Interconnect Technology and Computational Speed*

*From Chapter 1 of B. Wilkinson et al., "PARALLEL PROGRAMMING. Techniques and Applications Using Networked Workstations and Parallel Computers", augmented by Chapter 12 of "High Performance Computing"

Interconnect Technology

Strong similarity between high performance computing and high performance networking.

Both share many technologies.

Parallel processors can be thought of a variation on "network of workstations." They just have faster interconnection.

Processors need to communicate with one another to solve problems.

Need some form of interconnect.

Interconnect: Network connecting all parts of parallel system.

Interconnect Technology Cont.

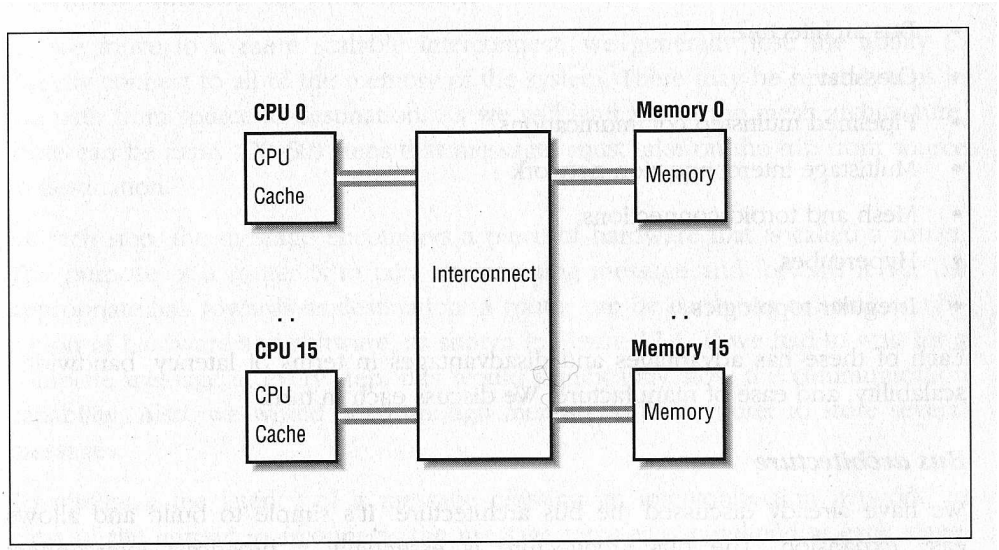

Figure 12-2: Connecting processors to memory

*

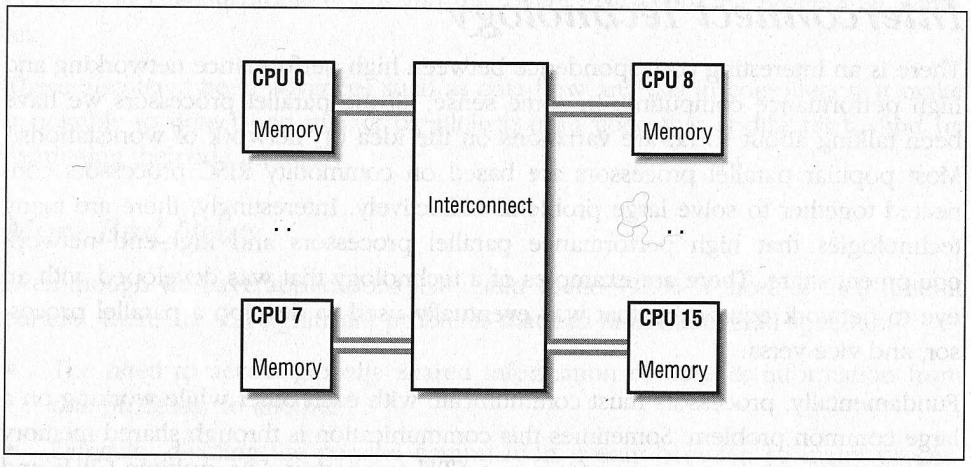

Figure 12-3: Connecting nodes to one another

*K. Dowd and C. Severance, *High Performance Computing, 2nd Ed.*, O'reilly, 1998.

Bus Architecture

Easy to build and expand.

Every component connected to bus.

Can watch every operation on bus. Cache coherency easy to implement.

Limitation: Only 1 CPU can use bus at time.

Doesn't scale well.

Crossbar

Like a network switch.

Offers multiple paths between processors and memory.

Offers overall better data throughput.

Scales better than bus-based approach.

Expensive to build, and costly as number of CPUs increases.

Doesn't scale to 1000 of processors.

Related: *Completely connected network.* For n nodes, there is a link to any other node.

Requires $n(n-1)/2$ links. Only practical for small n.

Pipelined Multistep Communication

More scalable means no longer able to directly connect to each node.

May be several steps in path.

At each step message encounters a router.

Takes incoming message and forward it on appropriate link towards destination.

If had to wait for entire message at each stop, would be slow. Also, need lots of memory at each router.

Minimize latency by pipelining data.

Dynamically establishes a temporary link through network.

See Figure 12-4.

Pipelined Multistep Communication Eg.

Figure 12-4 on following slide shows an example of a multistep communication network.

Only a small part of the network is shown, a source and destination, and three routers.

Each input of a router is hardwired to a compute node (first row of routers), or another router (2nd and 3rd row).

The output of last row of router are hardwired to compute nodes. All links are bidirectional.

Each router has four output ports, labelled 0 to 3, starting from the top. By selecting a specific output port on a router, you are selecting the router/node that is connected to that port.

Pipelined Multistep Comm. Eg. II

For example, selecting output port 2 of router B, means the message should be sent to input port 3 of router C (see Figure 12-4).

When the source node sends a message to the destination node, it starts transmitting data to router A.

The first two bits of the message tell router A which port to transmit the message to. in our example, the first two bits would be: $a_1a_0 = 01$ (output port 1 - the least significant bit of a number is sent first, just like in the parallel-access shift register from your digital logic course).

Router strips bits a_1a_0 from the message and starts transmitting the rest of the message to router B via its output port 1.

Pipelined Multistep Comm. Eg. III

The next two bits of the message tell router B which port to send the message to. In our example, the next two bits would be $b_1 b_0 = 10$. (output port two, thus router C).

Router B strips off bits $b_1 b_0$, and starts transmitting the rest of the message via its output port 2.

The next two bits of the message tell router C which port to send the message to. In our example, the next two bits would be $c_1 c_0 = 11$. (output port three, thus to destination node).

Pipelined Multistep Comm. Eg. IV

Router C strips off bits $c_1 c_0$, and starts transmitting the rest of the message via its output port 3. It thus starts transmitting the main data part of message to destination.

After the time to set up the links (bit c_1 arrived at router C), the remaining message time is the time to transmit the message over the last link due to the pipeline effect.

Pipelined Multistep Communication II

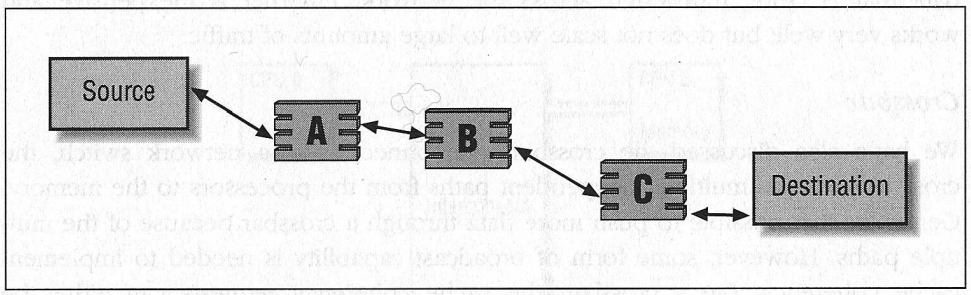

Figure 12-4: Pipelined multistage interconnection

*

*K. Dowd and C. Severance, *High Performance Computing, 2nd Ed.*, O'reilly, 1998.

Multistage Interconnection Network

A multistage interconnection network (MIN) is a network of small crossbars.

Scales far better than a bus or crossbar.

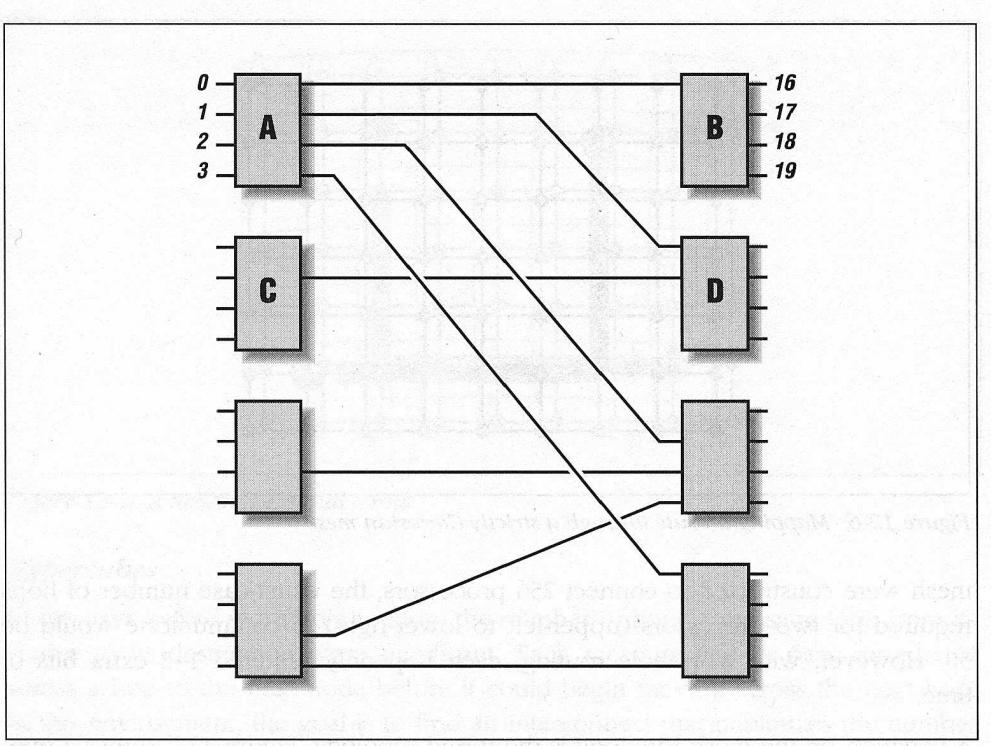

Figure 12-5: Multistage interconnection network *

*K. Dowd and C. Severance, *High Performance Computing, 2nd Ed.*, O'reilly, 1998.

Multistage Interconnection Network Cont.

A 4x4 crossbar requires 1 step to communicate. The MIN in Fig 12-5 requires 2.

Can minimize this by pipelining memory access.

Static Networks

Common form for message-passing multicomputer.

Have direct fixed physical links between nodes. An example would be a mesh network that we will look at soon.

A crossbar interconnect is not a static network.

Links may be bidirectional or 2 uni-directional links.

May be transmitted serially or in parallel.

Network Criteria

Network Bandwidth: Number of bits that can be transmitted in unit time (bits/sec).

Network Latency: Time to make a message transfer through network.

Communication Latency: Total time to send the message. Includes software overhead and interface delays.

Message Latency: This is the total time to send a zero length message. In other words, just the software/hardware overhead in sending a message. Also called startup time.

Network Criteria Cont.

Diameter: Minimum number of links between the two furthest apart nodes.

Used to determine worst case delay.

bisection Width: The number of links that must be cut to divide network in half.

Can also provide lower bound for messages in parallel algorithm.

Bisection Width Eg.

Assume network is 3×3 mesh (see diagram on board in class).

Assume each link is serial and bi-directional.

For our example, we have n numbers, thus we have $n/2$ numbers on each side of network (assuming uniformally distributed).

We need to move each set of numbers to the other half of the network.

For our network, the bisection width, w, is four.

This determines the maximum number of paths that can be used to exchange information from one side of network to the other.

Bisection Width Eg. Cont.

Let $n = 16$. This means we have $n/2 = 8$ numbers on each side of the network.

To move one half of the numbers to the other half of network will require $\frac{n/2}{w}$ messages phases.

As links are bidirectional, can only transmit in one direction at a time.

To swap both sets of numbers, will thus require $\frac{n/2}{w} * 2 = \frac{n}{w}$ message phases.

Substituting for n and w, we get $\frac{n}{w} = \frac{16}{4} = 4$.

For our example, we find that it would take 4 messages phases to swap the two sets of numbers.

Line/Ring

A line is a row of nodes, with connections limited to adjacent nodes.

Formed into a ring structure by connecting the two opposite ends.

Applicable to certain types of computations.

n node ring requires n links. Diameter, $n-1$ if links unidirectional.

Diameter is $\lfloor n/2 \rfloor$ if bidirectional.

When not completely connected, a network needs a routing algorithm.

Mesh Networks

Two dimensional mesh created by connecting node to its 4 nearest neighbors. See Figure 12-6.

Assuming $\sqrt{n}$ (n number of nodes) is an integer, then we have a $\sqrt{n} \times \sqrt{n}$ mesh.

Diameter is $2(\sqrt{n} - 1)$

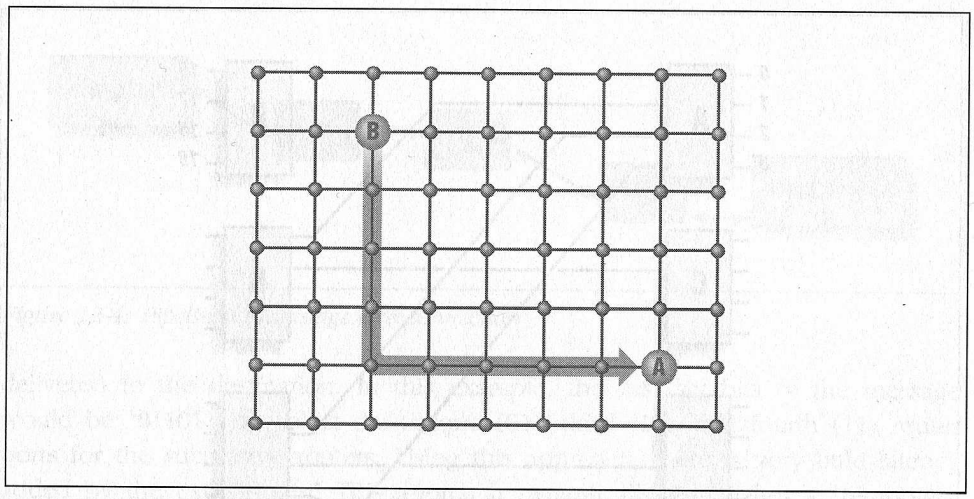

Figure 12-6: Mapping a route through a strictly Cartesian mesh *

*K. Dowd and C. Severance, *High Performance Computing, 2nd Ed.*, O'reilly, 1998.

Mesh Networks Cont.

To move data through mesh, map Cartesian (x,y) route between two points.

Like a MIN, first bits of message used to route. Bits stripped off, and rest of message sent on.

Visualize as a worm passing though mesh. Where head goes, data trails after it.

This technique called *wormhole routing.*

Torus Networks

Variation of Mesh network.

The unused ends of the mesh are wrapped around to the opposite end.

Reduces worst case number of hops by 50%.

Each node in an $\sqrt{n} \times \sqrt{n}$ torus has four links, with $2n$ links in all.

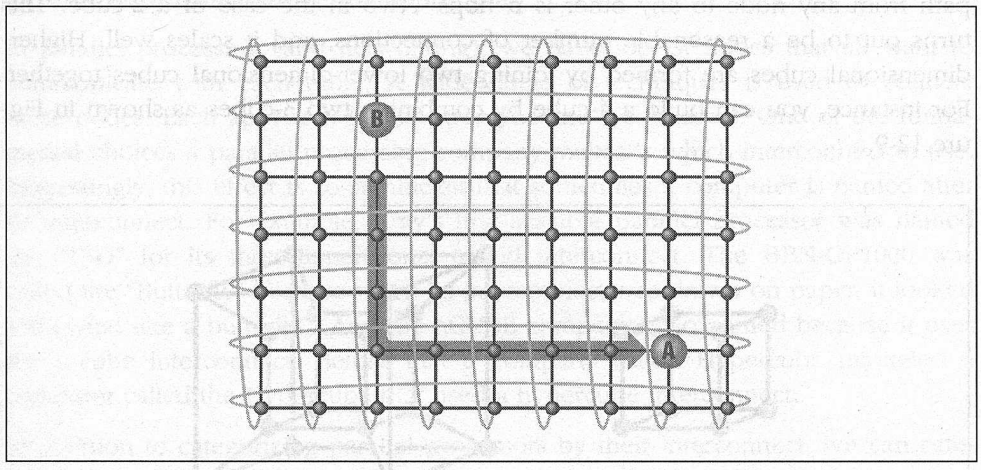

Figure 12-7: A two-dimensional toroid *

*K. Dowd and C. Severance, *High Performance Computing, 2nd Ed.*, O'reilly, 1998.

Tree Networks

For a *binary tree network* (see Figure 1.12), the first node is called the *root*.

Each node has two links to the two nodes below.

At j^{th} level below root node, there are 2^j nodes.

There are $2^{j+1} - 1$ nodes in the network, from the j^{th} level and above.

Such a tree is called a *complete binary tree*, as it is fully populated.

Height of tree is number of links from root to lowest leaves.

Useful for *divide and conquer algorithms.*

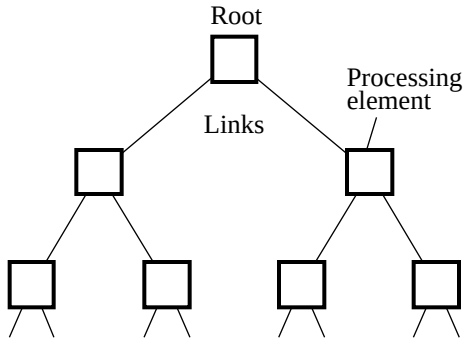

Figure 1.12 Tree structure.

Hypercubes

Before *wormhole routing,* number of hops a message must pass through important.

Want interconnect to minimize # of hops, but easy/cheap to build.

Used n-dimensional *hypercubes.* Each node connects to one node in each of the n dimensions.

With 4 processors, could build 2-cube. Number of nodes, N, is 2^n.

A 3-cube, has 8 processors, and is an actual cube.

Number of hops between any two nodes is n. Diameter is thus $\log_2(N) = n$. Bisection width is $N/2$.

Hypercubes Cont.

Higher dimension cubes created by combining two lower dimension cubes (see 4-cube).

Each node assigned a n-bit binary address. Each bit corresponds to a dimension. See Figure 1.13

Each node connects to nodes that differ by one bit.

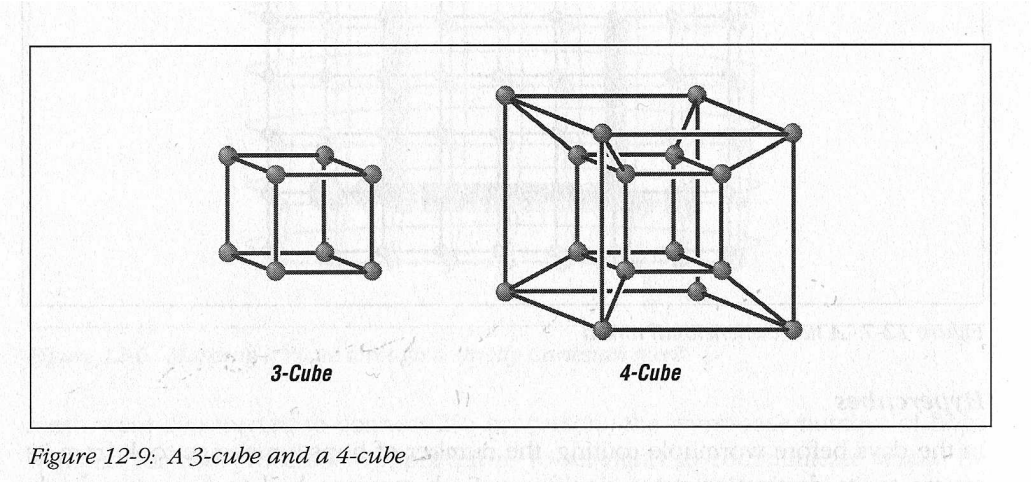

Figure 12-9: A 3-cube and a 4-cube *

*K. Dowd and C. Severance, *High Performance Computing, 2nd Ed.*, O'reilly, 1998.

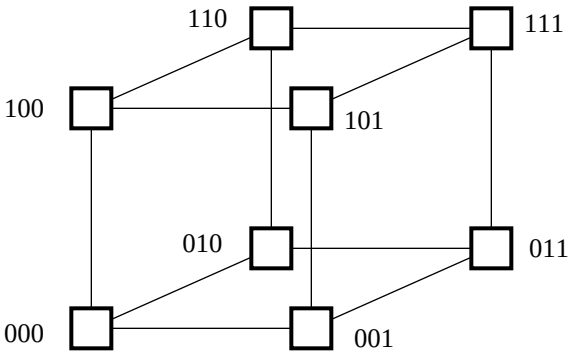

Figure 1.13 Three-dimensional hypercube.

Hypercube Routing

Has simple minimal distance deadlock-free routing algorithm. Called *e-cube routing algorithm.*

Route from node S, with address $s_{n-1}s_{n-2}\ldots s_0$, to Node D, with address $d_{n-1}d_{n-2}\ldots d_0$.

Each bit different represents a direction that must be traversed.

Found by doing exclusive-OR, $R = S \oplus D$, on bit pairs.

Dimensions to travel are bits of R that are "1."

At each stage of route, exclusive-OR of current node and destination node made.

Most significant (left-most bit) "1" bit used to choose next direction.

Embedding

For static networks, *embedding* means mapping nodes of one type of network, onto nodes of another.

For instance, a ring network (the embedded network), can be embedded in a torus (the embedding network). See Figure 1.15.

Above called a *perfect* embedding. Each link in ring actually exists in torus. ie. didn't have to use extra node to make link.

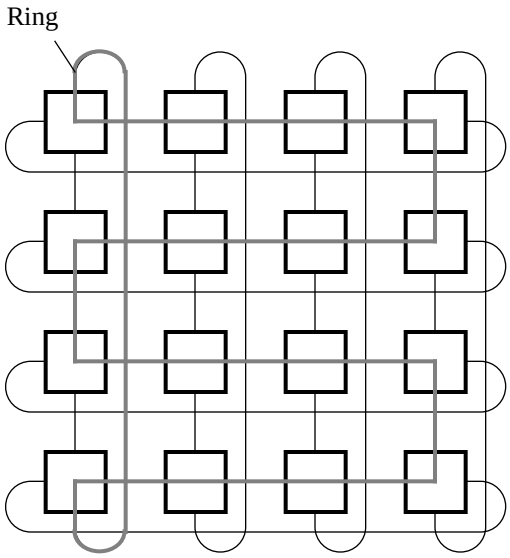

Figure 1.15 Embedding a ring onto a torus.

Embedding Cont.

Can embed a mesh or torus in a hypercube. See 4x4 mesh example in Figure 1.16.

Every node in mesh has an x and a y coordinate, labelled in a *gray code.*

Each node connects to another node whose addresses differs by one bit (thus one link).

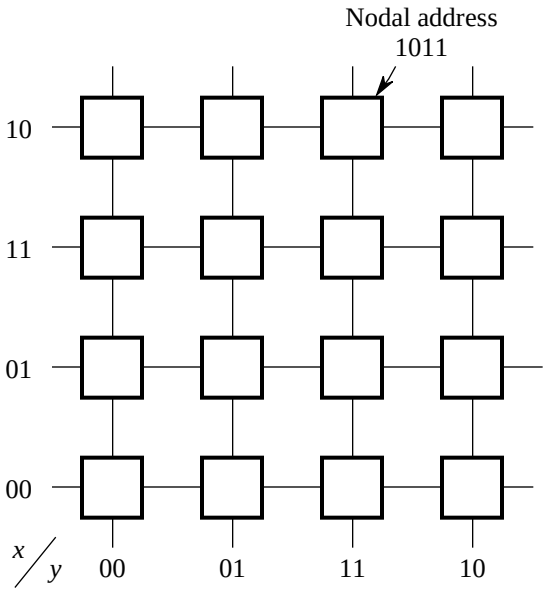

Figure 1.16 Embedding a mesh into a hypercube.

Embedding Tree Networks

Dilation refers to quality of embedding. Dilation is maximum number of links in embedding network that correspond to one link in embedded network.

Mapping a tree network into a mesh or hypercube usually doesn't result in a dilation of 1.

In Figure 1.17, the dilation is two.

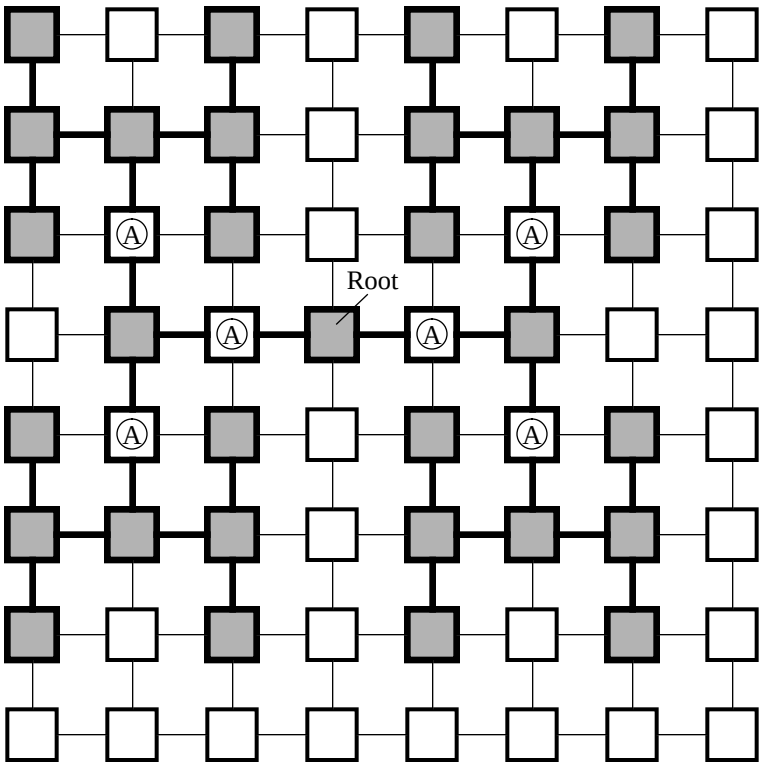

Figure 1.17 Embedding a tree into a mesh.

Communication Methods

To get message from source to destination, often need to route through intermediate nodes.

Two methods:

- Circuit Switching
- Packet Switching.

With circuit switching, you establish path and maintain all links until entire message has passed through (ie. wormhole routing).

Packet Switching

Message broken up into "packets" of data. Each has a source and destination address.

Network has maximum size. If too large, must be broken down into multiple packets.

Nodes have buffers to hold packet before re-transmission.

Called *store-and-forward packet switching.* High latency!

Alternative is *virtual cut-through.* If outgoing link available, message is immediately transmitted and not stored in buffer.

Wormhole Routing for Packet Switching

Alternative to normal store-and-forward routing: goal to reduce latency and size of buffers.

In *wormhole routing*, message divided into smaller units called *flits.* Size is usually 1-2 bytes.

Only head of message (first flit) is transmitted from source node to next node when link available.

Following flits transmitted when links available.

When head flit moves to next node, the next flit can move ahead.

Requires a request/acknowledge system. See Figure 1.18

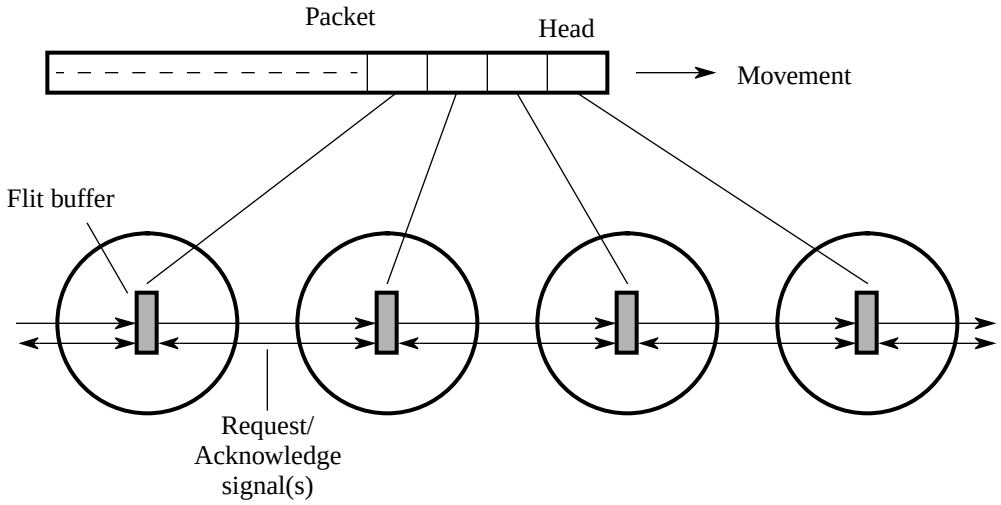

Figure 1.18 Distribution of flits.

Wormhole Routing for Packet Switching Cont.

Necessary to reserve entire path for message as the flits are linked.

Other packets can be interleaved along the same links.

Approach requires less storage and has latency independent of path length.

Network Latency Comparison

We will now evaluate the network latency for four different communication methods.

Let B stand for bandwidth (bits/s), l be the number of links the message passes through, and L the message size in bits.

For sending L bits over a single link, the transmission time is: L/B

Circuit Switching: Let L_c be the length (in bits) of control packet sent to establish path.

$$Latency = (\frac{L_c}{B})l + (\frac{L}{B})$$

Latency reduces to L/B if $L_c \ll L$

Network Latency Comparison Cont.

Store and Forward: $Latency = (\frac{L}{B})l$

Virtual Cut Through: Let L_h be the length (in bits) of header field of first packet of message.

$$Latency = (\frac{L_h}{B})l + (\frac{L}{B})$$

Latency reduces to L/B if $L_h \ll L$

Wormhole Routing: Let L_f be the length (in bits) of each flit.

$$Latency = (\frac{L_f}{B})l + (\frac{L}{B})$$

Latency reduces to L/B if $L_f \ll L$

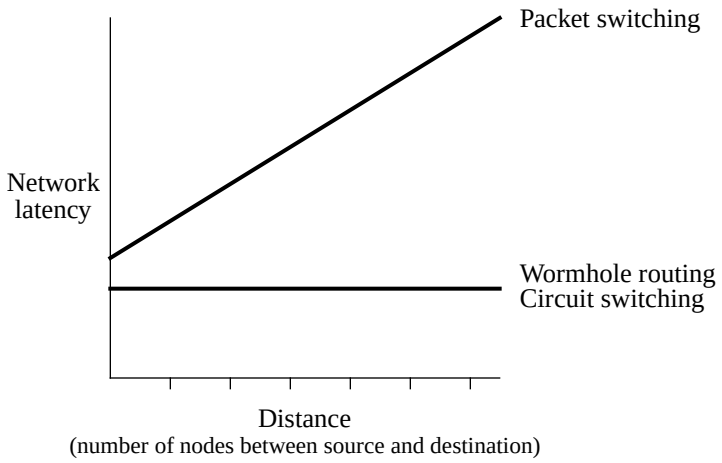

Figure 1.20 Network delay characteristics.

Deadlock

Interconnection networks use routing algorithms to determine path.

Can be adaptive; They can choose alternative paths depending on criteria - ie. local traffic

Algorithm may *deadlock* or *livelock.*

Livelock: When packet continually moves through network, but can never reach destination.

Deadlock: When a packet is stuck at a node, and can go no further do to other packets.

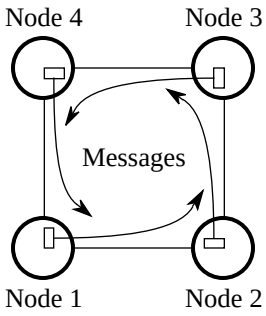

Figure 1.21 Deadlock in store-and-forward networks.

Deadlock Cont.

A solution is to provide *virtual* channels, each with separate buffers.

Physical links or channels or the actual links between nodes.

Multiple virtual channels are associate with a physical channel, and time-multiplexed onto the physical channel. See Figure 1.22.

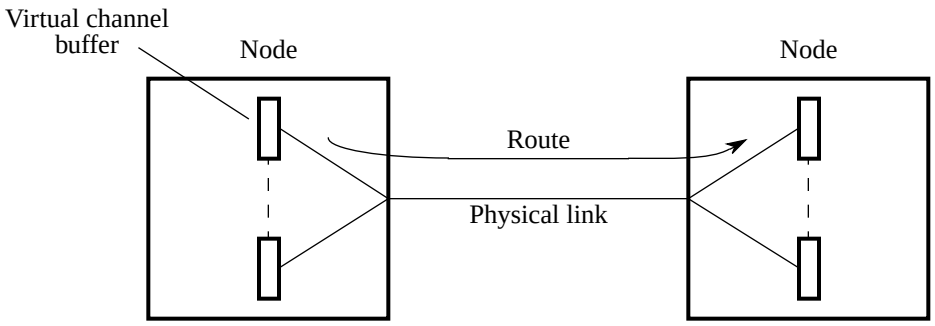

Figure 1.22 Multiple virtual channels mapped onto a single physical channel.

Networked Computers as Multicomputer Platform

Attractive alternative to expensive supercomputers.

- High performance PCs and workstations inexpensive and easily available.
- Latest processors can easily be added as they become available.
- Current software can be used or modified.
- Many networks of workstations already exist, and can be used.

Commonly connected using Ethernet (10 Mbit/sec, 100 Mbit/sec (fast), or 1000 Mbit/sec (Gigabit)).

Networked Computers as Multicomputer Platform Cont.

Originally, Ethernet was single cable connecting all workstations.

Bus topology, with high latency.

Other common networks used are ring structures (ie Token rings/FDDI networks - see Figure 1.25) and star networks (See Figure 1.26).

Point-to-Point communications have highest bandwidth. Created using switches.

Examples: High Performance Parallel Interface (HIPPI), fast and Gigabit Ethernet, and fiber optics.

Networks usually organized into subnetworks. May be mixed type.

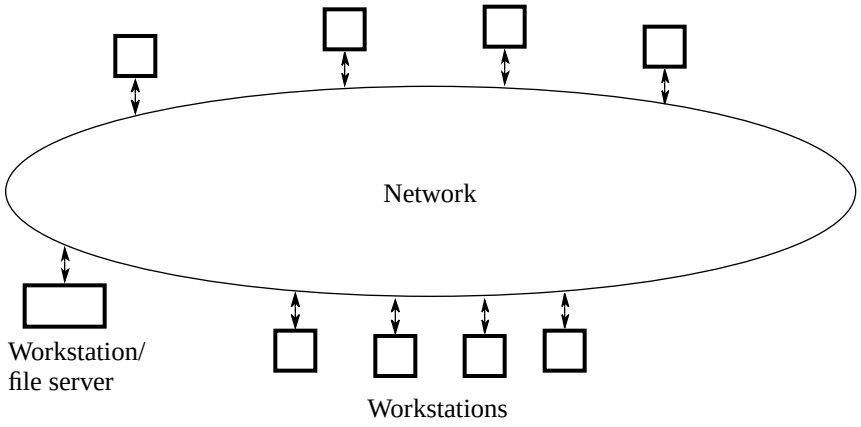

Figure 1.25 Network of workstations connected via a ring.

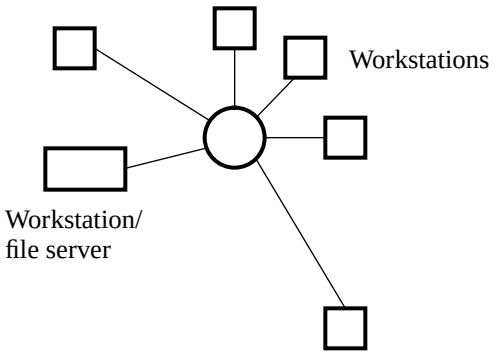

Figure 1.26 Star connected network.

Networked Computers vs Static Linked Computers

Networked workstations typically have higher latency, but more flexible communication.

For static linked, want to map communicating processes onto nearby processors in the interconnect topology.

Networked computers tend to be of mixed type and speed. Static linked usually has CPUs of the same type and speed.

Dedicated Programming Cluster

Workstations on a private network. Dedicated to cluster tasks.

Except for the cluster head, usually don't have keyboards, or monitors etc.

Simplest/cheapest to use Ethernet hubs. Would prefer something higher performance and that responds better to simultaneous traffic.

Collisions can be significantly reduced by multiple, overlapping, communication paths as in Figure 1.27.

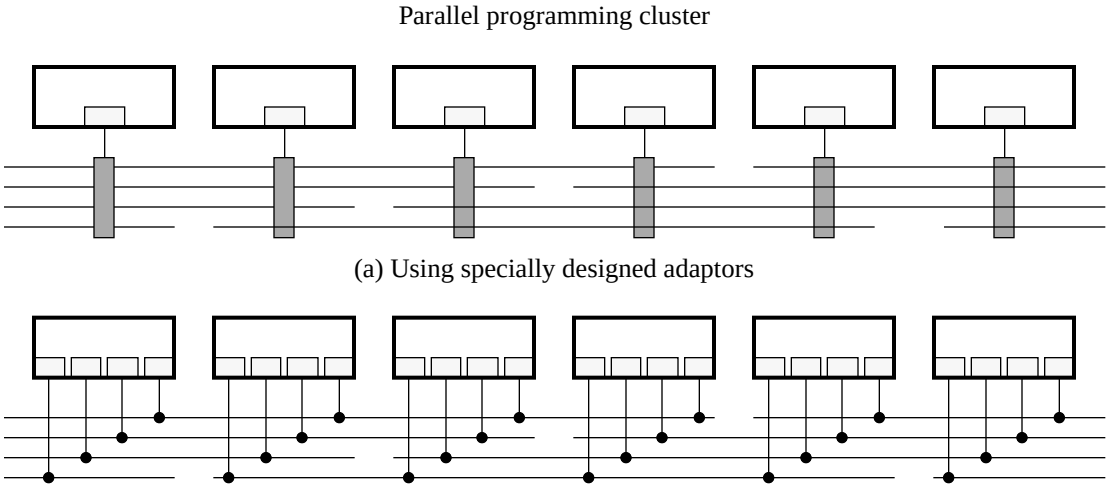

Figure 1.27 Overlapping connectivity Ethernets.

Potential for Increased Speed

No matter what form of parallel computer, must break computation into multiple *processes* (tasks) that can run simultaneously.

Granularity describes the size of a process.

Coarse Granularity: Each process has large number of instructions and takes a while to run.

Fine Granularity: Process contains only a few instructions.

Medium Granularity: Somewhere in between.

Granularity can also be the amount of computation between communication or synchronization points.

Granularity Metric

Want to find compromise between cost of process creation and communication, against how much we can parallelize.

For message passing, want to reduce amount of intercomputer communication.

As we divide problem, there's a point when the communication time dominates execution time.

Following ratio can be used as granularity metric:

$$\text{Computation/communication ratio} =$$

$$\frac{\text{Computation time}}{\text{Communication time}} = \frac{t_{comp}}{t_{comm}}$$

Want to maximize ratio, but still maintain sufficient parallelism.

Granularity Metric Example

We have two computers connected by a network. Each CPU has clock period T, and the communication latency for the network is L.

We examine the performance of the following task:

Each process (1 per CPU) must process 10 commands. Each command takes 30 clock cycles to execute. Process 1 then sends a message to process 2, who then replies. For our system, T = 12.5ns, and L = 500us.

$t_{comp} = 10 \cdot 30 \cdot T = 10 \cdot 30 \cdot 12.5 \times 10^{-9} s = 3.75 \times 10^{-6} s$

$t_{comm} = 2 \cdot L = 2 \cdot 500 \times 10^{-6} s = 0.01 s$

$ratio = \frac{t_{comp}}{t_{comm}} = \frac{3.75 \times 10^{-6} s}{0.01 s} = 0.00375$

NOTE: total execution time of task is: $t_{comp} + t_{comm}$

Granularity Metric Cont.

Granularity related to number of processors.

For domain decomposition (partitioning the data), the size of the per CPU data can be increased to improve ratio.

For a fixed data size, could reduce number of CPUs used.

Want to design a program where it is easy to vary granularity.

Speedup Factor

Measure of performance improvement between a single processor and parallel computer is the *speedup factor*, $S(n)$ (n the number of processors).

$$\begin{aligned} S(n) &= \frac{\text{Execution time using single CPU}}{\text{Execution time using } n \text{ processors}} \\ &= \frac{t_s}{t_p} \end{aligned}$$

For theoretical analysis, can also express as computational steps.

$$S(n) = \frac{\text{Number of steps using single CPU}}{\text{Number of parallel steps using } n \text{ processors}}$$

Speedup

The maximum speedup factor due to parallelization is n with n processors. This is called *linear speedup.* For example:

$$S(n) = \frac{t_s}{t_s/n} = n$$

Superlinear speedup is when $S(n) > n$.

Unusual, and usually due to a suboptimal sequential algorithm or special features of the architecture which favor parallel programs.

Normally, can't fully parallelize a program. Some part must be done serially.

Will be periods when only one processor doing work, and others idle. See Figure 1.28.

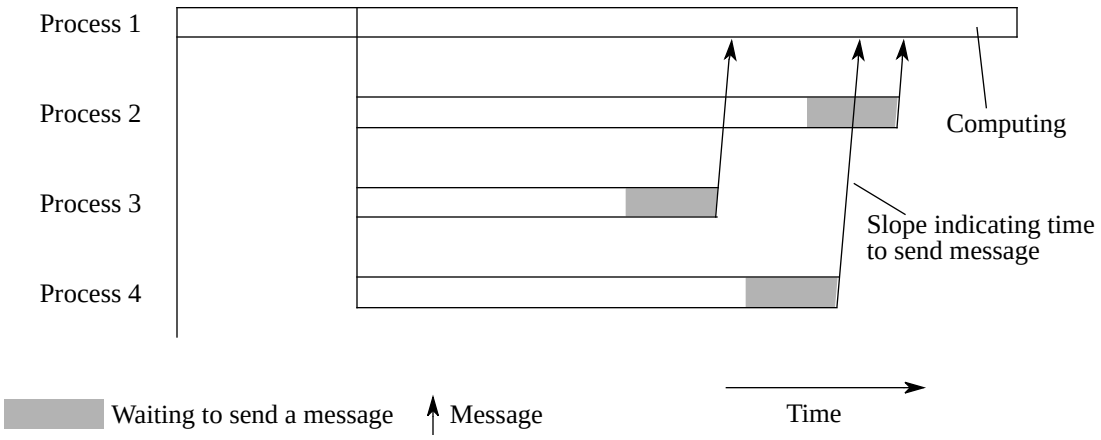

Figure 1.28 Space-time diagram of a message-passing program.

Overhead

Overhead is the time delays that are added due to the parallelization. Factors that cause overhead and reduce speedup are:

- Periods when all processors are not doing useful work.

- Additional computations not in serial version.

- Communication time for sending messages.

Maximum Speedup

Normally, there will always be some fraction of the program that must be performed serially.

Let the serial fraction of the program be f; thus, $1 - f$ is the fraction that can be parallelized.

Assuming no overhead results when program divided into concurrent parts, then computation time with n processors is:

$$t_p = ft_s + (1 - f)t_s/n$$

See Figure 1.29.

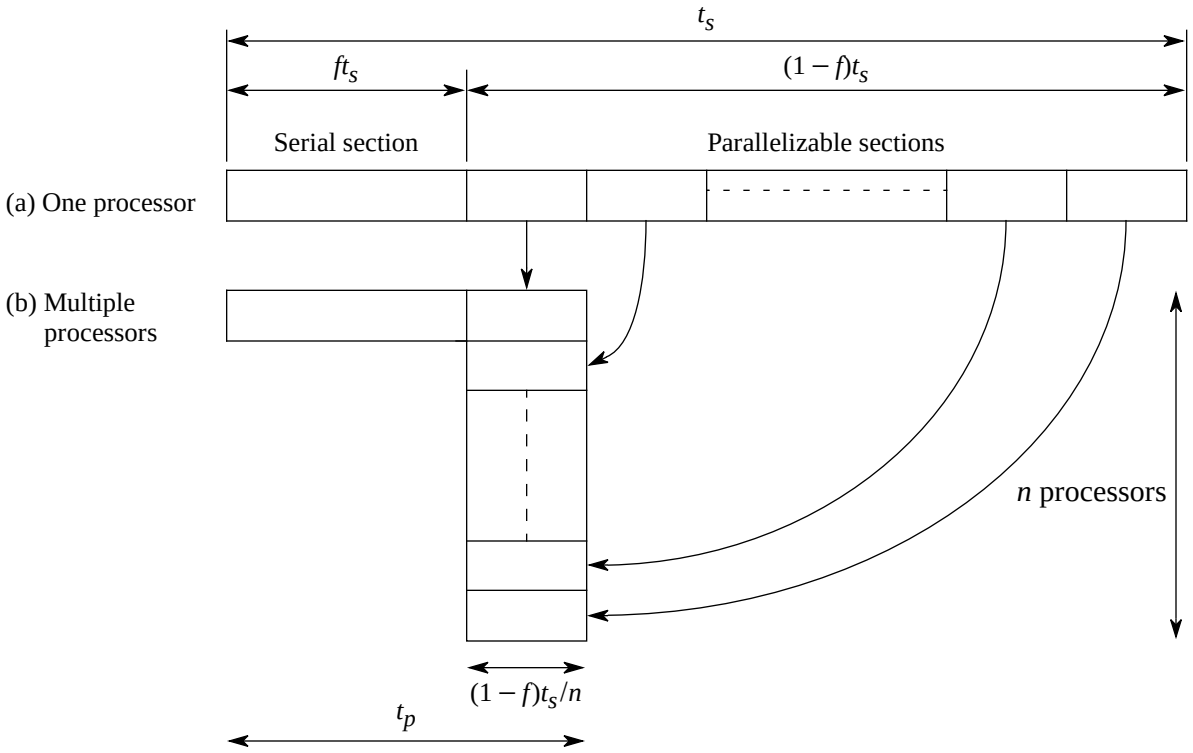

Figure 1.29 Parallelizing sequential problem — Amdahl's law.

Maximum Speedup Cont.

This gives us the equation below known as *Amdahl's law.*

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1 + (n-1)f}$$

See Figure 1.30.

Limit of speedup with infinite number of processors:

$$S(n)_{n\to\infty} = \frac{1}{f}$$

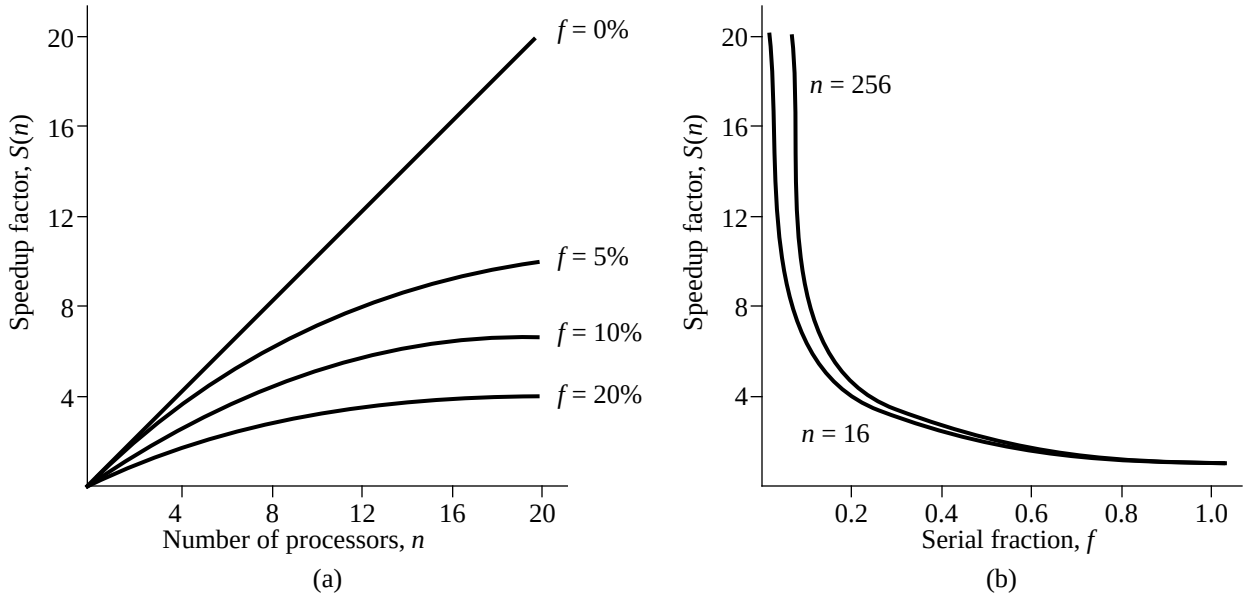

Figure 1.30 (a) Speedup against number of processors. (b) Speedup against serial fraction, *f*.

Efficiency

The system *efficiency*, E, is defined as:

$$\begin{aligned} E &= \frac{\text{Execution time with single CPU}}{\text{Execution time with multiprocessor} \times \text{number of CPUs}} \\ &= \frac{t_s}{t_p \times n} \end{aligned}$$

Efficiency as a percentage is given by:

$$E = \frac{S(n)}{n} \times 100\ \%$$

The *processor-time* product, or *cost* of a computation is:

$$\textit{Cost} = t_p \times n = \frac{t_s n}{S(n)} = \frac{t_s}{E}$$

Scalability

Has different meanings.

Architecture or Hardware Scalability: A hardware design that permits system to be enlarged which results in increased performance.

Usually, adding more processors means network must be increased. Means greater delay and contention thus lower efficiency.

Algorithmic Scalability: Means parallel algorithm can handle increase in data with low and bounded increase in computational steps.

One way to define *problem size* is the number of data elements to be processed.

Problem: doubling data size doesn't necessarily mean doubling number of computation steps.

Scalability Cont.

Alternative: equate *problem size* with number of steps in best sequential algorithm.

Still, increasing number of data elements will mean increasing "problem size."

Setting for Gustafson's Law

Gustafson argued that Amdahl's law is not as limiting as it seems.

Observed that a large multiprocessor means you can handle bigger problems.

In practice, problem size is related to number of processors.

Assume parallel processing time fixed, not problem size.

Gustafson claimed that serial section doesn't increase as problem size increases.

Setting for Gustafson's Law Cont.

Let s be the serial computation time. Let p be the time required to execute the parallel portion of the program on a single processor.

Assume $s + p$ is fixed. For convenience, we choose $s + p = 1$, so they represent fractions.

We can then represent Amdahl's law as:

$$S(n) = \frac{s + p}{s + p/n} = \frac{1}{s + (1 - s)/n}$$

Gustafson's Law

For Gustafson's *scaled speedup factor*, $S_s(n)$, he assumes the parallel execution is constant, and the serial time (t_s) changes.

Again, we have $s + p = 1$. Execution time on single processor is now $s + pn$. This gives us:

$$S_s(n) = \frac{s + np}{s + p} = s + np = n + (1 - n)s$$

Gustafson's Law Cont.

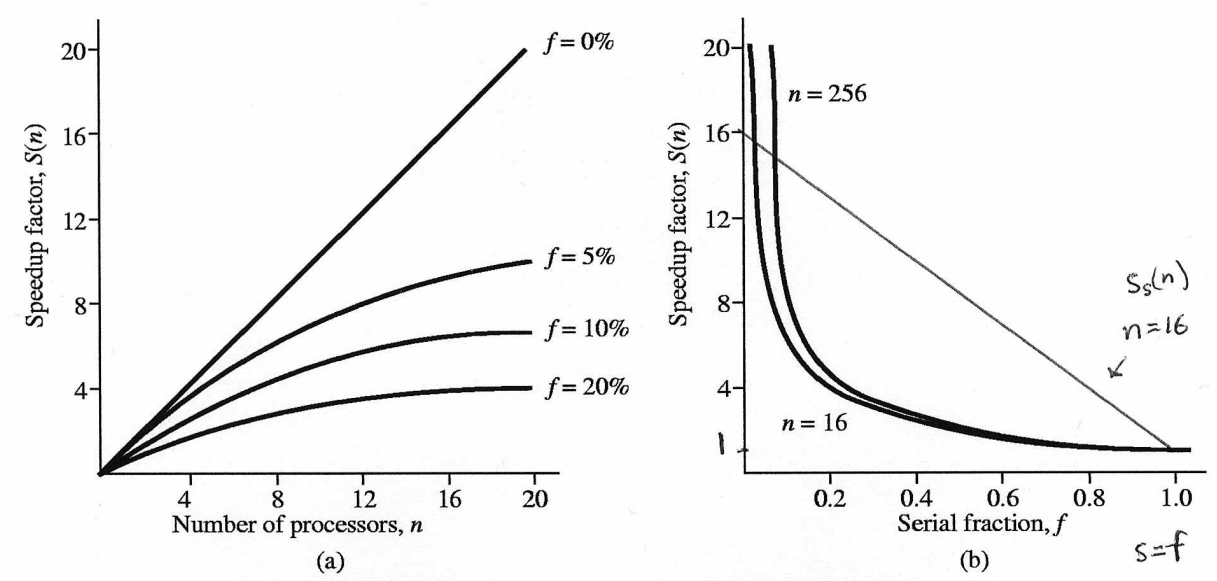

Figure 1.30 (a) Speedup against number of processors. (b) Speedup against serial fraction, f.