Slide 1

Administrivia

- (Reminders about what's due when.)
- Sample solutions for all regular homeworks on Web, or will be soon. (Quick review of my performance results?)
- Project presentations morning of December 14. Scheduled time is 8:30am but we can start at 9:30am. Everything else due the same day at noon.
- My office hours this week — I'm not quite sure. I should be around Wednesday and late Friday; I'll let you know when by e-mail.

Slide 2

More Administrivia

- "What about our grades?" You will get information by e-mail as soon as I have it. (Remember that I did send you grades for Homework 1.)

  (As a general rule, though — if you turned in code you think works, and you got reasonable speedups for Homeworks 2 and 3 and *some* speedup for Homework 4, you're probably doing okay.)

Slide 3

Programming Environments, Revisited

- Choice of environments for book was based on how things were when it was written — MPI fairly dominant for distributed memory and OpenMP for shared memory, with Java not so widely used for parallel programming but more familiar/available.
- All three include more than we had time to cover in class, and have continued to evolve, and then there's a whole new hardware platform (GPUs) . . .

Slide 4

OpenMP Revisited

- OpenMP worksharing constructs define "implicit tasks" (one per thread). We looked only at parallel loops, but there are also "parallel sections", which allow for nesting/recursion.
- OpenMP 3.0 adds support for explicit tasks, which may help with some kinds of problems (irregular and recursive).

Slide 5

MPI Revisited

- Even MPI 1.0 includes far more than we could cover in class — many collective communication operations, communicators, process topologies, and support for user-defined data types in messages.
- MPI 2.0 and later versions add more — e.g., process spawning and one-sided communication.

Slide 6

Java Revisited

- Package `java.util.concurrent` (new with Java 1.5) brought into the standard library a lot of classes previously available as third-party additions — thread pools, locks, various shared-data classes, etc.
- (Curiously enough, though, the need for explicit multithreading in GUIs seems to have declined from early versions of Java, with the notion of the EDT and classes such as `SwingWorker` and timers.)

Slide 7

OpenCL Revisited

- Graphics processors emerging as a new platform for parallel computing — hardware is becoming sophisticated enough to support computation beyond the cards' original purpose, so why not put it to use?
- No consensus yet about programming environments, but OpenCL might emerge as a semi-standard, as MPI and OpenMP did.
- We barely scratched the surface of this environment but perhaps did enough to get past the initial intimidation factor. A brief recap . . .

Slide 8

A Little About GPU Hardware

- Processing hardware typically includes many processors working more or less in lockstep, each able to do pipelined/vector operations — i.e., SIMD, making a comeback!
- Typical hardware also includes a possibly-complex memory hierarchy separate from the memory hierarchy of the "host computer".
- (Look again at performance of heat-equation problem. Performance is dismal with small number of work units, less so with a lot more — though still not exactly good. A quick Web search suggests that more work units mask latency in accessing global memory. "Hm!"?)

Slide 9

A Little About Programming for GPU Hardware

- SIMD hardware makes a data-parallel style of programming a good fit. Not something we really address in our pattern language (yet?), but conceptually similar to *Geometric Decomposition* but more closely synchronized. A.k.a. "stream processing"?
- So, you might express computations as a sequence of whole-array operations, or in terms of applying a "computational kernel" in parallel to many data elements. Whole-array operations included in some programming environments (e.g., Fortran). Current programming environments for GPUs (NVIDIA's CUDA, e.g., and OpenCL) use the computational-kernel idea.
- Currently moving data back and forth between host's memory and GPU's memory must be done explicitly. Actually maybe not a bad idea given that it does take time?

Slide 10

Review of Course

- "CS1 for parallel programming"? We covered:
  - Four languages/libraries — OpenMP, MPI, Java, OpenCL.
  - How to find and exploit concurrency in programs.
- We also did several running examples and some homeworks . . .

Slide 11

Review of Homeworks

- Homeworks 1 and 2 — estimating π with Monte Carlo methods. Basic structure is *Task Parallelism*. Complication is that you need a thread-safe RNG.
- Homework 3 — Conway's game of life. Basic structure is *Geometric Decomposition*. Basic idea easy, details a bit messy (particularly for MPI).
- Homework 4 — quicksort. Basic structure is *Divide and Conquer*. Probably not the best algorithm to parallelize this way, but relatively straightforward.
- For all programs, probably need large problem sizes to get any benefit from multiple UEs. Even then performance may not be amazingly good, but the primary goal is pedagogical rather than practical.

Slide 12

Minute Essay

- None really; sign in.
- And best wishes for a successful end of semester and a good holiday!