Slide 1

Administrivia

- (I really do apologize about Monday. Not a good start to the semester, but things happen. I'm trying! "it's complicated" maybe.)
- In the syllabus I say there will be reading quizzes, but — let's drop that.
- With so few students minute essays also don't make that much sense, so let's drop that too.

Slide 2

Turning Work In

- In the past I've asked students to turn in work via e-mail. In principle that works well; in practice not. Also students say they can't remember.
- So this semester I'm going to ask you to use a setup similar to how I communicate grade information to you:

  One Google Drive folder per student, shared only between me and student. "Grades" folder for me to communicate with you; "TurnIn" folder for you to turn work in. Subfolders for each assignment.
- Folders should be set up and shared, though I didn't have Google notify you. Folder names of the form:

  CSCI1120-lastname,_initial
- Do use them; I set things up this way so it fits well with the rest of my course infrastructure.

Slide 3

Course Infrastructure

- Many instructors use some sort of "learning management system" such as TLEARN or Google Classroom.
- I don't — I have my own system, going back to before these were so popular, and partially automated using a collection of programs and scripts. Switching would be possible but painful.

Slide 4

The Pandemic and Trinity

- News from outside continues to be scary. But inside the Trinity bubble, we're safe? Maybe. Latest "Covid by the numbers" e-mail from the adminministration sounds good.
- Only three cases among faculty/staff! But one of those cases is — one of our visiting faculty! Somehow this makes it more real. Hers is a cautionary tale: She visited some family, all of them fully vaccinated except an 11-year-old. The 11-year-old is back in school in person, but had recently tested negative. They felt safe, took off their masks, and . . . all got sick. (The negative test result was misleading.) No one got *very* sick, but.

Slide 5

The Pandemic and Trinity, Continued

- Take-away message (my view!): Trinity insisting on masks in the classroom is more than hygiene theater.
- In fact in general caution is probably still a good idea. You may not be worried for yourself, but consider others, some of whom may be more vulnerable, some of whom are legitimately worried about exposing people they live with, etc.
- My two cents' worth!

Slide 6

Terminology — Parallel Versus Distributed Versus Concurrent

- Key idea in common — more than one thing happening "at the same time". Distinctions among terms (in my opinion) not as important, but:
- "Parallel" connotes processors working more or less in synch. Examples include multiple-processor systems. Analogous to team of people all in the same room/building, working same hours.
- "Distributed" connotes processors in different locations, not necessarily working in synch. Example is `SETI@home` project. Analogous to geographically distributed team of people.
- "Concurrent" includes apparent concurrency. Example is multitasking operating systems. Analogous to one person "multitasking". Can be useful for "hiding latency".

Slide 7

Hardware for Parallel Computing — Overview

- Hardware for sequential computing pretty much all builds on the same model — "von Neumann architecture".
- Hardware for parallel computing is more diverse. Some major categories (using classification scheme proposed by Flynn in 1972(!)):
  - SIMD / vector architectures.
  - MIMD with shared memory.
  - MIMD with distributed memory.
- All of these have a long history, going back to early days of computing (1960-something — see history link on class "useful links" page).

Slide 8

SIMD Architectures

- Basic idea sort of implied by name (Single Instruction, Multiple Data) — many identical arithmetic units all executing the same instruction stream in lockstep (via single control unit), each on its own data. Can have separate memory for each AU or all can share.
- Vector processor — addition(s) to CPU meant to speed up operations on arrays (vectors) by using pipelining and/or multiple AUs. Can be thought of as a special case of (pipelined) SIMD.
- Both used fairly extensively early on and then abandoned, except for special-purpose hardware such as graphics cards. *But* the latter are emerging as a computing platform ("GPGPU"). Here as in other things fashion(?) is cyclical?
- Worth noting that GPGPU typically has its own memory distinct from "host" memory.

Slide 9

MIMD Architectures

- Again, basic idea implied by name (Multiple Instruction, Multiple Data) — many processors, each executing its own stream of instructions on its own data.
- Category is broad enough, and popular enough, to consider two subcategories (shared and distributed memory).

Slide 10

Shared-Memory MIMD Architectures

- Basic idea here — multiple processors, all with access to a common (shared) memory.
- Details of access to shared memory vary — shared bus versus crossbar switch, management of caches, etc. Textbook for CSCI 2321 has (some) details. Access to memory can be "constant-time" (SMP) or can vary (NUMA).
- Attractive from programming point of view, but not as scalable as distributed memory. (Up until recently, not scalable enough to be really attractive.)
- Many, many examples, from early mainframes to dual-processor PCs to multicore chips.
- Conceptually, each processor has access to all memory locations via normal memory-access instructions (e.g., load/store). Convenient, but has some potential drawbacks ("race conditions"). Hardware and/or programming environment must provide "synchronization mechanism(s)".

Slide 11

Distributed-Memory MIMD Architectures

- Basic idea here — multiple processors, each with its own memory, communicating via some sort of interconnect network.
- Details of interconnect network vary — can be custom-built "backplane" or standard network. Various "topologies" possible. Textbook for CSCI 2321 has (some) details.
- Not initially as attractive from a programming point of view, but very scalable.
- Examples include "massively parallel" supercomputers, Beowulf clusters, networks of PCs/workstations, etc.
- Conceptually, each processor has access only to its own memory via normal memory-access instructions (e.g., load/store). Communication between processors is via "message passing" (details depending on type of interconnect network). Not so convenient, but much less potential for race conditions.

Slide 12

"Parallel Hardware is Becoming Mainstream"?

- It's been an article of faith for a long time that eventually we'd hit physical limits on speed of single CPUs, despite interpretation of Moore's law as "CPU speed doubles every 1.5 years."
- But — strictly speaking, Moore's law says that the number of transistors that can be placed on a die doubles every 1.5 years.
- Historically that has meant — more or less — doubling speed and memory size. That seems to be at an end (for now?) — tricks hardware designers use to get more speed require higher power density, generate more heat, etc.
- So, what to do with all those transistors? Provide hardware support for parallelism!

Slide 13

"Parallel Hardware is Becoming Mainstream"?, Continued

- Ubiquity of networking makes almost any PC part of a "cluster", hence suitable for distributed-memory parallel computing.
- Current buzzphrases for hardware that supports shared-memory parallel computing are "multicore chips" and "Hyper Threading" (more in next slides).
- High-end graphics cards are emerging as a platform for "GPGPU" parallel computing, which is in some ways a cross between shared memory and distributed memory.

Slide 14

One Approach — Multicore Chips

- Key idea here — chip includes several "cores", all sharing one connection to memory.
- Each "core" is a CPU in the sense we talk about in Computer Design; each typically has its own first-level cache.
- To fully exploit this for a single application, probably need multiple threads.

Slide 15

Another Approach — Hyper Threading

- Key idea here — chip includes hardware support for having more than one thread at a time "active", but strictly speaking only a single processing core. Replicated components include program counter, ALU.
- What this allows is very fine-grained concurrency ("more than one logical CPU"), which can hide latency. (Note, though, that performance improvements range from zero to about 30 percent.)
- To fully exploit this for a single application, probably need multiple threads.

Slide 16

GPGPU

- GPUs typically contain many identical special-purpose processors that operate more or less in SIMD fashion, and also have their own memory hierarchy.
- Using this platform for general-purpose computing is still somewhat cumbersome, but possible . . .
- (More about this soon.)

Slide 17

Programming Models

- Two broad categories of hardware currently popular (shared-memory MIMD and distributed-memory MIMD). (Really probably three, with GPUs. Later!)
- Analogously, two basic programming models: shared memory and message passing. Obviously shared-memory model works well with shared-memory hardware, etc., but can also do message-passing on shared-memory hardware, or (with more difficulty) emulate shared memory on distributed-memory hardware.
- (It's not clear where GPGPU fits in here. More about it later in the semester.)

Slide 18

One Programming Model: Shared Memory

- Key idea — threads executing concurrently, all sharing one memory.
- Maps well onto hardware platforms for smaller-scale parallel computing, can be implemented on other platforms too (with some work).
- Challenge for programmers is to break up the work, figure out how to get threads to interact *safely* — sharing variables has its pitfalls.
- (How would the "add up a lot of numbers" example work here?)

Slide 19

Another Programming Model: Distributed Memory With Message Passing

- Key idea — processes executing concurrently, each has its own memory, all interaction is via messages.
- Maps well onto most-common hardware platforms for large-scale parallel computing, can be implemented on others too.
- Challenge for programmers is to break up the work, figure out how to get separate processes to interact *by message-passing* — no shared memory.
- (How would the “add up a lot of numbers” example work here?)