Slide 1

Administrivia

- Sample solutions for all regular homeworks on Web.
- Project presentations morning of May 4. Scheduled time is 8:30am but we could start later — 10am? Everything else due the same day at 11:59pm.
- Information about office hours next week coming by e-mail soon.

Slide 2

More Administrivia

- "What about our grades?" You will get information by e-mail as soon as I have it.
- Recall(?) weights from syllabus:
  - 20 points class participation (attendance).
  - 140 points homework.
  - 80 points project.

  (Be advised also that in the past I have sometimes given extra credit for particularly good projects.)

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

- 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.
- Java memory model also cleaned up a bit.
- (Curiously enough, though, the need for explicit multithreading in GUIs seems to have declined, with the notion of the EDT and new classes such as `SwingWorker` and timers.)

Slide 7

"GPGPU"

- 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.
- Some interesting challenges, though . . .

Slide 8

A Little About GPU Hardware

- Processing hardware seems to typically include many processors working more or less in lockstep, each able to do pipelined/vector operations — i.e., SIMD, making a comeback!
- Typical hardware seems to also include a possibly-complex memory hierarchy separate from the memory hierarchy of the "host computer".

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) seem to use the computational-kernel idea.

Slide 10

A Little About Programming for GPU Hardware, Continued

- 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?
- (Short example?)

Slide 11

Review of Course

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

Slide 12

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*.
- 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 13

Sort of for Fun — Performance Results versus Hype

- Fifteen years ago one David Bailey wrote a paper called “Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers”. Somewhat tongue in cheek, but many very valid points.
- Link to original text on course “Useful links” page. Let’s skim . . .
- Points for discussion: Have we been guilty, in this course, of doing any of the things he warns against, or have we been careful to avoid them? What if anything does it mean when your parallel program doesn’t seem to run faster as you increase the number of UEs? (It could mean that “multicore is the wave of the future!” is hype, right? Does it?)

Slide 14

Minute Essay

- How did the course compare with your expectations/goals? Did you learn what you hoped to learn?