Slide 1

Administrivia

- Reminder: Homework 1 (first part) due Thursday. (How are people doing?)

Slide 2

OpenMP — Overview (Review)

- Parallel programming environment for shared-memory programming, possibly emerging as de facto standard.
- Set of extensions to selected sequential programming languages — compiler directives, library functions.

Slide 3

OpenMP Constructs — Basic Categories

- Parallel regions ("replicate the following in all threads").
- Worksharing ("divide the following among threads").
- Data environment (shared variables versus per-thread variables).
- Synchronization.
- Runtime functions / environment variables.

Slide 4

Library Functions

- `omp_get_num_threads`, `omp_set_num_threads`, `omp_get_thread_num` — as in examples and appendix.
- `omp_get_wtime` — as in examples and appendix.
- Functions to do locking — more about them shortly.
- Functions to do other things — in specification.

Slide 5

Synchronization Constructs

- `critical` — only one thread at a time executes this block of code. (Example — `synch-2.c` on sample programs page.)
- `barrier` — threads wait here until all have arrived. Implicit barrier at end of parallel region.
- `single` — only one thread executes this block.
- Several others — `atomic`, `flush`, `ordered`, `master`. More about them in the specification.

Slide 6

Locks

- `omp_lock_t` — declares a lock variable.
- `omp_init_lock`, `omp_destroy_lock` — create and destroy.
- `omp_set_lock` — acquire lock (wait if necessary).
- `omp_unset_lock` — release lock.
- Other functions described in specification.
- Example — `synch-3.c` on sample programs page.

Slide 7

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.
- Original reference implementation — MPICH (Argonne National Lab). LAM/MPI (Local Area Multicomputer) is another free implementation. Latest / most popular may be OpenMPI (installed here).

Slide 8

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.)

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.

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 11

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 OpenMPI as installed on the F7 lab machines. There should be `man` pages for all commands and functions.
- Compile with `mpicc`.
- Execute with `mpirun`.

Slide 12

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 13

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 14

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.

Minute Essay

Slide 15

- 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

Slide 16

- 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.