Slide 1

Administrivia

- Review sheet, extra-credit problems to be on Web soon. I'll let you know by e-mail. I will try to have all homework graded by Friday, or at least distribute solutions (and have it graded early next week).
- Review session next week?

Slide 2

A Little About Parallel Processing

- Single processors get faster and faster, and yet it always seems like the computation people want to do exceeds what one processor can do. What to do? Multiple processors.
- Multiple processors can help in two ways:
  - Allow multiple applications to run together faster. Current operating systems provide illusion of a virtual CPU per application; multiple actual CPUs can speed this up. No change needed to applications, but o/s must be smarter.
  - Allow single applications to run faster — "parallel processing". Analogous to taking a job too big for one human and assigning it to a team — similar potential benefits, similar tricky parts (how to divide work, how to coordinate).

Slide 3

Hardware for Parallel Processing

- One approach — single shared memory with multiple processors (figure 9.2). Can be easier to program, but doesn't scale well. Usually have one cache per processor, and tricky to keep them consistent. Variation that scales better is "NUMA" (non-uniform memory access) — looks like single shared memory but isn't really.
- Another approach — connect multiple processor/cache/memory units through some kind of network (figure 9.8). Network can be custom-built for high-end "massively parallel computers", off-the-shelf for "clusters". Can be more difficult to program, but scales better.

Slide 4

Programming Models for Parallel Processing — Shared Memory

- Single address space: All processors (potentially) share access to all memory locations.
- In some ways easier to program, but must be careful to synchronize access. Simplest mechanism is "lock", for which hardware usually provides some support.
- Programming languages/libraries: Java, POSIX threads (Pthreads), OpenMP.
- Maps nicely onto shared-memory hardware, but can be simulated without it ("distributed shared memory").

Slide 5

Programming Models for Parallel Processing — Distributed Memory

- Multiple address spaces: Each processor has its own memory, and they communicate by sending each other messages.
- Considered more difficult to program (because variables must be distributed / duplicated), but avoids potential hazards of shared variables. Also allows larger total memory.
- Programming languages/libraries: PVM, MPI.
- Maps nicely onto no-shared-memory hardware, but can run with shared memory too.

Slide 6

Performance Considerations

- How much does parallel processing actually help? "Speedup" usually defined as time using one processor divided by time using N processors. Linear / "perfect" speedup would be — what?
- Actual speedup limited by two things:
  - Inherently sequential parts of program (e.g., I/O — though not always). Amdahl's law gives limit on speedup based on this.
  - Overhead of setting up multiple processes/threads and communicating/synchronizing.
- Notice, however, that less-than-perfect speedup can still be a win, and parallel processing may give access to larger memory or otherwise help.

Slide 7

Minute Essay

- Do you have concerns about the exam (availability of graded homework, solutions, review session, etc., etc.) I can address?