Slide 1

Administrivia

- (Review slide on makefiles in notes for 1/25.)
- Homework 1 on the Web. Several parts (one for each programming environment). First part due a week from today.
- A request: You will turn in most if not all work for this course by e-mail. Please include the name or number of the course in the subject line of your message, plus something about which assignment it is, to help me get it into the correct folder for grading.

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

Parallel Regions in OpenMP

- `#pragma omp parallel` tells compiler to do following block in all threads (starting team of threads if necessary). Execution doesn’t proceed in main thread until all are done. Example — “hello world” shown earlier.
- Block must be a “structured block” — block with one point of entry (at top) and one point of exit (at bottom). In C/C++, this is a statement or statements enclosed in brackets (with no `goto`s into / out of block).

Slide 5

Worksharing Constructs in OpenMP

- `#pragma omp parallel for` tells compiler to split iterations of following `for` loop among threads. By default, main thread doesn't continue until all are done, but can override that (might be useful if you have two consecutive such loops).
- How loop iterations are mapped onto threads — controlled by `schedule` clause. More about this later.
- To make different threads do different things — `#pragma parallel sections`, etc. (More in standard.)

Slide 6

A Little About Variables in OpenMP

- Most variables are shared by default, including any global variables.
- Some things, though, aren't — variables within a statement block, stack (local) variables in subprograms called from parallel region.
- Can specify that each thread gets its own copy with `private` clause. `firstprivate` and `lastprivate` can be used to start/end with shared value.
- Can specify that each thread gets its own copy, and copies are combined at the end, with `reduction` clause. Operations include sum, product, and/or. No max or min in C/C++.

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

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 9

Example — Numerical Integration

- Compute π by integrating $\int_0^1 \frac{4}{1+x^2}\,dx$.
- Do this numerically by approximating area under curve by many small rectangles, computing their area, adding results.
- Sequential program fairly straightforward. (`num-int-seq.c` on "sample programs" page).
- "Parallelize" how? (`num-int-par.c` on "sample programs" page).

Slide 10

Homework 1 Background

- In Homework 1, you will make a first pass at writing a set of programs (one using OpenMP, one using MPI, and one using Java) to solve the following problem. (We'll talk more about it in class after you've tried it.)
- We talked about computing π using numerical integration. Another interesting (surprising?) approach uses a "Monte Carlo" method:

  Consider a square with sides of length 2 (any unit you like), enclosing a circle of radius 1.

  Approximate the area of the circle by "throwing darts" at the square, counting how many fall within the circle, and calculating the ratio of those within the circle to the total number.

  Model "throwing darts" by using pseudorandom number generator to generate coordinates of a point.

Slide 11

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 12

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.

Minute Essay

- Running the numerical integration example with different numbers of threads gives different results. Why do you think that happens?

Minute Essay Answer

- The order in which the partial results (produced by the iterations of the loop to compute areas of rectangles) are added together depends on the number of threads and the scheduling — and floating-point arithmetic is not associative (!).