Slide 1

Administrivia

- (None?)

Slide 2

Minute Essay From Last Lecture

- (Review . . .)
- Many people got the intended point, which is that you really can't be sure.
- An answer that puzzled me, though — 15 seconds?

Slide 3

Executing Programs — Recap/Review

- Several ways source code can be executed:
- Interpreted directly (e.g., shell scripts).
- Compiled to intermediate form, interpreted/executed by possibly-language-specific runtime system (e.g., Scala and Java).
- Compiled to “native code” (usually producing “executable”) and executed.

Slide 4

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.
- 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).

Slide 5

Some Key Abstractions

- “Instruction set architecture” (ISA) — specification for processor, including supported instructions and other low-level-but-still-abstract details, such as how many registers and what they’re used for.
- “Application Binary Interface” (ABI) — specification addressing how program interacts with environment (hardware and operating system).
- The word “specification” here implies potential for multiple implementations. Means that compiled programs can run on any system that implements the right ISA and ABI.

Slide 6

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 7

Calculating Program Execution Time (CPU Only)

- CPU execution time for program X is given by

$$\text{CPU cycles} \times \text{clock cycle time}$$

- We can expand this a bit to get

$$\text{instruction count} \times \text{cycles per instruction} \times \text{clock cycle}$$

- We can then come up with many variations — e.g., one that uses clock rate rather than clock cycle time — based largely on consideration of units of measure (e.g., clock cycle time is seconds per cycle, while clock rate is cycles per second).

Slide 8

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*.
- 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?

Parallelism (Hardware), Continued

- All that time there were people saying we would hit a limit on single-processor performance, and the only answer would be paralleism at a higher level — executing multiple instruction streams at the same time.
- So . . . use all those transistors to put multiple *cores* (processing elements) on a chip!
- Why wasn't this done even earlier? because alas the "magic parallelizing compiler" —- the one that would magically turn "sequential" programs into "parallel" versions — has proved elusive, and (re)training programmers is not trivial.

Slide 9

Parallelism (Hardware/Software)

- Multicore computers offer one kind of potential parallelism — "multithreading".
- Networks of computers offer another — "message-passing".
- Sufficiently advanced graphics processors offer yet another — limited form of multithreading.
- Exploiting any of these traditionally requires significant programmer effort. Hiding the details in libraries — research topic for many years, becoming much more mainstream now that the hardware is.

Slide 10

Slide 11

One More Thing About Performance — Amdahl's Law

- 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 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.
- Textbook points out that this is more broadly applicable!

Slide 12

Minute Essay

- None — sign in. (Unless questions?)