Slide 1

Administrivia

- Reminder: Reading Quiz 1 due Monday. (If you won't have your copy by then, tell me so and I can let you have longer. Not ideal but I want to be reasonable!)
- About subject lines and e-mail to me . . . Yes, I'm picky, but there's method to the madness(?): The program I use for e-mail makes it very easy to pick out ones that match a subject line, and file them away for my semi-automated grading scripts.

  Most people are cooperating, at least partly, and it really has helped!
- I plan to communicate grade information to you fairly regularly. TLEARN does not work well for me for that, so I am setting up Google Drive folders for each of you, and I'll put grade information there. You'll get a "shared with me" about it soon.
- And finally, something I meant to do last time . . .

Slide 2

A Little About Me

- Short version of biography: Undergrad degrees from UT Austin, math and Plan II. More than ten years in what we now call IT. Back to school for master's and PhD in computer science. Two years as a postdoc, then at Trinity since Fall 1999.
- I teach a variety of courses, but currently focusing more on courses "close to the machine". My research area (sadly neglected for some years) is parallel computing.
- (What do I do for fun? well . . .)

Slide 3

Introduction

- "Computers are everywhere" — you know about servers and desktops and smaller computing devices, all of which are more and more central to our lives, but also consider "embedded processors", largely invisible but even more prevalent.

  How far they've come, and how fast — astonishing to those of who grew up in different times.

- It seems to be a truism that however fast computers can process information, they can't keep up with humans' ability to imagine things for them to do. So performance matters. (Sometimes it makes things not just faster but feasible!).

- We'll start with an overview of hardware and software and how they interact (cf. textbook subtitle) and also talk a little about measuring performance.

Slide 4

Overview of Hardware

- Lots of material in this chapter that's interesting (to some) but not vital to the course. No need to understand details! Reading indicates that some sections are "skim/skip". I'll try to at least mention in class things you shouldn't miss.

- Some things won't make a lot of sense at this point — e.g., "great ideas" — but will as the semester goes along.

Slide 5

"Below Your Program" — Review?

- Most programming these days is done in a high-level language (HLL), often using a lot of library code. Processors, however, can't execute it directly. How do we get from HLL to something the processor can do? Traditional view:
- First step is to *compile* — conceptually, to *assembly language* (symbolic representation of instructions the processor can execute).
- Next step is to translate assembly language into *machine language* (actual instructions for processors, in 1s and 0s), a.k.a. *object code*. Might be combined with compiling.

Slide 6

"Below Your Program", Continued

- Final step is to combine object code for your program with library object code. Can be done as part of compiling process to create an *executable file* or at runtime, or some combination of the two.
- Actual execution of program typically involves operating system (something that manages physical resources / provides abstraction for applications). Contents/format of executable files depends on operating system as well as hardware.
- Worth noting that some languages/implementations don't exactly follow this scheme: Some languages (e.g., shell scripts) are translated/interpreted at runtime, and others (e.g., Scala and Java) are compiled to machine language for a virtual processor (the JVM), which may then be translated into "native code" at runtime.

Slide 7

Hardware

- Textbook presents simplified model of hardware in terms of key elements — input/output, processor.
- Right now what's inside the processor box ("control", "datapath") may not make a lot of sense. It will when we start talking about processor design, much later in the course.

Slide 8

Abstraction

- You're presumably familiar with the idea of abstraction from programming courses — e.g., ADT (abstract data type) versus different implementations of it. Point is to "manage complexity", and also to allow reuse and flexibility.
- Same idea applies here too:
  - Instruction set architecture (ISA) and application binary interface (ABI).
  - Implementations of ISA.

Slide 9

Minute Essay

- Do you have a copy of the textbook? Rented or owned? Hard-copy or electronic?