Slide 1

Administrivia

- OpenMP compiler installed on publicly available multiprocessor machines (SnowWhite, Dione01, Dione02). Details of use linked from "Sample programs" and "Useful links" pages.

  (It turns out that there are, or have been, several efforts to develop an open-source OpenMP compiler. None quite production-quality yet, at least for FC4 systems. I installed the one that seems to work best.)
- "Useful links" Web page has links to MPI and OpenMP sites. Go there to find complete documentation (standard/specification).
- *Please do not* reboot the machines in HAS 340! People use these remotely, and you may cause someone's program to crash. If you think a reboot is needed, ask a faculty member.

  (If the machines seem to be very slow, odds are it's a background program running. Try another machine.)

Slide 2

Multithreaded Programming with OpenMP — Review

- Basic idea — fork/join programming model, all threads share memory.
- Can duplicate code in all threads (`parallel` directive), split a loop among threads (`parallel for`), have different threads do different things (`parallel sections`).

  More details in specification — can combine these in various ways.

  Various ways to assign loop iterations to threads — later.

Slide 3

Variables in OpenMP

- Most variables are shared by default; exceptions are variables local to a block within a parallel region.
- Some things, though, aren't — variables within a statement block, stack (local) variables in subprograms called from parallel region.
- To give each thread a separate copy — `private` clause. `firstprivate` and `lastprivate` can be used to start/end with shared value.
- To create a partial result in each thread and then combine ("reduce") — `reduction` clause. Operations include sum, product, and/or. No max or min in C/C++.

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 — later.
- 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

Assigning Work to Threads — `schedule` clause

- `static` (with optional chunk size) — divide iterations into fixed-size blocks, distribute evenly among threads.
- `dynamic` (with optional chunk size) — queue of iterations, threads grab blocks of iterations until all done.
- `guided` (with optional chunk size) — like `dynamic`, but with decreasing blocks of iterations.
- `runtime` — get from `OMP_SCHEDULE` environment variable.

Slide 8

Sidebar — Environment Variables (in `bash`)

- To set environment variable `FOO` for the rest of the session:

  ```
  export FOO=fooval
  ```

  (To set every time you log in, put in `.bash_profile`.)
- To run `bar` with a value for `FOO`:

  ```
  FOO=fooval bar
  ```

Slide 9

Numerical Integration, Revisited

- Recall numerical integration program from a couple of classes ago. Let's try parallelizing with OpenMP.
- One approach — use parallel region to create an SPMD program, conceptually identical to MPI program except for details of computing/combining partial sums. Look at code ... (`num-int-par-spmd-1.c` on sample programs page).
- Performance is terrible, though, and there are other problems (see minute essay). Other approaches next time.

Slide 10

Minute Essay

- We ran `num-int-par-spmd-1` on SnowWhite with varying numbers of threads. It always printed a value of 1 for `nthreads`, no matter how many threads were being used.
  - Why?
  - Could this program crash? (It did for me on another machine — segmentation fault — when run with more than one thread.)

Slide 11

Minute Essay Answer

- At the time we call `omp_get_num_threads`, there *is* only one thread. (Only later are there more.)
- Something bad could easily happen if we run the program with more than one thread: We declare the array `partsum` with one element, but with more than one thread we access elements with indices greater than zero.