Slide 1

Administrivia

- First homework will be on the Web soon.

Slide 2

Minute Essay From Last Lecture

- People mentioned many things, from obviously electronic gadgets to appliances to cars . . .

Slide 3

A Little About Integrated Circuits

- Conceptual view of hardware:
  - *Transistor* — on/off switch controlled by electrical current.
  - Combine/connect a lot of transistors to get *circuit* that does interesting things (e.g., addition).
  - Put a bunch of circuits together to get a *chip* / *integrated circuit* (IC). If lots of transistors, *VLSI chip*.

Slide 4

A Little About Integrated Circuits, Continued

- Manufacturing process starts with a thin flat piece of silicon, adds metal and other stuff to make wires, insulators, transistors, etc.
- Of course, this is all automated! Low-level chip designers use CAD-type tools, which save designs in a standard format, which the chip designers simulate/test with other software, and then send off to be *fabricated*.
- Typically make many chips on a *wafer*, discard those with defects, bond each good one to something larger with *pins* to allow connections to other parts of computer.

Slide 5

Parallelism

- 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 6

Parallelism, 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 7

Defining Performance

- What does it mean to say that computer A "has better performance than" computer B?
- Really — "it depends". Some answers:
  - Computer A has better response time / smaller execution time.
  - Computer A has higher throughput.
- We'll use execution time, and say

$$\frac{\text{Performance}_A}{\text{Performance}_B} = n$$

exactly when

$$\frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Slide 8

Measuring Performance

- If we use execution time as criterion, how to measure?
- Wall-clock time seems fairest, since it includes
  - Time for CPU to execute instructions.
  - Any waiting for memory access.
  - Any waiting for I/O.
  - Any waiting for operating system.
- Is that easy to measure reliably / repeatably?

Slide 9

Measuring Performance, Continued

- No — to get repeatable measure of wall clock time, need an otherwise unused system.
- So instead we could use “CPU performance” — amount of time CPU needs to run program. Easier to measure, more consistent.
- Or we could try “clock speed”. Can define in terms of “clock period / cycle” or “clock rate” (inverse of clock period).
- Example — for 1GHz processor, what’s its clock cycle?

Slide 10

How These Metrics Relate

- CPU execution time for program X is given by

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

- How would you write that using clock rate instead of clock cycle?
- How would you write it if you know number of instructions and (average) number of cycles per instruction?
- What if you can define different classes of instructions, each with a different number of cycles per instruction?
- So, to double performance for a program, is it enough to double the clock rate?

Slide 11

How These Metrics Relate, Continued

- Not necessarily —
  - Could number of instructions change?
  - Could cycles per instruction change?
- Well, but at least it's better to have fewer instructions?

Slide 12

How These Metrics Relate, Continued

- Also not necessarily — e. g., if you replace instructions that take a few cycles each with a few that take a lot of cycles.

Slide 13

Evaluating / Comparing Performance

- Trickier than it sounds to come up with one number that means something.
- Approaches include
  - Use the actual workload, on the actual hardware platform(s), and compare times.
  - Put together a representative simulated workload — "benchmark"; run and compare times.
  - Compare code size.
  - Compare number of instructions per second ("MIPS" or "MFLOPS").
- Alas, all of these are flawed in some way.

  (Paraphrasing someone whose name I don't remember, "peak MIPS is just the number you can't go any faster than.")

Slide 14

Minute Essay

- None — sign in.