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 Forecasting

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.

Suppose we consider the 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 are necessary.

If we were 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.

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

Predicting the motion of the astronomical bodies in space.

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

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, there will be $N - 1$ forces to calculate for each body, or approximately N^2 calculations, in total.

After determining the new positions of the bodies, the calculations must be repeated.

A galaxy might have, say, 10^{11} stars. This suggests 10^{22} calculations that have to be repeated.

Even if each calculation could be done in 1μs (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.

Parallel Computers and Programming

Using multiple processors operating together on a single problem.

The overall problem is split into parts, each of which is performed by a separate processor in parallel.

Not a new idea; in fact it is a very old idea.

Gill writes about parallel programming 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 "the continued drive for higher- and higher-performance systems … 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:

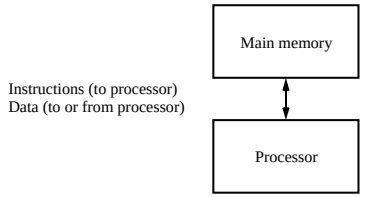


Figure 1.2 Conventional computer having a single processor and memory.

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 System

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:

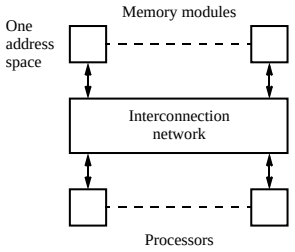


Figure 1.3 Traditional shared memory multiprocessor model.

Programming Shared Memory Multiprocessor

Involves having executable code stored in the memory for each processor to execute. Can be done in different ways:

Parallel Programming Languages

Designed 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 from the programmer's specification.

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 Multicomputer

Complete computers connected through an interconnection network:

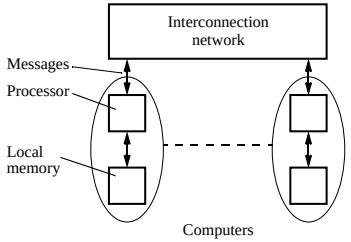


Figure 1.4 Message-passing multiprocessor model (multicomputer).

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.

Shared Virtual Memory,

Gives the illusion of shared memory even when it is distributed.

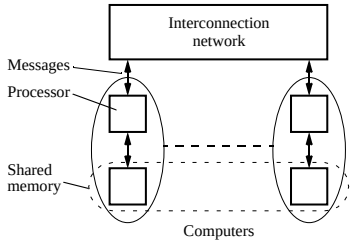


Figure 1.5 Shared memory multiprocessor implementation.

MIMD and SIMD Classifications

In a single processor computer, a single stream of instructions is generated from the program. The instructions operate upon data items.

Flynn (1966) created a classification for computers and called this single processor computer a s*ingle 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 Stream-Multiple Data Stream (SIMD) Computer

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 (MPMD) Structure

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

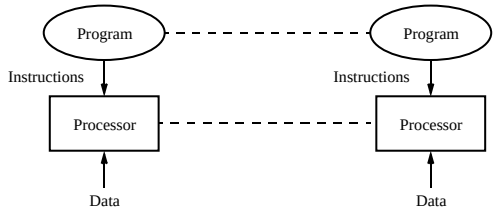


Figure 1.6 MPMD structure.

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.

Architectural Features of Message-Passing Multicomputers

Static Network Message-Passing Multicomputers

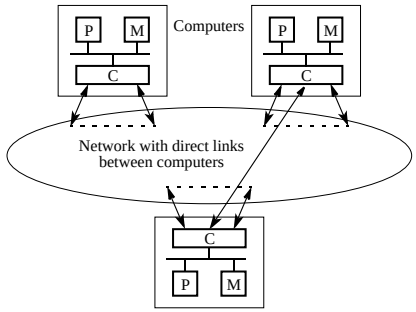


Figure 1.7 Static link multicomputer.

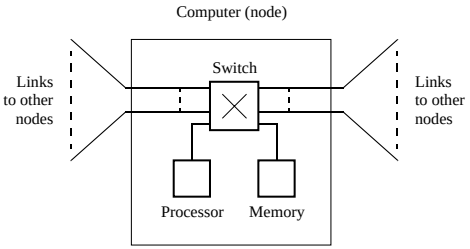


Figure 1.8 Node with a switch for internode message transfers.

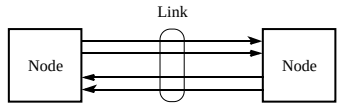


Figure 1.9 A link between two nodes with separate wires in each direction.

Network Criteria

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

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

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

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

Message latency or startup time - the time to send a zero-length message. Essentially the software and hardware overhead in sending a message (finding the route, packing, unpacking, etc.) onto which must be added the actual transmission time to send the data along the link.

Number of links in a path between two nodes is a major factor in determining the delay for a message.

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

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

Interconnection Networks

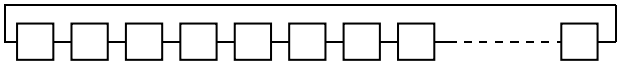


Figure 1.10 Ring.

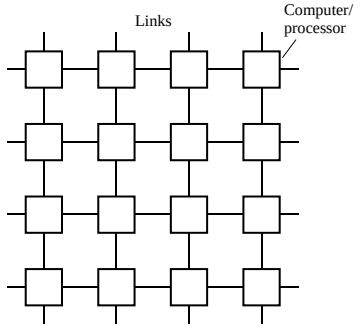


Figure 1.11 Two-dimensional array (mesh).

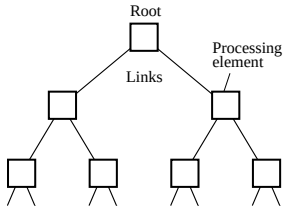


Figure 1.12 Tree structure.

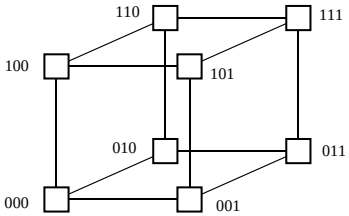


Figure 1.13 Three-dimensional hypercube.

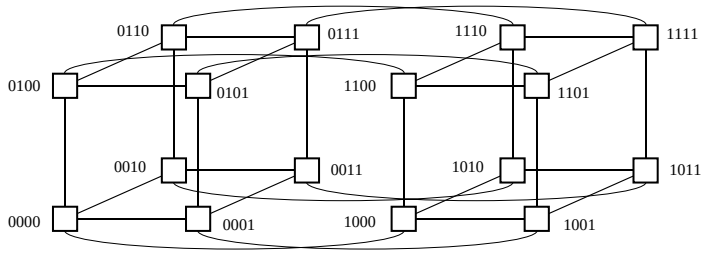


Figure 1.14 Four-dimensional hypercube.

Embedding

As applied to static networks, describes mapping nodes of one network onto another network.

Example - a ring can be embedded in a torus:

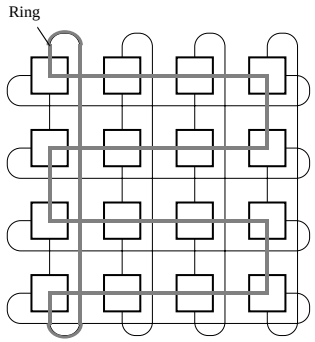


Figure 1.15 Embedding a ring onto a torus.

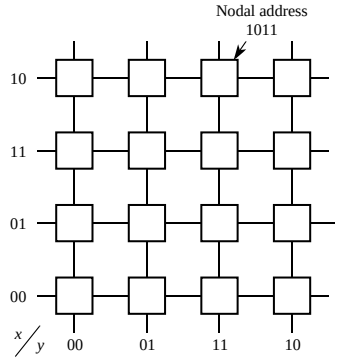


Figure 1.16 Embedding a mesh into a hypercube.

D*ilation* - 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:

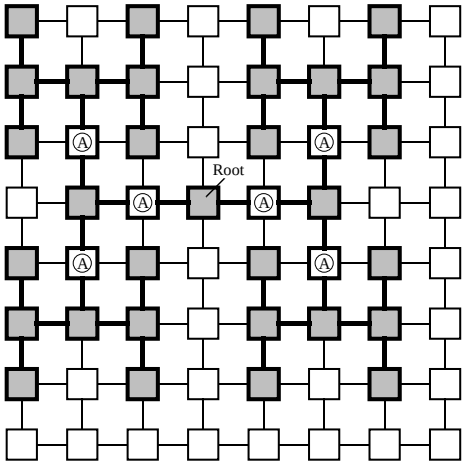


Figure 1.17 Embedding a tree into a mesh.

Communication Methods

Ways that messages can be transferred from a source to a destination.

Circuit Switching

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

A 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 has been used on some multicomputers (for example, the Intel IPSC-2 hypercube system), but it 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.

Packet Switching,

Message is divided into “packets” of information, each of which includes the source and destination addresses for routing the packet through the interconnection network. There is a maximum size for the packet, say 1000 data bytes, and if the message is larger than this, more than one packet must be sent through the network.

Buffers are provided inside nodes to hold packets before they are transferred onward to the next node. A packet remains in a buffer if blocked from moving forward to the next node. This form called *store-and-forward packet switching*.

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

Store-and-forward packet switching 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.

Virtual Cut-Through,

Eliminated storage latency
I
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 the complete path were available, the message would pass immediately through to the destination.

However, if the path is blocked, storage is needed for the complete message/packet being received.

Wormhole routing

Alternative to normal store-and-forward routing to reduce the size of the buffers and decrease the latency.

The message is divided into smaller units called *flits* (flow control digits). A flit is usually one or two bytes. The link between nodes may provide for one wire for each bit in the flit so that the flit can be transmitted in parallel.

Only the head of the message is initially transmitted from the source node to the next node when the connecting link is available.

Subsequent flits of the message are transmitted when links become available, and the flits can become distributed through the network.

When the head flit moves forward, the next one can move forward and so on. A request/acknowledge system is necessary between nodes to “pull” the flits along:

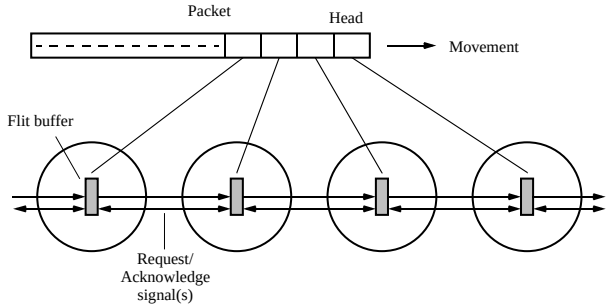


Figure 1.18 Distribution of flits.

A Signaling System

Only requires a single wire between the sending node and receiving node, called *R*/*A* (request/acknowledge).

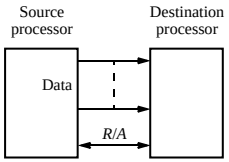


Figure 1.19 A signaling method between processors for wormhole routing (Ni and McKinley, 1993).

R/*A* is reset to a 0 by the receiving node when the receiving node is ready to receive the flit (its flit buffer is empty).

R/*A* is set to a 1 by the sending node when the sending node is about to send the flit.

The sending node must wait for *R*/*A* = 0 before setting it to a 1 and sending the flit.

The sending node knows that the data has been received when the receiving node resets *R*/*A* to a 0.

Message Latency

Suppose the message length is *L*, the bandwidth of the links *B*, and the number of links used is *l*.

Circuit Switching

$$\text{Latency} = \left\lceil \frac{L_c}{B} \right\rceil l + \left\lceil \frac{L}{B} \right\rceil$$

where L_c is the length of control packet sent to establish the path. If $L_c \ll L$, the latency is essentially constant (L/B).

Store-and-forward Packet Switching

$$\text{Latency} = \left\lceil \frac{L}{B} \right\rceil l$$

i.e., a latency proportional to the number of links used.

Virtual Cut-Through

$$\text{Latency} = \left\lceil \frac{L_h}{B} \right\rceil l + \left\lceil \frac{L}{B} \right\rceil$$

where L_h is the length of the header field.

Wormhole

$$\text{Latency} = \left\lceil \frac{L_f}{B} \right\rceil l + \left\lceil \frac{L}{B} \right\rceil$$

where L_f is the length of each flit.

If the length of a flit is much less than the total message, the latency of wormhole routing will be appropriately constant irrespective of the length of the route. (Circuit switching will produce a similar characteristic.) In contrast, store-and-forward packet switching produces a latency that is approximately proportional to the length of the

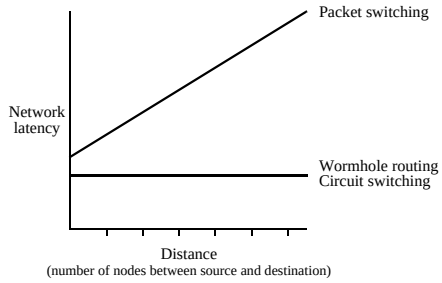


Figure 1.20 Network delay characteristics.

Livelock - occur particularly in adaptive routing algorithms and describes the situation in which a packet keeps going around the network without ever finding its destination.

Deadlock - occurs when packets cannot be forwarded to the 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.

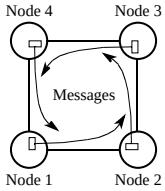


Figure 1.21 Deadlock in store-and-forward networks.

All the messages are blocked because the node buffers are not free to accept packets:

Virtual Channels

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,

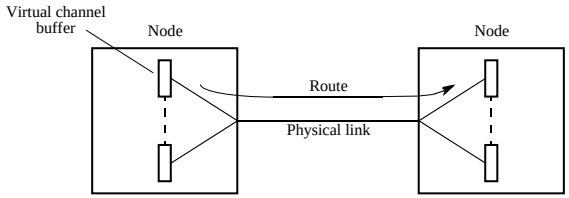


Figure 1.22 Multiple virtual channels mapped onto a single physical channel.

Networked Computers as a Multicomputer Platform

Now widely recognized that 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:

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

Parallel Programming Software Tools for Workstations

Parallel Virtual Machine (PVM) - develped inthe late 1980's. Became very popular.

Message-Passing Interface (MPI) - standard was defined in 1990s.

Ethernet

Common communication network for workstations

Consisting of a single wire to which all the computers attach:

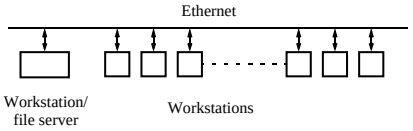


Figure 1.23 Ethernet-type single wire network.

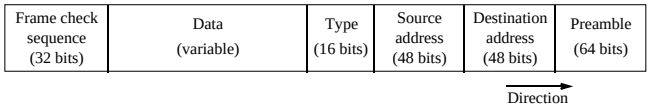


Figure 1.24 Ethernet frame format.

Ring Structures

Examples - token rings/FDDI networks

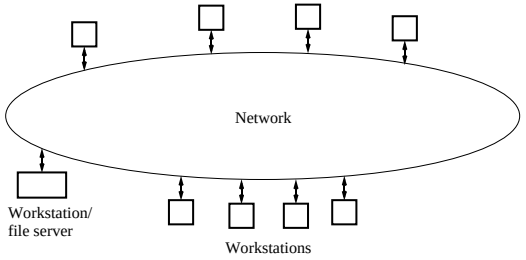


Figure 1.25 Network of workstations connected via a ring.

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.

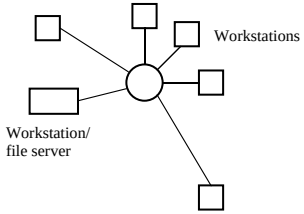


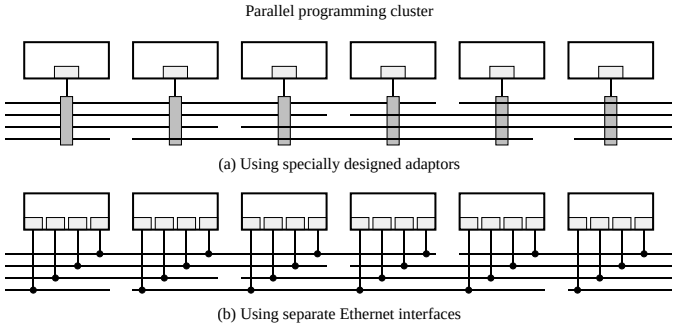
Figure 1.26 Star connected network.

Overlapping Connectivity Networks

Have the characteristic that regions of connectivity are provided and the regions overlap.

There are several ways overlapping connectivity can be achieved;

In the case of overlapping connectivity Ethernets, achieved by having Ethernet segments:



(a) Using specially designed adaptors

(b) Using separate Ethernet interfaces

Figure 1.27 Overlapping connectivity Ethernets.

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 the execution time on a single processor and t_p is the execution time on a multiprocessor. $S(n)$ gives the increase in speed in using a multiprocessor.

For comparing a parallel solution with a sequential solution, we will use the fastest known sequential algorithm for running on a single processor. The underlying algorithm for the parallel implementation might be (and is usually) different.

In a theoretical analysis, speedup factor will also be cast in terms of computational steps:

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

Example

Suppose a parallel sorting algorithm requires $4n$ steps and the best sequential sorting algorithm requires $n \log n$ steps (compare and exchange sorting). The speedup factor would be $(1/4)\log n$.

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

Superlinear 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.

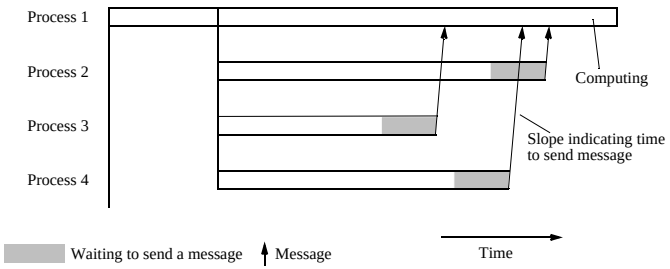


Figure 1.28 Space-time diagram of a message-passing program.

Maximum Speedup

If the fraction of the computation that cannot be divided into concurrent tasks is f, and no overhead incurs when the computation is divided into concurrent parts, the time to perform the computation with n processors is given by $ft_s + (1 - f)t_s/n$, as illustrated below:

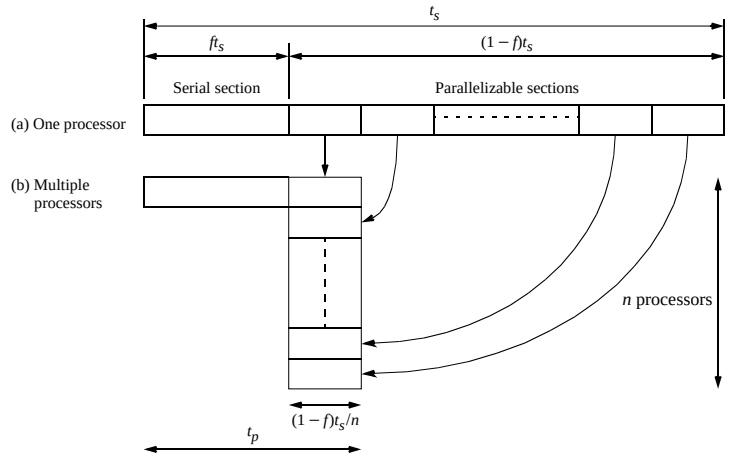


Figure 1.29 Parallelizing sequential problem — Amdahl's law.

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*

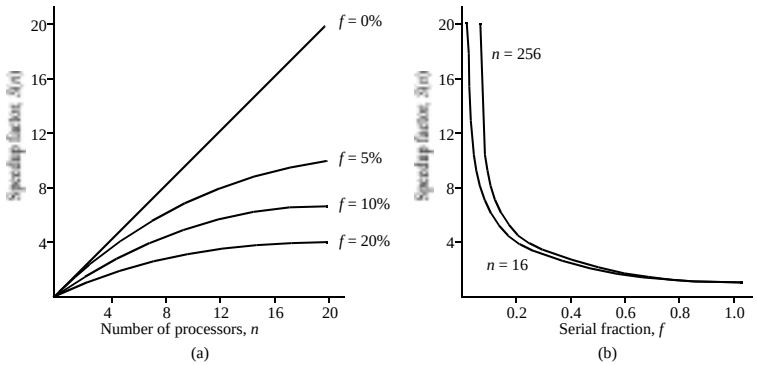


Figure 1.30 (a) Speedup against number of processors. (b) Speedup against serial fraction, *f*.

Even with an infinite number of processors, the maximum speedup is limited to 1/*f*; i.e.,

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

For example, with only 5% of the computation being serial, the maximum speedup is 20, irrespective of the 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 the fraction of the time that the processors are being used on the 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

Combined architecture/algorithmic scalability suggests that increased problem size can be accommodated with increased system size for a particular architecture and algorithm.

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, as discussed in Chapter 10, 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. An 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 speedup factor when the problem is scaled).

Let s be the time for executing the serial part of the computation and p the time for executing the parallel part of the computation on a single processor.

Suppose we fix the total execution time on a single processor, $s + p$, as a constant.

Let the execution time of the parallel computer be a serial part and a parallel part, $s + p$, and the same time as the original serial computation.

For algebraic convenience, let $s + p = 1$.

The scaled speedup factor becomes

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

called *Gustafson's law.*

Gustafson's observation here is that the scaled speedup factor as a function of s is a line of (negative) slope $(1 - n)$ rather than the rapid reduction previously illustrated in Figure 1.30.

For example, suppose we had 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.)

PROBLEMS

1-1. Suppose a galaxy has 10^{11} stars. Estimate the time it would take to perform 100 iterations of the basic O(N^2) *N*-body algorithm using a computer that is capable of 500 MFlops.

1-2. What is the diameter of a torus?

1-3. What is the diameter of a tree network?

1-4. What is the diameter of a *d*-dimensional mesh?

1-5. Determine the route taken in a five-dimensional hypercube network from node 7 to node 22 using the deadlock free *e*-cube routing algorithm described in the chapter. Repeat for an 8×8 mesh, assuming that nodes are numbered in row order (across the rows starting at the top left corner).

1-6. Draw a *k*-ary *n*-cube (a hypercube with *k* processors in each of *n* dimensions) when $k = 4$ and $n = 4$. Determine the appropriate number representation and number the processors. (Recall that a binary hypercube uses $k = 2$ and the binary number representation.)

1-7. Suppose we wanted to embed a 9×9 mesh onto a hypercube. List the *x*- and *y*- coordinates of the mesh.

1-8. Determine how the largest possible complete binary tree can be embedded into a hypercube and a mesh. What is the dilation of your mappings? Embed two disjoint trees into a hypercube. Determine the average distance between two nodes in a mesh network and a hypercube network.

1-9. Determine the communication lower bound for a complete binary tree, based upon diameter and based upon bisection width.

1-10. Suppose *n* numbers are distributed uniformly in a mesh. What is the communication lower bound for a sorting algorithm that distributes the sorted numbers uniformly across the network, based upon diameter and based upon bisection width?

1-11. A multiprocessor consists of 10 processors, each capable a peak execution rate of 200 MFLOPs (millions of floating point operations per second). What is the performance of the system as measured in MFLOPs when 10% of the code is sequential and 90% is parallelizable?

1-12. Suppose the best sequential algorithm for a problem requires $2n \log \sqrt{n}$ steps for *n* data items. What is the minimum number of steps for a parallel algorithm to be cost optimal using n^2 processors?

1-13. It is possible to construct a system that is a hybrid of a message-passing multicomputer and a shared memory multiprocessor. Write a report on how this might be achieved and its relative advantages over a purely message-passing system and a purely shared memory system.

1-14. (Research question) For each of the following interconnection networks, identify one commercial multiprocessor system having that network:

(i) Single bus

(ii) Two-dimensional mesh

(iii) Three-dimensional mesh

(iv) Hypercube

(v) Tree

(vi) Another interconnection network