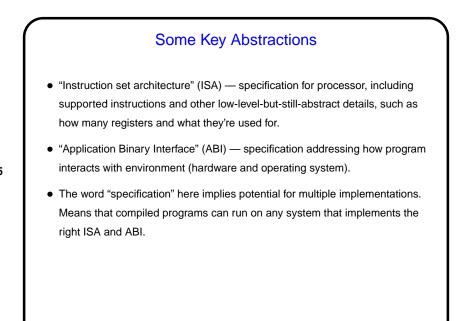


## Running Executable Files — Recap/Review?

- What a processing element can do is fetch machine-language instructions from memory (RAM) and execute them one at a time.
- So to execute a program somehow get machine-language instructions into memory and transfer control to a starting instruction.
- Slide 4
- Several ways to do that, but most typical in general-purpose systems involves operating system that reads contents of "executable file" from storage device.
  Executable file contains machine-language instructions (a.k.a. "object code") and possibly other information (e.g., how much space to reserve for fixed data).
- Programs can be completely self-contained or can contain instructions that request operating-system services (e.g., I/O).



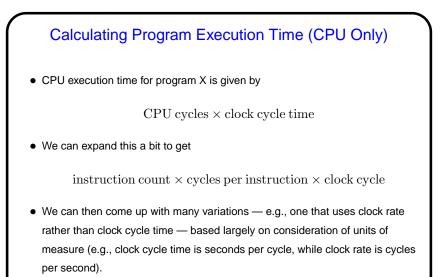
Measuring Performance — Recap/Review

• Many, many factors influence execution time for programs, from choice of algorithm to "processor speed" to system load, as discussed previously.

 Textbook chooses to focus in this chapter on "execution time" by which the authors mean processor time only, excluding delays caused by other factors. Might not be meaningful for comparing systems but seems like reasonable way to compare processors at least.

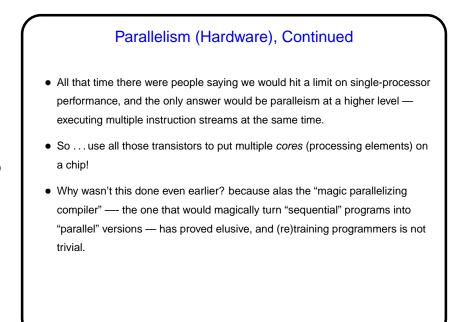
Slide 6

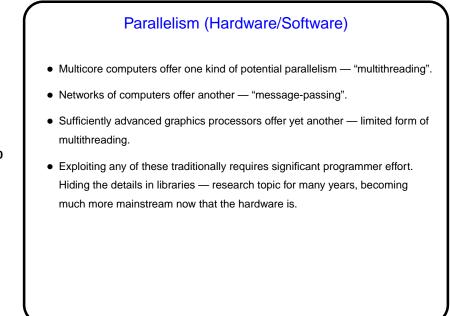
Slide 5



## Parallelism (Hardware)

- Executive-level definition of "parallelism" might be "doing more than one thing at a time". In that sense, it's been used in processors for a very long time, via *pipelining*,and (in high-performance processors) *vector processing*.
- Slide 8
- For a (relatively!) long time, hardware designers were able to make single processors faster using these and other techniques (e.g., reducing sizes of things). In the mid-2000s, however, they ran out of ways to do that. But they could still put larger numbers of transistors on the chip. How to use that to get better performance?





Slide 10



- Parallel-computing version: Can define "speedup" gained by using *P* processors as ratio of execution time using 1 processor to execution time using *P* processors. (So, in a perfect world it would be *P*).
- But 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  $\gamma$  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.

• Textbook points out that this is more broadly applicable!

