to get actual count.
- Look at sample program `send-recv.c`.

Slide 8

Not-So-Simple Point-to-Point Communication in MPI

- For not-too-long messages and when readability is more important than performance, `MPI_Send` and `MPI_Recv` are probably fine.
- If messages are long, however, buffering can be a problem, and can even lead to deadlock. Also, sometimes it's nice to be able to overlap computation and communication.
- Therefore, MPI offers several other kinds of send/receive functions — "synchronous" (blocks both sender and receiver until communication can take place), "non-blocking" (doesn't block at all, program must later test/wait for communication to take place).

  (More about these later.)

Slide 9

Collective Communication in MPI

- "Collective communication" operation — one that involves many processes (typically all, or all in MPI "communicator").
- Could implement using point-to-point message passing, but some operations are common enough to be library functions — broadcast (`MPI_Bcast`), "reduction" (`MPI_Reduce`), etc.

Slide 10

Minute Essay

- If you add the following lines to sample program `send-recv.c`, right after the call to `printf()` for process 0

```
buff[0] = 30;
buff[1] = 40;
```

what does process 1 print?

Minute Essay Answer

- The same thing as before — the old data has already been sent to process 1 (or at least copied to a system buffer somewhere), so the change doesn't affect what happens in process 1.