L1

Grand challenge problems

- large DNA structures, global weather forcasting, motion of astronomical bodies

Shared Memory Multiprocessor Systems

Parallel programming languages

- with special parallel programming constructs and statements that allow shared variables and parallel code sections to be declared.

Threads

- high-level language code sequences for individual processors

Message Passing Multiprocessor Systems

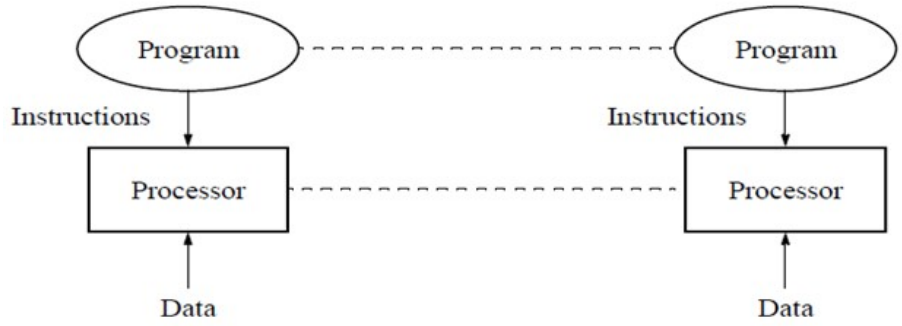
- messages processor
- local memory
- interconnection network

SISD (Flynn分类法)

- calles this single processor Single Instruction Stream-Single Data Stream (SISD)
  - 所有指令都是串行执行，CPU只能处理一个数据流
- Multiple Instruction Stream-Multiple Data Stream (MIMD)
- Single Instruction Streaming-Multiple Data Streaming (SIMD)
  - multiple processors. Each processor executes the same instruction **in synchronism**, but using different data

Multiple Program Multiple Data Streaming (MPMD) Structure

- each processor will have its own program to execute

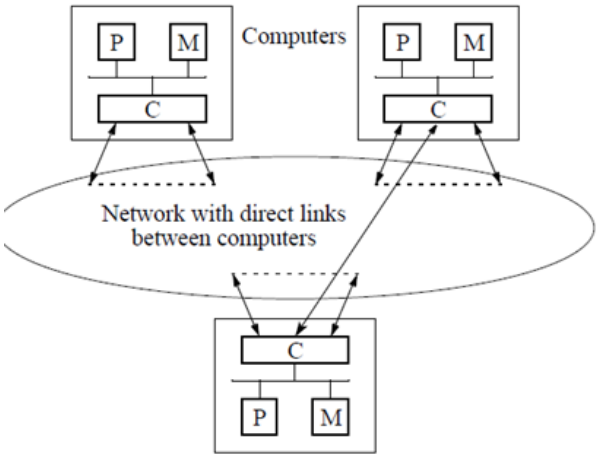


Single Program Multiple Data (SPMD) Structure

- Single source program is written, each processor will execute its personal copy of this program
- **not in synchronism**

Message-Passing Multicomputers

- static network message-passing multicomputers



- node with a switch for internode message transfters

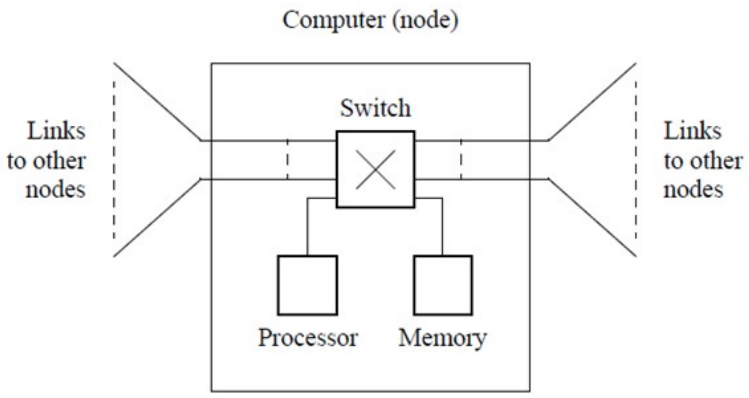


Figure 1.8 Node with a switch for internode message transfers.

- link between two nodes with separate wires in each direction

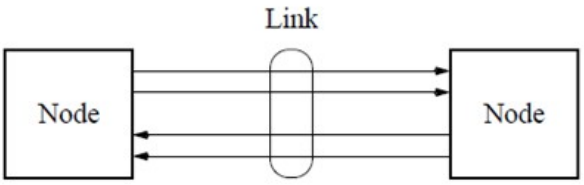


Figure 1.9 A link between two nodes with separate wires in each direction.

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.

Embdding

- describes mapping nodes of one network onto another network
- dilation - used to indicate the quality of embdding, is the maximum number of links in the "embedding" network corresponding to one link in the "embedded" network
  - 详见sub4.pdf

Definition. An embedding of a graph $G = (V, E)$ into a graph $G' = (V', E')$ is a function ϕ from V to V'.

Definition. The dilation of the embedding ϕ is defined as follows. $dil(\phi) = max\{dist(\phi(u), \phi(v)) : (u, v) \in E\}$, where $dist(a, b)$ is the distance in edges between $a, b \in V'$.

Claim 1. A 1d-array can be embedded into a 2d-array with dilation 1.

Claim 2. A ring can be embedded into a 2d-array with dilation 1, if and only if the number of vertices of the array is even.

Communication Methods

Circuit Switching

- establishing path and maintaining all links in path for message to pass. All links are reserved for the transfer until message transfer is complete.

- e.g. simple telephone system: once a telephone connection is made, the connection is maintained until the completion of the telephone call

Packing Switching

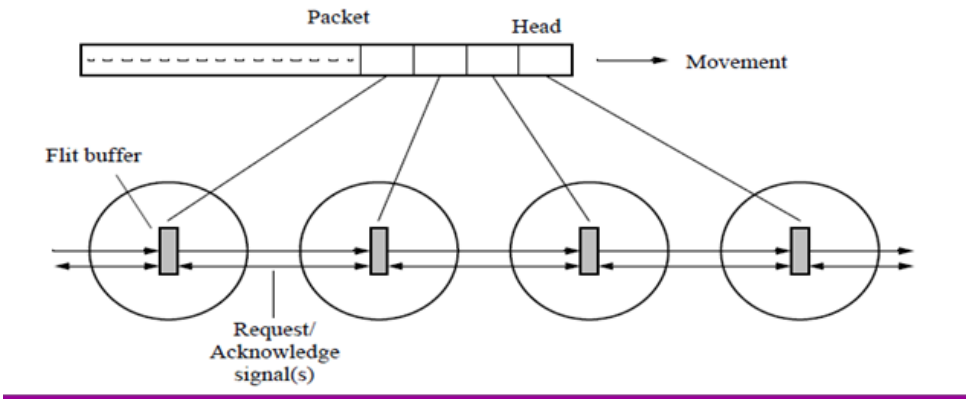
- Message divided into "packets" of information, each of which includes source and destination addresses for routing packet through interconnection network.
- If message is larger than maximum size for the packet (e..g 1000 data bytes), more than one packet must be sent through network.
- Buffers provided inside nodes to hold packets before they are transferred onward to the next node
- *store-and-forward packet switching*
- e.g. mail system. letters moved from mailbox to post office and handled at itermediate sites before delivered to destination

Virtual Cut-Through

- eliminated storage latency
- if outgoing link is available, the message is immediately passed forward without being stored in the nodal buffer; if path is blocked, storage is needed for complete message/packet being received.

Wormhole Routing

- 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.



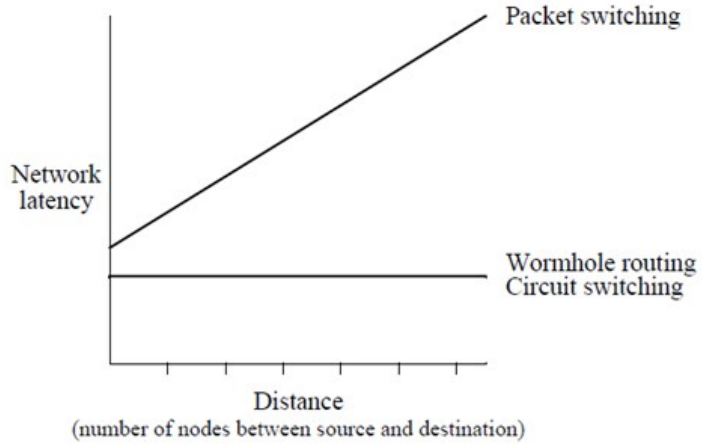
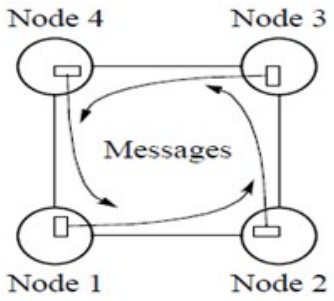


Figure 1.20 Network delay characteristics.

Deadlock

- Occurs when packets can't be forwarded to next node because they are blocked by other packets waiting to be forwarded and these packets are blocked in a similary way such that none of the packets can move
- e.g.



1.21 Deadlock in store-and-forward networks.

Solution 1: Virtual Channel

- Multiple *virtual* channels are associated with a physical channel and time-multiplexed onto the physical channel

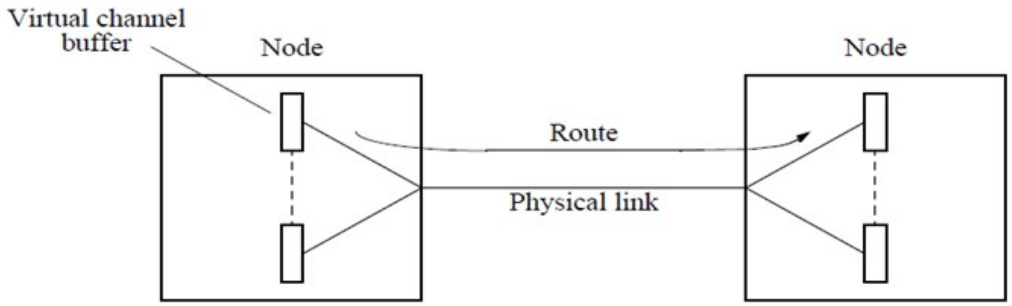


Figure 1.22 Multiple virtual channels mapped onto a single physical channel.

Network

Ring Structures

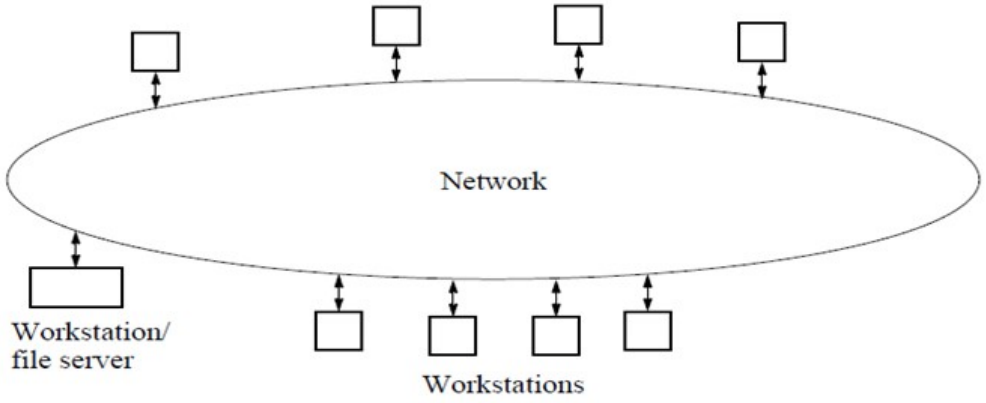


Figure 1.25 Network of workstations connected via a ring.

Point-to-Point Communication

- Provides the highest interconnection bandwidth

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

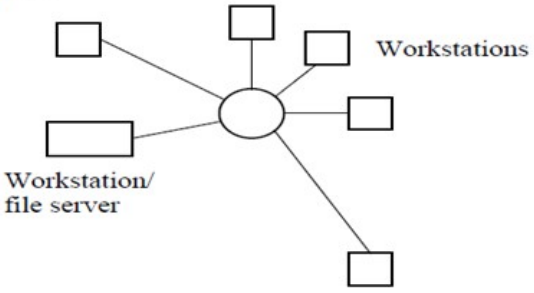


Figure 1.26 Star connected network.

Overlapping Connectivity Networks

- e.g.

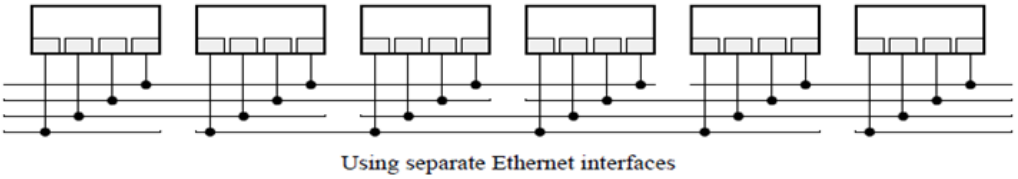


Figure 1.27 Overlapping connectivity Ethernet.

Speedup Factor

- Maximum speedup is n with n processors (linear speedup)

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

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

Amdahl's Law

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

Efficiency

- $E = \frac{Executiontimeusingoneprocessor}{Executiontimeusingamultiprocessor \times numberofprocessors} = \frac{t_s}{t_p \times n}$
  $= \frac{S(n)}{n} \times 100$

Cost

- $Cost = (executiontime)$
  $\times (totalnumberofprocessorsused) = \frac{t_s n}{S(n)} = \frac{t_s}{E}$
  - t_s is the execution time of a sequential computation
  - $t_p \times n$ is the cost of a parallel computation

Scalability

- architecutre scalability: a hardware design that allows the system to be increased in size and in doing so to obtain increased performance
- algorithmic scalability: a parallel algorithm can accommodate increased data items with a low and bounded increase in computational steps

Gustafson' Law

- $S_s(n) = \frac{s+np}{s+p} = s + np = n + (1-n)s$
- e.g. serial section of 5% and 20 processors. 0.05+0.95(20) = 19.05

L2 Message Passing Computing and Programming

Single Program Multiple Data Model (SPMD)

- Different processes are merged into one program.

MPMD Model

- Separate program written for each processors. Master-slave approachusually taken whereby a single processor executes a master process and other processes are started from within the master process

Synchronous Message Passing

- Routines that actually return when the messages transfer has been completed
- Do not need message buffer storage.
- A synchronous send routine could wait until the complete message can be accepted by the receiving process before sending the message
- A s receive routine will wait until the message it is expectig arrives.

Blocking and Nonblocking Message Passing

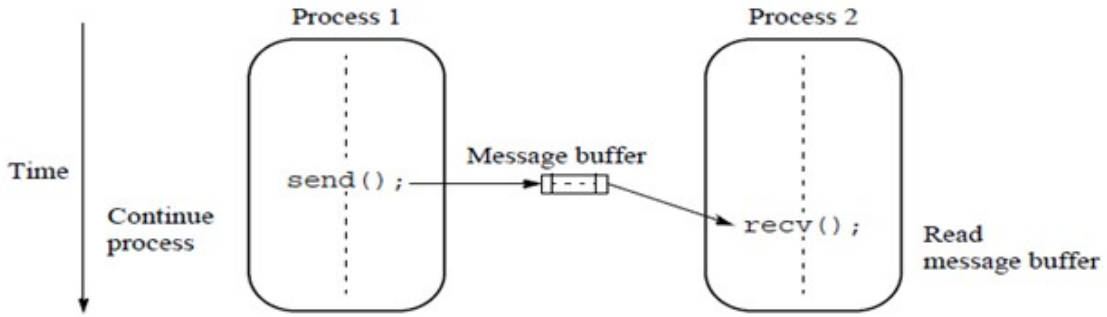
Blocking

- has been used to describe routines that do not return until the transfer is completed
- *synchronous* and *blocking* were synonymous
- In MPI, it is: return after their local actions complete, though the message transfer may not have been completed

Non-blocking

- routines that return whether or not the message had been received
- In MPI, return immediately.
- How message-passing routines can return before the message transfer has been completed
    - generally, a message buffer needed between source and destination to hold message

- for send routine, once the local actions have been completed and the message is safely on its way, the process can continue with subsequent work.
- Buffer 长度一定，满了之后，send routine is held up



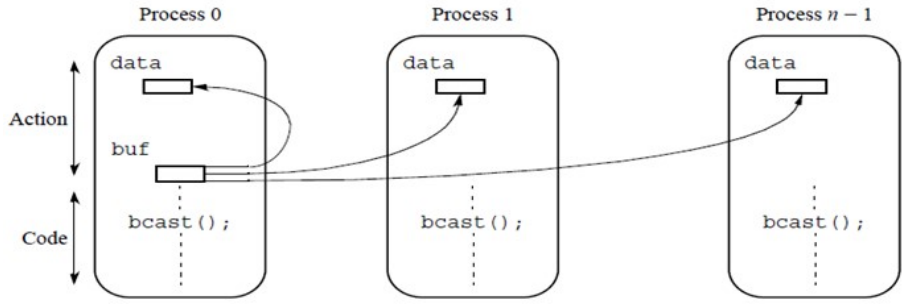
Functions

Message Tag

- If special type matching is not required, a *wild card* message tag is used, so that the rece() will match with any send()
- e.g. message x with tag 5. The message tag is carried within the message
  - send(&x,2,5)
  - recv(&y,1,5)

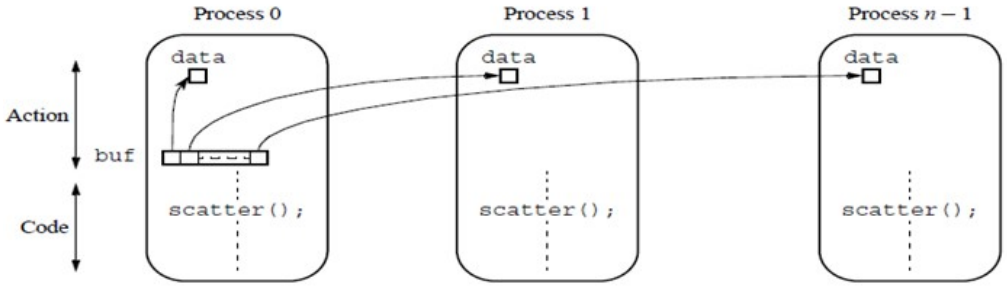
Broadcast

- sending the same message to all the processes concerned with the problem



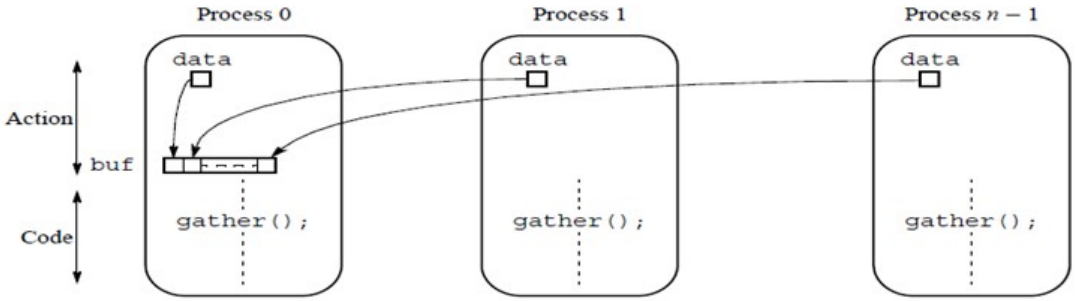
Scatter

- sending each element of an array of data in the root to a separate process. ith location of array is sent to ith process



Gather

- having on process collect individual values from a set of processes



Reduce

- gather operation combined with a specified arithmetic or logical operation

Parallel Virtual Machine (PVM)

- Provides for a software environment for message passing between homogeneous or heterogeneous computes and has a collection of library routines that the user can employ with C or FORTRAN programs
- the set of computers used on a problem first must be defined prior to running the programs
- 最方便的方法是 creating a list of the names of the computers available in a *hostfile*. The hostfile is then read by PVM
- Routing of message between computers is done by PVM daemon processes installed by PVM on the computers that form the virtual machine

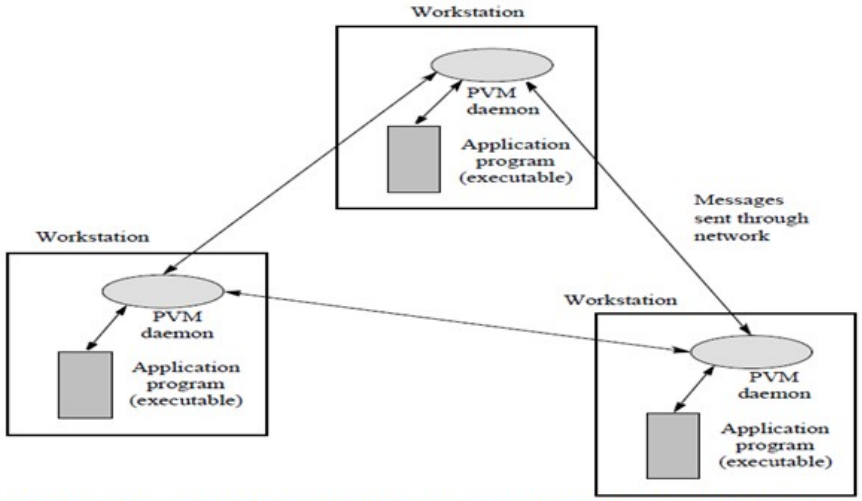


Figure 2.10 Message passing between workstations using PVM.

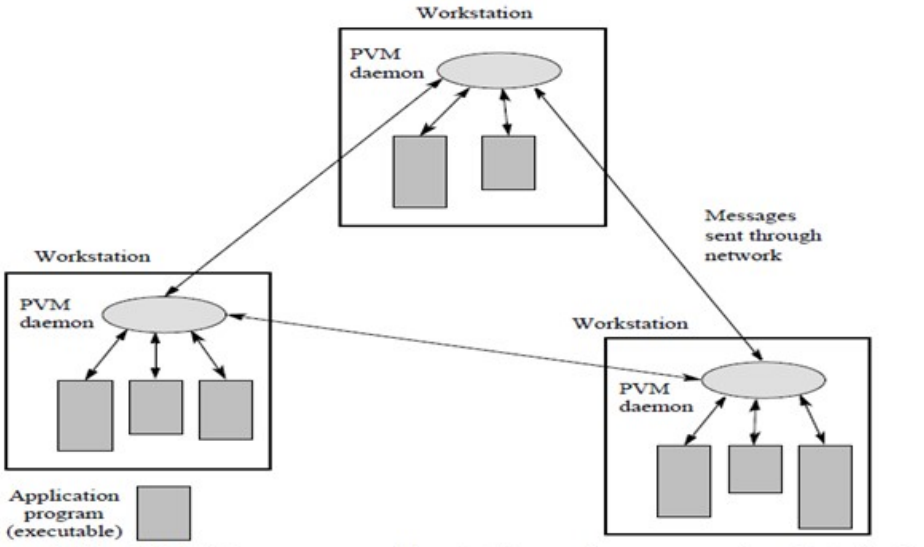


Figure 2.11 Multiple processes allocated to each processor (workstation).

Basic Message-Passing Routines

- All PVM sending routines are nonblocking, receive routines can be either blocking or nonblocking
- If data being sent is a list items of the same data type, the PVM routines $pvm_psend()$ and $pvmprecv()$ can be used

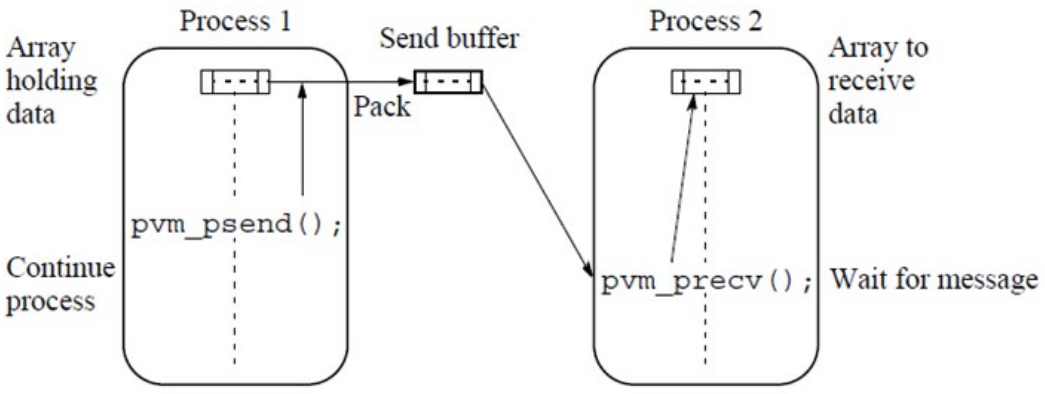


Figure 2.12 `pvm_psend()` and `pvm_precv()` system calls.

- broadcast, scatter, gather, and reduce operations (pvm_bcast(),pvm_scatter()...)use group of processes
- the PVM multicast operation, pvm_mcast(), is not a group operation

MPI

- Only static process creation is supported in version 1. Use SPMD model of computation.

Communication

- communicators: is a *communication domain* that defines a set of processes that are allowed to communicate between themselves
- Types
  - intracommunicator: for communication within a group
  - intercommunicator: for communication between groups
- A process has unique rank in a group, and it could be a member of more than one group

Point-to-Point Communication

- Blocking Routines
  - A blocking send will send the message and return. This does not mean the message has been received, just the process is free to move on without adversely affecting the message
  - MPI_Send(buf, count, datatype, dest, tag, comm)
  - MPI_Recv(buf, count, datatype, src, tag, comm, status)
- Nonblocking Routines
  - send: will return "immediately" even before source location is safe to be altered.
    - MPI_Isend (buf, count, datatype, dest, tag, comm, request)
  - receive: will return even if there is no message to accept
    - MPI_Irecv(buf, count, datatype, source, tag, comm, request)
  - Completion detected by MPI_Wait() and MPI_Test()

Send Communication Modes

Standard Mode Send

Not assumed that corresponding receive routine has started. Amount of buffering not defined by MPI. If buffering is provided, send could complete before receive reached.

Buffered Mode

Send may start and return before a matching receive. Necessary to specify buffer space via routine `MPI_Buffer_attach()` - removed with `MPI_Buffer_detach()`.

Synchronous Mode

Send and receive can start before each other but can only complete together.

Ready Mode

Send can only start if matching receive already reached, otherwise error. Use with care.

Each of the four modes can be applied to both blocking and nonblocking send routines. Only the standard mode is available for the blocking and nonblocking receive routines. Any type of send routine can be used with any type of receive routine.

Collective Communication

- Involves set of processes, defined by an intra-communicator

Time Complexity of MPI Program

Parallel Execution Time

- two parts: a computation part, and a communication part
- $t_p = t_{comp} + t_{comm}$

Communication Time

- $t_{comm} = t_{startup} + nt_{data}$
  - $t_{startup}$ is startup time, sometimes called message latency, assumed constant
  - t_{data} is transmission time to send data word, also assumed a constant

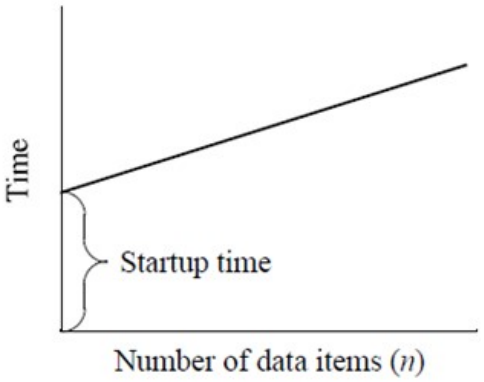


Figure 2.18 Theoretical communication time.

Important Note on Interpretation of Equations

Only intended to give a starting point to how an algorithm might perform in practice.

Parallel execution time, t_p, normalized in units of an arithmetic operation, which will depend upon computer system.

We might find that the computation requires m computational steps so that

$$t_{comp} = m$$

Since we are measuring time in units of computational steps, the communication time has to be measured in the same way.

All data types are assumed to require the same time.

Suppose q messages are sent, each containing n data items. We have

$$t_{comm} = q(t_{startup} + nt_{data})$$

To add n numbers on two computers, where each computer adds $n/2$ numbers together, and the numbers are initially all held by first computer. Second computer submits its result to first computer for adding the two partial sums together. Several phases:

1. Computer 1 sends $n/2$ numbers to computer 2.
2. Both computers add $n/2$ numbers simultaneously.
3. Computer 2 sends its partial result back to computer 1.
4. Computer 1 adds the partial sums to produce the final result.

Computation (for steps 2 and 4):

$$t_{comp} = n/2 + 1$$

Communication (for steps 1 and 3):

$$t_{comm} = (t_{startup} + n/2t_{data}) + (t_{startup} + t_{data}) = 2t_{startup} + (n/2 + 1)t_{data}$$

The computational complexity is $O(n)$. The communication complexity is $O(n)$. The overall time complexity is $O(n)$.

Cost Optimal Algorithms

A *cost-optimal* (or *work-efficient* or *processor-time optimality*) algorithm is one in which the cost to solve a problem is proportional to the execution time on a single processor system (using the fastest known sequential algorithm); i.e.,

$$\text{Cost} = t_p \times n = k \times t_s$$

where k is a constant.

Given time complexity analysis, we can say that a parallel algorithm is cost-optimal algorithm if

(Parallel time complexity) × (number of processors) = sequential time complexity

Example

Suppose the best known sequential algorithm for a problem has time complexity of $O(n \log n)$. A parallel algorithm for the same problem that uses n processes and has a time complexity of $O(\log n)$ is cost optimal, whereas a parallel algorithm that uses n^2 processors and has time complexity of $O(1)$ is not cost optimal.

Time Complexity of Broadcast

on a Hypercube Network

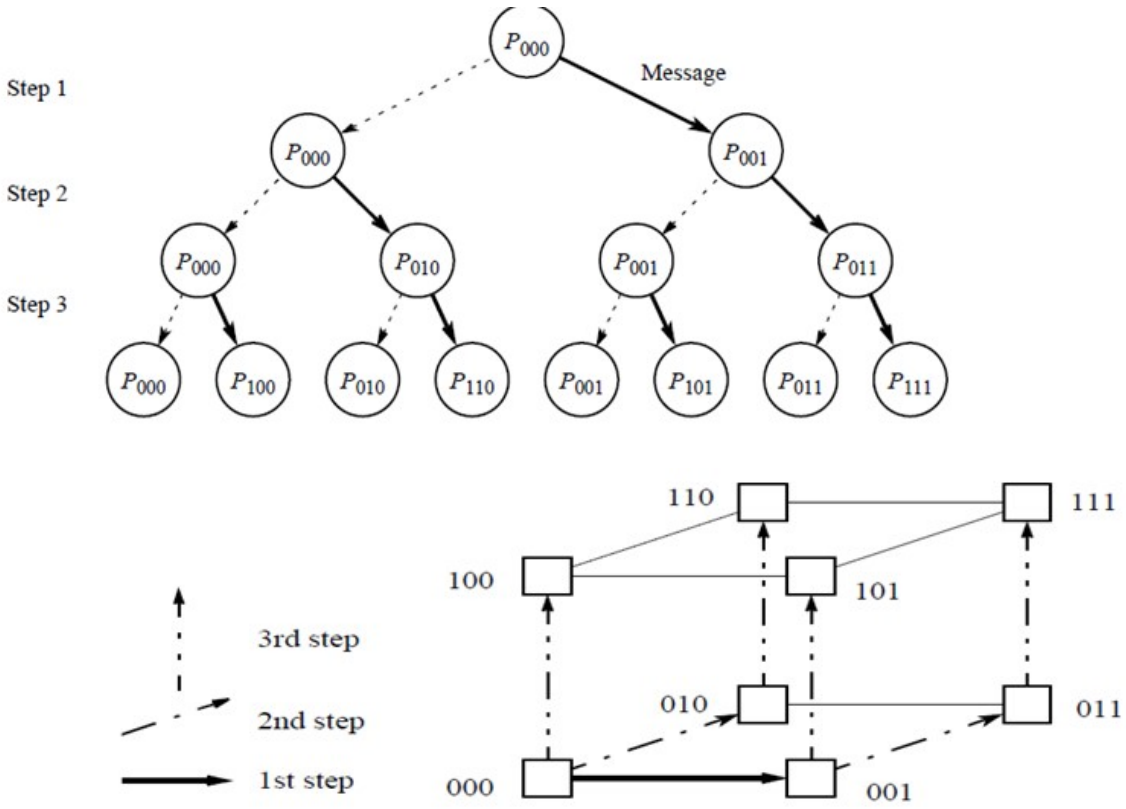


Figure 2.20 Broadcast in a three-dimensional hypercube.

The time complexity for a hypercube system will be $O(\log n)$, using this algorithm, which is optimal because the *diameter* of a hypercube network is $\log n$. It is necessary at least to use this number of links in the broadcast to reach the furthest node.

on a Mesh Network

Send message across top row and down each column as it reaches top of that column.

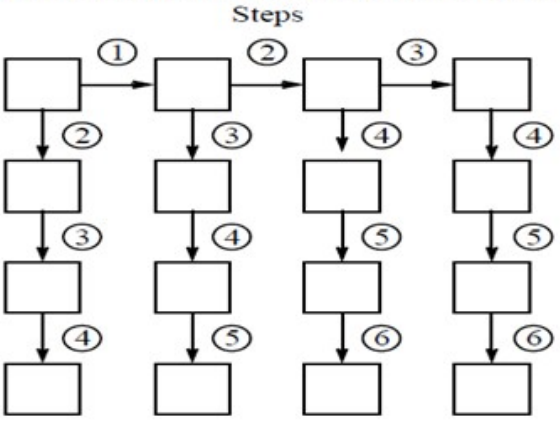
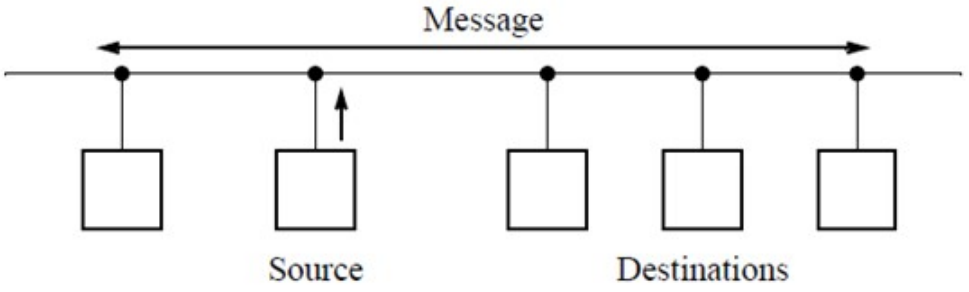


Figure 2.22 Broadcast in a mesh.

Optimal in terms of number of steps because same as diameter of mesh.

on a Workstation Cluster

Broadcast on a single Ethernet connection can be done using a single message that is read by all the destinations on the network simultaneously



Gather on a Hypercube Network

The reverse algorithm can be used to gather data from all nodes to, say, node 000; i.e., for a three-dimensional hypercube,

	Node		Node
1st step:	100	→	000
	101	→	001
	110	→	010
	111	→	011
2nd step:	010	→	000
	011	→	001
3rd step:	001	→	000

In the case of gather, the messages become longer as the data is gathered, and hence the time complexity is increased over O(log n).