

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".

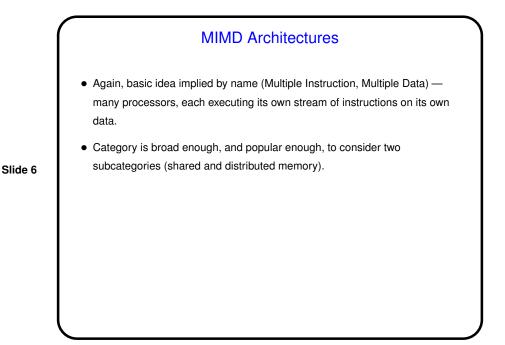
Hardware for Parallel Computing — Overview

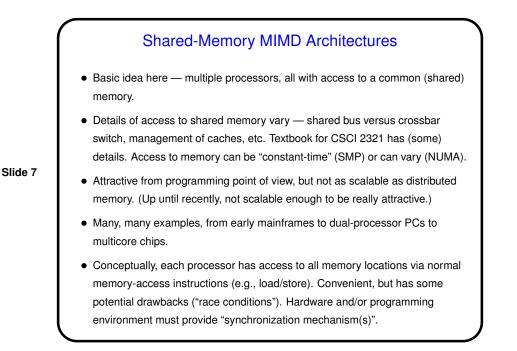
- Hardware for sequential computing pretty much all builds on the same model — "von Neumann architecture".
- Hardware for parallel computing is more diverse. Some major categories (using classification scheme proposed by Flynn in 1972(!)):

- 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).

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.

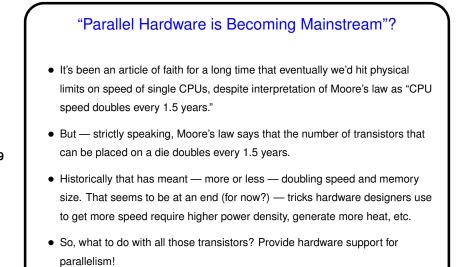




Distributed-Memory MIMD Architectures

- 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.

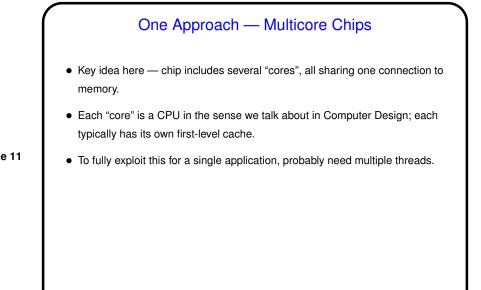
- Not initially as attractive from a programming point of view, but very scalable.
- Examples include "massively parallel" supercomputers, Beowulf clusters, networks of PCs/workstations, etc.
- Conceptually, each processor has access only to its own memory via normal memory-access instructions (e.g.,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

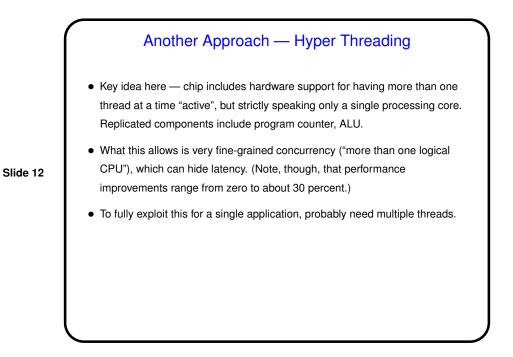


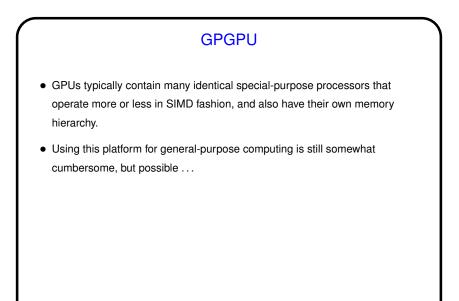
"Parallel Hardware is Becoming Mainstream"?, Continued

- 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.

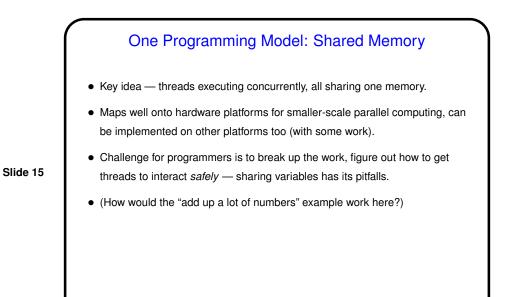
Slide 10







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.)



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?)
- Slide 16

