Administrivia

- Note that for minute essays like the one last week, where there's a "right" answer, my answer will be in the slides online sometime after class.
- Homework 1 on the Web; due in a week. Note requirement for explicit pledge and a brief statement about collaboration.

Slide 1

Minute Essay From Last Lecture

- (Review question and intended answer.)
- Not everyone got the "trick" aspect, but many did.

Slide 2

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., for 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), and how various program components (functions etc.) interact at the machine-code level as opposed to the source-language level (as in API).
- 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

Sidebar: Dimensional Analysis

- (Or at least I think that's close to the term I want.)
- Idea here is to approach "word problems" in terms of units, treating them almost like factors in multiplication and division. (Example is converting, say, inches to cm by multiplying by 1 in the form 2.54cm/1in.)
- If the formula you propose to use produces the right units (e.g., seconds for execution time), there's at least a good chance it's the right one.

Slide 9

Calculating Execution Time, Continued

- One factor in the basic formula is cycles per instruction. What if that isn't the same for all instructions?
- Common sense(?) may tell you ...

Slide 10

Calculating Execution Time, Continued

- If different types of instructions need different numbers of cycles, have to do something like a weighted sum. Usually instructions fall into one of a few "classes", each with a common number of cycles per instruction.
- So, compute times for each "class" of instruction and add. Would also allow you to compute an average CPI.
- Simple example: For a processor with clock rate 2GHz and two classes of instructions taking 1 and 2 CPI, and a program that requires 10^{10} instructions, evenly split between the two classes, how much processor time does it need?

Slide 11

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?

Slide 12

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 13

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 14

Parallelism — Performance

- One use of multithreading is simply to make the code simpler, at least for the programmer — as an example consider the typical GUI-based program, where it makes sense to think in terms of one thread of control for getting user input and one for drawing. Doable on a single processor via interleaving.
- But it *can* also be used to improve performance. Often a discussion of “how much” is in terms of “speedup”.
- Here, “speedup” is defined as some sort of function of the number of processing elements (cores, fully independent processors, etc.), where the speedup for P processing elements is the ratio of execution time using 1 PE to execution time using P PEs.

Slide 15

Parallel Performance, Continued

- While it might seem like with P processing elements you could get a speedup of P, in fact most if not all programs have at least a few parts that have to be executed sequentially. This limits P, and if we can estimate what fraction of the program is sequential we can compute speedups for some values of P.
- Further, typically "parallelizing" programs involves adding some sort of overhead for managing and coordinating more than one stream of control.
- But even ignoring those, as long as any part must remain sequential . . .

Slide 16

One More Thing About Performance — Amdahl's Law

- (Named after Gene Amdahl, a key figure in developing some of IBM's early mainframes who left to start his own company to made hardware "plug-compatible" with IBM's. Interaction between the two companies was — complicated.)
- His observation ("Amdahl's law") can be more generally stated, but in the context of parallel programming it's stated thus:

  If γ is the "serial fraction", speedup on P PEs is (at best, i.e., ignoring overhead)

  $$S(P) = \frac{1}{\gamma + \frac{1-\gamma}{P}}$$

  and as P increase, this approaches $\frac{1}{\gamma}$ — upper bound on speedup.

Slide 17

Preview — "Architecture" as Interface Definition

- From software perspective, "architecture" defines lowest-level building blocks — what operations are possible, what kinds of operands, binary data formats, etc.
- From hardware perspective, "architecture" is a specification — designers must build something that behaves the way the specification says.

Slide 18

Architecture — Key Abstractions

- Memory: Long long list of binary "numbers", encoding all data (including programs), each with "address" and "contents".

  When running a program, program itself is in memory; so is its data.
- Instructions: Primitive operations processor can perform.
- Fetch/execute cycle: What the processor does to execute a program — repeatedly get next instruction (from memory, location defined by "program counter"), increment program counter, execute instruction.
- Registers: Fast-access work space for processor, typically divided into "special-purpose" (e.g., program counter), "general-purpose" (integer and floating-point).

Design Goals for Instruction Set

- From software perspective — expressivity.
- From hardware perspective — good performance, low cost.
- (Yes, these can sometimes be opposing forces!)

Slide 19

Why Study MIPS Architecture?

- Goal is not to become assembly-language programmers, but to understand how things work at this level. Once you understand basic principles, learning another assembly language is easier.
- MIPS architecture is simple but representative.

  Aside: SPIM simulator will let you experiment (commands `spim` and `xspim`).

Slide 20

Minute Essay

Slide 21

- We did a simple example earlier, as follows: For a processor with clock rate 2GHz and two classes of instructions taking 1 and 2 CPI, and a program that requires 10^{10} instructions, evenly split between the two classes, how much processor time does it need?
- How much processor time would the program need if the clock rate increased to 4GHz but all instructions took 3 cycles?
- (And — any other questions?)

Minute Essay Answer

Slide 22

- For the original example, time is

$$\frac{(0.5 \times 10^{10} \times 1) + (0.5 \times 10^{10} \times 2)}{2 \times 10^9}$$

which is 7.5 (seconds).

- For the changed problem, time is

$$\frac{10^{10} \times 3}{4 \times 10^9}$$

which is also 7.5 seconds. (I didn't really plan it that way, but interesting?)