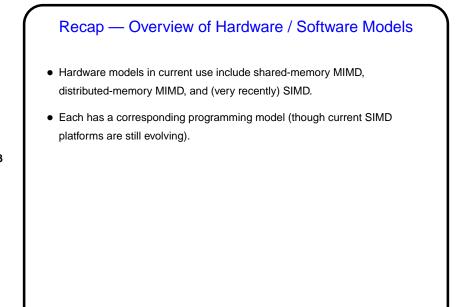


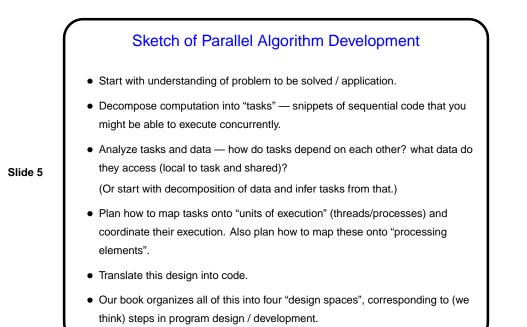
Minute Essay From Last Lecture

- One person asked about hardware and O/S involvement in parallel processing . . .
- Hardware manages lowest level of sharing memory among processors/cores (only one at a time can access a location) and is involved in transferring data and/or routing messages (though not in a way that matters for application programming?).
- O/S is responsible for scheduling processes/threads. Reasonable to assume that it will usually make "good" decisions (i.e., if there are enough processors/cores to run all non-blocked processes/threads, then all will run).



Recap — Parallel Programming Environments

- A regular sequential language with a parallelizing compiler: Attractive, but such compilers are not easy.
- A language designed to support parallel programming (Java, Ada, PCN): Perhaps the most expressive, but more work for programmers and implementers.
- A regular sequential language plus calls to parallel library functions (PVM, MPI, Pthreads): More familiar for users, easier to implement.
- A regular sequential language with some added features (CC++, OpenMP): Also familiar for users, can be difficult to implement.



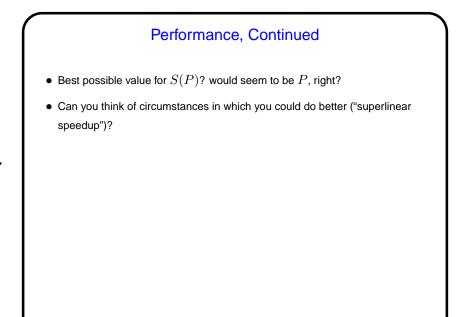
A Few Words About Performance

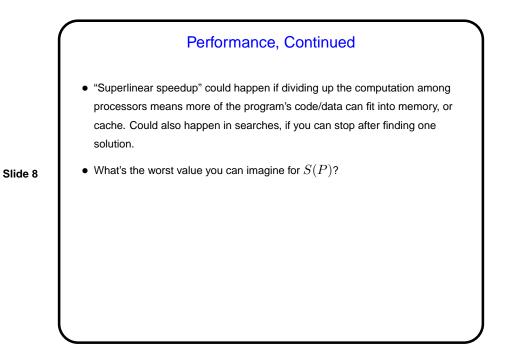
- If the point is to "make the program run faster" can we quantify that?
- Sure. Several ways to do that. One is "speedup" --

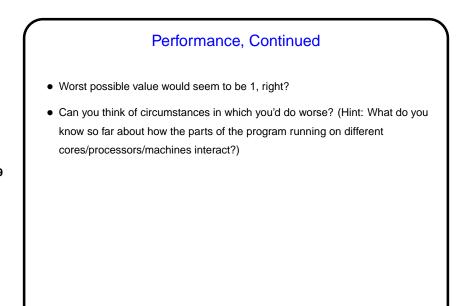
$$S(P) = \frac{T_{total}(1)}{T_{total}(P)}$$

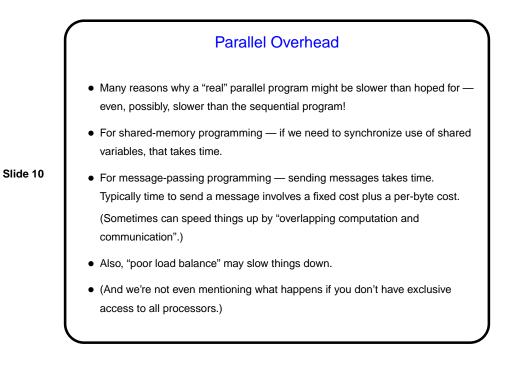
Slide 6

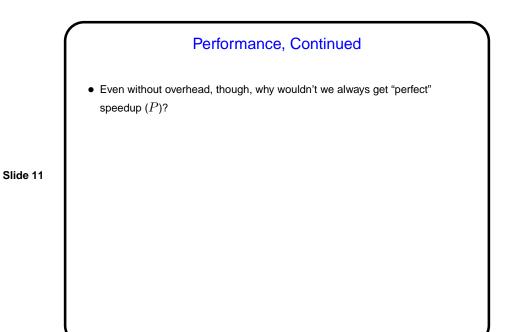
• What's the best possible value you can imagine for S(P)?



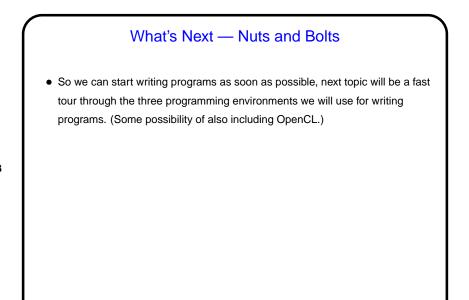




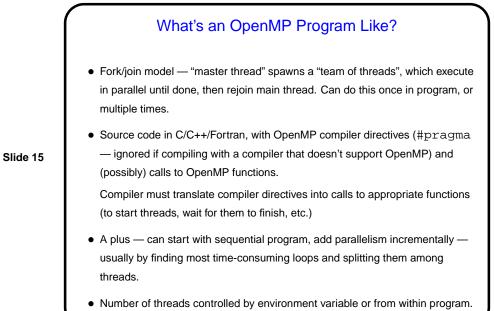




Amdahl's Law • And most "real programs" have some parts that have to be done sequentially. Gene Amdahl (principal architect of early IBM mainframe(s)) argued that this limits speedup — "Amdahl's Law": If γ is the "serial fraction", speedup on P processors is (at best — this ignores overhead) $S(P) = \frac{1}{\gamma + \frac{1-\gamma}{P}}$ and as P increase, this approaches $\frac{1}{\gamma}$ — upper bound on speedup. (Details of math in chapter 2.)



Slide 14 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. Slide 14 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. !!



Simple Example / Compiling and Executing

- 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.



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.) Slide 18 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.

