Slide 1

Administrivia

- One purpose of the syllabus is to spell out policies, especially about:
  - Course requirements and grading.
  - Exam or other dates (can only be changed if you all agree). “Please plan accordingly” means “don’t schedule something else for these dates”.
  - Late work.
  - Academic integrity.
- Most other information will be on the Web, either on my home page (office hours) or the course Web page.

Slide 2

More Administrivia

- Part of my job is to answer your questions outside class, so if you need help, please ask! in person or by e-mail or phone.

Slide 3

More Administrivia

- Why are we using “my” book when there are books that are more textbook-like? because (1) I think it emphasizes the right things, which many textbooks don’t, and (2) learning from a not-really-a-textbook and other resources should be good practice for whatever you do after you graduate.

  (I don’t actually think I’m going to be able to retire on the extra royalty income — but it might be enough to finance a trip to Java City for the class?)

  Also — if you spot errors, even typos, please let me know. The first person to report any legitimate error I don’t know about is eligible for extra-credit points.

Slide 4

A Few Words About Computer Use in Class

- Checking your e-mail when you first get here is okay.
- Taking notes online is okay.
- Surfing the Web or playing games during lecture is not okay.
- Remember that I can lock all screens, project what’s on one student’s screen, etc. — and I will if need be. But I’d rather you’d all be responsible enough to resist this distraction!

Slide 5

What is Parallel/Distributed Computing?

- Some computational jobs are just too much for one processor — no way to get them done in reasonable time.
- For jobs done by people, what do you do when the job is too much for one person?

Slide 6

What is Parallel/Distributed Computing?

- For jobs done by people, if too much for one person you assign a team — but you have to figure out
  - How to divide up work among team members.
  - How to coordinate activities of team members.
- Same idea applies to computing — if too much for one processor, use multiple processors. Issues are similar — how to divide up work, how to coordinate.

Slide 7

Simple Examples

- "People job" examples:
  - Digging a hole.
  - Building a house.
  - Baby.

  What do you notice about the last one in particular?
- Computer examples:
  - Adding up a lot of numbers.
  - Computing Fibonacci numbers.

  But these don't seem too "big" . . .

Slide 8

How Much Calculating is "A Lot"?

- Examples from computational biology — how many operations per second are needed to get things done fast enough to be useful?:
  - Sequence the genome — 10^{12} ops/second (500 2-Gigahertz processors).
  - Protein/protein interactions — 10^{14} ops/second (25,000 4-Gigahertz processors).
  - Simulating whole-body response to a drug — 10^{16} operations/second (1,250,000 8-Gigahertz processors).
- (Source — Intel's former life sciences industry manager.)

Slide 9

How Much Calculating is "A Lot"?

- Simplified example — weather simulation:

  Divide earth's surface into 1-square-km cells (about 5×10^8 of them); examine from surface to 14 km out. Gives 7.5×10^9 3D cells.

  Typically need to update least five variables per cell (temperature, humidity, wind (3D), etc.). So, 37.5×10^9 updates.

  To model 24 hours in 1-minute chunks: 86400 minutes. Total of 3.24×10^{15} updates.

  Optimistically assuming 10^9 updates per second, 3.24×10^6 seconds — 900 hours.

- (Adapted from example by Dr. Eggen.)

Slide 10

What Are Some Other Hard Problems?

- Crash simulation / structural analysis.
- Oil exploration.
- Explosion simulations (why Los Alamos is interested).
- Astrophysics simulations (e.g,, Dr. Lewis's work on Saturn's rings).
- Fluid dynamics.
- "Rendering" for computer-generated animation.
- And many others . . .

The Need for Speed

- Solving the same problems faster — reducing wall-clock time.
- Solving bigger problems.
- Solving problems more exactly — to get better answers, need more detail, hence more processing.

Slide 11

Can't You Just Get a Faster Computer?

- Up to a point — yes. Moore's law predicts that processor speed and memory both double about every 1.5 years. Over 30 years, that's a factor of about a million!
- But . . .
  - As you know — however fast processors are, it's never fast enough.
  - Faster is more expensive, and price/performance is not constant.
  - Eventually we'll run into physical limitations on hardware — speed of light limits how fast we can move data along wires (in copper, light moves 9 cm in a nanosecond — one "cycle" for a 1GHz processor), other factors limit how small we can make chips. (We may be there — cf. Intel's announcement last year.)
  - Maybe we can switch to biological computers or quantum computers, but those are pretty big paradigm shifts . . .

Slide 12

Slide 13

“The Answer” — Parallel Computing

- Analogous to “team of people” idea — if one processor isn’t fast enough, use more than one.
- Also useful when there’s something “inherently parallel” about the problem — e.g., operating systems, GUI-based applications, etc.
- `http://www.top500.org` tracks fastest computers; for many years now all have been “massively parallel”.

Slide 14

But I Don’t Want To Solve Problems Like Those!

- What if you aren’t interested in solving problems like these “grand challenge” problems, Is there still a reason to be interested in parallel computing?
- The hardware is there, and it’s becoming mainstream — multicore chips, hyperthreading, etc. (The Intel person says “the chip makers can put more and more transistors on a chip, and this is the best way to use that.”)

  To get best use of it for single applications, will probably need parallelism.
- Also, for some applications, thinking of them as parallel/multithreaded can lead to a solution that lets you do something useful while waiting for I/O, etc.

Slide 15

About the Course

- Can think of this course as the equivalent of PAD I for parallel (and to some extent concurrent and distributed) programming. As with PAD I, many things to learn all at once:
  - A new “box of tools” — or several boxes of tools (different languages/libraries/paradigms). Must learn syntax/functions, plus tools such as compilers and runtime systems.
  - How to use the stuff in the box of tools to solve interesting problems — from low-level “what is this syntax good for?” to algorithm design.
  - How to think about “does it work?”
  - How to think about “how fast is it?”
- Also as with PAD I, the idea will be to teach a mix of technical skills and basic concepts, with emphasis on learning by doing.

Slide 16

Minute Essay

- What are your goals for this course?
- Are you reasonably comfortable with Java and C? How about C++?
- Do you have any experience already with parallel or multithreaded programming? (If so, tell me about it, briefly.)
- Will it be a problem for you if I assign homework that will be hard to do without access to our Linux lab machines?