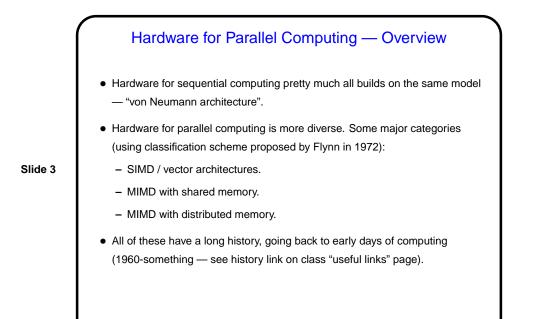
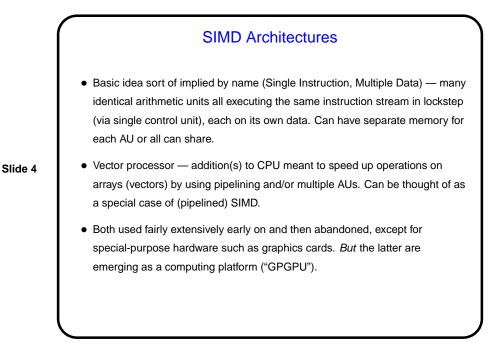


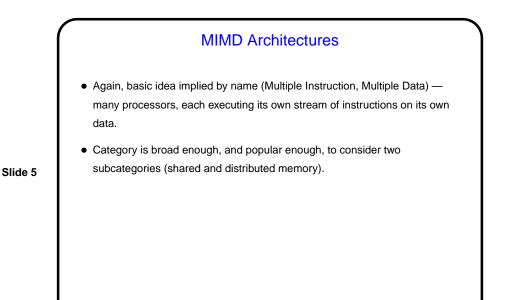
## Terminology — Parallel Versus Distributed Versus Concurrent

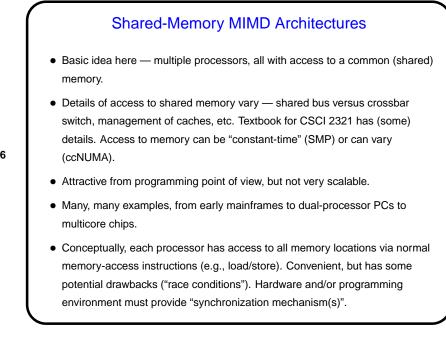
- Key idea in common more than one thing happening "at the same time". Distinctions among terms (in my opinion) not as important, but:
- "Parallel" connotes processors working more or less in synch. Examples include multiple-processor systems. Analogous to team of people all in the same room/building, 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 2











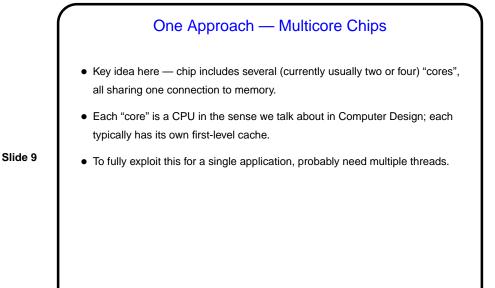
- Basic idea here multiple processors, each with its own memory, communicating via some sort of interconnect network.
- Details of interconnect network vary can be custom-built "backplane" or standard network. Various "topologies" possible. Textbook for CSCI 2321 has (some) details.

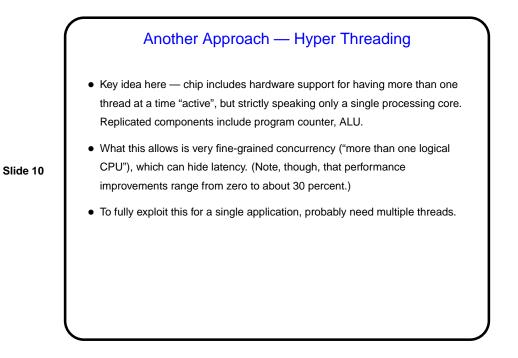
- Not initially as attractive from a programming point of view, but very scalable.
- Examples include "massively parallel" supercomputers, Beowulf clusters, networks of PCs/workstations, etc.
- Conceptually, each processor has access only to its own memory via normal memory-access instructions (e.g., 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.

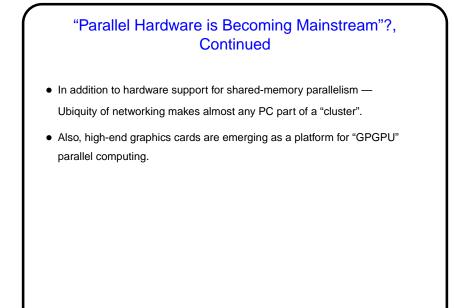


- It's been an article of faith for a long time that eventually we'd hit physical limits on speed of single CPUs, despite interpretation of Moore's law as "CPU speed doubles every 1.5 years."
- But strictly speaking, Moore's law says that the number of transistors that can be placed on a die doubles 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! current buzzphrases are "multicore chip" and "Hyper Threading".

Slide 8

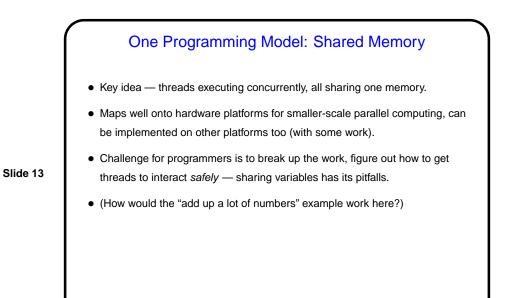






Slide 12

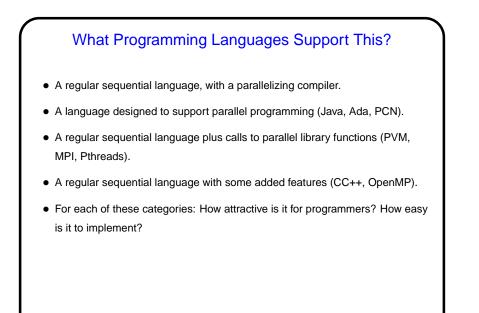
## Programming Models Two broad categories of currently-popular hardware (shared-memory MIMD and distributed-memory MIMD). Analogously, two basic programming models: shared memory and message passing. Obviously shared-memory model works well with shared-memory hardware, etc., but can also do message-passing 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 if possible.)



## 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?)





## What Programming Languages Support This?, Continued

- A regular sequential language with a parallelizing compiler: Attractive, but such compilers are not easy.
- Slide 16
- A language designed to support parallel programming (Java, Ada, PCN): Perhaps the most expressive, but more work for programmers and implementers.
- A regular sequential language plus calls to parallel library functions (PVM, MPI, Pthreads): More familiar for users, easier to implement.
- A regular sequential language with some added features (CC++, OpenMP): Also familiar for users, can be difficult to implement.

