Administrivia

Slide 1

- Homework 1 on Web; first part due next Thursday.

OpenMP

Slide 2

- Early work on message-passing programming resulted in many competing programming environments — but eventually, MPI emerged as a standard.

- Similarly, many different programming environments for shared-memory programming, but OpenMP may be emerging as a standard.

- In both cases, idea was to come up with a single standard, then allow many implementations. For MPI, standard defines concepts and library. For OpenMP, standard defines concepts, library, *and compiler directives*.

- First release 1997 (for Fortran, followed in 1998 by version for C/C++).

- Several production-quality commercial compilers available. Up until very recently, free compilers were, um, "research software" or in work. Latest versions of GNU compilers, though, offer support. !!

What's an OpenMP Program Like?

Slide 3

- Fork/join model — "master thread" spawns a "team of threads", which execute in parallel until done, then rejoin main thread. Can do this once in program, or multiple times.

- Source code in C/C++/Fortran, with OpenMP compiler directives (`#pragma` — ignored if compiling with a compiler that doesn't support OpenMP) and (possibly) calls to OpenMP functions.

  Compiler must translate compiler directives into calls to appropriate functions (to start threads, wait for them to finish, etc.)

- A plus — can start with sequential program, add parallelism incrementally — usually by finding most time-consuming loops and splitting them among threads.

- Number of threads controlled by environment variable or from within program.

Simple Example / Compiling and Executing

Slide 4

- Look at simple program — `hello.c` on sample programs page.

- Compile with compiler supporting OpenMP.

- Execute like regular program. Can set environment variable `OMP_NUM_THREADS` to specify number of threads. Default value seems to be one thread per processor.

Slide 5

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 6

How Do Threads Interact?

- With OpenMP, threads share an address space, so they communicate by sharing variables. (Contrast with MPI, to be discussed next, in which processes don't share an address space, so to communicate they must use messages.)

- Sharing variables is more convenient, may seem more natural.

- However, "race conditions" are possible — program's outcome depends on scheduling of threads, often giving wrong results.

  What to do? use synchronization to control access to shared variables. Works, but takes (execution) time, so good performance depends on using it wisely.

OpenMP Constructs — Basic Categories

Slide 7

- 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.

Parallel Regions in OpenMP

Slide 8

- `#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).

Worksharing Constructs in OpenMP

Slide 9

- `#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.)

A Little About Variables in OpenMP

Slide 10

- 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 11

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 12

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.

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 13

Minute Essay

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

Slide 14

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 (!).

Slide 15