## Parallel and distributed computing

- What is sequential computing?
- How is parallel computation different?
- Concurrent processing, multitasking, and networks
- Parallel theoretical models
- Memory models
- Languages for parallel programming
- Parallel-prefix computation



## Why parallel computing is worth looking at

- Networked PCs can cooperate today to perform gigantic tasks in parallel cheaply
- Multi-CPU servers are common today to meet the needs of client/server computing
- "Moore's Law", which pictures computer speed doubling every 1.5 years, is losing steam; loophole is lack of real-life performance guarantees
- In a 1-gigaherz CPU, electricity will travel only 1 inch per clock cycle, close to size of chip
- Your brain's billions of neurons work in parallel

# Sequential computing

- The 50-year-old *Von Neumann architecture* defines most computing today:
  - one processor (e.g., Pentium)
  - data is in memory
  - program is in memory
  - enables general-purpose computing

- The microprocessor (central processing unit, CPU) carries out a *fetch/execute cycle*, retrieving and executing machine instructions one at a time



# Today's sequential computing has parallel features

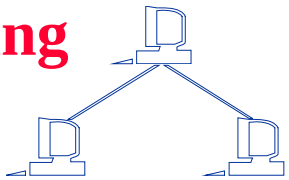
- The Pentium *pipelines* its fetch-execute cycle: as an instruction executes, the processor fetches the next instruction
- The Pentium *looks ahead* and speculatively executes certain instructions
- Multitasking executes different programs *concurrently*
- PCs operate on networks

# Concurrent processing

- A processor may run two or more programs at the same time (multitasking)
- To do so, it saves its current state in one program's fetch/execute cycle, and loads the state of another program's cycle, to run a time slice of that program, numerous times per second
- A *process* is a programming abstraction that simulates ownership of all computer resources



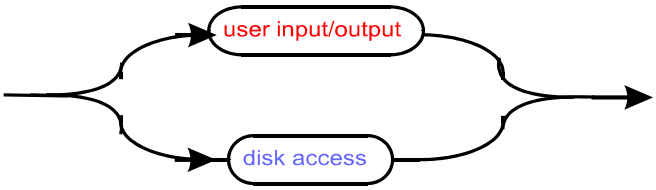
# Networked computing



- Users may communicate with each other through a network, built around one or more servers
- Examples: local area networks; the Internet
- In client/server computing, the server runs a program to fulfill requests from client programs
- Client/server example: a user on a client PC queries a database located on the server via a database management system running on the server

## A *thread* simulates ownership of a CPU

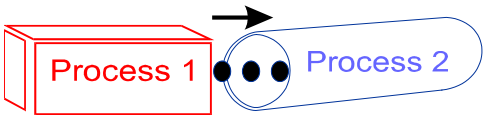
- One program may run multiple threads, e.g., for disk access and user I/O
- The processor executes multiple threads concurrently





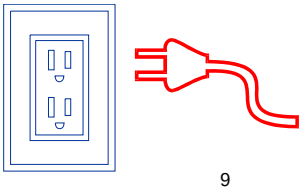

## A *pipe* simulates interprocess communication

- A process that produces data may make results available before it completes
- A second process works immediately on the first data that enters the pipeline
- Familiar example: the conveyer on an assembly line



## *Sockets* are used for inter-CPU communication

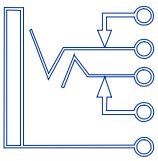
- Sockets, unlike pipes, are a bidirectional abstraction
- The Berkeley sockets standard, introduced with BSD UNIX, permits unrelated processes to communicate
- Processes may communicate via sockets using a protocol





## Theoretical models of parallel computation

- PRAM (Parallel Random-Access Memory): an extension of the Von Neumann architecture to multiple CPUs sharing memory
- Circuit model: Processing is hard-wired into a combinational circuit that executes an algorithm



## Data and instruction parallelism

- SIMD (single-instruction, multiple-data): each CPU applies the same program instruction to its own data
- MIMD (multiple-instruction, multiple-data): each CPU runs its own program
- SIMD lost ground to MIMD because it required specialized hardware; even simple specialized CPUs can't compete with Pentiums on cost

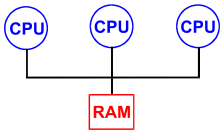


## Great speed gains are possible with parallelism

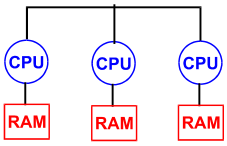
- *The folklore speedup theorem:* with $p$ processors, the maximum speedup, versus one processor, is $p$
- *False!* Counter-examples:
  - if RAM access is slower than network communication
  - if doubling data doubles cache size
  - if intermediate calculations occupy great time resources

# Memory models

- *Shared memory:* assumes uniform memory access time




- *Distributed shared memory* (DSM): assumes data is distributed, so latency (memory-access delay) may be longer if data is owned by a remote CPU than if it is owned by the CPU using it






# Cache coherence

- To make slow RAM access appear fast, a *cache* is used on CPUs today, parallel and sequential
- Using a cache is somewhat like keeping a phone list on your desk to avoid paging through a large phone book
- CPUs that share memory need to keep a common view of RAM

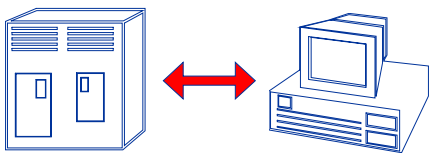
## Languages for parallel computing

- MPI (Message-Passing Interface): A low-level language that is gaining momentum
- TOP-C (Task-Oriented Parallel C): A languaged derived from C for use in distributed-memory environments; supports high-level structures such as arrays, objects
- CILK: For shared memory, developed at MIT

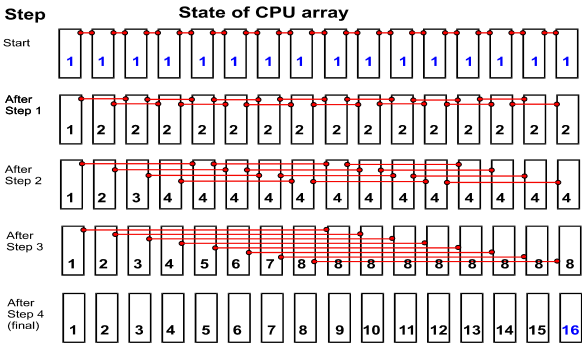


## Goals in parallel-computing software development tools

- *Heterogeneity:* A software tool should be usable on different computer systems
- *Interoperability:* Different systems on which different implementations of a tool were used should be able to take part in the same computation



# Parallel prefix computation




- CPUs (boxes here) exchange data and CPU addresses across wider and wider gaps
- In four steps, a 16-CPU array can apply an associative operator (such as +, here) to 16 data items and accumulate the result (blue) in rightmost CPU.



# References

Lecture notes and drawings by Gene Cooperman, COM 3640 Parallel Algorithms, Northeastern University, Winter 1998.