CSC4005 – Distributed and Parallel Computing

Prof. Yeh-Ching Chung

School of Data Science

Chinese University of Hong Kong, Shenzhen

Outline

Parallel Computers

The Demand for Computational Speed

Continual demand for greater computational speed from a computer system than is currently possible. Areas requiring great computational speed include numerical modeling and simulation of scientific and engineering problems. Computations must be completed within a "reasonable" time period.

Grand Challenge Problems

A grand challenge problem is one that cannot be solved in a reasonable amount of time with today's computers. Obviously, an execution time of 10 years is always unreasonable.

Examples: Modeling large DNA structures, global weather forecasting, modeling motion of astronomical bodies.

Weather Forecast

Atmosphere is modeled by dividing it into three-dimensional regions or cells. The calculations of each cell are repeated many times to model the passage of time.

Example

Whole global atmosphere divided into cells of size 1 mile $\times$ 1 mile $\times$ 1 mile to a height of 10 miles (10 cells high) - about 5×10^8 cells.

Suppose each calculation requires 200 floating point operations. In one time step, 10^{11} floating point operations necessary.

To forecast the weather over 10 days using 10-minute intervals, a computer operating at 100 Mflops (10^8 floating point operations/s) would take 10^7 seconds or over 100 days.

To perform the calculation in 10 minutes would require a computer operating at 1.7 Tflops (1.7×10^{12} floating point operations/sec).

Modeling Motion of Astronomical Bodies (1)

Predicting the motion of the astronomical bodies in space. Each body is attracted to each other body by gravitational forces. Movement of each body can be predicted by calculating the total force experienced by the body.

If there are N bodies, $N - 1$ forces to calculate for each body, or approximately N^2 calculations, in total. ($N \log_2 N$ for an efficient approximate algorithm.) After determining the new positions of the bodies, the calculations must be repeated.

A galaxy might have, say, 10^{11} stars. Even if each calculation could be done in $1\,\mu\text{s}$ (10^{-6} seconds, an extremely optimistic figure), it would take 10^9 years for one iteration using the N^2 algorithm and almost a year for one iteration using the $N \log_2 N$ efficient approximate algorithm.

Modeling Motion of Astronomical Bodies (2)

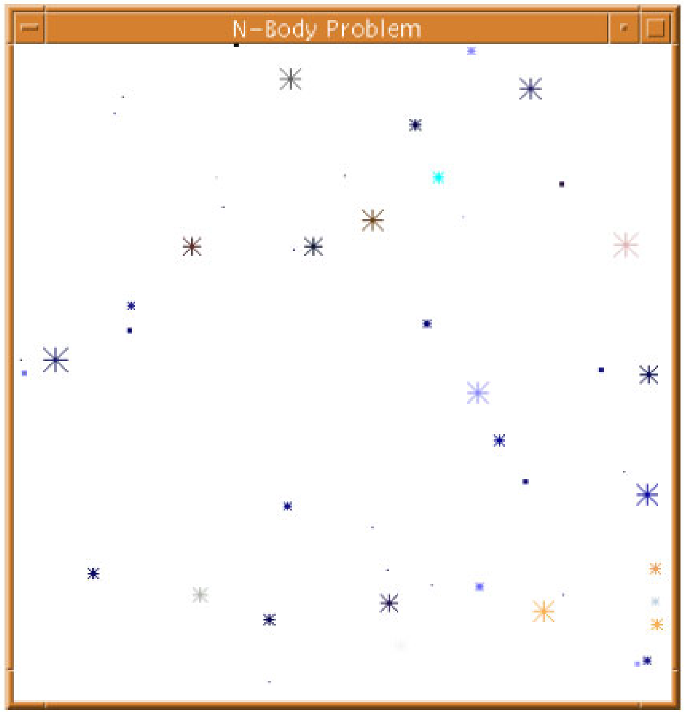


Figure 1.1 Astrophysical *N*-body simulation by Scott Linssen (undergraduate University of North Carolina at Charlotte [UNCC] student).

Parallel Computers and Programming

Using multiple processors operating together on a single problem. Not a new idea; in fact it is a very old idea. Gill writes in 1958:

"... There is therefore nothing new in the idea of parallel programming, but its application to computers. The author cannot believe that there will be any insuperable difficulty in extending it to computers. It is not to be expected that the necessary programming techniques will be worked out overnight. Much experimenting remains to be done. After all, the techniques that are commonly used in programming today were only won at the cost of considerable toil several years ago. In fact the advent of parallel programming may do something to revive the pioneering spirit in programming which seems at the present to be degenerating into a rather dull and routine occupation ..."

Gill, S. (1958), "Parallel Programming," The Computer Journal, vol. 1, April, pp. 2-10.

Notwithstanding the long history, Flynn and Rudd (1996) write that " ... leads us to one simple conclusion: the future is parallel." We concur.

Types of Parallel Computers

A conventional computer consists of a processor executing a program stored in a (main) memory:

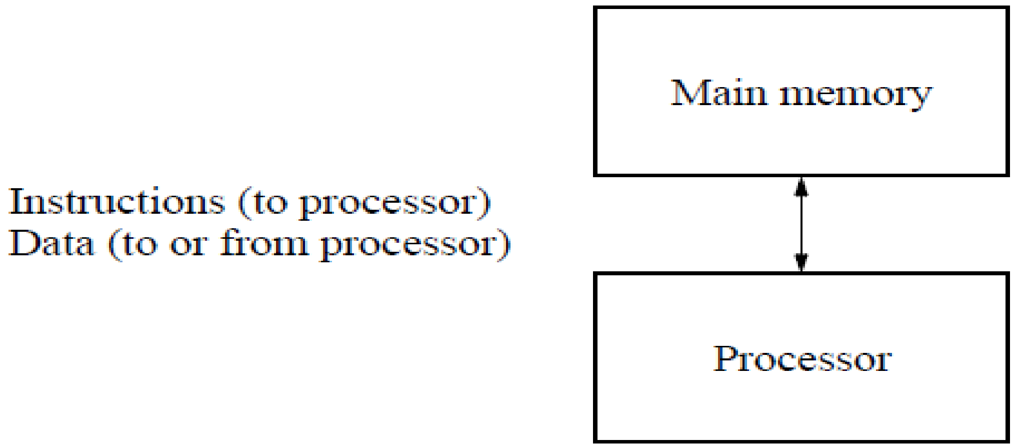


Figure 1.2 Conventional computer having a single processor and memory.

Each main memory location in the memory in all computers is located by a number called its *address*. Addresses start at 0 and extend to $2^n - 1$ when there are n bits (binary digits) in the address.

Shared Memory Multiprocessor Systems

A natural way to extend the single processor model is to have multiple processors connected to multiple memory modules, such that each processor can access any memory module in a so-called *shared memory* configuration:

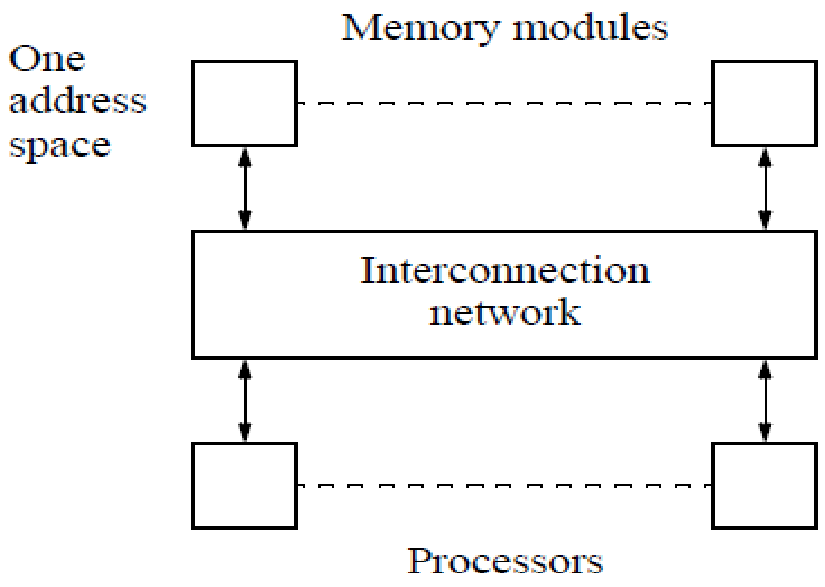


Figure 1.3 Traditional shared memory multiprocessor model.

Programming Shared Memory Multiprocessor

Can be done in different ways:

Parallel Programming Languages

With special parallel programming constructs and statements that allow shared variables and parallel code sections to be declared. Then the compiler is responsible for producing the final executable code.

Threads

Threads can be used that contain regular high-level language code sequences for individual processors. These code sequences can then access shared locations.

Message Passing Multiprocessor Systems

Complete computers connected through an interconnection network:

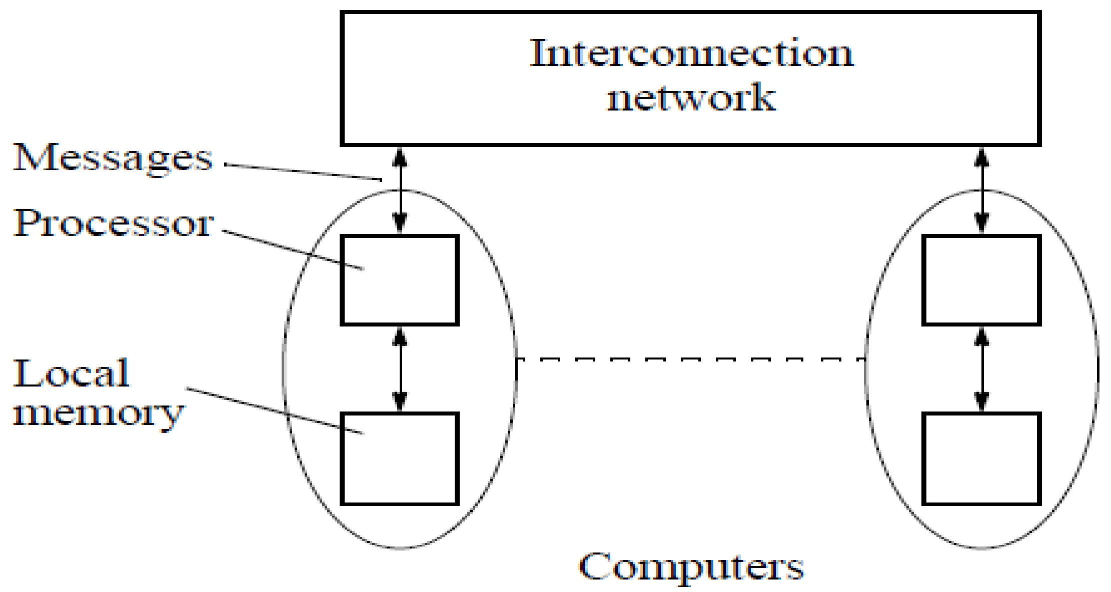


Figure 1.4 Message-passing multiprocessor model (multicomputer).

Programming

Still involves dividing the problem into parts that are intended to be executed simultaneously to solve the problem

Common approach is to use message-passing library routines that are linked to conventional sequential program(s) for message passing.

Problem divided into a number of concurrent processes.

Processes will communicate by sending messages; this will be the only way to distribute data and results between processes.

Distributed Shared Memory

Each processor has access to the whole memory using a single memory address space. For a processor to access a location not in its local memory, message passing must occur to pass data from the processor to the location or from the location to the processor, in some automated way that hides the fact that the memory is distributed.

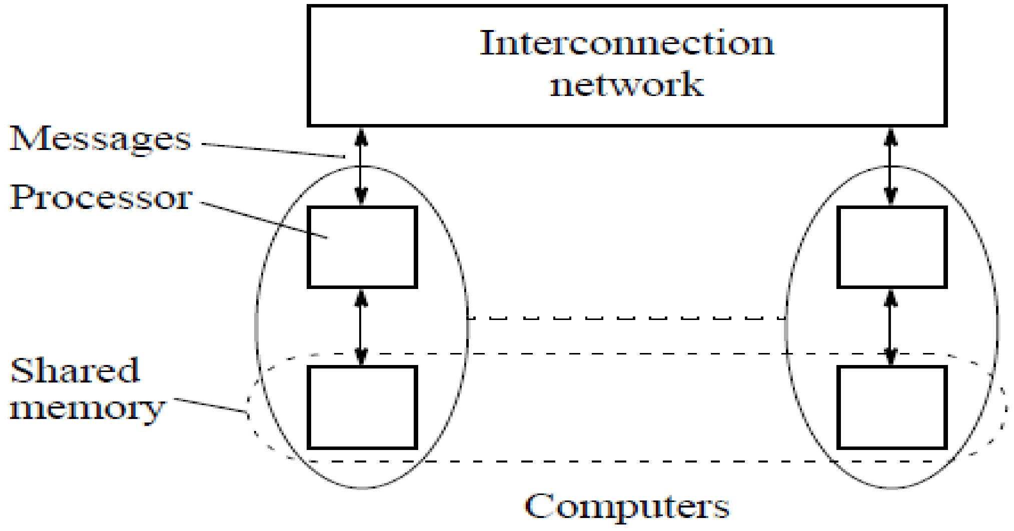


Figure 1.5 Shared memory multiprocessor implementation.

MIMD and SIMD Classifications

In a single processor computer, a single stream of instructions is generated from the program. The instructions operate upon a single stream of data items. Flynn (1966) created a classification for computers and called this single processor computer a *single instruction stream-single data stream* (SISD) computer.

Multiple Instruction Stream-Multiple Data Stream (MIMD) Computer

General-purpose multiprocessor system - each processor has a separate program and one instruction stream is generated from each program for each processor. Each instruction operates upon different data.

Both the shared memory and the message-passing multiprocessors so far described are in the MIMD classification.

Single Instruction Streaming-Multiple Data Streaming (SIMD) Computers

A specially designed computer in which a single instruction stream is from a single program, but multiple data streams exist. The instructions from the program are broadcast to more than one processor. Each processor executes the same instruction in synchronism, but using different data.

Developed because there are a number of important applications that mostly operate upon arrays of data.

Multiple Program Multiple Data Streaming (MPMD) Structure

Within the MIMD classification, which we are concerned with, each processor will have its own program to execute:

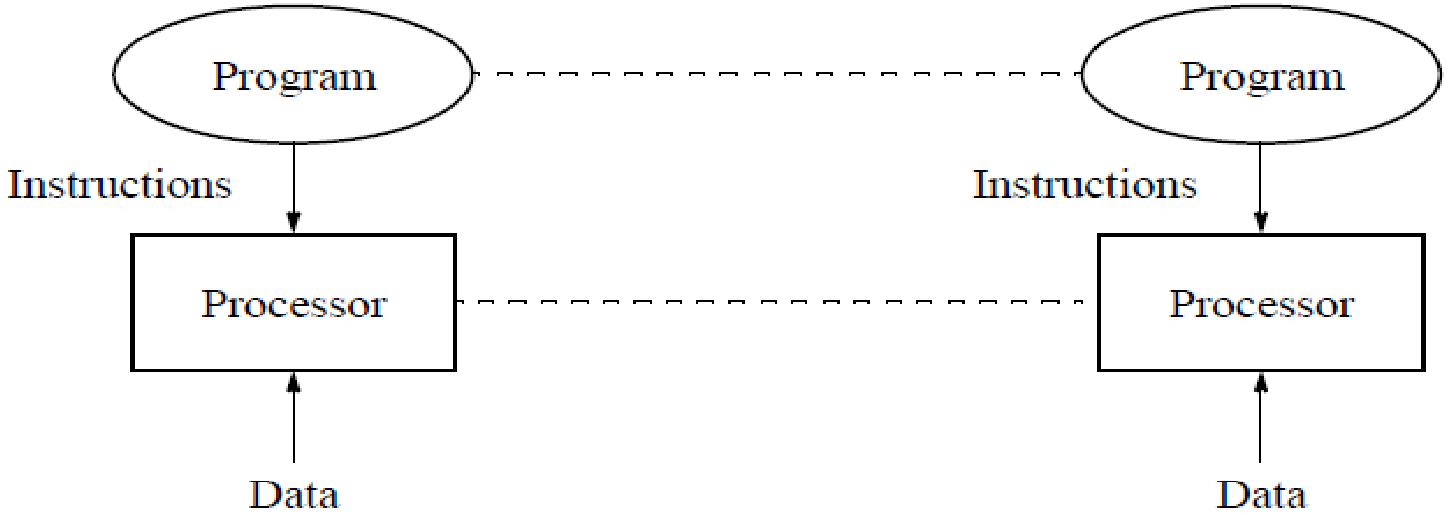


Figure 1.6 MPMD structure.

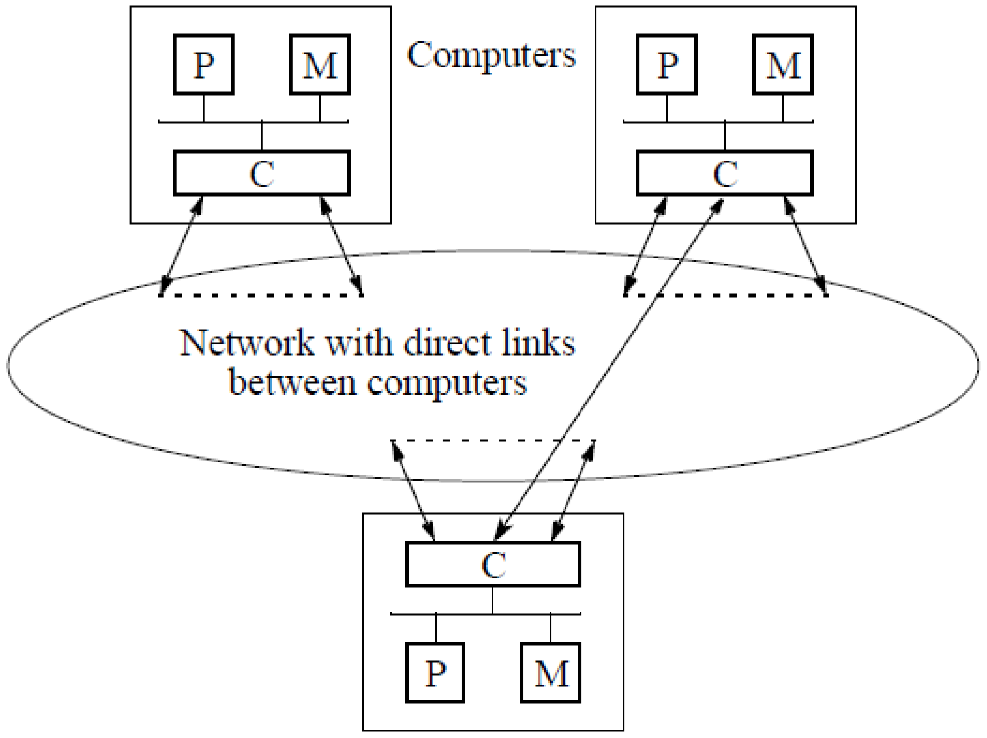
Single Program Multiple Data (SPMD) Structure

Single source program is written and each processor will execute its personal copy of this program, although independently and not in synchronism.

The source program can be constructed so that parts of the program are executed by certain computers and not others depending upon the identity of the computer.

Message-Passing Multicomputers (1)

- **Static network message-passing multicomputers**



Message-Passing Multicomputers (2)

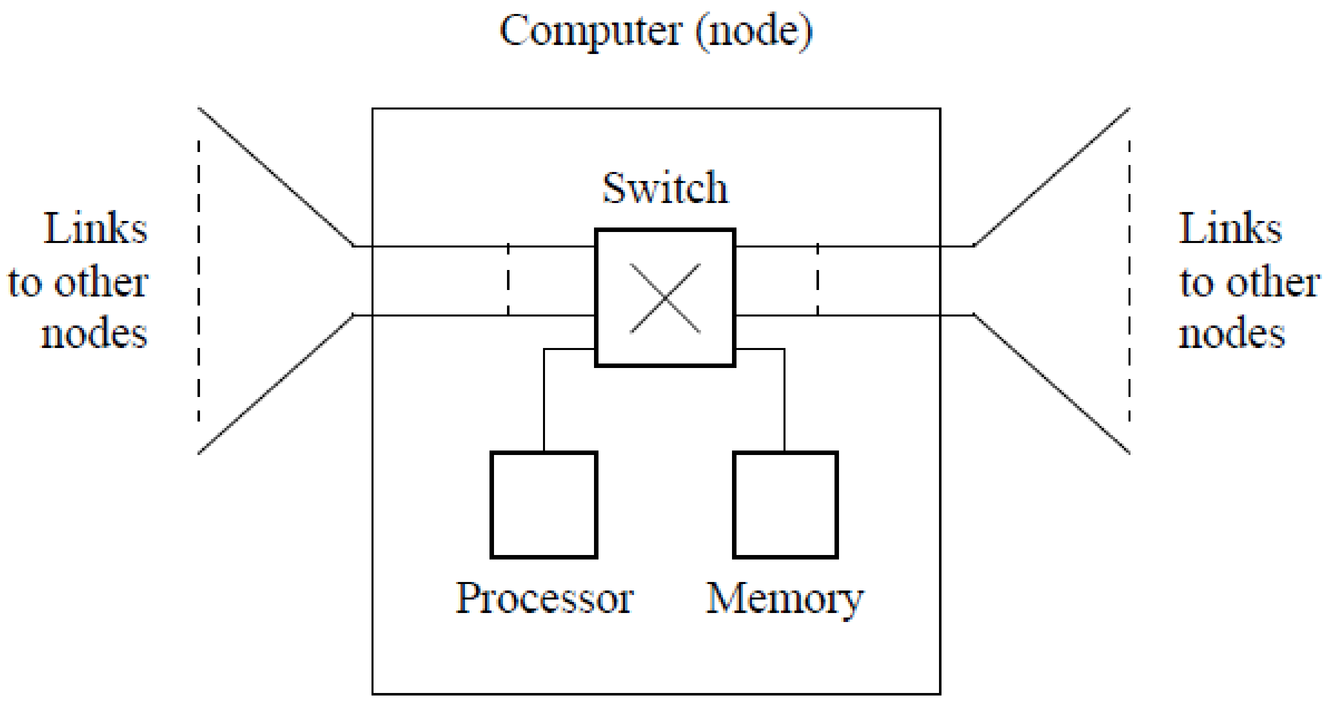


Figure 1.8 Node with a switch for internode message transfers.

Message-Passing Multicomputers (3)

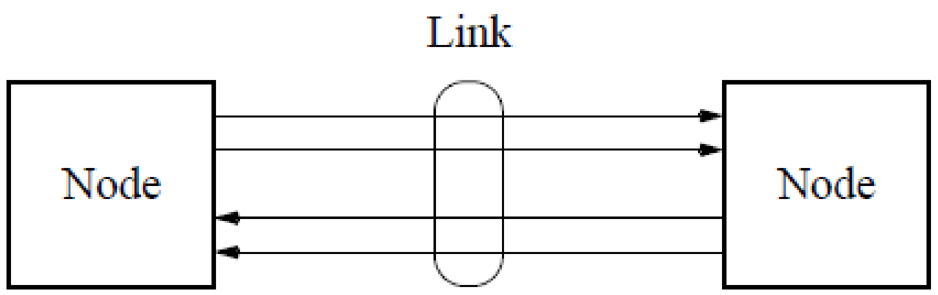


Figure 1.9 A link between two nodes with separate wires in each direction.

Network Criteria

Cost - indicated by number of links in network. (Ease of construction is also important.)

Bandwidth - number of bits that can be transmitted in unit time, given as bits/sec.

Network latency - time to make a message transfer through network.

Communication latency - total time to send message, including software overhead and interface delays.

Message latency or startup time - time to send a zero-length message. Essentially the software and hardware overhead in sending message and the actual transmission time.

Diameter - minimum number of links between two farthest nodes in the network. Only shortest routes used. Used to determine worst case delays.

Bisection width of a network - number of links (or sometimes wires) that must be cut to divide network into two equal parts. Can provide a lower bound for messages in a parallel algorithm.

Interconnection Networks (1)

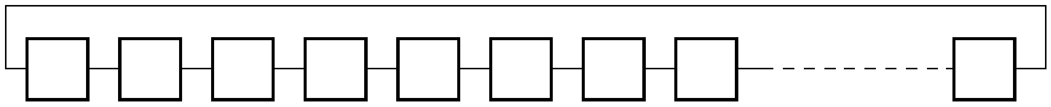


Figure 1.10 Ring.

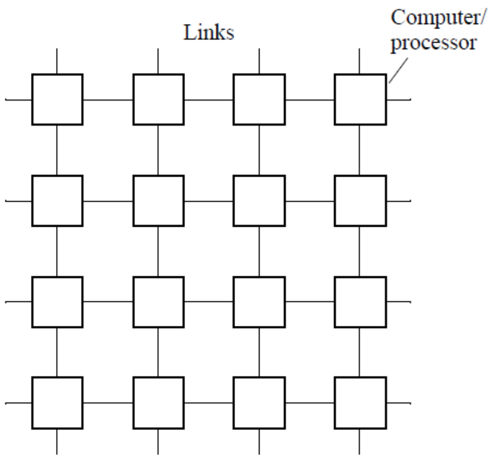


Figure 1.11 Two-dimensional array (mesh).

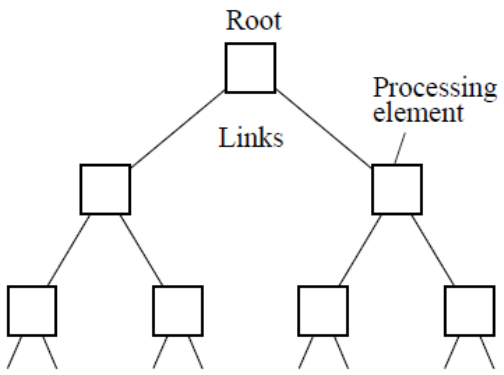


Figure 1.12 Tree structure.

Interconnection Networks (2)

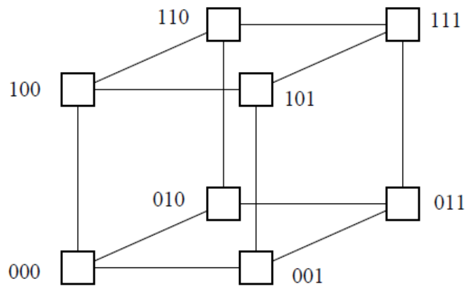


Figure 1.13 Three-dimensional hypercube.

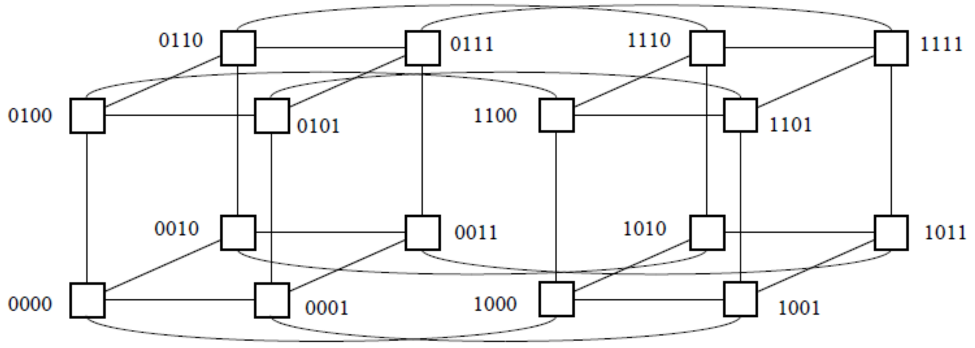
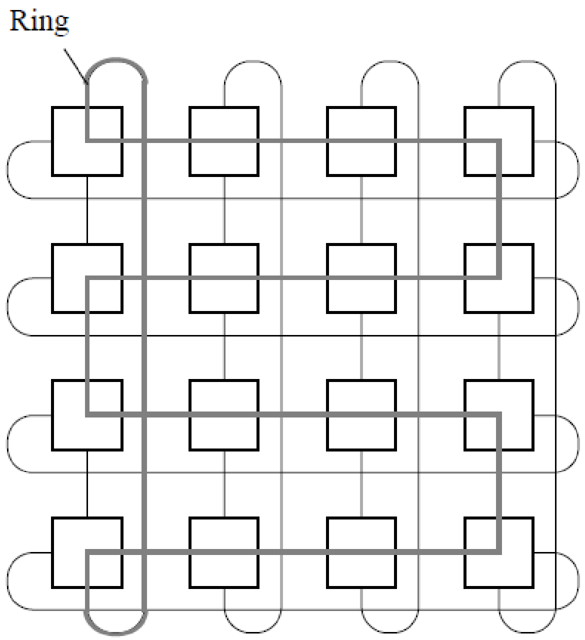
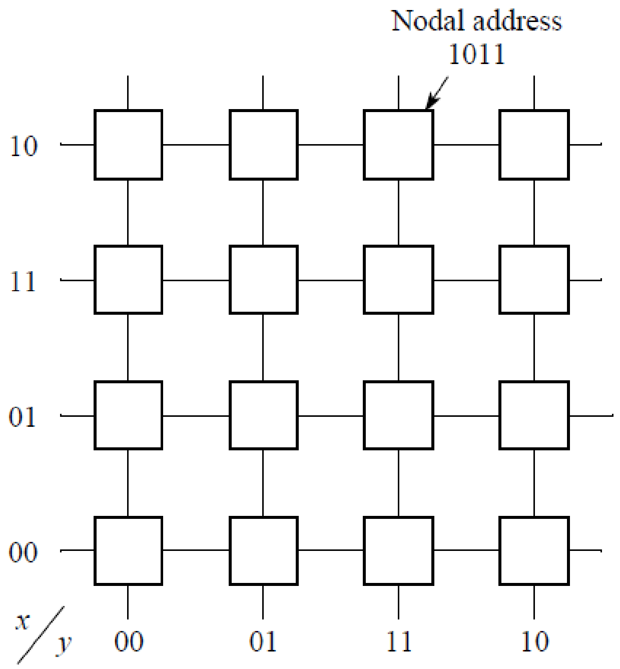


Figure 1.14 Four-dimensional hypercube.

Embedding (1)

Describes mapping nodes of one network onto another network. Example - a ring can be embedded in a torus:





Embedding (2)

Dilation - used to indicate the quality of the embedding.

The dilation is the maximum number of links in the "embedding" network corresponding to one link in the "embedded" network.

Perfect embeddings, such as a line/ring into mesh/torus or a mesh onto a hypercube, have a dilation of 1.

Sometimes it may not be possible to obtain a dilation of 1.

Example, mapping a tree onto a mesh or hypercube does not result in a dilation of 1 except for very small trees of height 2:

Embedding (3)

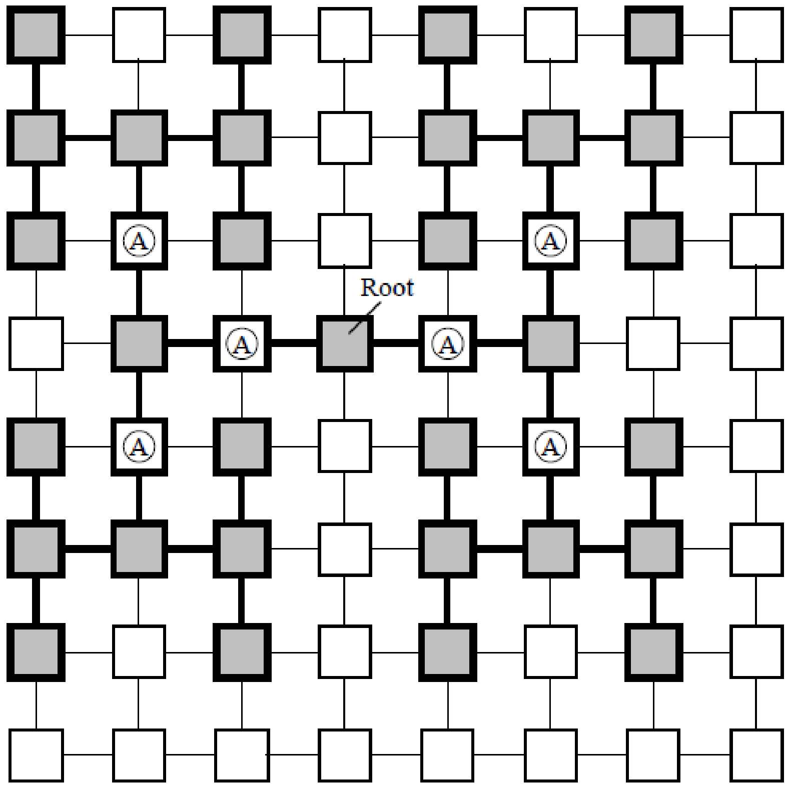


Figure 1.17 Embedding a tree into a mesh.

Communication Methods – Circuit Switching

Involves establishing path and maintaining all links in path for message to pass, uninterrupted, from source to destination. All links are reserved for the transfer until message transfer is complete.

Simple telephone system (not using advanced digital techniques) is an example of a circuit-switched system. Once a telephone connection is made, the connection is maintained until the completion of the telephone call.

Circuit switching suffers from forcing all the links in the path to be reserved for the complete transfer. None of links can be used for other messages until the transfer is completed.

Communication Methods – Packing Switching

Message divided into "packets" of information, each of which includes source and destination addresses for routing packet through interconnection network. Maximum size for the packet, say 1000 data bytes. If message is larger than this, more than one packet must be sent through network. Buffers provided inside nodes to hold packets before they are transferred onward to the next node. This form called *store-and-forward packet switching*.

Mail system is an example of a packet-switched system. Letters moved from mailbox to post office and handled at intermediate sites before being delivered to destination.

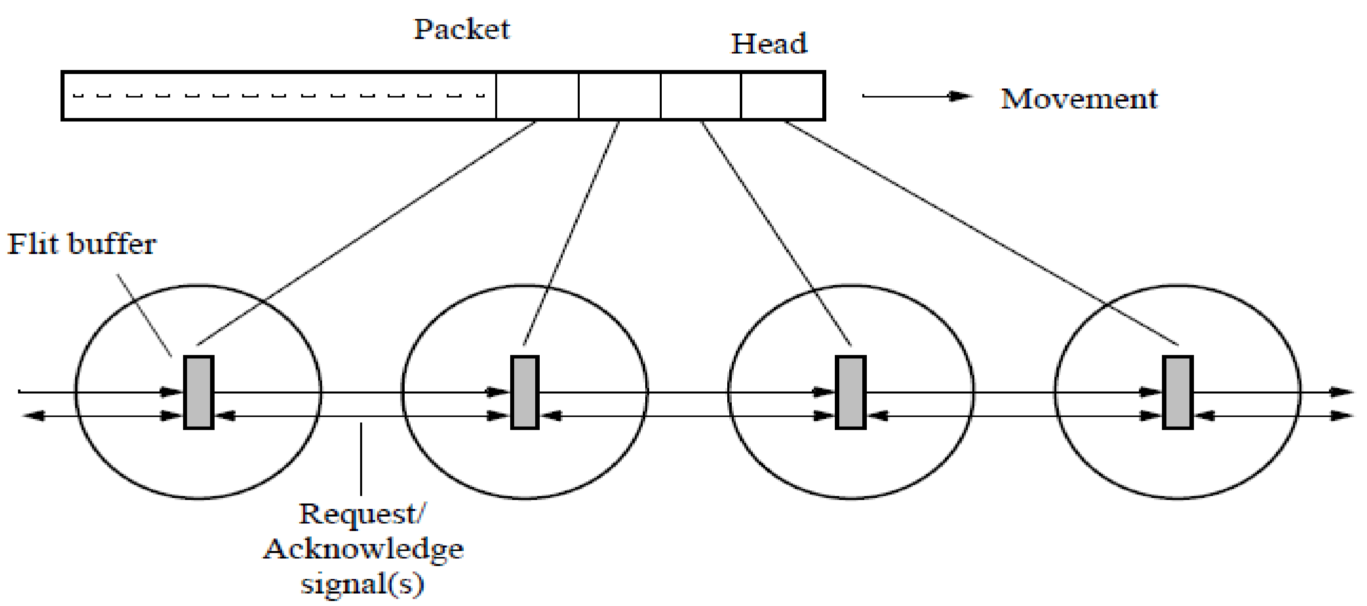
Enables links to be used by other packets once the current packet has been forwarded. Incurs a significant latency since packets must first be stored in buffers within each node, whether or not an outgoing link is available.

Communication Methods – Virtual Cut-Through

Can eliminated storage latency. If the outgoing link is available, the message is immediately passed forward without being stored in the nodal buffer; i.e., it is "cut through." If complete path were available, the message would pass immediately through to the destination. However, if path is blocked, storage is needed for the complete message/packet being received.

Communication Methods – Wormhole Routing (1)

Message divided into smaller units called *flits* (flow control digits). Only head of message initially transmitted from source node to next node when connecting link available. Subsequent flits of message transmitted when links become available. Flits can become distributed through network.



Communication Methods – Wormhole Routing (2)

Request/acknowledge system

A way to "pull" flits along. Only requires a single wire between the sending node and receiving node, called *R/A* (request/acknowledge).

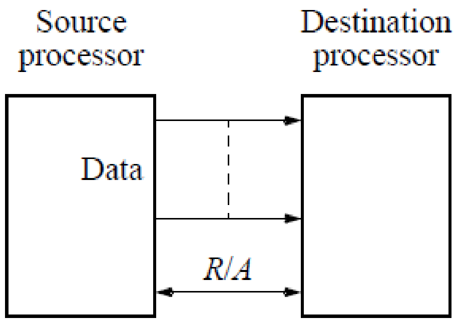


Figure 1.19 A signaling method between processors for wormhole routing (Ni and McKinley, 1993).

R/A reset to 0 by receiving node when ready to receive flit (its flit buffer empty).
R/A set to 1 by sending node when sending node is about to send flit.
Sending node must wait for *R/A* = 0 before setting it to a 1 and sending the flit.
Sending node knows data has been received when receiving node resets *R/A* to a 0.

Communication Methods – Wormhole Routing (3)

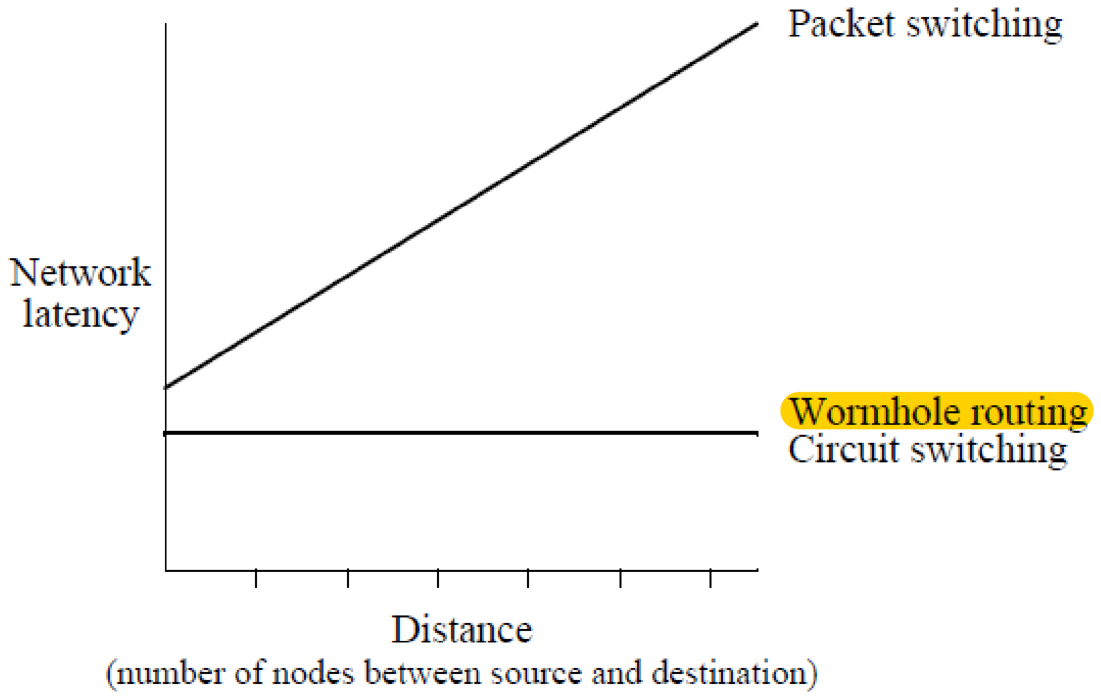


Figure 1.20 Network delay characteristics.

Deadlock

Occurs when packets cannot be forwarded to next node because they are blocked by other packets waiting to be forwarded and these packets are blocked in a similar way such that none of the packets can move.

Example

Node 1 wishes to send a message through node 2 to node 3. Node 2 wishes to send a message through node 3 to node 4. Node 3 wishes to send a message through node 4 to node 1. Node 4 wishes to send a message through node 1 to node 2.

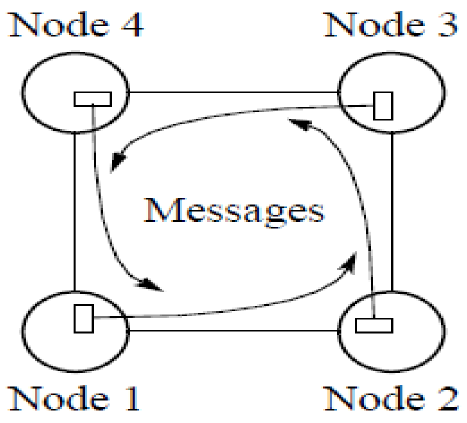


Figure 1.21 Deadlock in store-and-forward networks.

Virtual Channel

A general solution to deadlock. The *physical* links or channels are the actual hardware links between nodes. Multiple *virtual* channels are associated with a physical channel and time-multiplexed onto the physical channel.

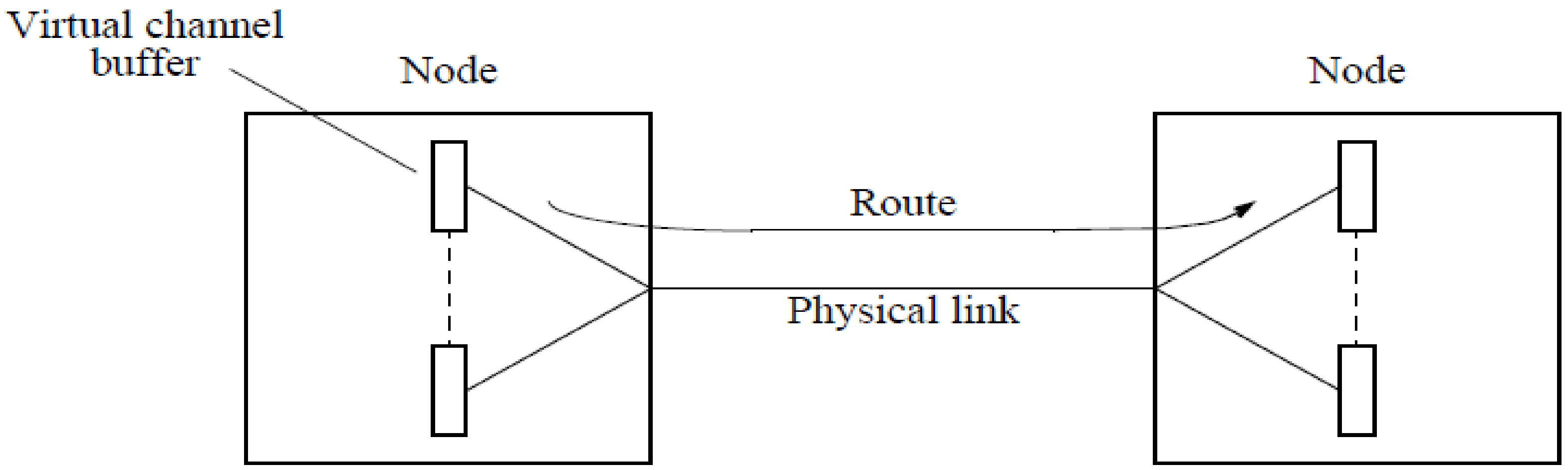


Figure 1.22 Multiple virtual channels mapped onto a single physical channel.

Network Computers as a Multicomputer Platform

A *cluster of workstations* (COWs), or *network of workstations* (NOWs), offers a very attractive alternative to expensive supercomputers and parallel computer systems for high-performance computing. Key advantages are as follows:

- Very high performance workstations and PCs are readily available at low cost.
- The latest processors can easily be incorporated into the system as they become available.
- Existing software can be used or modified.

Ring Structures

Examples - token rings/FDDI networks

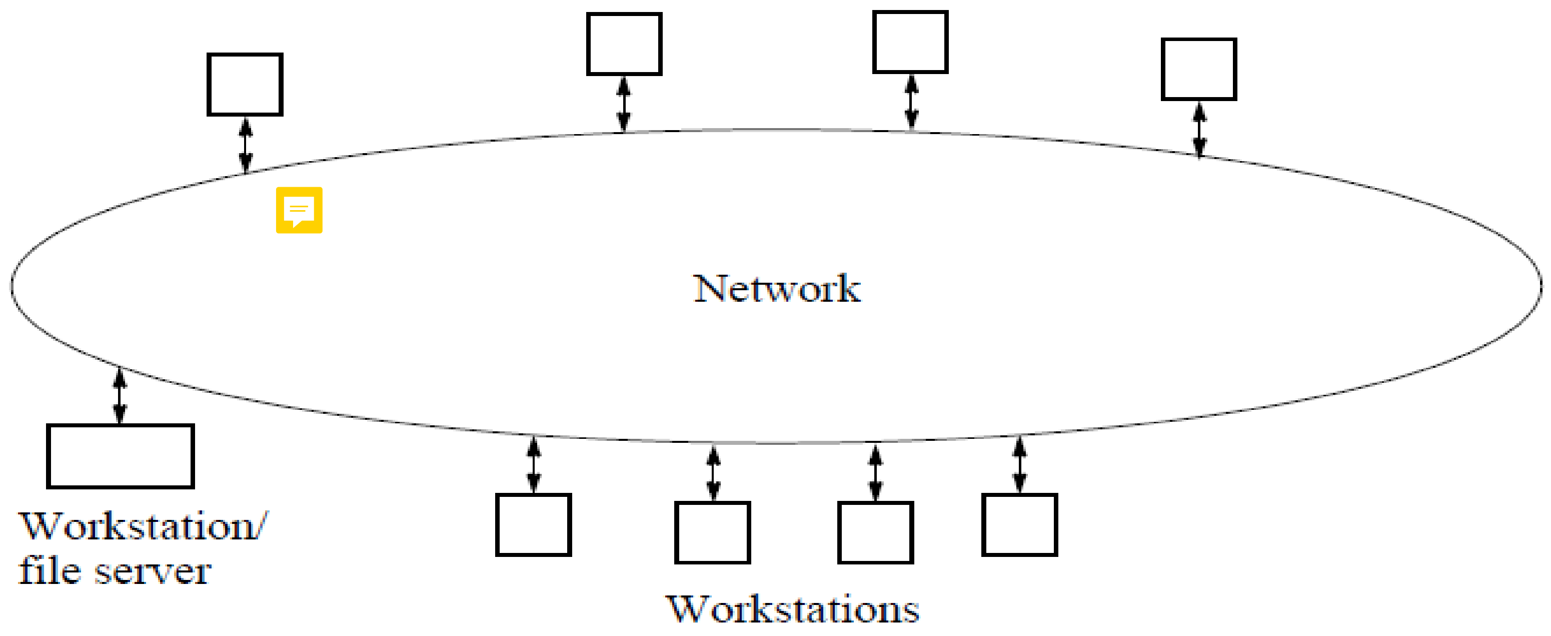


Figure 1.25 Network of workstations connected via a ring.

Point-to-Point Communication

Provides the highest interconnection bandwidth. Various point-to-point configurations can be created using hubs and switches.

Examples - High Performance Parallel Interface (HIPPI), Fast (100 MHz) and Gigabit Ethernet, and fiber optics.

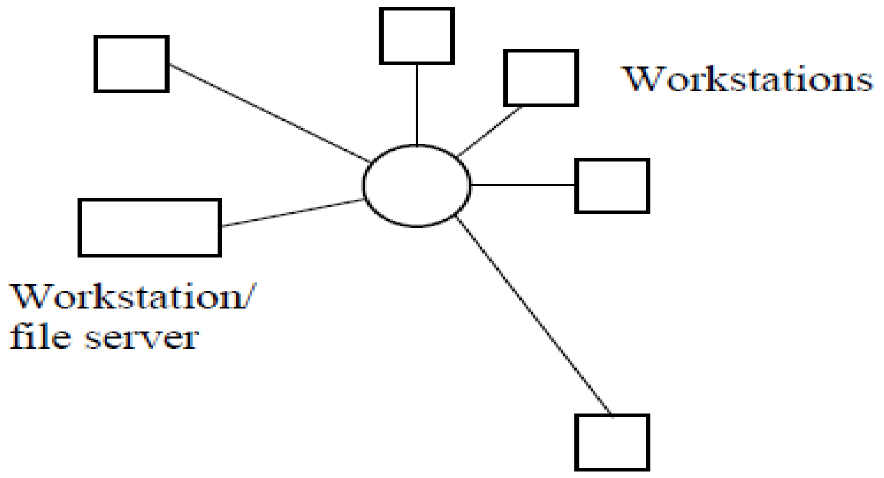
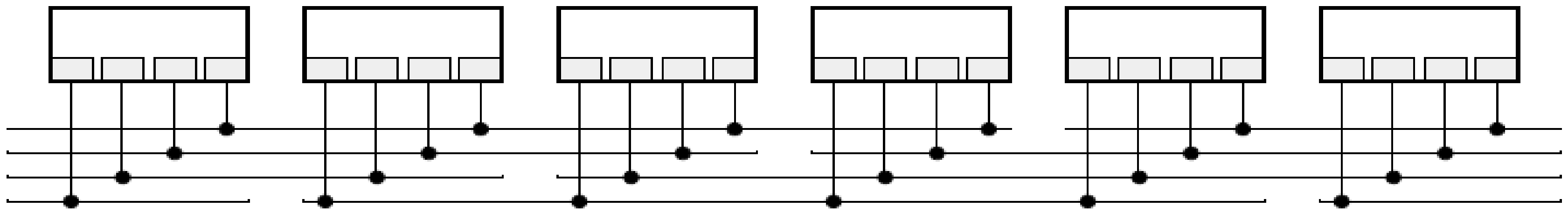


Figure 1.26 Star connected network.

Overlapping Connectivity Networks

Have characteristic that regions of connectivity are provided and regions overlap. Several ways overlapping connectivity can be achieved. Example using Ethernet:



Using separate Ethernet interfaces

Figure 1.27 Overlapping connectivity Ethernet.

Speedup Factor

$$S(n) = \frac{\text{Execution time using one processor (single processor system)}}{\text{Execution time using a multiprocessor with } n \text{ processors}} = \frac{t_s}{t_p}$$

where t_s is execution time on a single processor and t_p is execution time on a multiprocessor. $S(n)$ gives increase in speed in using a multiprocessor. Underlying algorithm for parallel implementation might be (and is usually) different.

Speedup factor can also be cast in terms of computational steps:

$$S(n) = \frac{\text{Number of computational steps using one processor}}{\text{Number of parallel computational steps with } n \text{ processors}}$$

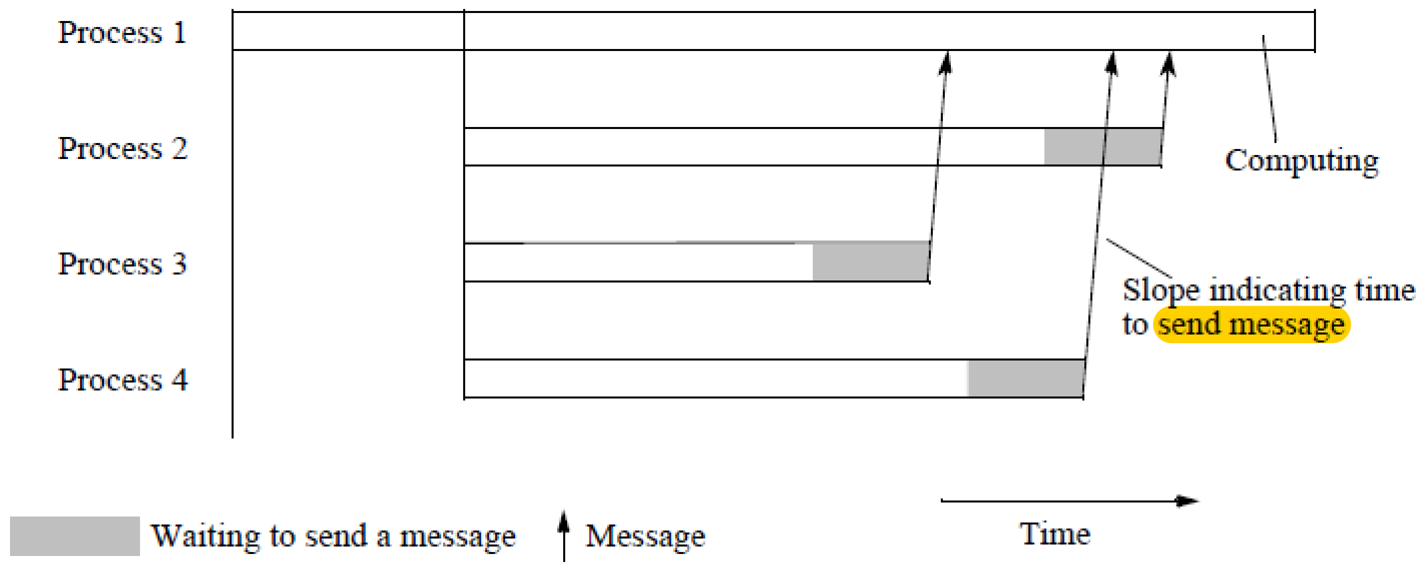
The maximum speedup is n with n processors (*linear speedup*).

Super-linear Speedup

where $S(n) > n$, may be seen on occasion, but usually this is due to using a suboptimal sequential algorithm or some unique feature of the architecture that favors the parallel formation.

One common reason for superlinear speedup is the extra memory in the multiprocessor system which can hold more of the problem data at any instant, it leads to less, relatively slow disk memory traffic. Superlinear speedup can occur in search algorithms.

Space-Time Diagram



Process 1
Process 2
Process 3
Process 4
Computing
Slope indicating time to send message
Waiting to send a message
Message
Time

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

Amdahl’s Law (1)

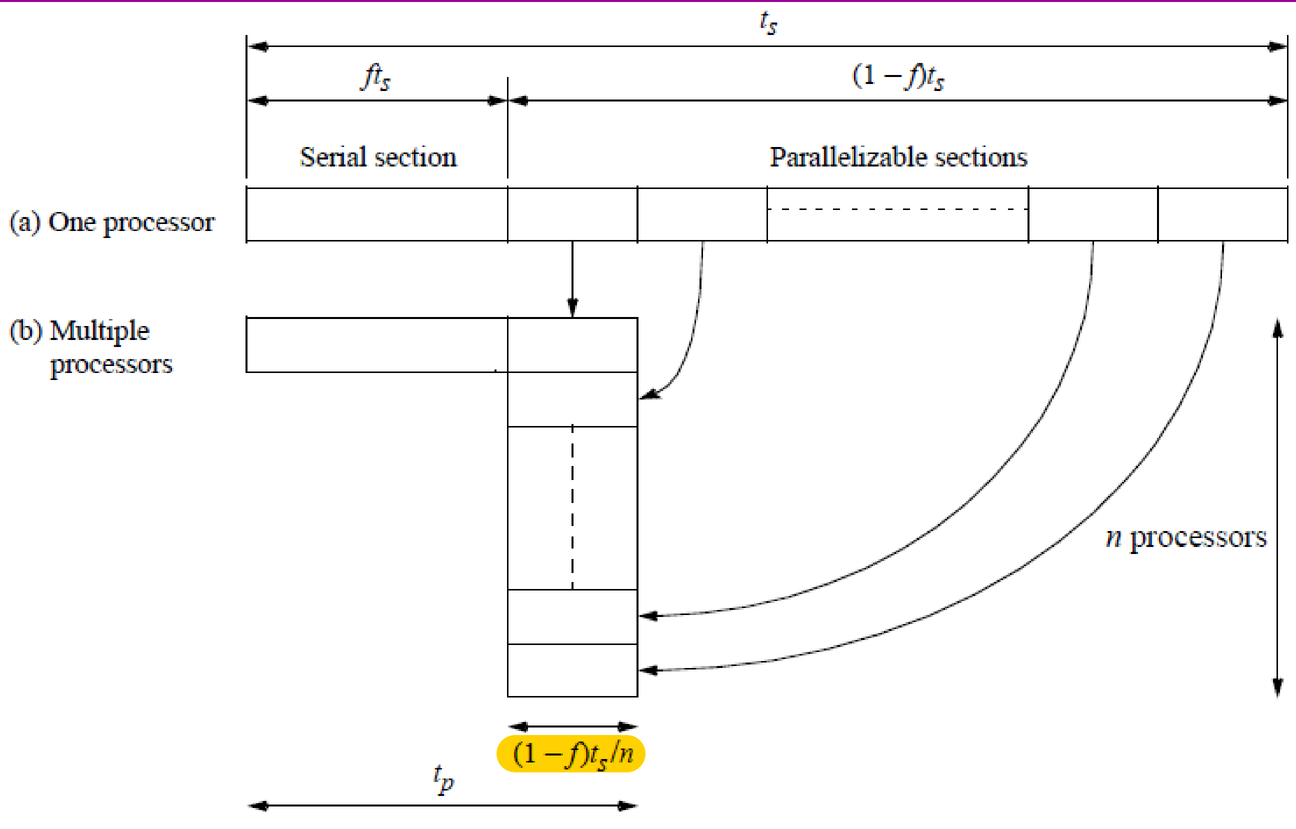


Figure 1.29 Parallelizing sequential problem — Amdahl’s law.

Amdahl's Law (2)

Speedup factor is given by

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

This equation is known as *Amdahl's law*

Amdahl's Law (3)

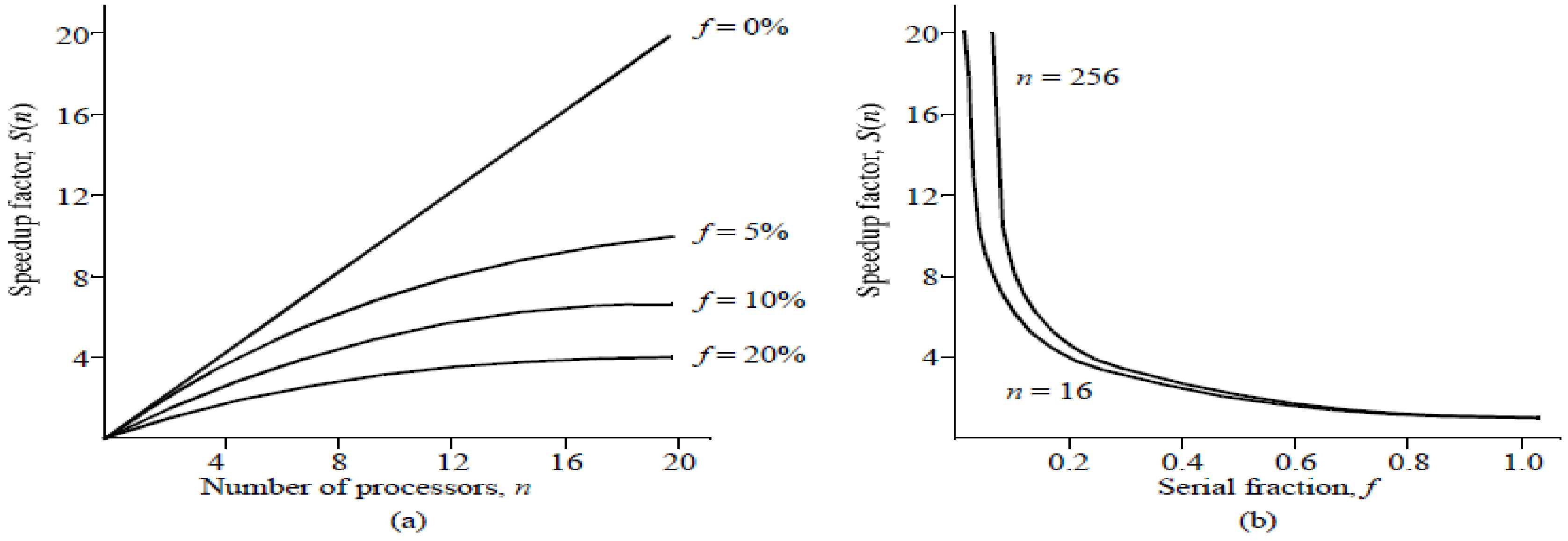


Figure 1.30 (a) Speedup against number of processors. (b) Speedup against serial fraction, f.

Even with infinite number of processors, maximum speedup limited to $1/f$. For example, with only 5% of computation being serial, maximum speedup is 20, irrespective of number of processors.

Efficiency

$$E = \frac{\text{Execution time using one processor}}{\text{Execution time using a multiprocessor} \times \text{number of processors}}$$

$$= \frac{t_s}{t_p \times n}$$

which leads to

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

when E is given as a percentage.

Efficiency gives fraction of time that processors are being used on computation.

Cost

The *processor-time* product or *cost* (or *work*) of a computation defined as

$$\text{Cost} = (\text{execution time}) \times (\text{total number of processors used})$$

The cost of a sequential computation is simply its execution time, t_s. The cost of a parallel computation is $t_p \times n$. The parallel execution time, t_p, is given by $t_s/S(n)$. Hence, the cost of a parallel computation is given by

$$\text{Cost} = \frac{t_s n}{S(n)} = \frac{t_s}{E}$$

Cost-Optimal Parallel Algorithm

One in which the cost to solve a problem on a multiprocessor is proportional to the cost (i.e., execution time) on a single processor system.

Scalability

Used to indicate a hardware design that allows the system to be increased in size and in doing so to obtain increased performance - could be described as *architecture* or *hardware scalablity.*

Scalability is also used to indicate that a parallel algorithm can accommodate increased data items with a low and bounded increase in computational steps - could be described as *algorithmic scalablity.*

Problem Size

Intuitively, we would think of the number of data elements being processed in the algorithm as a measure of size.

However, doubling the problem size would not necessarily double the number of computational steps. It will depend upon the problem.

For example, adding two matrices has this effect, but multiplying matrices does not. The number of computational steps for multiplying matrices quadruples.

Hence, scaling different problems would imply different computational requirements. Alternative definition of *problem size* is to equate problem size with the number of basic steps in the best sequential algorithm.

Gustafson's Law

Rather than assume that the problem size is fixed, assume that the parallel execution time is fixed. In increasing the problem size, Gustafson also makes the case that the serial section of the code does not increase as the problem size.

Scaled Speedup Factor

The scaled speedup factor becomes

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

called *Gustafson's law*.

Example

Suppose a serial section of 5% and 20 processors; the speedup according to the formula is 0.05 + 0.95(20) = 19.05 instead of 10.26 according to Amdahl's law. (Note, however, the different assumptions.)