UNIT- 5 MULTI-PROCESSOR ORGANIZATION

UNIT STRUCTURE

5.1 LEARNING OBJECTIVES

After going through this unit, you will able to

- know the multiprocessor motivation
- describe the classification of parallel computation
- explain the multiprocessor organizations
- know the shared memory multiprocessors
- illustrate the message-passing multiprocessors
- explain the performance of multiprocessor architecture
- know the future direction of multiprocessor architecture

5.2 INTRODUCTION

Today computers are not used for solving only scientific and military applications but are used in all other areas such as banking, share markets, Universities, reservation such as airline reservations and train reservations. Although kind of data to be processed in each application is different but there is one common factor among the data processed in all these applications. The common factor is data to be processed is huge and in most of the applications there is a symmetry among the data in each applications. For example airline reservation has to process the data that has information about passengers , flight and flight schedules. Thus instead of using high end server like mainframe to process these related large volume of data, we can use multiple programs running on different computers which are interconnected to obtain the same result. This and many more detailed observations listed below give insight into benefit of executing the programs in parallel.

- Many scientific applications such as Modeling of weather patterns, astrophysics, chemical reactions, ocean currents, etc take too long to run on a single processor machine. Many of these are parallel applications which largely consist of loops which operate on independent data. Such applications can make efficient use of a multiprocessor machine with each loop iteration running on a different processor and operating on independent data.

- Many multi-user environments require more compute power than available from a single processor machine. Example of this category of applications are Airline reservation system, department store chain inventory system, file server for a large department, web server for a major corporation, etc. These consist of largely parallel transactions which operate on independent data. In this unit we will basically concentrates on the multiprocessor organization.

5.3 CLASSIFICATION OF PARALLEL COMPUTATION

Michael J. Flynn created one of the earliest classification systems for parallel (and sequential) computers and programs, now known

as Flynn's taxonomy. Flynn classified programs and computers by whether they were operating using a single set or multiple sets of instructions, whether or not those instructions were using a single or multiple sets of data. In this system classifications are based upon the number of concurrent instructions and data streams present in the computer architecture. These classification are

- The single-instruction-single-data (SISD)
- The single-instruction-multiple-data (SIMD)
- Multiple-instruction-single-data (MISD)
- Multiple-instruction-multiple-data (MIMD)

Except SISD, other three category of applications such as SIMD, MISD,MIMD are candidate applications for multiprocessor architecture. A typical multiprocessor architecture performing the parallel processing is shown in the following Fig-5.1.

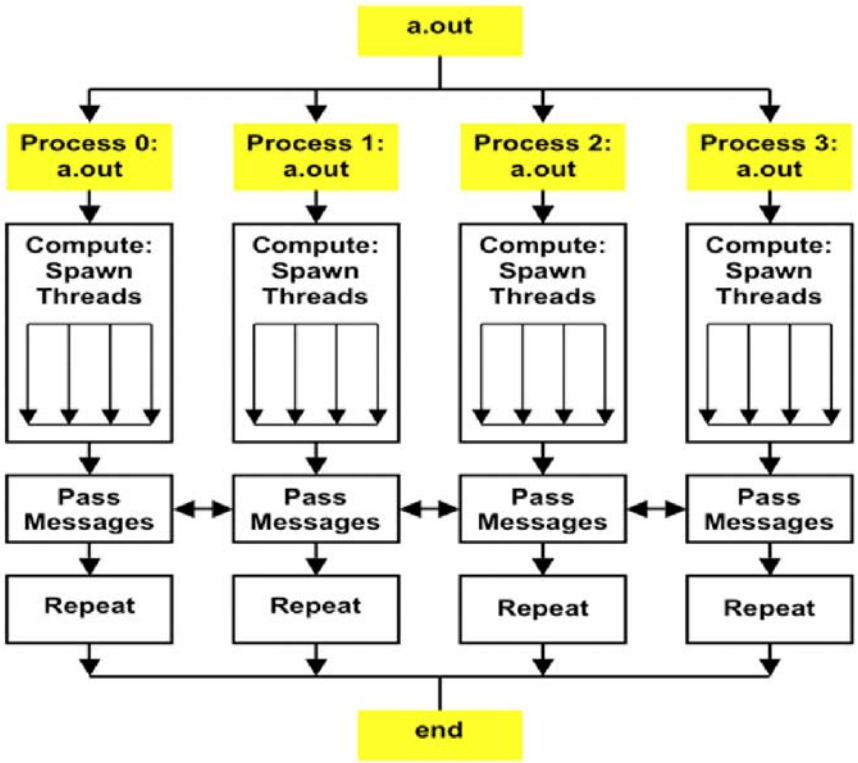

Fig. 5.1 Block diagram for multiprocessor architecture

Instead of executing whole program on a single machine. Execution of a program is partitioned into different subprograms and each of

these subprograms are executed on different machines. During time of execution, programs running on different machines may want to communicate with each other. Finally result from each processor is combined to obtain the desired final result.
Now, we consider in detail the each category of architecture of Flynn's classification.

5.3.1 The Single-Instruction-Single-Data (SISD)

SISD architecture is equivalent to an machine executing program entirely in sequential manner. The program of type calculating a cuberoot of a given number. A single program computes the cuberoot on a given number. A Uniprocessor architecture is best suited for this kind of programs. This is the classic "Von Neumann" architecture dating from 1945, where a control unit coordinates the traditional "Fetch - Execute" cycle, obtaining a single instruction from memory, feeding it to the processor for execution on the data specified within the instruction, then storing the result back into memory. Although a modern computers especially desktop do not adhere completely to this arrangement,they are still classed as sequential rather than parallel processors. Modern computers may use separate data and instruction buses and the use pipelining at a low level to increase the speed of execution but still considered executing the program in sequential fashion. Following block diagram Fig-4.2 shows main functional components of the SISD architecture.

5.3.2 The Single-Instruction-Multiple-Data (SIMD)

SIMD architecture is equivalent to a machine doing the same operation repeatedly over a large data set. This is commonly done in signal processing applications. If we extend the previous program to compute cuberoot for a set of given numbers then that program is classified into SIMD. The architecture is also referred to as an "Array processor", comprising a single control unit and several processing elements which all execute the same instruction at the same time on different data. Machines using this architecture are rather specialized compare to "Von nuemann". These are designed to be

for greater performance in their use when one wants to employ their full parallelism. SIMD perform excellently on digital signal and image processing and on certain types of Monte Carlo simulations. Though limited to a small number of applications such as image processing and the solution of 2D and 3D field problems, the speed up factor is significant in that it is almost directly proportional to the number of processing elements. Fig-5.3 above gives the block diagram for SIMD architecture.

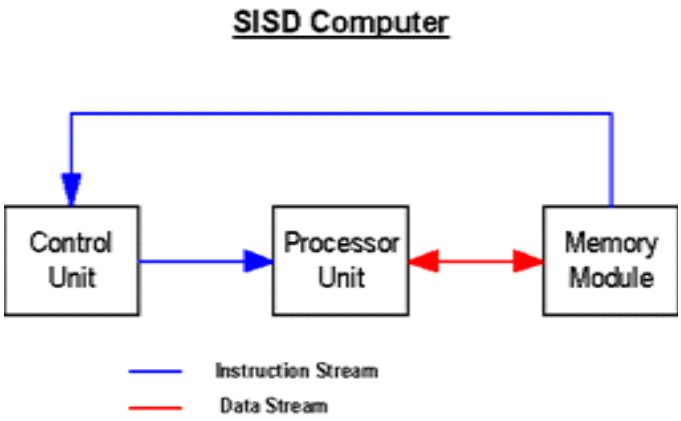

Fig. 5.2 Block diagram of SISD architecture

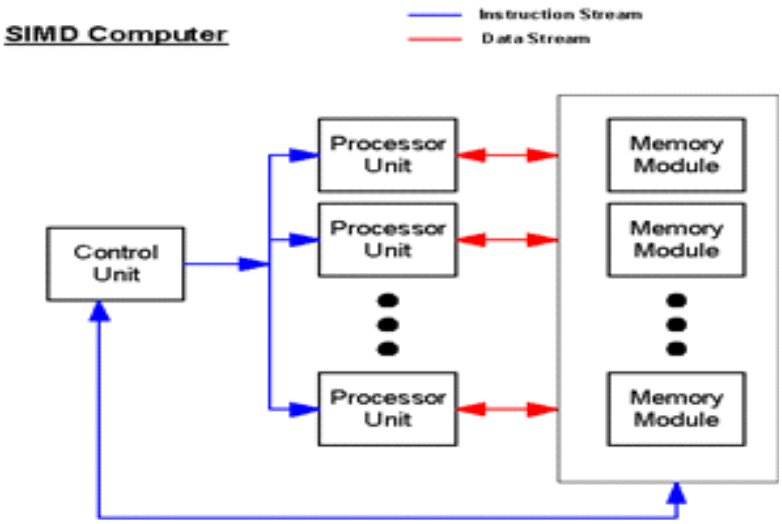

Fig. 5.3 Block diagram of SIMD architecture

5.3.3 Multiple-Instruction-Single-Data (MISD)

Conceptually there are a number of processing elements operating different instructions on the same data. Though this type of processor can be conceptually illustrated, such that each processor output becomes the input to the next in a form of pipe, there are very few actual applications currently using this architecture. It has been argued that Systolic Array processing may fit this criteria. Another applications such as stream based processing may fit into this class. Fig-5.4 gives the block diagram for MISD architecture.

5.3.4 Multiple-Instruction-Multiple-Data (MIMD)

MIMD programs are by far the most common type of parallel programs. If we extend the program of SIMD to compute cube root for different set of numbers, wherein each set contains different data types. One set may contain integer data type and other set may contain floating-point numbers. In this architecture each processor has it's own set of instructions which are executed under the control of a Control Unit associated within that processor. Usually each processor will have its own local memory upon which the instructions operate. Thus synchronization can only be achieved by the provision of explicit interprocessor mechanisms, which provide access to an area of shared memory. Fig-5.5 gives the block diagram for MIMD architecture.

Among four classification two classifications SIMD and MIMD are two architectures which are well suited for parallel processing. We discuss in detail these two classifications and finally we discuss related problem that may arise because of parallel processing and a way of overcoming these problems.

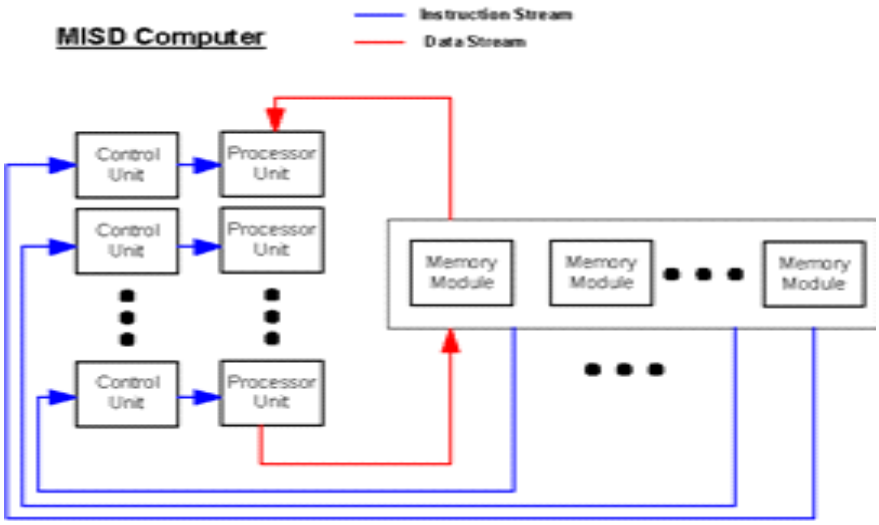

Fig. 5.4 Block diagram of MISD architecture

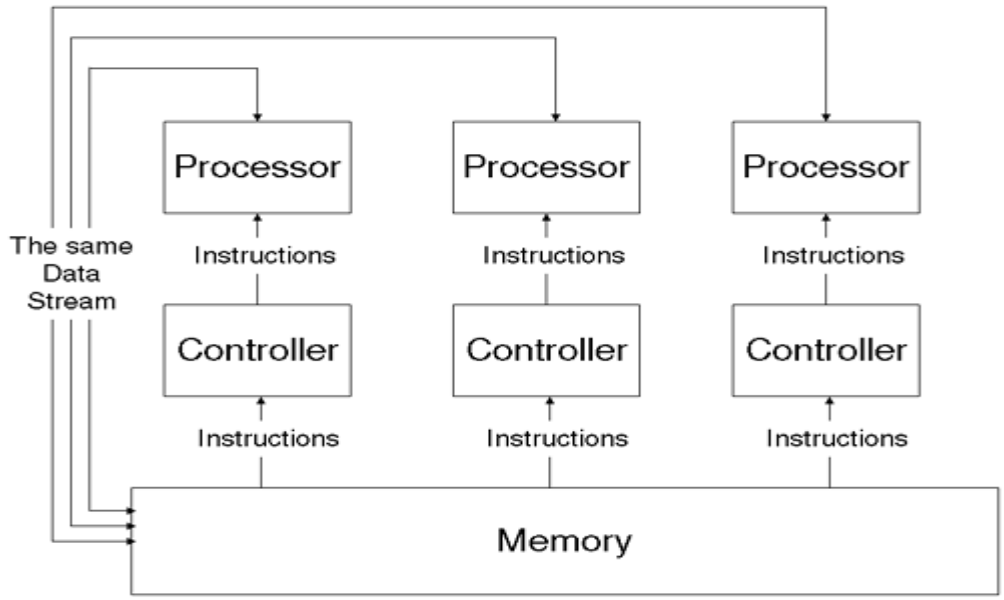

Fig. 5.5 Block diagram of MIMD architecture

5.4 SINGLE-INSTRUCTION-MULTIPLE-DATA (SIMD)

In this section, we will explain SIMD class of architecture. It is important to know that SIMD are mostly designed to exploit the inherent parallelism encountered in matrix (array) operations. Array operations are most required operations in applications such as image processing. There are two main SIMD configurations in parallel processing. These two schemes are shown in Fig-5.6.

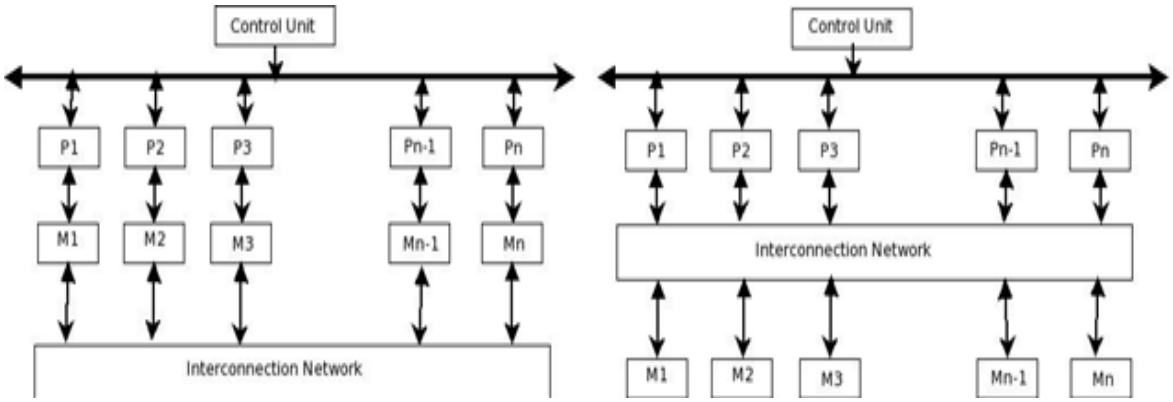

Fig. 5.6 Two SIMD schemes

* The ILLIAC IV was one of the most infamous super computers ever built

** The Burroughs Scientific Processor (BSP),a high-performance computer system combined parallelism and pipelining, performing memory-to-memory operations

In the first scheme, each processor is independent. Independent means each has its own local memory. When processors want to communicate with each other they done through the interconnection network. If the interconnection network does not provide direct connection between a given pair of processors, then this pair can exchange data via an intermediate processor. Super computer ILLIAC-IV* used this kind of inter connection scheme. The interconnection network in the ILLIAC-IV allowed each processor to communicate directly with four neighboring processors in an 8 X 8 matrix pattern such that the i^{th} processor can communicate directly with the $(i-1)^{th}$, $(i+1)^{th}$, $(i-8)^{th}$, and $(i+8)^{th}$ processors. This similar to movement of the soldier in the chess. He can move one step above, below or top, two but not move in diagonal.

In the second SIMD scheme, processors and memory modules communicate with each other via the interconnection network. Two

processors can transfer data between each other via intermediate one or more memory modules or through one or more intermediate processors. We illustrate this with an example. Suppose processor i is connected to memory modules (i-1), i, and (i+1). In this case, processor 1 can communicate with processor 5 via memory modules 2, 3, and 4 as intermediaries. We explain why memory modules 2,3,4 are used. First processors 1 and 3 are connected to memory module 2. Next processor 3 and 4 are connected to memory module 3 . Finally processor 4 and five are connected to memory module 4. Now if we trace movement of data, data was first places into memory module 2 by processor 1. Processor 3 transfer from memory module 2 to memory module 3. Then processor 4 translate from memory module 3 to memory module 4. Finally memory module 4 is read from processor 5. The BSP** (Burroughs' Scientific Processor) used the second SIMD scheme. In order to illustrate the effectiveness of SIMD in handling array operations, consider the operations of adding the corresponding elements of two one dimensional arrays A and B and storing the results in a third one-dimensional array C . Assume also that each of the three arrays has N elements.

Assume SIMD scheme 1 is used. The N additions required can be done in one step if the elements of the three arrays are distributed such that M0 contains the elements A (0), B (0), and C (0), M1 contains the elements A (1), B (1), and C (1), . . . , and MN-1 contains the elements A (N - 1), B (N - 1), and C (N - 1). In this case, all processors will execute simultaneously an add instruction of the form C = A + B . After executing this single step by all processors, the elements of the resultant array C will be stored across the memory modules such that M0 will store C (0), M1 will store C(1), . . . , and MN-1 will store C (N - 1). It is customary to formally represent an SIMD machine in terms of five-tuples (N , C , I , M , F). The meaning of each argument is given below.

1. N is the number of processing elements ($N = 2^k$, $k >= 1$).
2. C is the set of control instructions used by the control unit, for example, *do,for,step .*
3. I is the set of instructions executed by active processing units.
4. M is the subset of processing elements that are enabled.
5. F is the set of interconnection functions that determine the

communication links among processing elements.

5.5 MULTIPLE-INSTRUCTION MULTIPLE-DATA (MIMD)

Multiple-instruction multiple-data streams (MIMD) parallel architectures has multiple processors and multiple memory modules connected together through interconnection network. We can divide MIMD architecture into two broad categories one shared memory based and another message passing. Fig-7 illustrates the general architecture of these two categories. In shared memory there is central shared memory and processors exchange information through this central shared memory. Processors communicate through a bus and cache memory controller. As multiple servers are accessing the shared memory, shared memory need to be an expensive multiported memories supporting simultaneous reads and writes. Some example architectures that use Shared memory for information exchange are Sun Microsystems multiprocessor servers, and Silicon Graphics Inc. multiprocessor servers.

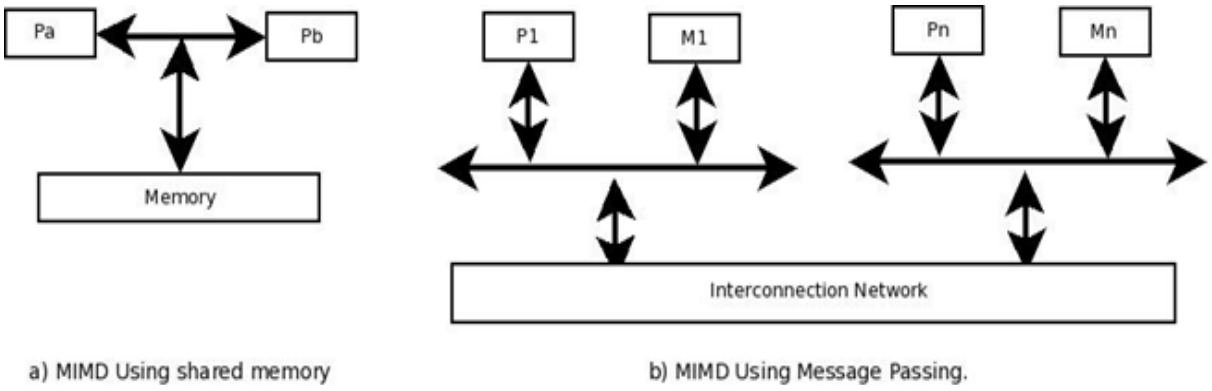

Fig. 5.7 MIMD variants a) Shared memory b) Message Passing

In case of message passing system exchange of information is done by exchange of messages through interconnection network. In message passing system (also referred to as distributed memory) each nodes combines the local memory and processor. Thus there is no global memory. Because of this it is necessary to move data from one local memory to another by means of message passing. Send/ Receive pair of commands are used to achieve the passing message from one node to another node. Usually message passing

system calls are available at application level. Thus message passing mechanism is implemented at the application layer. Programmers wanting to achieve parallel processing using message passing method need to know message-passing paradigm, which involves data copying and dealing with consistency issues. Commercial examples of message passing architectures c. 1990 were the nCUBE, iPSC/ 2, and various Transputer-based systems. Concept of Internet is similar to multiprocessor systems using message passing. In Internet a node is either Internet servers or clients and information is exchanged using messages.

Next question we ask is which among these is better. Distributing memory is one way of efficiently increase the number of processors managed by a parallel and distributed system. Instead if we use centralized memory then by increasing number of processors results in greater conflicts. Thus to scale to larger and larger systems (as measured by the number of processors) the systems had to use distributed memory techniques. These two forces created a conflict: programming in the shared memory model was easier, and designing systems in the message passing model provided scalability.

5.5.1 Shared Memory Organization

In shared memory model all processors communicate with each other by reading and writing locations in a shared memory. The shared memory is made equally accessible by all processors. Apart from centralized shared memory each processor may have registers, buffers, caches, and local memory as additional memory resources. Since centralized resource is shared simultaneously by number of processors many basic issues in the design of shared memory systems have to be taken into consideration. These include access control, synchronization, protection, and security.

Access Control.

Access control determines which process can access which of possible resources. This list is often maintained in separate table. For every access request issued by the processors to the shared

memory, access control module make the required check against the contents of the access control table to determine whether to give or not to give(deny) access permission. If there are access attempts to resources, then until the desired access is completed, all disallowed access attempts and illegal processes are blocked. Requests from sharing processes may change the contents of the access control table during execution. Access control mechanism along with synchronization rules guide the sharing of shared memory among multiple processors.

Synchronization.

Synchronization controls and coordinates access of shared resources by multiple processors. Appropriate synchronization ensures that the information flows properly and ensures system functionality. Many synchronization primitives such as semaphores are used along the shared memory access.

Protection

Protection is a system feature that prevents processes from making arbitrary access to resources belonging to other processes. One must not confuse between access control and protection. Access control is something to determine what are you allowed to do. This has to do with the policy making regarding who are all can use which are all resources. Whereas in protection question asked is who are you. Protection is often done by authorization. There are many authorization mechanisms ranging from simpler password based to more powerful cryptographic methods may be used for authorization.

Sharing

Sharing and protection are incompatible; sharing allows access, whereas protection restricts it. Sharing often leads to resource conflicts and synchronization problem. One extreme of sharing is not to share the resources. Another extreme is to share the resources completely. Shared memory when used along with the cache results in greater data inconsistency leading to cache coherence problem. We explain cache coherence problem briefly here. For example when two processors cache the data then when one processor write-Back the modified data back into memory, the

updated data is not visible to other process. Cache coherence is way of ensuring data consistency in the case of write-backs.

There are many shared memory variants for the multiprocessor architectures. The simplest shared memory system consists of one memory module that can be accessed from two processors. In this requests arrive at the memory module through its two ports. An arbitration unit within the memory module passes requests to a memory controller. If the memory module is not busy and a single request arrives, then the arbitration unit passes that request to the memory controller and the request is granted. During this time module is placed in the busy state . If a new request arrives while the memory is busy , then requesting processor put the request in the queue until the memory becomes free or it may repeat its request sometime later.

Depending on the interconnection network, a shared memory systems are classified as: **Uniform Memory Access (UMA), NonUniform Memory Access (NUMA), and Cache-Only Memory Architecture (COMA).** In the UMA system, a central shared memory is accessible by all processors through an interconnection network .Therefore, all processors have equal access time to any memory location. The interconnection network used in the UMA can be a single bus, multiple buses, a crossbar, or a multiport memory. Instead of centralized shared memory, in the NUMA system, each processor has part of the shared memory attached. But the memory has a single address space. This can be visualized as single memory is dived into different pages and these pages are distributed among processors. Therefore, any processor could access any memory location directly using its real address. However, the access time to modules depends on the distance of processor with respect to memory module it try accessing . This results in a nonuniform memory access time. COMA is similar to the NUMA wherein each processor has part of the shared memory. However, in this case the shared memory consists of cache memory. In COMA system data is migrated to the processor requesting it. After data has been migrated to the cache, then the data in cache is used by processor.

5.5.2 Message Passing Organization

Multiprocessor architecture with message passing systems is made up of set of processor each having their own local memory. Communications in message passing systems are performed via send and receive operations. A node in such a system consists of a processor and its local memory. Nodes are typically able to store messages in buffers (temporary memory locations where messages wait until they can be sent or received), and perform send/ receive operations at the same time as processing. Simultaneous message processing and message processing are handled by the underlying operating system. Each nodes in this system are interconnected in a many ways ranging from architecture-specific interconnection structures to geographically dispersed networks. As stated earlier, message passing approach is scalable. By scalable, its meant that the number of processors can be increased without significant decrease in efficiency of operation. Two important design factors must be considered in designing interconnection networks for message passing systems. These are the link bandwidth and the network latency . The link bandwidth is defined as the number of bits that can be transmitted per unit time (bits/ s). The network latency is defined as the time to complete a message transfer.

An message routing protocol called Wormhole routing is an improvement over traditional store-and-forward routing. In Wormhole routing in order to reduce the size of the required buffers and to decrease the message latency, a packet is divided into smaller units that are called flits (flow control bits).Each flits move in a pipeline fashion with the header flit moves first followed by remaining flits. When the header flit is blocked due to network congestion, the remaining flits are blocked as well. This not only reduces the unnecessary traffic but also decrease time for transmission of messages.

Interconnection Networks

There are many ways of classifying interconnection networks. Classification is often based on mode of operation, control system, switching techniques used and topology used.

According to the mode of operation, they are classified as synchronous versus asynchronous . In synchronous mode of operation, a single global clock is used by all components in the system. Asynchronous mode of operation, on the other hand, does not require a global clock. Handshaking signals are used instead in order to coordinate the operation of asynchronous systems.

According to the control strategy, Interconnection networks can be classified as centralized versus decentralized. In centralized control systems, a single central control unit is used to oversee and control the operation of the components of the system. In decentralized control, the control function is distributed among different components in the system.

Interconnection networks can be classified according to the switching mechanism as circuit versus packet switching networks. In the circuit switching mechanism, a complete path has to be established prior to the start of communication between a source and a destination. The established path will remain in existence during the whole communication period. In a packet switching mechanism, communication between a source and destination takes place via messages that are divided into smaller entities, called packets. On their way to the destination, packets can be sent from one node to another in a store-and-forward manner until they reach their destination. While packet switching tends to use the network resources more efficiently, compared to circuit switching, it suffers from variable packet delays.

Based on topology used for interconnection, interconnection networks are classified as single bus, Crossbar networks, Multistage networks and Hypercube Networks.

Single bus

In this interconnection system a common communication path is shared by all functional units. As each unit try to same same path mechanisms such as bus arbitration are needed when multiple units try accessing the shared bus simultaneously. Further performance of shared bus is limited by number of processors connecting to it. Performance of shared bus decrease according to number of processors connected to it.

Crossbar networks

In crossbar networks memory units are connected to processor through separate paths as shown in the figure below wherein each processor is connected each memory module through a cross point switch. This enables all processors to send memory requests independently and asynchronously. This kind of interconnection scales well but increases the hardware complexity.

Each crossbar switch in a crossbar network can be set open or closed, providing a point-to-point path between processor and memory module. On each row of a crossbar mesh multiple switches can be connected simultaneously but only one switch in a column is opened at any point of time. This provides a processor to interact with multiple memory modules simultaneously but one memory module can communicate with only one processor at any point of time.

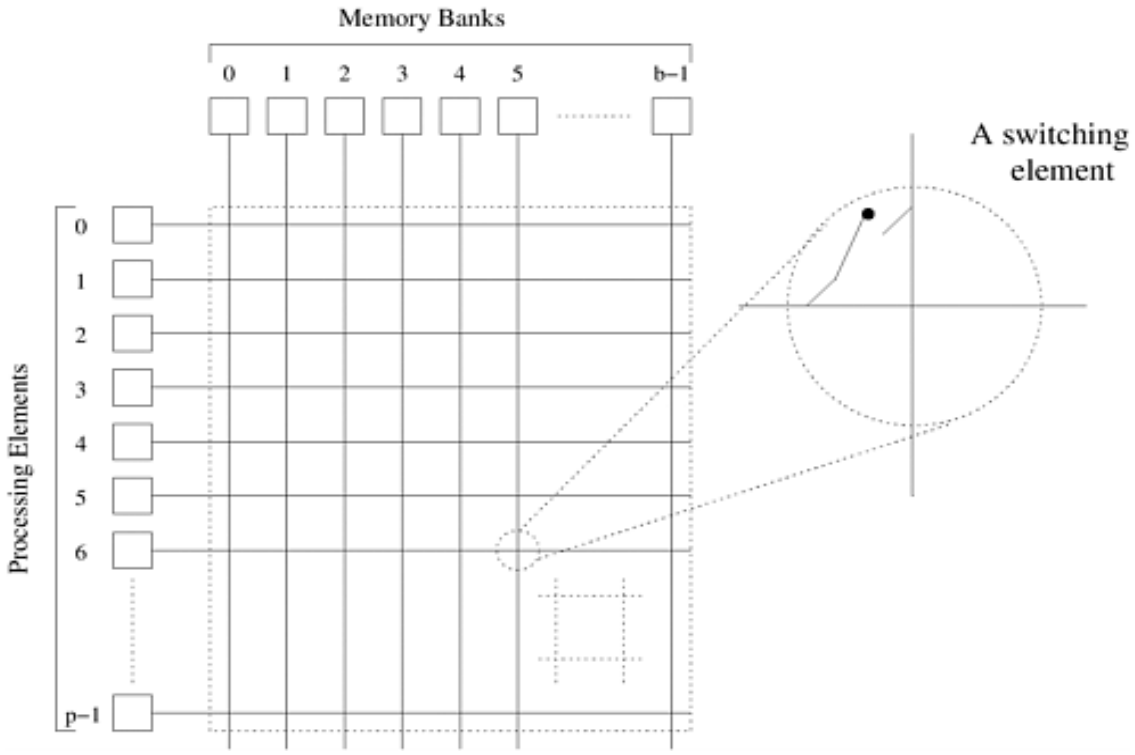

Fig. 5.8 Multiprocessor connected through crossbar switches

Multistage networks

Multistage Interconnection Networks(MINs) are extension of

crossbar networks connecting processors and memory modules through switching elements. The switching elements themselves are usually connected to each other in stages, hence the name. The main difference between MINs and crossbar networks is number of stages used. In crossbar networks a complete mesh connects processors to memory. This can be thought as single stage. In MINs multiple such stages are used for interconnections. This provides reliability and efficiency by deriving the paths through permutation among different stages. Banyans, Clos , Batcher sorters are examples of this interconnections. Following figure shows one such multistage network.

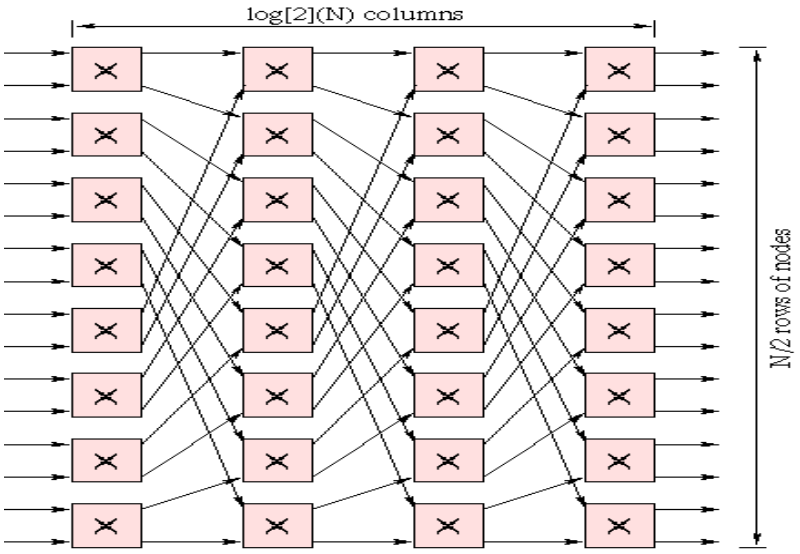

Fig. 5.9 Multiprocessor connected through multistage network

Hypercube Networks

Hypercube networks are derived from cubic structures. Hypercubes can be thought of an hierarchical cubic design wherein higher level is designed formed from connecting many lower levels cubic designs. Usually coordinates of the cubes are connected to provide multistage connectivity between processors and memory modules. Often hypercubes are considered to be more reliable interconnections as it provides multiple paths through its different dimensions.

5.6 ANALYSIS AND PERFORMANCE OF MULTIPROCESSOR ARCHITECTURE

In this section, we provide some basic ideas about the performance issues in multiprocessors. A fundamental question that is usually asked is how much faster a given problem can be solved using multiprocessors as compared to a single processor? This question can be formulated into the speed-up factor defined below.

S(n) = speed-up factor

$$= \frac{\text{Execution time using a single processor}}{\text{Execution time using n processors}}$$

A related question is that how efficiently each of the n processors is utilized. The question can be formulated into the efficiency defined below.

E(n) = Efficiency

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

Where s(n) is the speedup gained by dividing task into n subtasks.

In executing tasks (programs) using a multiprocessor, it may be assumed that a given task can be divided into n equal subtasks each of which can be executed by one processor. Therefore, the expected speed-up will be given by the S(n)=n while the efficiency E(n) =100%. The assumption that a given task can be divided into n equal subtasks, each executed by a processor, is unrealistic. Even if we assume that time taken for a doing task and the time taken to execute the task using multiprocessor, then also we cannot achieve 100 % efficiency. Because time of executing job using multiprocessor not only involves time for execution but also time for dividing the task into subtasks, assigning them to different machines, time taken for multiprocessor communication and time for combining the results.

5.7 FUTURE DIRECTION OF MULTIPROCESSOR ARCHITECTURE

Initially when the idea of parallel processing was conceived there were limited parallel processing applications such as vector manipulation which are extensively used in computational science and engineering and problems too large to solve on one computer, may use 100s or 1000s of computers to solve these.

Many companies in the 80s/90s "bet" on parallel computing and failed to make profit because computers got faster too quickly. What initially thought of computer as slow computing device as taken a momentum according to Moore's law and become very fast computing device such that it overshadowed the concept of parallel processing. But we still claim the importance of parallel processing because following two reasons :

- There are more applications that can take benefit of parallelism
- Moore's law does not hold any more

Now-a-days, there are many parallel frameworks that wanted to exploit the benefits of parallel processing. Examples such as MapReduce, Graph traversal, Dynamic programming, Backtracking/B&B, Graphical models,N-Body, (Un) Structured Grid primitively exhibit the nature of parallel execution. These new applications have open up new avenues in the area of parallel processing.

Further, Moore's law no longer holds good. Following graph gives the number of transistors used in a processor as the years progressed.

In a present situation we cannot follow Moore's law because although VLSI and nanotechnologies enabled putting large transistors on the chip but clock speed in not possible to increase in that speed. Further overcrowding the transistors on the chip is leading into new problem of heat dissipation. Thus new design philosophy is designing multicore processors. All the aspects we had discussed with multiprocessor architecture such as processor communication, data

sharing and cache coherence still holds good in multicore architectures. Only difference between these two is in multiprocessor architecture each processor uses bus exclusively whereas in multicore a bus is shared by many cores. Problems such as bus arbitration are newly introduced in multicore architectures.

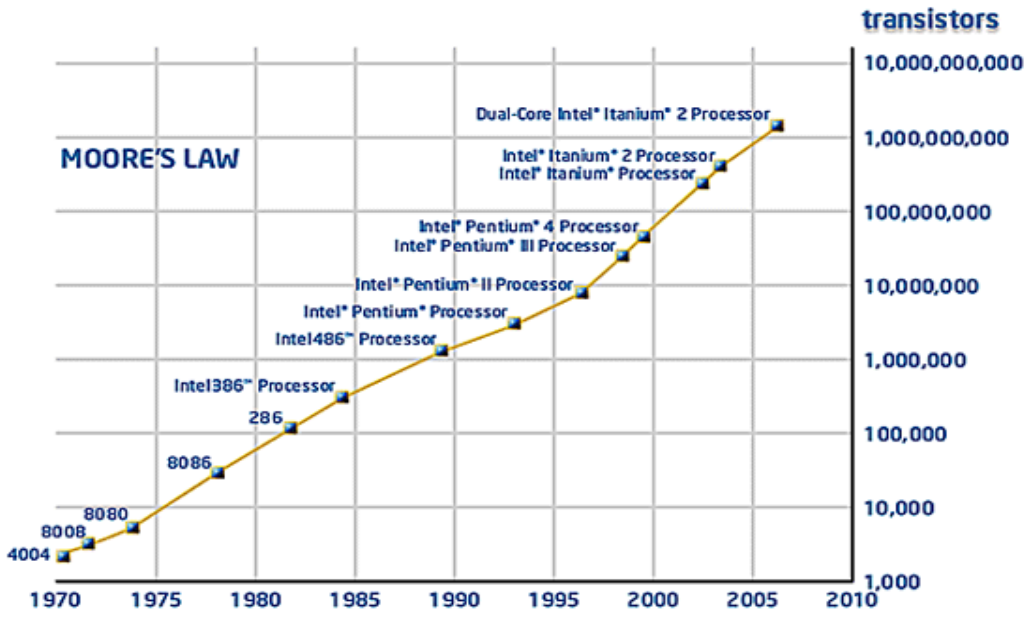

Fig. 5.10 Moore's law showing number of transistors against year