1 Message Passing Architecture

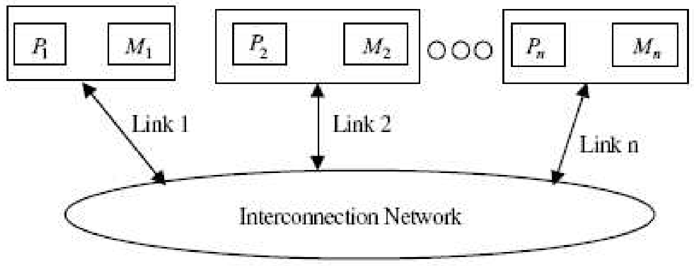


Figure 1: Message passing systems.

- Message passing systems provide alternative methods for communication and movement of data among multiprocessors (compared to shared memory multiprocessor systems, see Fig. 1).
- There is no global memory so it is necessary to move data from one local memory to another by means of message passing (send/receive pairs).
- Each processor has access to its own local memory and can communicate with other processors using the interconnection network. These systems eventually gave way to Internet-connected systems where the processor/memory nodes are cluster nodes, servers, clients, or nodes in a greater grid.
- Aspects of message passing systems;
  - programming model,
  - message routing,
  - network switching,
  - processor support for message passing,
  - examples of message passing systems.

1.1 Introduction to Message Passing

- A message passing architecture is used to communicate data among a set of processors without the need for a global memory.

- The elimination of the need for a large global memory together with its synchronization requirement, gives message passing schemes an edge over shared memory schemes.

- Each processor has its own address space. Nodes communicate with each other by
  - links (called *external channels*)
  - via an *interconnection network*, normally a static-type network.
    - In particular, hypercubes and the nearest-neighbor two-dimensional and three-dimensional mesh interconnection networks have received considerable attention over the years.
    - Two important factors must be considered in designing message passing interconnection networks:
      - link bandwidth; defined as the number of bits that can be transmitted per unit of time (bits/s).
      - network latency; defined as the time to complete a message transfer through the network.

- In executing a given application program, the program is divided into concurrent processes; each is executed on a separate processor.

- If the number of processes is larger than the number of processors, then more than one process will have to be executed on a processor in a time-shared fashion.

- Processes running on a given processor use what is called *internal channels* to exchange messages among themselves.

- Processes running on different processors use the external channels to exchange messages (see Fig. 2, in this figure, a horizontal line represents the execution of each process and lines extended among processes represent messages exchanged among these processes.).

- An important advantage of this form of data exchange is the elimination of the need for synchronization constructs, such as semaphores, which results in performance improvement.

- In addition, a message passing scheme offers flexibility in accommodating a large number of processors in addition to being readily scalable.

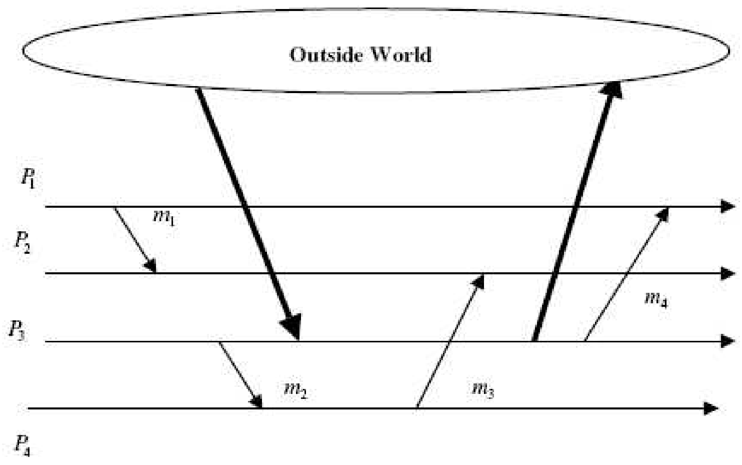


Figure 2: An example of a message passing system.

- A message is defined as a logical unit for internode communication; it is considered as a collection of related information that travels together as an entity. A message can be
  - an instruction,
  - data,
  - synchronization,
  - interrupt signals.
- A message passing system interacts with the outside world by receiving input message(s) and/or outputting message(s). It is essential that the outside world perceives a consistent behavior of a given message passing system.
- *Process Granularity;* The size of a process in a message passing system can be described by a parameter called process granularity.

$$Process\ Granularity = \frac{computation\ time}{communication\ time}$$

  Three types of granularity can be distinguished. These are:

  1. *Coarse granularity:* Each process holds a large number of sequential instructions and takes a substantial amount of time to execute.

2. *Medium granularity:* Since the process communication overhead increases as the granularity decreases, medium granularity describes a middle ground where communication overhead is reduced.
3. *Fine granularity:* Each process contains a few sequential instructions (as few as just one instruction).

Message passing multiprocessors uses mostly medium or coarse granularity.

1.2 Routing in Message Passing Networks

- Routing is defined as the techniques used for a message to select a path over the network channels.
- The identification of a set of permissible paths that may be used by a message to reach its destination, and a function, ν, that selects one path from the set of permissible paths.
- A routing technique is said to be *adaptive* if, for a given source and destination pair, the path taken by the message depends on network conditions, such as network congestion.
- Contrary to adaptive routing, a *deterministic* routing technique, determines the path using only the source and destination, regardless of the network conditions.
  - Although simple, deterministic routing techniques make inefficient use of the bandwidth available between the source and destination.
- Routing techniques can also be classified based on the method used to make the routing decision as
  - centralized (self), the routing decisions regarding the entire path are made before sending the message. Centralized routing requires complete knowledge of the status of the rest of the nodes in the network.
  - distributed routing, each node decides by itself which channel should be used to forward the incoming message. Distributed routing requires knowledge of only the status of the neighboring nodes.

- Examples of the *deterministic* routing algorithms include the e-cube or dimension order routing used in the mesh and torus multicomputer networks and the XOR routing in the hypercube.

1.2.1 Routing for Broadcasting and Multicasting

- There are two types of communication operations in message passing systems,
    - one-to-one (point-to-point or unicast), a node is allowed to communicate a message to only a single destination, which may be its immediate neighbors.
    - collective communications, a number of routing operations are defined under collective communication.
        - Among these, broadcast and multicast are the most widely used.
        - In broadcast, also known as the one-to-all operation, one node sends the same message to all other nodes. Broadcast is mainly used to distribute data from one node to others during computation of a distributed memory program.
        - In multicast, also known as the one-to-many operation, one node sends its messages to k distinct destinations. Multicast has several uses in large-scale multiprocessors, including parallel search algorithms and single program multiple data (SPMD) computation.
- Practical broadcast and multicast routing algorithms must be deadlock-free and should transmit the message to each destination node in as little time and using as short a path as possible.
- One technique to achieve this is to deliver the message along a common path to as many destinations as possible
    - and then replicate the message and forward each copy on a different channel band for a unique set of destination nodes.
    - The path followed by each copy may further branch in this manner until the message is delivered to every destination node.
    - In such a tree-based communication model, the destination set is partitioned at the source and separate copies are sent on one or more outgoing links.

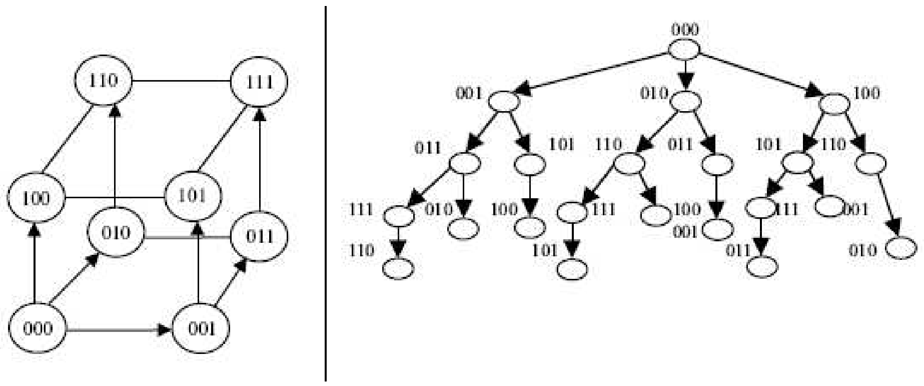


Figure 3: Hypercube broadcast tree-based communication.

  - A message may be replicated at intermediate nodes and forwarded along multiple outgoing links towards disjoint subsets of destinations.

- Another method to implement a multicast operation uses separate addressing.
  - In this case, a separate copy of the message is sent directly from the source to every destination. Clearly, this is an inefficient technique.

A hypercube broadcast tree-based nearest-neighbor communication is shown in Fig. 3.

1.2.2 Routing Potential Problems

- A number of possible problems can result from the use of certain routing mechanisms in message passing systems. These include *deadlock*, *livelock*, and *starvation*.
- **Deadlock;** When two messages each hold the resources required by the other in order to move, both messages will be blocked. This is called a deadlock.
- Resources must be allocated in a manner that avoids deadlock. A straightforward, but inefficient, way to solve the deadlock problem is to allow rerouting (maybe discarding) of the messages participating in a deadlock situation.

  - Rerouting of messages gives rise to non-minimal routing, while discarding messages requires that messages be recovered at the source and retransmitted. This preemptive technique leads to long latency and, therefore, is not used by most message passing networks.
  - A more common technique is to avoid the occurrence of deadlock. This can be achieved by ordering network resources and requiring that messages request use of these resources in a strict monotonic order. This restricted way for using network resources prevents the occurrence of circular wait, and hence prevents the occurrence of deadlock.

- **Livelock;** Livelock describes a situation in which a message keeps going around the network and never reaches its destination.
- It is a phenomenon that results from using *adaptive* routing algorithms where messages are rerouted in the hope to find another path to their destinations.
- When nodes need to communicate, they inject their messages into the network.
  - A static injection model results when all nodes inject their messages at the same moment, with the network clear of messages.
  - This is to be compared to dynamic injection, according to which nodes can inject their messages at arbitrary times. Livelock can take place if dynamic injection is used. It cannot occur if static injection is used.
- A number of routing policies can be used to avoid livelock. They are based on the following. Let S be a set of priorities that is totally ordered. Whenever a message is injected into the network, some priority is assigned to it. In order to avoid livelock, the following must hold.
  - Messages are routed according to their priorities;
  - Once a message has been injected, only a finite number of messages will be injected with higher or equal priority.
- **Starvation;** A node is said to suffer from starvation if it has a message to inject into the network but is never allowed to do so.
  - Starvation cannot arise if static injection is used.

- A number of routing policies can be used in order to avoid starvation taking place.
    * The simplest among them is to allow each node to have its injection queue, where it stores the messages it wants to inject into the network.
    * This queue is considered in the same way as the queues of the incoming links to that node and it competes with them.
    * As long as a fair queue management policy is used, this method prevents starvation from happening.
    * The main disadvantage is that a node with a high message injection rate can slow down all the other nodes in the network.

1.3 Switching Mechanisms in Message Passing

- Switching mechanisms refer to the mechanisms used to remove data from an input channel and place it on an output channel.
- Network latency is highly dependent on the switching mechanism used. A number of switching mechanisms have been in use.
    - store-and-forward,
    - circuit-switching,
    - virtual cut-through,
    - wormhole,
    - pipelined circuit-switching.
- In *circuit-switching* networks, the path between the source and destination is first determined, all links along that path are reserved, and no buffers are needed in each node. After data transfer, reserved links are released for use by other messages.
    - An important characteristic of the circuit-switching technique is that the source and destination are guaranteed a certain bandwidth and maximum latency when communication is established between them.
    - This static bandwidth allocation regardless of the actual use is the main drawback of the circuit-switching approach.

  - However, static bandwidth allocation leads to a simple buffering strategy. In addition, circuit-switching networks are characterized by having the smallest amount of delay.
  - This is because message routing overhead is only needed when the circuit is set up; subsequent messages suffer no, or minimal, additional delay.
  - Therefore, circuit-switching networks can be advantageously used in the case of a large number of message transfers.

- The *store-and-forward* switching mechanism provides an alternate data transfer scheme. The main idea is to offer dynamic bandwidth allocation to messages as they flow through the network, thus avoiding the main drawback of the circuit-switching mechanism.
  Two main types of store-and-forward networks are common. These are packet-switched and virtual cut-through networks.
  - In packet-switched networks,
    * Each message is divided into smaller fixed size parts, called *packets*, before being transmitted.
    * Each node must contain enough buffers to hold received packets before transmitting them.
    * A complete path from source to destination may not be available at the start of transmission. As links become available, packets are moved from node to node until they reach the destination node.
    * Since packets are routed separately through the network, they may follow different paths to the destination node.
    * This may lead to packets arriving out of order at the destination. Therefore, an end-to-end message assembly scheme is needed, incurring additional overhead.
    * Packet-switched networks suffer also from the need for routing overhead for each packet, rather than message, sent into the network.
    * In addition to dynamically allocating bandwidth, packet-switched networks have the advantage of reduced buffer requirements in each node.
  - In virtual cut-through, a packet is stored at an intermediate node only if the next required channel is busy.

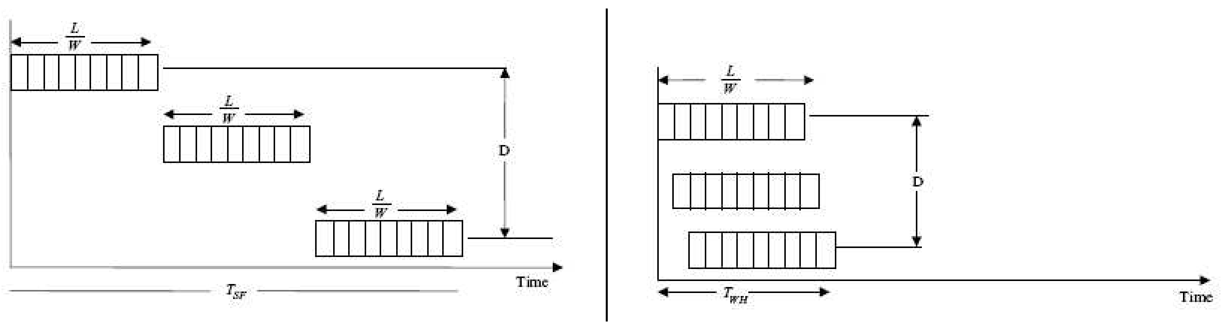


Figure 4: Communication latencies in the store-and-forward (SF) and wormhole (WH) techniques.

        * Virtual cut-through is similar to the packet-switching technique, with the following difference. In contrast to packet switching, when a packet arrives at an intermediate node and its selected outgoing channel is free, the packet is sent out to the adjacent node towards its destination before it is completely received.
        * Therefore, the delay due to unnecessary buffering in front of an idle channel is avoided.

- In order to reduce the size of the required buffers and decrease the incurred network latency, a technique called *wormhole* routing has been introduced. Here, a packet is divided into smaller units called flits (flow control bits).

- These flits move in a pipeline fashion with a header flit leading the way to the destination node.

- When the header flit is blocked due to network congestion, the remaining flits are also blocked.

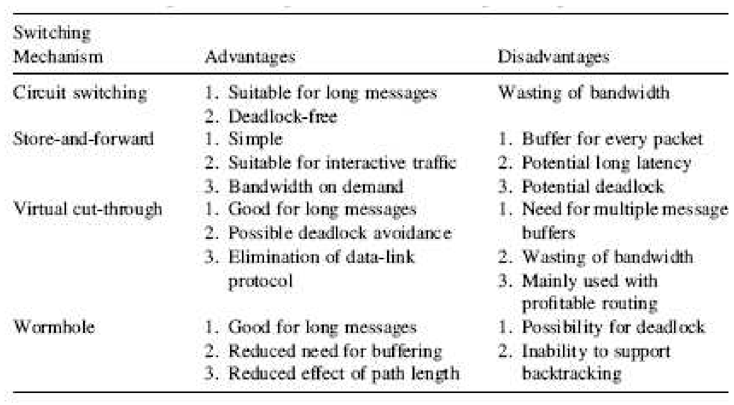
- Only a buffer that can store a flit is required for a successful operation of the wormhole routing technique. The technique is known to produce a latency that is independent of the path length and it requires less storage at all nodes compared to the store-and-forward packet-switching technique. Fig. 4 illustrate the difference in performance between the store and-forward (SF) and wormhole (WH) routing in terms of communication latency.

- In these figures, L represents the packet length in bits, W represents the channel bandwidth in bits/cycle, D is the number of channels, and

T_c is the cycle time. As can be seen from the figures, the latency of the SF and that of the WH are given respectively by

$$T_{SF} = T_c \langle \frac{L}{W} * D \rangle, \ and \ T_{WF} = T_c \langle \frac{L}{W} + D \rangle,$$

Table 5 shows an overall comparison of a number of switching mechanisms.



Switching Mechanism	Advantages	Disadvantages
Circuit switching	1. Suitable for long messages 2. Deadlock-free	Wasting of bandwidth
Store-and-forward	1. Simple 2. Suitable for interactive traffic 3. Bandwidth on demand	1. Buffer for every packet 2. Potential long latency 3. Potential deadlock
Virtual cut-through	1. Good for long messages 2. Possible deadlock avoidance 3. Elimination of data-link protocol	1. Need for multiple message buffers 2. Wasting of bandwidth 3. Mainly used with profitable routing
Wormhole	1. Good for long messages 2. Reduced need for buffering 3. Reduced effect of path length	1. Possibility for deadlock 2. Inability to support backtracking

Figure 5: Comparison Among a Number of Switching Techniques.

1.4 Processor Support for Message Passing

- Processors that support message passing are those processors that contain the special instructions needed to support interprocess message communications. A number of features are required:

    1. A port is a communication channel. It is a reference object for tasks and threads. Two main operations can be performed on ports: send and receive.
    2. Messages are used as communication among objects. A message is divided into a header and a body. The size of the body is variable while that of the header is fixed. A message holds information exchanged between processes.
    3. Port sets: A task can hold multiple access rights (send and receive) on ports.
        - Multiple tasks can hold send access to a single port.

    - On the other hand, one task can hold receive access at a given time.
    - In port set, a task can have either all or none of the access rights to a group of ports. Ports must be mutually exclusive in the sense that a port cannot be in two different sets at a given time.

- The Intel iPAX 432 uses message passing communications and supports them directly. It also uses port objects that work as a competitor to the path of the message. The processor contains a message queue. A message communication can be arranged depending on the following:
    - Time of arrival (such as the “first-in-first-out”, FIFO);
    - Priority;
    - Deadline within priority.

- The IBM AS/400 supports message passing by having an event object type that contains a field supporting the contents of the message. This field is called the event data field. AS/400 processor operations are send and receive.

1.5 Example Message Passing Architectures

- Examples of message passing machines include Caltech Hypercube, the Inmos Transputer systems, Meiko CS-2, Cosmic Cube, nCUBE/2, iPSC/2, iPSC/860, CM-5. Other recent systems include the IBM Scalable Power Series (IBM POWERparallel 3, SP 3).

- The Caltech Hypercube (the Cosmic Cube) was an n-dimensional hypercube system with a single host, known as the Intermediate Host (IH), for global control. The original system was based on the simple store-and-forward routing mechanism. The system started with a set of routine libraries known as the crystalline operating system (CrOS), which supported C and FORTRAN. The system supported only collective operations (broadcast) to/from the IH. Two years later, the Caltech project team introduced a hardware wormhole routing chip.

- The Cosmic Cube is considered the first working hypercube multicomputer message passing system. The Cosmic cube system has been constructed using 64 node for the Intel iPSC. Each node has 128 KB of dynamic RAM that has parity checking for error detection but no correction. In addition, each node has 8 KB of ROM in order to store the initialization and bootstrap programs. The basic packet size is 64 bits with queues in each node. In this system, messages are communicated via transmissions (send/receive).

- The Meiko Computing Surface CS-1 was the first Inmos Transputer T800-based system. The Transputer was a 32-bit microprocessor with fast task-switching capability through hardware intercommunication. The system was programmed using a communication sequential processes (CSP) language called Occam. The language used abstract links known as channels and supported synchronous blocking send and receive primitives.

- The Intel iPSC is a commercial message passing hypercube developed after the Cosmic Cube. The iPSC/1 used Intel 286 processors with a 287 floating-point coprocessor. Each node consists of a single board computer having two buses, a process bus and I/O bus. Nodes are controlled by the Cube manager. Each node has seven communication channels (links) to communicate with other nodes and a separate channel for communication with the Cube Manager. FORTRAN message passing routines are supported. The software environment used in iPSC1 was called NX1, and has a more distributed processes environment than those included in the Caltech CrOS.

- The nCUBE/2 has up to a few thousand nodes connected in a binary hypercube network. Each node consists of a CPU-chip and DRAM chips on a small double-sided printed circuit board. The CPU chip contains a 64 bit integer unit, an IEEE floating-point unit, a DRAM memory interface, a network interface with 28 DMA channels, and routers that support cut-through routing across a 13-dimensional hypercube. The processor runs at 20 MHz and delivers roughly 5 MIPS or 1.5 MFLOPS.

- The Thinking Machine CM-5 had up to a few thousand nodes interconnected in a hypertree (incomplete fat tree). Each node consists of a 33 MHz SPARC RISC processor chip-set, local DRAM memory, and a network interface to the hypertree and broadcast/scan/prefix control networks. Compared to its predecessors, CM-5 represented a true distributed memory message passing system. It featured two interconnection networks, and Sparc-based processing nodes. Each node has four vector units for pipeline arithmetic operations. The CM-5 programming environment consisted of the CMOST operating system, the

CMMD message passing library, and various array-style compilers. The latter includes CMF, supporting a F90-like SIMD programming style.

- The IBM Scalable POWERparallel 3 (SP 3) is the most recent IBM supercomputer series (1999/2000). The SP 3 consists of 2 to 512 POWER3 Architecture RISC System/6000 processor nodes. Each node has its own private memory and its own copy of the AIX operating system. The POWER3 processor is an eight-stage pipeline processor. Two instructions can be executed per clock-cycle except for the multiply and divide. A multiply instruction takes two clock cycles while a divide instruction takes 13 to 17 cycles. The FPU contains two execution units using double precision (64 bit). Both execution units are identical and conform to the IEEE 754 binary floating-point standard.

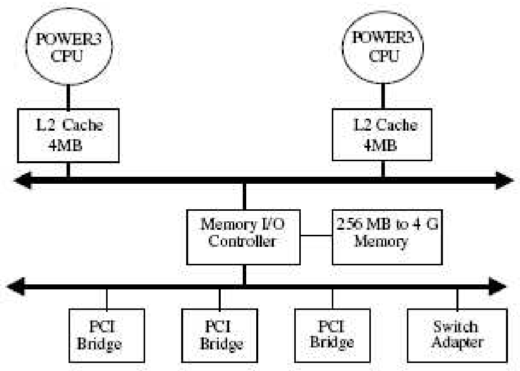


Figure 6: Typical SP 3 node.

  - Figure 6 shows a block diagram of a typical SP 3 node. Nodes are connected by a high-performance scalable packet-switched network in a distributed memory and message passing.
  - The network's building block is a two-staged 16×16 switch board, made up of 4×4 bidirectional crossbar switching elements (SEs).
  - Each link is bidirectional and has a 40 MB/s bandwidth in each direction. The switch uses buffered cut-through wormhole routing. This interconnection arrangement allows all processors to send messages simultaneously.
  - For full connectivity, at least one extra stage is provided. This stage guarantees that there are at least four different paths between every pair of nodes. This form of path redundancy helps in

reducing network congestion as well as recovery in the presence of failures.

  - The communication protocol supports end-to-end packet acknowledgment. For every packet sent by a source node, there is a returned acknowledgment after the packet has reached the destination node. This allows source nodes to discover packet loss. Automatic retransmission of a packet is made if the acknowledgment is not received within a preset time interval.

1.6 Message Passing vs Shared Memory

- Shared memory enjoys the desirable feature that all communications are done using implicit loads and stores to a global address space.

- Another fundamental feature of shared memory is that synchronization and communication are distinct. Special synchronization operations (mechanisms), in addition to the loads and stores operations, need to be employed in order to detect when data have been produced and/or consumed.

- On the other hand, message passing employs an explicit communication model. Explicit messages are exchanged among processors.

- Synchronization and communication are unified in message passing. The generation of remote, asynchronous events is an integral part of the message passing communication model.

- It is important, however, to indicate that shared memory and message passing communication models are universal; that is, it is possible to employ one to simulate the other.

  - However, it is observed that it is easier to simulate shared memory using message passing than the converse.
  - This is basically because of the asynchronous event semantics of message passing as compared to the polling semantics of the shared memory.

- The shared memory communication model allows the programmer to concentrate on the issues related to parallelism by relieving him/her of the details of the interprocessor communication.

- In that sense, the shared memory communication model represents a straightforward extension of the uniprocessor programming paradigm. In addition, shared memory semantics are independent of the physical location and therefore they are open to the dynamic optimization offered by the underlying operating system.

- On the other hand, the shared memory communication model is in essence a polling interface. This is a drawback as far as synchronization is concerned.

- Message passing can be characterized as employing an interrupt-driven communication model. In message passing, messages include both data and synchronization in a single unit. As such, the message passing communication model lends itself to those operating system activities in which communication patterns are explicitly known in advance, for example, I/O, interprocessor interrupts, and task and data migration.

- On the other hand, message passing suffers from the need for marshaling cost, that is, the cost of assembling and disassembling of the message.

- One natural conclusion arising from the above discussion is that shared memory and message passing communication models each lend themselves naturally to certain application domains. Shared memory manifests itself to application writers while message passing manifests itself to operating systems designers.

- It is therefore natural to consider combining both shared memory and message passing in general-purpose multiprocessor systems. This has been the main driving force behind systems such as the Stanford FLexible Architecture for SHared memory (FLASH) system. It is a multiprocessor system that efficiently integrates support for shared memory and message passing while minimizing both hardware and software overhead.