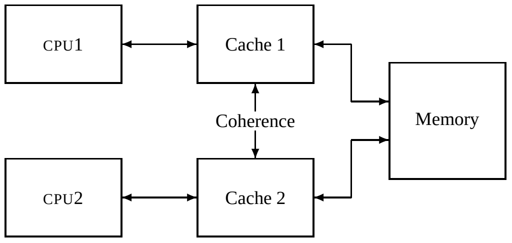
Motivation and History

Introduction

- Speed of computers
  - Faster machines mean faster computation
  - Running a machine ten times faster implies solving bigger problems in less time
  - Before 2002, the performance of microprocessors increased by about 50% per year
  - Since 2002, the single CPU performance improvement has slowed down to about 20% per year
  - Can you wait for machines to get faster to solve your problems?
  - Most of the new performance improvement comes in the form of multiple complete CPUs on a single chip
    - Adding more CPUs will not increase performance for sequential programs
- Problems with increasing clock speed
  - RAM speeds unable to keep up with increased processor speed; processors waste clock cycles waiting for data
  - Limit on the speed for electron flow through wires; delay within chip itself
  - Increasing clock speed also increases power requirements leading to processors generating more heat
- In current processors, clock speeds have settled in the 2–3 GHz range
  - The speed up for non-parallel applications usually occurs due to architectural changes rather than clock speed
- Parallel computing
  - Use of a parallel computer to reduce the time needed to solve a computational problem, specially on a given data set
  - Provides cost-effective means of meeting enormous resource demands of large-scale simulations
    - Run multiple datasets within fixed time
  - Leverages existing resources with relatively inexpensive commodity parts
  - Offers alternatives when individual processor speeds reach limits imposed by fundamental physical laws
  - Better suited to model the activites in everyday life of humans
- Need for ever-increasing performance
  - More complex problems requiring higher accuracy
  - Climate modeling – interaction between different components of atmosphere
  - Protein folding – studying causes for diseases like Parkinson's and Alzheimer's
  - Drug discovery – analyzing genomes of patients for better drug response
  - Energy research – efficient and clean energy sources
  - Data analysis – genomics, particle colliders, astronomy, web search
- Parallel computer
  - A computer with multiple processors, supporting parallel programming
  - Different processors need to share data and communicate with each other
  - Multicomputer
    - Parallel computer with multiple (sub)computers and an interconnection network
    - Processors interact by passing messages
    - No shared memory between processors

- Centralized multiprocessor/Symmetric multiprocessor SMP
  - Highly integrated systems with all CPUs share access to a single global memory, through a common bus
    - Common bus arbitrates between simultaneous memory requests by several processors
    - Ensures that processors are served fairly with minimum access delay
  - Communications and synchronization between processors through the shared memory
  - Major difficulty is the memory contention by the processors; more processors increase the memory contention
    - Memory contention may be reduced by using local cache for each processor
    - Cache contains copies of most recently used memory values
    - Introduced the problem of *cache coherence*



    - CPU 1 may have a copy of memory block that has been since modified by CPU 2, without notification to CPU 1

- Parallel programming
  - Programming in a language to allow you to explicitly indicate how different portions of a computation may be executed concurrently on different processors
- MPI and OpenMP
  - MPI – a standard specification for message passing libraries
  - OpenMP – a better way for SMP environment

Modern Scientific Method

- Classical – Observation, theory, physical experimentation
- Modern – Observation, theory, physical experimentation, numerical simulation
  - Simulation may be necessary for some applications, such as crash testing for cars
- Grand challenge problems, such as global weather and envrionment modeling

Evolution of Supercomputing

- Mostly funded by the U.S. government for security-related applications
- Most powerful computers that can be built
- Cray-1, 1976
  - First machine to be recognized as a supercomputer
  - Built at a cost of $8.86 million, first delivery to National Center for Atmospheric Research
  - Pipelined vector processor; not a multiple-processor
  - Clock speed of 80 MHz

    - More than 100 MFLOPS

- Speed increase in computers comes from
    - Faster clock rate, or more clock cycles per second
    - Greater system concurrency, including instruction optimization
        * More work per cycle
        * Pipelining, branch prediction, executing multiple instructions in the same clock cycle, reordering the instruction stream for out-of-order execution
        * Designed to make instructions flow better and/or execute faster
        * Squeeze maximum work out of each clock cycle by reducing latency
    - Modern computers are about a billion times faster than ENIAC
        * ENIAC performed about 1K arithmetic operations per second
        * Clocks are about a million times faster
        * The remaining thousand-fold increase comes from concurrency
- Today's supercomputers are parallel machines with thousands of CPUs

Modern parallel computers

- Supercomputers are too expensive for most organizations
- VLSI allowed a massive reduction in chip count to construct affordable, reliable parallel systems
- Cosmic Cube, 1981
    - Parallel computer with 64 Intel 8086 Microprocessors, donated by Intel
    - 5–10 MFLOPS; compared to 1 MFLOPS of VAX 11/780
- Commercial parallel computers
    - Connection machine, 1986
        * Built with 1 CPU and 64K ALUs by Thinking Machines
    - Today, most of the leading manufacturers have parallel computers in their product lines
    - High price per CPU compared to commodity PCs due to custom hardware for shared memory or low-latency, high-bandwidth interprocessor communications
- Beowulf, 1994
    - Built from commodity PCs and freely available software
    - 16 Intel CPUs connected by 10 Mbit/sec Ethernet links
    - Linux/Gnu/MPI
    - Extremely low cost
    - Not balanced between compute speed and communications speed
        * Communications quite slow compared to CPU speeds
- Advanced Strategic Computing Initiative (ASCI)
    - Sophisticated numerical computation required to guarantee the safety, reliability, and performance of nuclear stockpile
    - Currently known as Advanced Simulation and Computing Program
    - ASCI Red, 1997

    * 9,000 Intel Pentium II Xeon CPUs
    * First supercomputer to sustain more than 1 teraop on production codes
  - ASCI Blue, 1998
    * 5,856 PowerPC CPUs
    * Performance in excess of 3 teraop
  - ASCI White, 2000
    * Composed of three separate systems
    * 512 node production system; each node an SMP with 16 PowerPC CPUs
    * More than 10 teraops
  - Roadrunner
    * Installed in 2009 at Los Alamos National Labs
    * Based on dual-core Opteron X64 CPUs from AMD and IBM Cell Broadband Engine (Cell BE) processing elements
    * Built as an IBM cluster
    * Current configuration has 122,400 cores
    * Current performance ranks at 1.042 petaflops, with a peak of 1.375 petaflops
    * As of November 2009, it was the second fastest machine on the planet
    * Most of the details are classified

- Current champion
  - K Computer
  - Installed at RIKEN Advanced Institute for Computational Science (AICS), Kobe, Japan in June 2011
  - 68,544 SPARC64 VIIIfx CPUs, each with eight cores, giving a total of 548,352 cores
  - Performance at 8.162 petaflops, with a peak of 8.774 petaflops
  - Running Linux on SPARC64 VIIIfx 2.0GHz, using Tofu Interconnect

Measuring performance

- Performance of parallel systems quantified by the *speedup* with respect to sequential programs
  - Speedup defined as the ratio of sequential execution time to parallel execution time
  - Comparison of speedup relative to upper limit of the potential speedup
    * For an application that scales well, speedup should increase in proportion to the number of cores (threads)
    * If measured speedups fail to keep up, level out, or begin to go down with an increase in the number of threads, the application doesn't scale well on the data sets being measured
- Another measure of performance given by *efficiency*
  - The extent to which software utilizes the computational resources
  - Efficiency quantified by the ratio of observed speedup to the number of cores used
  - Expressed as a percentage
  - A $53X$ speedup on 64 cores equates to an efficiency of 82.8% (53/64 = 0.828)
    * On average, over the course of the execution, each of the cores is idle about 17% of the time
- Expressing speedup
  - Typically expressed using a multiplier
  - Using percentage to express speedup can cause confusion

    - A parallel code is 200% faster than the serial code
        - Does it run in half the time of the serial version or one-third of the time?
    - Is 105% speedup almost the same time as the serial execution or more than twice as fast?
    - Is the baseline serial time 0% speedup or 100% speedup?
  - If the parallel application were reported to have a speedup of 2X, it is clear that it took half the time
    - Parallel version could have executed twice in the same time it took the serial code to execute once

- Speedup in sequential code
  - Maximum speedup in an improved sequential program, where some part was sped up p times is limited by inequality (s denotes the part not improved in speed)

$$S \leq \frac{p}{1 + s \cdot (p-1)}$$

    - Assume that a task has two independent parts, A and B
    - B takes roughly 25% of the time of the whole computation
    - By working very hard, one may be able to make B 5 times faster, giving a speedup due to B as

$$S_B \leq \frac{5}{1 + 0.75 * (5-1)} \approx 1.25$$

    - One may need to perform less work to make part A be twice as fast, giving a speedup due to A as

$$S_A \leq \frac{2}{1 + 0.25 * (2-1)} \approx 1.60$$

  - Try to optimize the part of the code that takes longest time to execute

- Performance expressed in terms of [MGTP]flops (floating point operations per second)

- Machines ranked for their performance using LINPACK benchmark
  - Measures a system's floating point computation power
  - Time to solve an $N \times N$ system of linear equations of the form $Ax = b$
  - Solution based on Gaussian elimination with partial pivoting, with $2/3N^3 + 2N^2$ floating point operations
  - Supercomputers in the TOP500 list ranked by high-performance LINPACK, designed for large-scale distributed memory systems

- Amdahl's Law
  - Used to find the maximum expected or theoretical improvement to an overall system when only a part of the system is improved.
    - Uses percentage of sequential code that can be executed in parallel
    - Speedup limited by the inherently sequential fraction of the code
  - Example
    - Let a program require 20 hours to run on a single core
    - Let there be a part of the program that is sequential and requires 1 hour to execute
    - Let the remaining 19 hours be completely parallelizable
    - Then, no matter the number of cores in the system, it cannot complete in less than one hour
    - The maximum speedup is bound by $20x$
  - Formulation of Amdahl's Law
    - Let the fraction of code that can be executed in parallel be given by p
    - The fraction of code that is executed as sequential is given by $1 - p$

    * For n cores, the speedup S is limited by
      $$S \leq \frac{1}{(1-p)+\frac{p}{n}}$$
    * If there is an infinite number of cores, we have
      $$\begin{aligned} S &\leq \lim_{n\to\infty} \frac{1}{(1-p)+\frac{p}{n}} \\ &\leq \frac{1}{1-p} \end{aligned}$$
    * In practice, the performance to price ratio falls rapidly as n is increased once there is even a small component of $(1-p)$
  - Criticism of Amdahl's Law
    * Ignores real-world overheads such as communication, synchronization, and other thread management
    * Assumption of infinite number of processors/cores
    * Ignores the fact that as the number of cores increases, the amount of data handled is likely to increase as well
      · Works with an assumption of fixed data size, with fixed fraction of sequential and parallel parts in code
      · Leads to *Law of diminishing returns* – As you add more cores to the system, the speedup ratio actually decreases

- Gustafson's Law
  - Also known as *scaled speedup*
  - Tries to account for the fact that the *fraction of serial code execution time* may not be constant as the size of data set changes
    * Real-world data suggests that the serial code execution time stays constant even as the overall data size increases
  - Scaled speedup is given by
    $$S \leq n + (1-n)s$$
    where n is the number of cores, and s is the percentage of serial execution time in the parallel application for a given data set size
  - If 1% of execution time on 32 cores is spent in serial execution, the speedup over the same data set being run on a single core with a single thread is
    $$S \leq 32 + (1-32) * .01 \approx 31.68X$$
  - If the serial execution percentage is 1%, Amdahl's Law gives the speedup as
    $$S \leq \frac{1}{(1-0.99)+\frac{0.99}{32}} \approx 24.43X$$
    * Incorrect because given percentage of serial time is relative to execution on 32 cores

Multicore Programming

- More transistors keep on being added to the same space but we cannot increase their clock speed without overheating
- Recent trend towards *multicore* architectures – *concurrency revolution*
  - Multiples CPUs communicate directly through shared hardware caches
  - The *free-ride* of improved code performance due to increase in clock speed is over
  - Shared-memory multiprocessors or multicores
    * Began as dual core processors

        - Number of cores approximately doubling with each semiconductor generation
        - Intel Core i7
            - Quad core processor
            - Each core is out-of-order, multiple instruction issue processor with full x86 instruction set
            - Supports hyperthreading with two hardware threads
            - Designed to maximize execution speed of sequential programs

- New challenge to exploit parallelism to improve performance
    - Multicores coordinating access to shared memory locations at a small scale
    - CPUs in a supercomputer coordinating routing of data
    - Asynchronous execution of tasks
        - Unpredictable delays without warning by interrupts, preemption, cache misses
        - Cache miss may delay a task by < 10 instructions
        - Page fault may delay the task by a few million instructions
        - OS preemption may delay the task by hundreds of millions of instructions
- Safety property
    - *"Some bad thing never happens."*
        - A traffic light never displays green in all directions, even if the power fails
    - Easy to handle in sequential code
    - Harder to ensure in concurrent processes because the threads can be interleaved in a number of ways
- Some tasks are inherently sequential
    - "Nine women cannot make a baby in one month." – Brook's Law

CUDA

- Makes use of GPU
    - GPU – Display accelerators for hardware-assisted bitmap operations to help with display and usability of graphical OS
    - The popularity of special effects in games drove the market for consumer GPUs like Nvidia GeForce 256 that allowed for transform and lighting computations on graphics processor
    - Allowed for more of graphics pipeline to be implemented directly on GPU
- General purpose parallel computing architecture from NVIDIA
- Design goals
    - Programmable with a small set of extensions to standard C syntax
    - Support heterogeneous computation allowing the applications to use both CPU and GPU
        - CPU and GPU considered as separate devices with isolated memory
        - No contention of memory resources between CPU and GPU
- Hundreds of cores with thousands of computing threads
- On-chip shared memory
    - Allows parallel tasks to share data without using system memory bus
- Effect of branching

  - GPU better for regular work with few branches, that will keep most cores busy
  - CPU better for code with random memory accesses and data-driven instruction flow
- Data movement
  - Performance cannot scale with the number of cores because large portion of time is spent on data movement rather than computation

Seeking concurrency

- Identify operations that may be performed concurrently
- Data dependence graphs
  - Directed graph in which each vertex represents a task to be completed
  - $u \rightarrow v$ implies that task in vertex u must be completed before the task in vertex v is started
    - v is dependent on u
  - If there is no path from u to v, the two tasks are *independent* and may be performed concurrently
- Data parallelism
  - There are independent tasks applying the same operation to different elements of a data set
  - Adding two vectors to create a third vector
- Functional parallelism
  - There are independent tasks applying different operations to different data elements
  - Sequential computation:

$$\begin{array}{rcl} a & \leftarrow & 2 \\ b & \leftarrow & 3 \\ m & \leftarrow & (a+b)/2 \\ s & \leftarrow & (a^2+b^2)/2 \\ v & \leftarrow & s-m^2 \end{array}$$

  - m and s could be computed concurrently
- Pipelining
  - Computation divided into stages can have a degree of concurrency equal to the number of stages
  - More like an assembly line
- Size considerations
  - Parallelism worth doing only if a significant amount of work is given to each stage or processing element; otherwise the overhead of dividing the problem may negate the expenses

Data clustering

- Data mining or scientific data analysis
  - Clustering N text documents into k clusters based on subjects covered
  - Think of Amazon trying to make recommendations to you for purchase based on your past behavior

  - Compute-intensive operation

- Data clustering is the process of organizing a dataset into groups/clusters of similar items
  - Create a cluster of items looked together by many users
  - Search into multiple clusters simultaneously
  - Amazon asks the users to assign categories to different items
  - The categories are used for clustering, with many items in a cluster described by a cluster center

Programming parallel computers

1. Extend a compiler
   - Develop a compiler to detect and exploit parallelism in sequential programs
   - Will benefit the installed base of Fortran code
   - May not work very well due to limited "intelligence" of such compilers; we still have to spend time in manually optimizing code
   - Allow programmer to annotate the sequential program with compiler directives
2. Extend a sequential programming language
   - Allow for functions that allow a programmer to create and terminate parallel processes, synchronize them, and enable them to communicate with each other
   - Easiest, quickest, least expensive, and most popular approach
   - MPI approach
   - Maximum flexibility for programmers in terms of code design
   - Problems with debugging due to lack of compiler support
3. Add a parallel programming layer
   - A lower layer contains all sequential code – the core of computation
     - Abstracted as a set of processes
   - Upper layer to create and synchronize the processes, and partition data among processes
   - The two-layer parallel program translated by a compiler into code suitable for execution on a parallel machine
4. Create a parallel language
   - Programmer can express parallel operations explicitly
   - Languages like Occam and CSP/K
   - Some languages may be designed using the syntax of existing languages
   - No standards for the languages
   - The problem may be user resistance

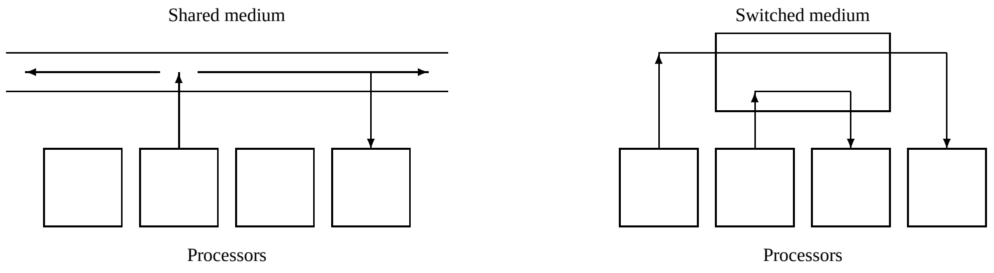
   Current status – leaning on MPI

Parallel Architectures

- Most contemporary parallel computers are constructed out of commodity CPUs

Interconnection networks

- Processors in a multiple processor configuration need to interact
  - Cannot have a direct connection between every pair of processors as that will imply $O(n^2)$ connections
  - It also affects the number of inputs and outputs for each PE
- Approaches include shared memory interconnection and network to pass messages, at a rapid rate
  - Reduce the cost and complexity of network while allowing for rapid communication
- Shared vs switched media



  - Shared medium
    - Allows only one message to be sent at a time
    - Messages broadcast by the processors over the medium
    - Each processor continuously listens and picks up the messages meant for it
    - Ethernet
    - Each processor, before sending the message, listens until the medium is unused
      - If medium is unused, processor sends a message
      - If two processors send messages simultaneously, messages get garbled and must be resent
      - Processors wait for a random amount of time and send the message again
      - Message collisions degrade the performance in shared medium, specially for heavy traffic networks
  - Switched interconnection medium
    - Supports point-to-point messages among pairs of processors
    - Each processor has its own path to the switch
    - Concurrent transmission of multiple messages among different pairs of processors
    - Network can be scaled to accommodate greater number of processors
    - Telephone network
- Switch network topologies
  - Graph with processors as nodes and switches/edges as communications paths
  - Each processor connected to one switch; switches connect processors and other switches
  - Direct topology
    - Same number of processors and switches
    - Each switch node connected to one processor node and one or more other switch nodes
  - Indirect topology
    - More switch nodes than processor nodes
    - Some switches simply connect to other switches
  - Evaluation of switch network topologies to understand effectiveness in implementing efficient parallel algorithms

    Diameter of network
      - Largest distance between two switch nodes

- Lower bound on the complexity of parallel algorithms requiring communication between arbitrary pair of nodes
- Lower diameter is better

Bisection width of network

- Minimum number of edge between switch nodes that must be removed to divide the network into two
- Connectivitiy of the network
- High bisection width is better
  - In algorithms requiring large amount of data movement, the size of data divided by bisection width puts a lower bound on the complexity of parallel algorithm

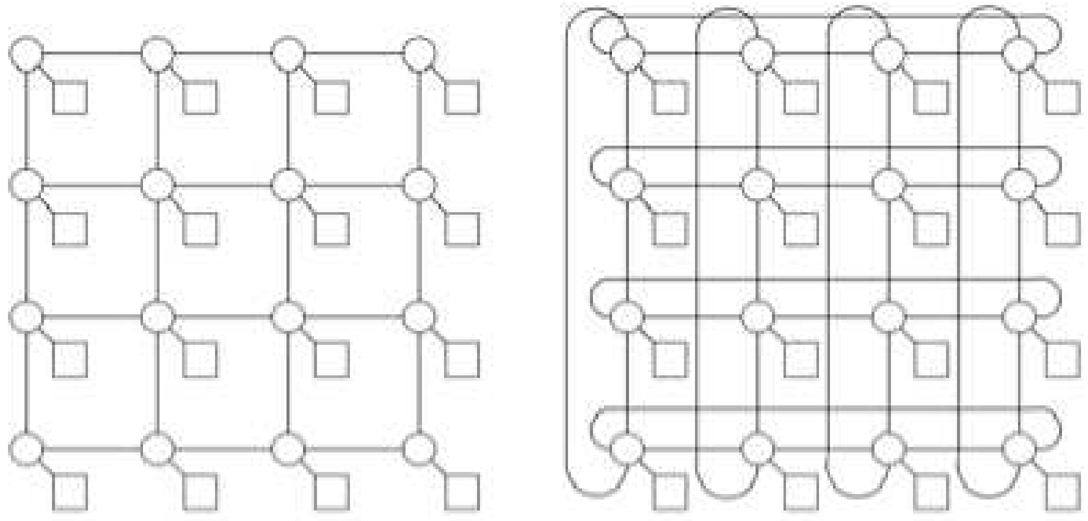
Edges per switch node

- Preferred to be constant, irrespective of network size
- Allows better scaling to systems with larger number of nodes

Edge length

- Preferred to be constant in 3D space
- Distance traveled as the number of PEs increases

- 2D mesh network
  - Direct topology
  - Switches arranged into a 2D lattice
  - Communication allowed only between neighboring switches
    - Interior switches communicate with four other switches



  - Variants allow wraparound connections between switches on edge of mesh
  - Evaluation

    Diameter $\Theta(\sqrt{n})$

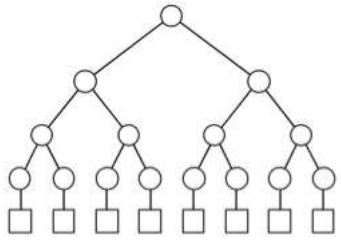
    Bisection width $\Theta(\sqrt{n})$

    Number of edges per switch 4

    Constant edge length Yes

- Binary tree network
  - Indirect topology
  - $n = 2^d$ processors; $2n - 1$ switches

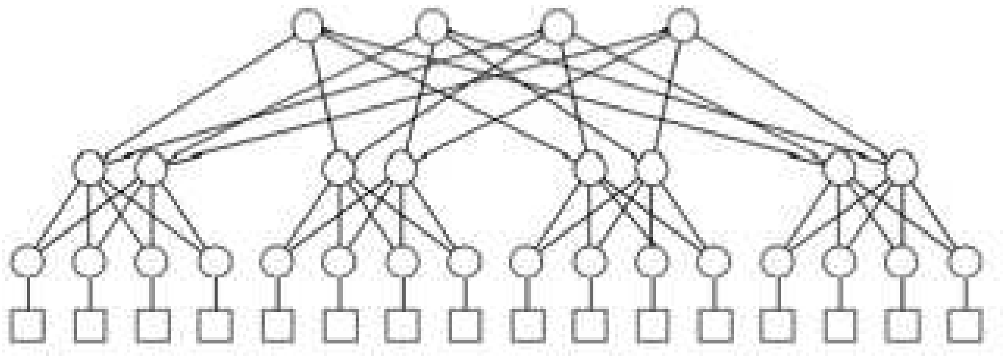
- Each processor connected to a leaf of binary tree
- Interior switch nodes have at most three links; two to children, one to parent



- Evaluation

  Diameter $2\lg n$
  Bisection width 1
  Number of edges per switch 3
  Constant edge length No

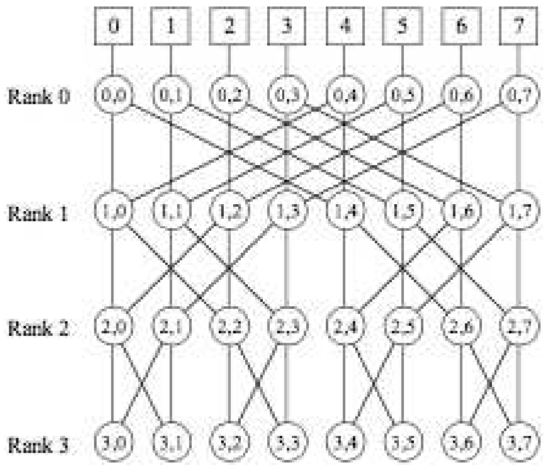
- Hypertree network
  - Indirect topology
  - Low diameter of binary tree but improves bisection width
  - Hypertree network of degree 4 and depth 2



  - Front view: Complete k-ary tree of height d
  - Side view: Upside down binary tree of height d
  - 4-ary hypertree
    * With depth d, can have 4^d processors
    * $2^d(2^{d+1} - 1)$ switching nodes
  - Evaluation of 4-ary hypertree

    Diameter $2d$ or $\lg n$
    Bisection width 2^{d+1} or $n/2$
    Number of edges per node 6
    Constant edge length No

- Butterfly network
  - Indirect topology
  - $n = 2^d$ processors connected by $n(\lg n + 1)$ switch nodes
  - Switch nodes divide into $\lg n + 1$ rows, or *ranks*, each with n nodes



- Ranks labeled 0 through $d = \lg n$
  - Rank 0 and d may be combined
  - Each switch node gets four connections to other switch nodes
- Connections
  - Node (i, j) refers to node j on rank i
  - Each node (i, j) connected to two nodes on rank $i + 1$
    - First connection to node $(i + 1, j)$
    - Second connection to node $(i + 1, k)$ where k is the integer found by inverting the ith most significant bit in the binary representation of j
  - A decrease in rank numbers results in an increase in connection width
    - Length of longest network edge increases with an increase in number of nodes
- Evaluation

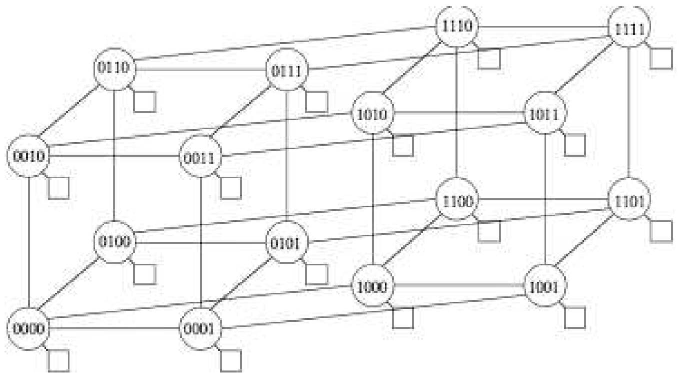
  Diameter $\lg n$

  Bisection width $n/2$

  Number of edges per node 4

  Constant edge length No
- Message routing based on the bit corresponding to $(d - \text{rank})$
  - If bit corresponding to rank is 0, go down left link, otherwise go down right link
  - Go from processor 2 to processor 5

- Hypercube network
  - Direct topology
  - Also called *Binary n-cube network*
  - Buttefly with each column of nodes collapsed into a single node
  - $n = 2^k$ processor nodes and same number of switch nodes
    - Number of nodes a power of 2
  - Processor and associated switch nodes labeled $0, 1, \ldots 2^k - 1$
  - Two switches are adjacent if their labels differ in exactly one bit position
  - Each node connected to k nodes



- Evaluation

  Diameter $\lg n$
  Bisection width $n/2$
  Number of edges per node $\lg n$
  Constant edge length No
- Routing messages
  - Edges always connect switches whose addresses differ in exactly one position
  - Go from 0101 to 0011
    - Difference of two bit positions implies a shortest distance of 2 hops
    - Two possible paths: $0101 \rightarrow 0001 \rightarrow 0011$ and $0101 \rightarrow 0111 \rightarrow 0011$

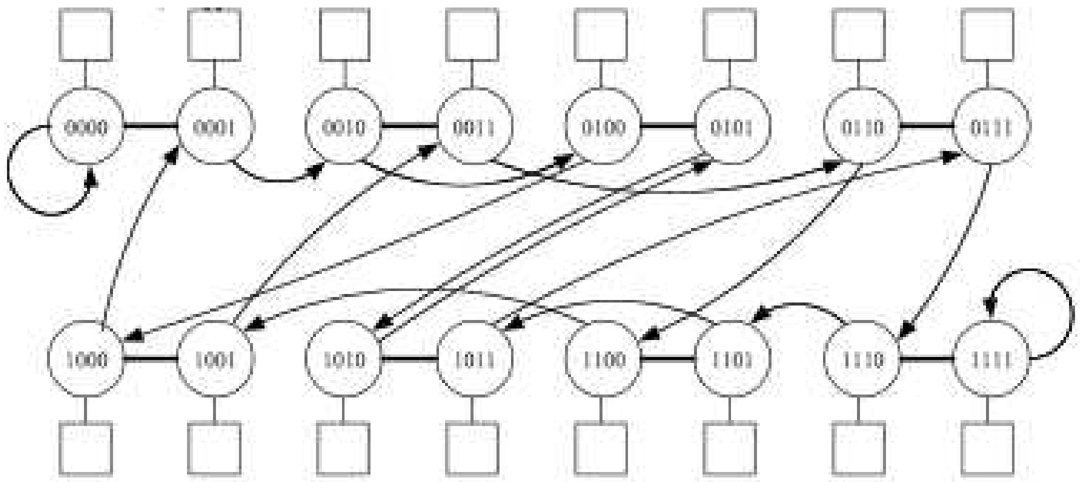
- Shuffle-exchange network
  - Perfect shuffle
    - Take a *sorted* deck of cards
    - Divide it in half and shuffle the two halves perfectly

$$0123|4567 \Rightarrow 04152637$$

    - Achieved by bit rotation as follows for n-bit number i, resulting in j

```
j = ( i << 1 ) | ( i \& ( 1 << ( n - 1 ) ) >> ( n - 1 ) )
```

  - Direct topology, with $n = 2^k$ processor/switch pairs
  - Pairs numbered $0, 1, \ldots, 2^k - 1$
  - Two outgoing links from node i
    1. Shuffle link to node j
    2. Exchange link to node computed by flipping the least significant bit



  - Every switch has constant number of edges: two outgoing and two incoming
  - Evaluation

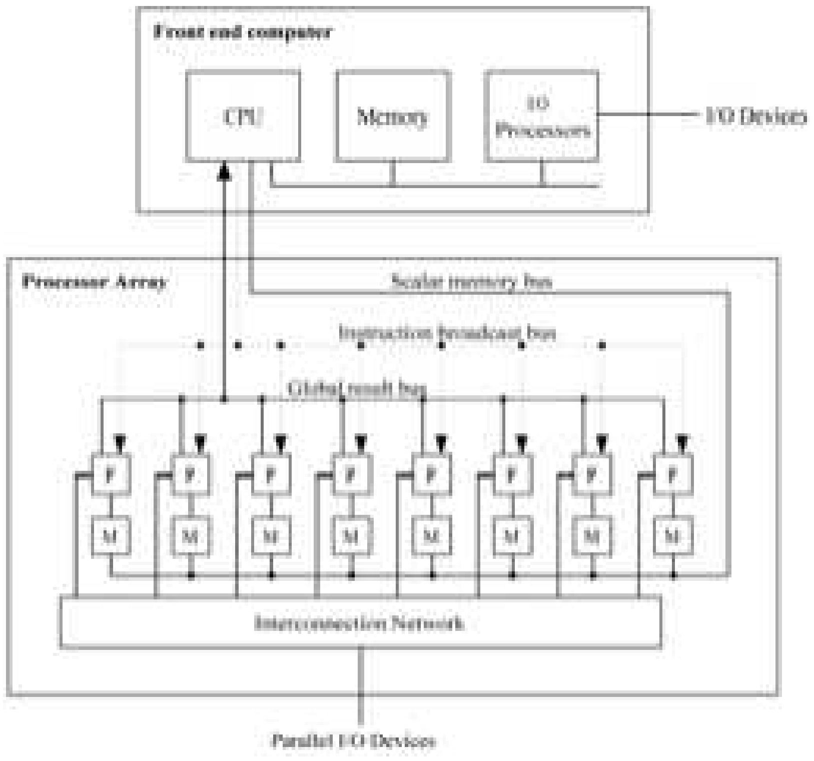
    Diameter $2 \lg n - 1$
    Bisection width $\approx n / \lg n$
    Number of edges per node 2
    Constant edge length No; length of longest edge increases with network size
  - Message routing
    * Path of maximum length follows $\lg n$ exchange links and $\lg n - 1$ shuffle links
    * Worst case scenario routing message from switch 0 to switch $n - 1$
      · For 16 nodes, routing requires $2 \lg n - 1 = 7$ steps

$$0000 \xrightarrow{E} 0001 \xrightarrow{S} 0010 \xrightarrow{E} 0011 \xrightarrow{S} 0110 \xrightarrow{E} 0111 \xrightarrow{S} 1110 \xrightarrow{E} 1111$$

- Comparing networks
  - All have logarithmic diameter except 2D-mesh
  - Hypertree, butterfly, and hypercube has bisection width $n/2$
  - All have constant edges per node except hypercube
  - Only 2D mesh keeps edge length constant as network size increases

Processor arrays

- Vector computer
  - Instruction set includes operations on vectors as well as scalars
  - Pipelined vector processor
    * Streams data through pipelined arithmetic units
    * Cray-1 and CDC Cyber 205
    * Not of much interest in this class
  - Processor array
    * Many identical, synchronized arithmetic PEs connected to a sequential computer
      · All PEs perform the same operation on different data
    * Justified by high cost of control unit
    * Data parallel computations
- Architecture and data-parallel operations
  - Collection of simple PEs controlled by a front-end computer



- Front-end computer
  - Standard uniprocessor
  - Data manipulated sequentially in its own memory
- Processor array
  - Individual processor-memory pairs
  - Data manipulated in parallel
  - Instruction transmitted by front-end to PEs after distributing data into PE memories
- Configuration of 1024 PEs labeled $p_0, p_1, \ldots, p_{1023}$
  - Two 1024-element vectors A and B distributed among PEs such that p_i contains a_i and b_i
  - Vector addition $A + B$ performed in a single instruction
    - Each PE fetches its own pair of values and performs addition concurrently with other PEs
  - Time to perform addition on two vectors of length ≤ 1024 is the same regardless of the number of elements in the vectors
  - Distributing 10,000 element vector on the 1024-PE machine
    - Give 784 PEs 10 elements each and 240 PEs 9 elements each
    - Mapping may or may not need to be managed by programmer

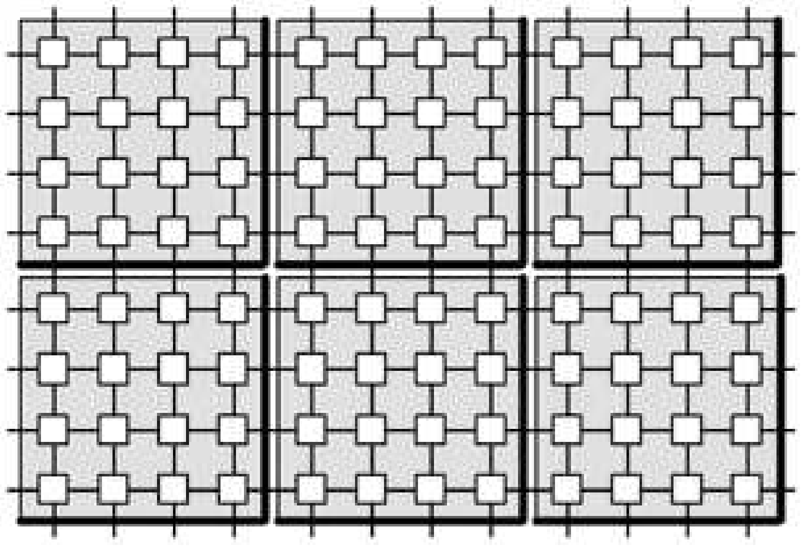
- Processor array performance
  - Measured by work done per unit time, or operations per second
  - Depends on utilization of PEs and size of data structures being manipulated
    - Performance is highest when all the PEs are active and size of data structures is a multiple of the number of PEs
  - Example 1
    - 1024 PEs
    - Each PE adds a pair of integers in 1 μs
    - Adding two 1024-elements integer vectors

        · Assume each vector is allocated to PEs in a balanced fashion
      * Performance = $\frac{1024\text{ops}}{1\mu\text{s}} = 1.024 \times 10^9$ ops/sec
  – Example 2
      * 512 PEs
      * Each adds a pair of integers in 1 μs
      * Adding two 600-element integer vectors
        · Assume each vector is allocated to PEs in a balanced fashion
        · Since $600 > 512$, 88 PEs must add two pairs of integers
      * Performance = $\frac{600\text{ops}}{\lceil 600/512 \rceil \mu\text{s}} = 3 \times 10^8$ ops/sec

- Processor interconnection network
  – Value of a vector or matrix element may be a function of other elements
      * Implementation of a finite difference method to solve a partial differential equation

$$a_i \leftarrow (a_{i-1} + a_{i+1})/2$$

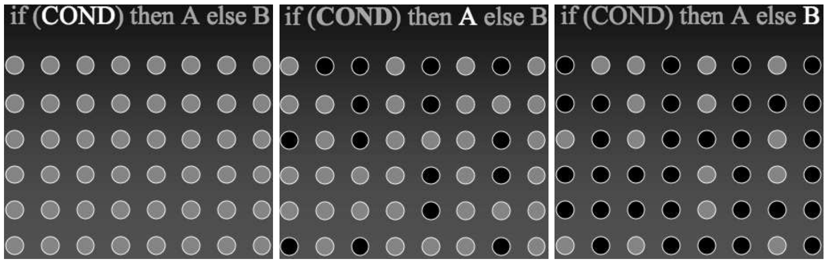
  – Operands in the memories of different PEs pass through interconnection network
  – 2D-mesh
      * Most popular interconnection network
      * Relatively straightforward implementation in VLSI
        · A single chip may contain a large number of PEs
  – Supports concurrent message passing
      * Each PE may simultaneously send a value to the PE to its north
  – 8×12 PE array with 2D mesh interconnection



      * Minimize wire length
      * Single VLSI chip with 16 PEs arranged in a 4×4 mesh
      * Figure does not show connections between PEs and the front-end computer

- Enabling and disabling processors
  – All individual PEs work in lockstep; are synchronized
  – Possible to make only a subset of PEs execute an instruction

- Masking bit
  - *Opt-out* for each PE
  - Useful when number of data items not an exact multiple of the number of PEs
  - Convert all nonzero values to 1 and all zero values to -1



- Conditional execution may decrease performance
  - Overhead of testing mask bits
  - Switching mask bits

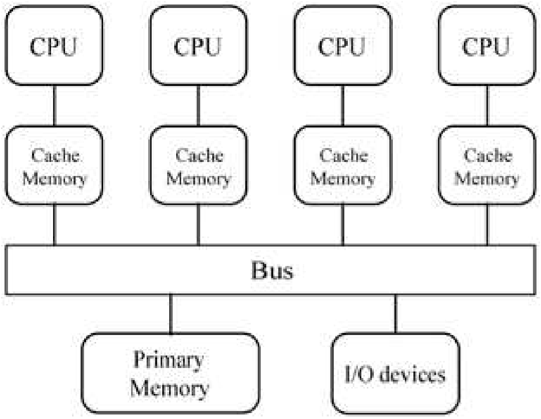
- Shortcomings of processor arrays
  - Not all problems are data-parallel
  - Speed drop for conditionally executed code
  - Don't adapt to multiple users well
  - Do not scale down well to "starter" systems
    - Cost of high bandwidth network does not change by much as the number of PEs changes drastically
  - Rely on custom VLSI for PEs
  - Expense of control units has dropped

Multiprocessors

- Multiprocessor
  - Multiple CPU computer with shared memory
  - Same address on two different CPUs refers to the same location
  - Avoid three problems of processor arrays
    1. Can be built from commodity CPUs
    2. Naturally support multiple users
    3. Maintain efficiency while executing conditional code

- Centralized multiprocessors
  - Typical uniprocessor
    - Bus to connect a CPU with primary memory and I/O processors
    - Cache memory to improve performance
  - Centralized multiprocessor as an extension of above architecture
    - Additional CPUs attached to the bus
    - All processors share same primary memory; same access time for each processor
    - Also called uniform memory access (UMA) multiprocessor or SMP
      - All memory is in one place and has same access time for every processor
      - Large efficient instruction and data caches reduce the load a single processor puts on the memory bus and memory

        - Number of processors limited by memory bus bandwidth



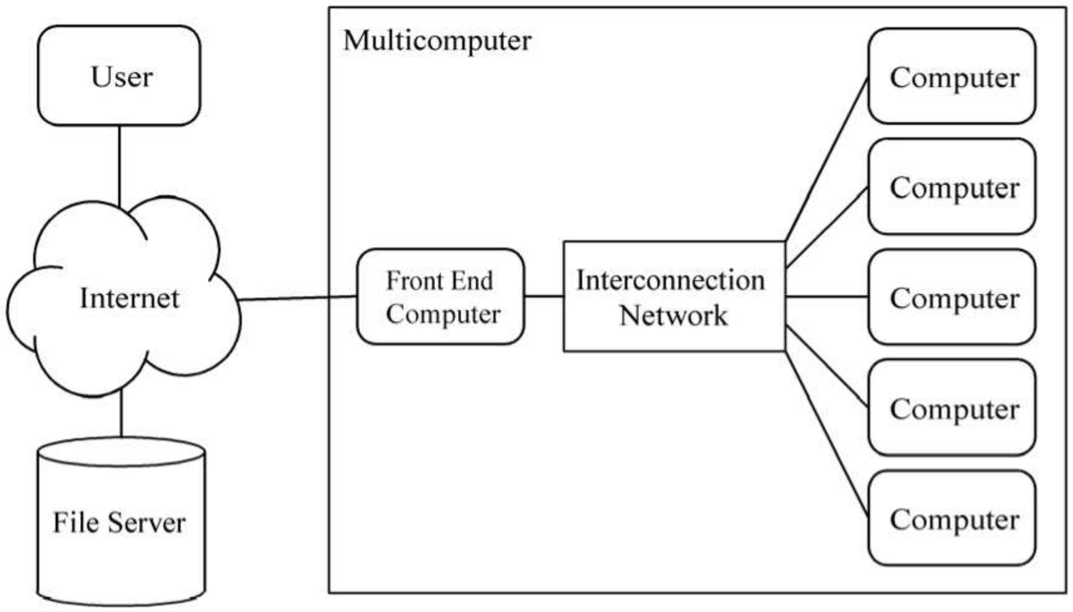
    - Private and shared data
        - Data items that may be local to a processor's memory, or visible globally
        - Processors communicate through shared data items
            - Example: linked list; a processor may advance a shared pointer after reading the address of the node to be processes
        - Leads to two probelms: cache coherence and synchronization
    - Cache coherence
        - Replicating data across multiple caches reduces contention for shared-data values
            - CPU A and B read the same memory values in their cache
            - CPU B writes a new value in its location, also updating the shared memory location in the process
            - CPU A still has old value in its cache
        - Ensure that different processors have same value for the same address
        - Typically solved by *snooping protocols*
            - Each CPU's cache controller snoops the bus to identify the cache blocks being requested by other CPUs
            - Give a process exclusive cache access to a data item before writing its value
        - Write invalidate protocol
            - Before the write occurs, all copies of the data item cached by other PEs are invalidated
            - Processor performs write, updating its cache and appropriate memory
            - When another PE attempts to read its cache block, it gets a cache miss forcing the retrieval of updated value from memory
        - Two PEs trying to write to the same memory location simultaneously
            - Leads to a race condition
            - Cache block of "losing" PE is invalidated
            - Losing PE must get a new updated copy of data before it can do its write
    - Synchronization
        - Mutual exclusion
        - Barrier synchronization
            - No process will proceed beyond a certain point in the program, until every process has reached the barrier
        - Software and hardware mechanisms to synchronize

- Distributed multiprocessors
    - Also known as non-uniform memory access (NUMA) multiprocessor
    - Distribute primary memory among processors

    - Make local memory accesses much faster than nonlocal accesses
  - Assume spatial and temporal locality in memory references
  - Higher aggregate memory bandwidth
  - Lower memory access time, allowing for higher processor count
  - Distributed collection of memories forms one logical address space
    - Same address on different processors refers to the same physical memory location
    - Non Uniform Memory Access
      - NUMA refers to variation in memory access time dependent on local or global memory reference for the PE
  - Cache coherence
    - Some NUMA multiprocessors, such as Cray T3D, do not support it in hardware
    - Only instructions and private data are allowed in cache
    - Large variation in memory access time
      - Nonlocal memory access takes 150 cycles in Cray T3D vs only 2 cycles for cache memory access
    - Implementation is more difficult
      - No shared memory bus to snoop
      - Directory-based protocol needed
  - Directory-based protocol
    - Single directory contains sharing information about cacheable memory blocks
    - One directory entry per cache block containing
      - Sharing status (uncached/shared/exclusive)
      - Which processors have copies
    - Sharing status
      - Uncached – Block not in any processor's cache
      - Shared – Read only block cached by one or more processors; copy in memory is correct
      - Exclusive – Cached by exactly one processor who has written into it; copy in memory is obsolete
  - Directory may be distributed among processors' memories to avoid access performance bottleneck
    - Contents are not replicated
    - Information about a given memory block is in exactly one location
  - Directory-based protocol example

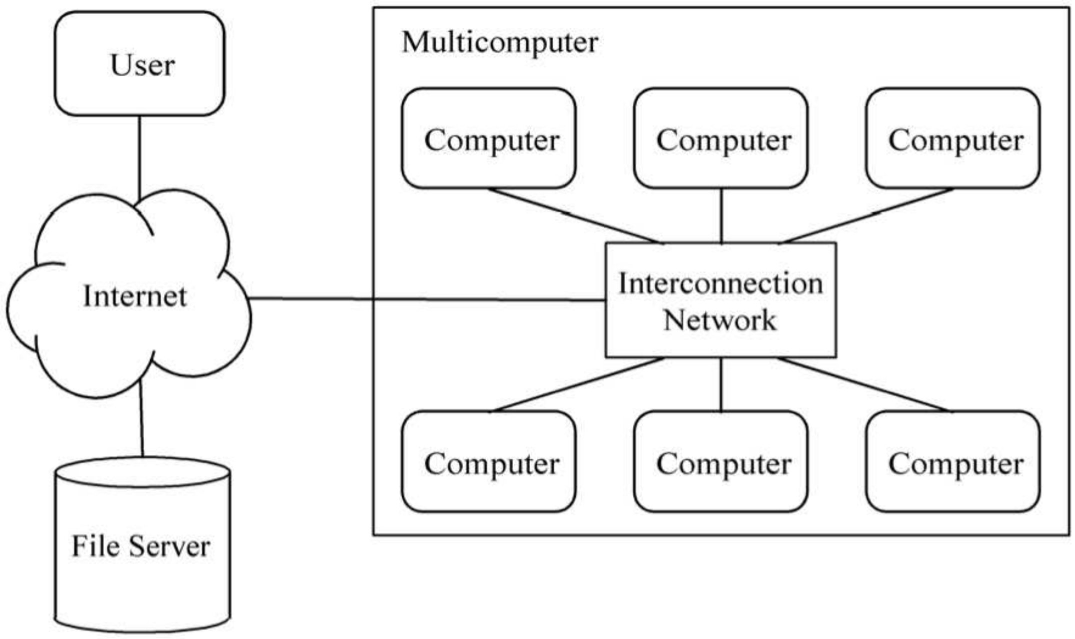
Multicomputers

- Distributed memory multiple CPU computer
  - Same address on different CPUs refers to different physical memory locations
  - Processor interaction through message passing
  - Commercial multicomputers
    - Custom switching network for low latency, high bandwidth access between processors
    - Good balance between speeds of processors and communications network
  - Commodity clusters
    - Mass-produced computers, switches, and other equipment for local area networks
    - Less expensive with higher message latency and lower communication bandwidth
- Asymmetrical multicomputers



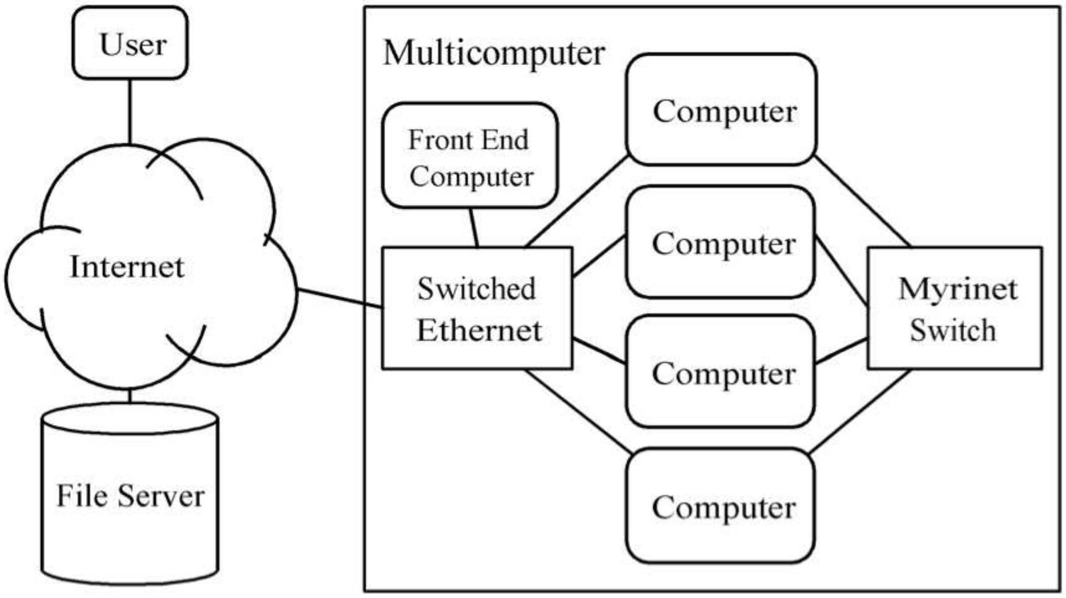
  - Asymmetrical design
  - Front-end computer for interaction with users and I/O devices
  - Back-end processors for number crunching/parallel computation
  - Easier to understand, model, tune performance
  - Only a simple back-end OS needed
    * Primitive OS in back-end machines may not support virtual memory, I/O, or multiprogramming
    * No other processes to take CPU cycles or send messages across network
  - Disadvantages of asymmetrical design
    * Single point of failure in front-end computer
    * Limited scalability due to single front-end computer
    * Primitive OS in back-end makes debugging difficult
    * Every application requires the development of both front-end and back-end code

- Symmetrical multicomputer



- Every computer executes same OS and has identical functionality
- Alleviate performance bottleneck caused by single front-end computer
- Better support for debugging
- Every processor executes same program
- Disadvantages
  - Difficult to maintain illusion of a single parallel computer
  - No simple way to balance program development workload among processors
  - Difficult to achieve high performance whene processes compete with other processes for CPU and other resources

- Model for commodity cluster
  - Put a full-fledged OS on each processor and give every computer access to file system
  - Mixed model with attributes of both asymmetrical and symmetrical designs
  - ParPar cluster



- Differences between clusters and networks of workstations
  - Network of workstations
    - Dispersed computers
    - First priority: person at keyboard
    - Parallel jobs run in background
    - Different OS
    - Different local images
    - Checkpointing and restarting important
  - Commodity clusters
    - Co-located computers
    - Dedicated to running parallel jobs
    - No keyboards or displays
    - Identical OS
    - Identical local disk image
    - Administered as an entity

Flynn's Taxonomy

- Best known classification scheme for parallel computers
- Based on instruction stream and data stream (single/multiple)
- SISD – Single Instruction Single Data
  - Single CPU systems
  - Concurrency of processing (pipelining) vs concurrency of execution
  - PCs up to 2005
- SIMD – Single Instruction Multiple Data
  - Pipelined vector processor (Cray 1)

  - Processor array (Connection Machine)
- MISD – Multiple Instructions Single Data
  - Systolic array
  - Primitive PEs that “pump” data
- MIMD – Multiple Instructions Multiple Data
  - Multiprocessors and multicomputers