Introduction to Parallel Processing

Introduction to Parallel Computing

Before taking a toll on Parallel Computing, first let's take a look at the background of computations of a computer software and why it failed for the modern era.

Computer software were written conventionally for serial computing. This meant that to solve a problem, an algorithm divides the problem into smaller instructions. These discrete instructions are then executed on Central Processing Unit of a computer one by one. Only after one instruction is finished, next one starts.

Real life example of this would be people standing in a queue waiting for movie ticket and there is only cashier. Cashier is giving ticket one by one to the persons. Complexity of this situation increases when there are 2 queues and only one cashier.

So, in short Serial Computing is following:

1. In this, a problem statement is broken into discrete instructions.
2. Then the instructions are executed one by one.
3. Only one instruction is executed at any moment of time.

Look at point 3. This was causing a huge problem in computing industry as only one instruction was getting executed at any moment of time. This was a huge waste of hardware resources as only one part of the hardware will be running for a particular instruction and of time. As problem statements were getting heavier and bulkier, so does the amount of time in execution of those statements. Example of processors are Pentium 3 and Pentium 4.

Now let's come back to our real life problem. We could definitely say that complexity will decrease when there are 2 queues and 2 cashier giving tickets to 2 persons simultaneously. This is an example of Parallel Computing.

Parallel Computing

It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.

Advantages of Parallel Computing over Serial Computing are as follows:

1. It saves time and money as many resources working together will reduce the time and cut potential costs.
2. It can be impractical to solve larger problems on Serial Computing.
3. It can take advantage of non-local resources when the local resources are finite.
4. Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of hardware.

Types of Parallelism:

1. **Bit-level parallelism:** It is the form of parallel computing which is based on the increasing processor's size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data.

Example: Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.

2. **Instruction-level parallelism:** A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.

3. **Task Parallelism:** Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform execution of sub tasks concurrently.

Why parallel computing?

- The whole real world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.
- Real world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.
- Parallel computing provides concurrency and saves time and money.
- Complex, large datasets, and their management can be organized only and only using parallel computing's approach.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.
- Also, it is impractical to implement real-time systems using serial computing.

Applications of Parallel Computing:

- Data bases and Data mining.
- Real time simulation of systems.
- Science and Engineering.
- Advanced graphics, augmented reality and virtual reality.

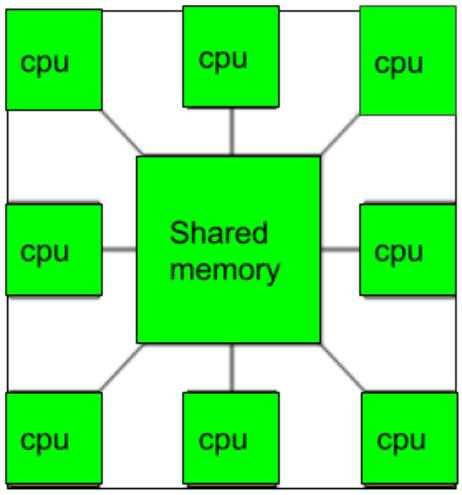
Limitations of Parallel Computing:

- It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.
- The algorithms must be managed in such a way that they can be handled in the parallel mechanism.
- The algorithms or program must have low coupling and high cohesion. But it's difficult to create such programs.
- More technically skilled and expert programmers can code a parallelism based program well.

Future of Parallel Computing: The computational graph has undergone a great transition from serial computing to parallel computing. Tech giant such as Intel has already taken a step towards parallel computing by employing multicore processors. Parallel computation will revolutionize the way computers work in the future, for the better good. With all the world connecting to each other even more than before, Parallel Computing does a better role in helping us stay that way. With faster networks, distributed systems, and multi-processor computers, it becomes even more necessary.

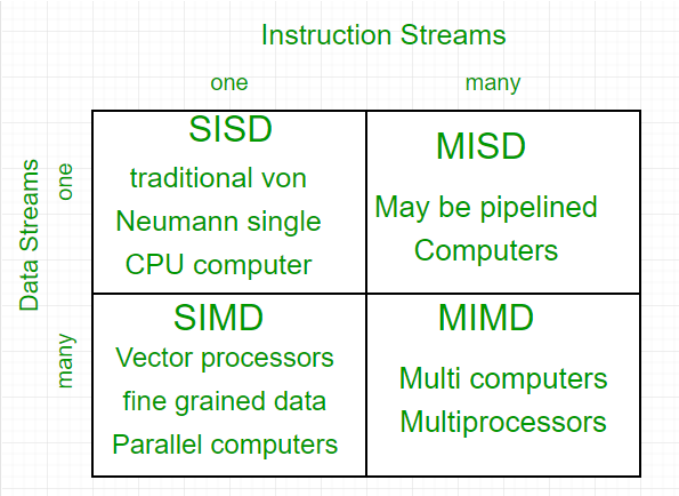
Multiprocessor System

A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.
There are two types of multiprocessors, one is called shared memory multiprocessor and another is distributed memory multiprocessor. In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own private memory.



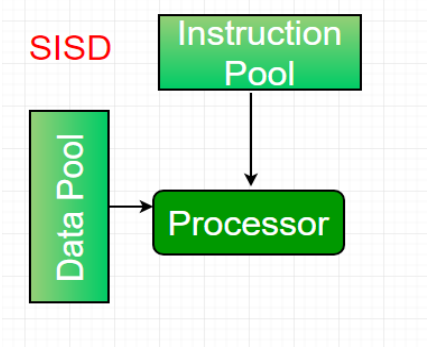
Flynn's taxonomy

Parallel computing is a computing where the jobs are broken into discrete parts that can be executed concurrently. Each part is further broken down to a series of instructions. Instructions from each part execute simultaneously on different CPUs. Parallel systems deal with the simultaneous use of multiple computer resources that can include a single computer with multiple processors, a number of computers connected by a network to form a parallel processing cluster or a combination of both. Parallel systems are more difficult to program than computers with a single processor because the architecture of parallel computers varies accordingly and the processes of multiple CPUs must be coordinated and synchronized.
The crux of parallel processing are CPUs. Based on the number of **instruction and data** streams that can be processed simultaneously, computing systems are classified into four major categories:



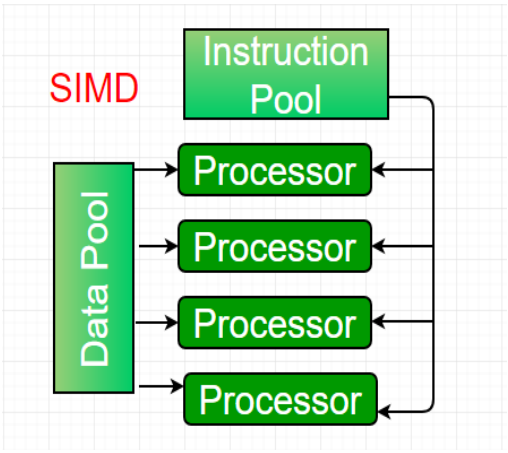
Flynn's classification

1. **Single-instruction, single-data (SISD) systems –**
   An SISD computing system is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream. In SISD, machine instructions are processed in a sequential manner and computers adopting this model are popularly called sequential computers. Most conventional computers have SISD architecture. All the instructions and data to be processed have to be stored in primary memory.



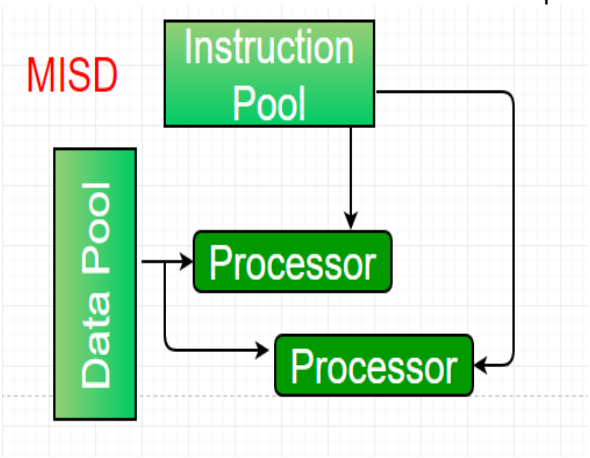
   The speed of the processing element in the SISD model is limited(dependent) by the rate at which the computer can transfer information internally. Dominant representative SISD systems are IBM PC, workstations.

2. **Single-instruction, multiple-data (SIMD) systems –**
   An SIMD system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams. Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations. So that the information can be passed to all the processing elements (PEs) organized data elements of vectors can be divided into multiple sets(N-sets for N PE systems) and each PE can process one data set.



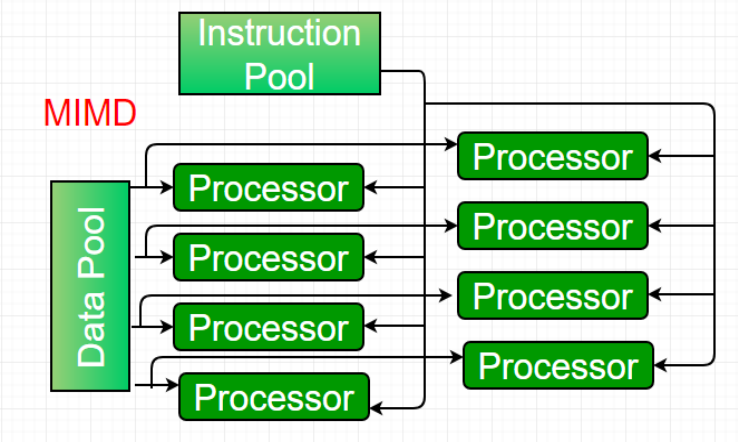
Dominant representative SIMD systems is Cray's vector processing machine.

3. **Multiple-instruction, single-data (MISD) systems –**
An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same dataset .



Example $Z = sin(x) + cos(x) + tan(x)$
The system performs different operations on the same data set. Machines built using the MISD model are not useful in most of the application, a few machines are built, but none of them are available commercially.

4. **Multiple-instruction, multiple-data (MIMD) systems –**
An MIMD system is a multiprocessor machine which is capable of executing multiple instructions on multiple data sets. Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application. Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.



MIMD machines are broadly categorized into **shared-memory MIMD** and **distributed-memory MIMD** based on the way PEs are coupled to the main memory.
In the **shared memory MIMD** model (tightly coupled multiprocessor systems), all the PEs are connected to a single global memory and they all have access to it. The communication between PEs in this model takes place through the shared memory, modification of the data stored in the global memory by one PE is visible to all other PEs. Dominant representative shared memory MIMD systems are Silicon Graphics machines and Sun/IBM's SMP (Symmetric Multi-Processing).
In **Distributed memory MIMD** machines (loosely coupled multiprocessor systems) all PEs have a local memory. The communication between PEs in this model takes place through the interconnection network (the inter process communication channel, or IPC). The network connecting PEs can be configured to tree, mesh or in accordance with the requirement.
The shared-memory MIMD architecture is easier to program but is less tolerant to failures and harder to extend with respect to the distributed memory MIMD model. Failures in a shared-memory MIMD affect the entire system, whereas this is not the case of the distributed model, in which each of the PEs can be easily isolated. Moreover, shared memory MIMD architectures are less likely to scale because the addition of more PEs leads to memory contention. This is a situation that does not happen in the case of distributed memory, in which each PE has its own memory. As a result of practical outcomes and user's requirement, distributed memory MIMD architecture is superior to the other existing models.

Interconnection Structure in Multiprocessor

CPUs, TOPs connect' to input-output device, and a memory unit form a multiprocessor system, they may be partitioned into a number of separate modules. The interconnection between the components can have different physical configurations, depending on the number of transfer paths that are available between the processors and memory in a shared memory system or among the processing elements in a loosely coupled system. There are several physical forms available for establishing an interconnection network. Some of them are:

1. Time-shared common bus
2. Multiport memory
3. Crossbar switch
4. Multistage switching network
5. Hypercube network

Time Shared Common Bus

- A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit.
- *Disadvantages :*
    - Only one processor can communicate with the memory or another processor at any given time.
    - As a consequence, the total overall transfer rate within the system is limited by the speed of the single path
- A more economical implementation of a dual bus structure is depicted in Fig. below.
- Part of the local memory may be designed as a *cache memory* attached to the CPU.

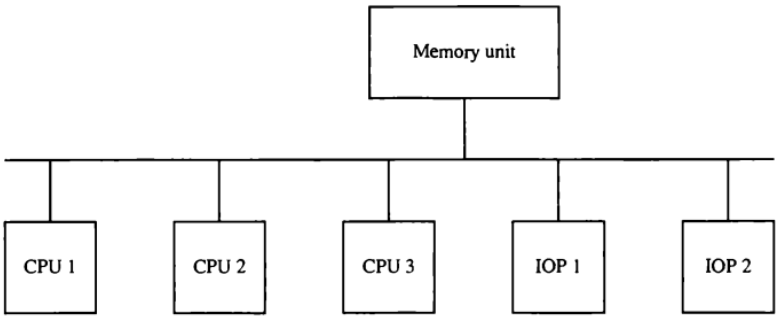


Figure 1 Time-shared common bus organization.

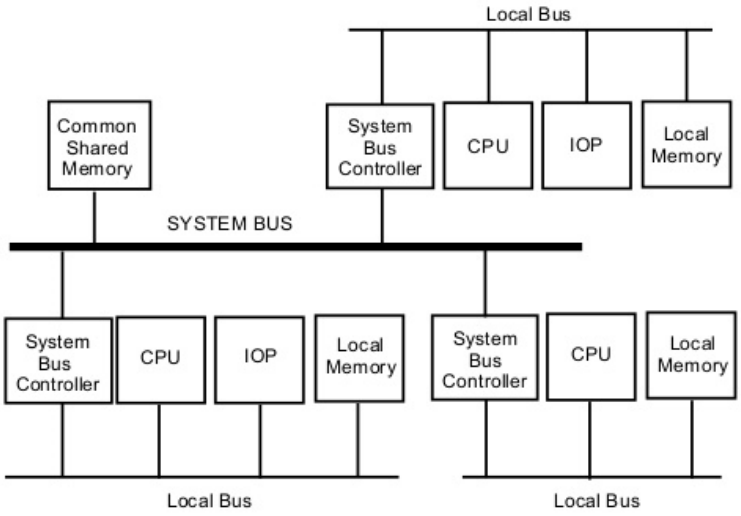


Fig: System bus structure for multiprocessors

Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU.
- The module must have internal control logic to determine which port will have access to memory at any given time.

- Memory access conflicts are resolved by assigning fixed priorities to each memory port.
- *Advantages*
  - The high transfer rate can be achieved because of the multiple paths.
- *Disadvantages*
  - It requires expensive memory control logic and a large number of cables and connections.

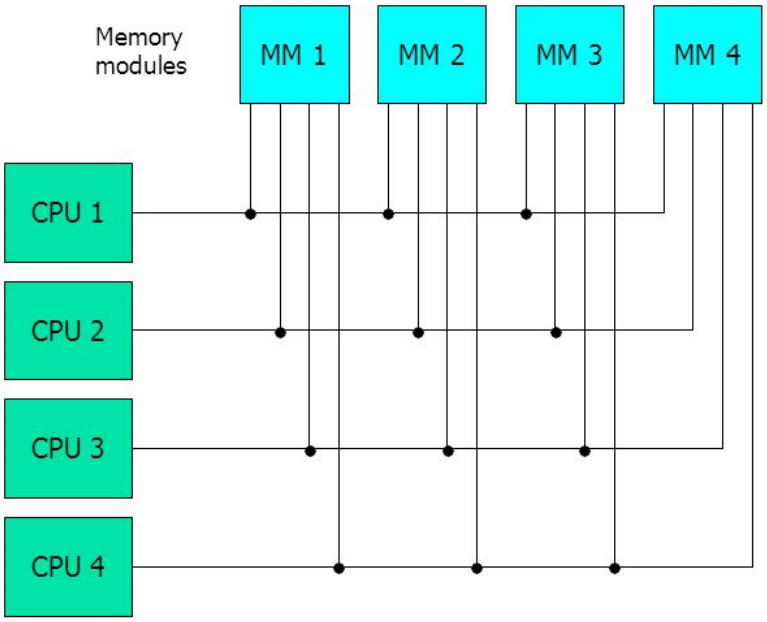
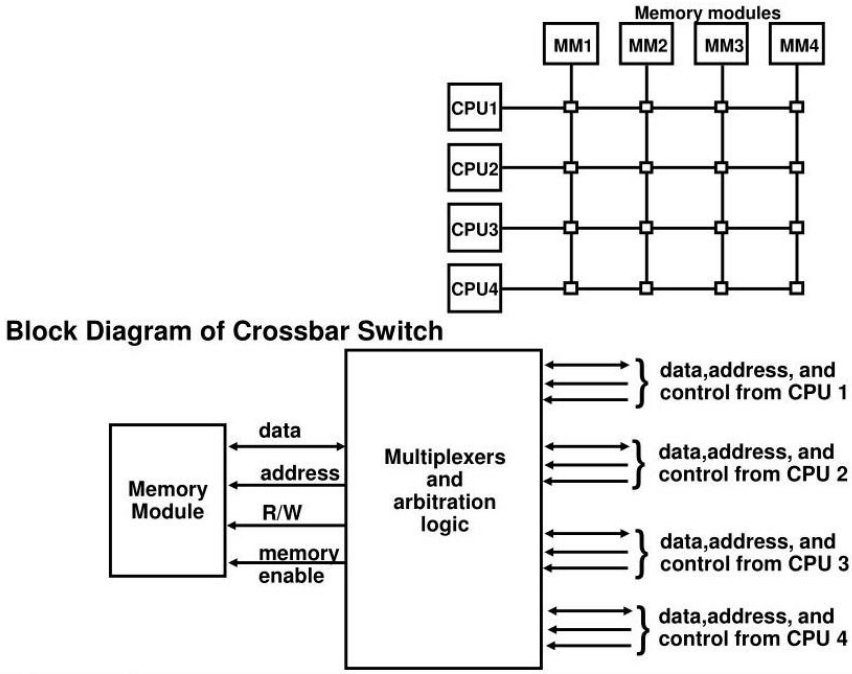


Fig: Multiport organization

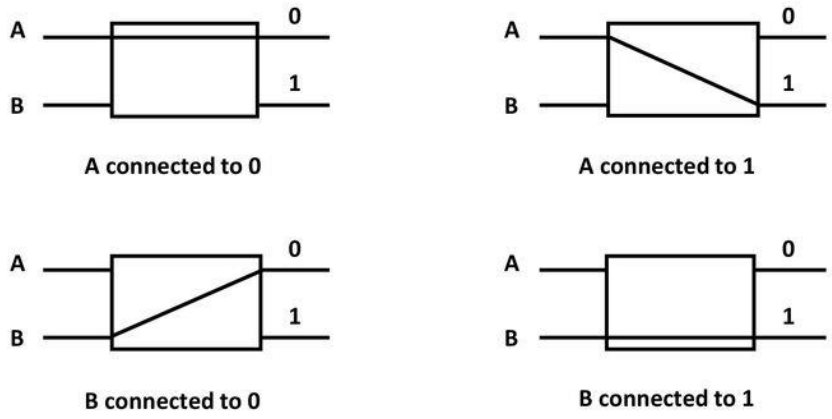
Crossbar Switch

- Consists of a number of *crosspoints* that are placed at intersections between processor buses and memory module paths.
- The small square in each crosspoint is a *switch* that determines the path from a processor to a memory module.
- Advantage
  - Supports simultaneous transfers from all memory modules
- Disadvantage
  - The hardware required to implement the switch can become quite large and complex.
- Below fig. shows the functional design of a crossbar switch connected to one memory module.

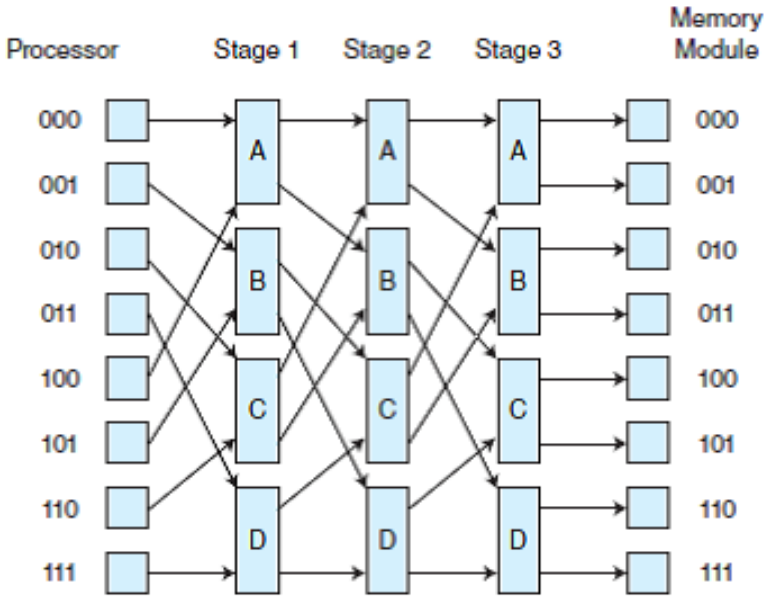


Multistage Switching Network

- The basic component of a multistage network is a two-input, two-output interchange switch as shown in Fig. below.

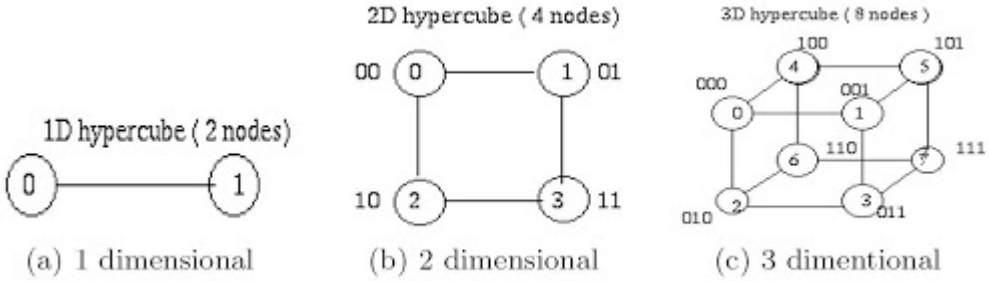


- One such topology is the omega switching network shown in Fig. below



Hypercube System

- The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of N=2n processors interconnected in an n-dimensional binary cube.
    - Each processor forms a node of the cube, in effect it contains not only a CPU but also local memory and I/O interface.
    - Each processor address differs from that of each of its n neighbors by exactly one bit position.
- Fig. below shows the hypercube structure for n=1, 2, and 3.
- Routing messages through an *n*-cube structure may take from one to *n* links from a source node to a destination node.
    - A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address.
    - The message is then sent along any one of the axes that the resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ.
- A representative of the hypercube architecture is the Intel iPSC computer complex.
    - It consists of 128(*n*=7) microcomputers, each node consists of a CPU, a floating-point processor, local memory, and serial communication interface units.



(a) 1 dimensional (b) 2 dimensional (c) 3 dimentional

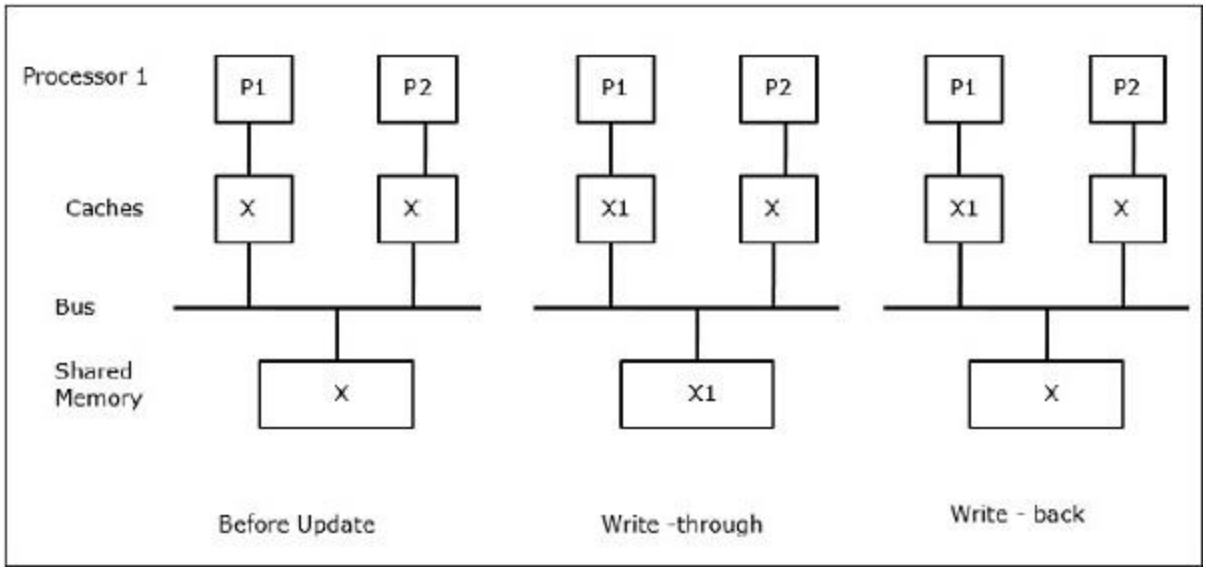
Loosely Coupled System

- There is no shared memory for passing information.
- The communication between processors is by means of message passing through I/O channels.
- The communication is initiated by one processor calling a procedure that resides in the memory of the processor with which it wishes to communicate.
- The communication efficiency of the interprocessor network depends on the communication routing protocol, processor speed, data link speed, and the topology of the network.

The Cache Coherence Problem

In a multiprocessor system, data inconsistency may occur among adjacent levels or within the same level of the memory hierarchy. For example, the cache and the main memory may have inconsistent copies of the same object.

As multiple processors operate in parallel, and independently multiple caches may possess different copies of the same memory block, this creates **cache coherence problem**. **Cache coherence schemes** help to avoid this problem by maintaining a uniform state for each cached block of data.



Let X be an element of shared data which has been referenced by two processors, P1 and P2. In the beginning, three copies of X are consistent. If the processor P1 writes a new data X1 into the cache, by using **write-through policy**, the same copy will be written immediately into the shared memory. In this case, inconsistency occurs between cache memory and the main memory. When a **write-back policy** is used, the main memory will be updated when the modified data in the cache is replaced or invalidated.

In general, there are three sources of inconsistency problem −

- Sharing of writable data
- Process migration
- I/O activity

Cache Events and Actions

Following events and actions occur on the execution of memory-access and invalidation commands −

- **Read-miss** − When a processor wants to read a block and it is not in the cache, a read-miss occurs. This initiates a **bus-read** operation. If no dirty copy exists, then the main memory that has a consistent copy, supplies a copy to the requesting cache memory. If a dirty copy exists in a remote cache memory, that cache will restrain the main memory and send a copy to the requesting cache memory. In both the cases, the cache copy will enter the valid state after a read miss.

- **Write-hit** − If the copy is in dirty or **reserved** state, write is done locally and the new state is dirty. If the new state is valid, write-invalidate command is broadcasted to all the caches, invalidating their copies. When the shared memory is written through, the resulting state is reserved after this first write.

- **Write-miss** − If a processor fails to write in the local cache memory, the copy must come either from the main memory or from a remote cache memory with a dirty block. This is done by sending a **read-invalidate** command, which will invalidate all cache copies. Then the local copy is updated with dirty state.

- **Read-hit** − Read-hit is always performed in local cache memory without causing a transition of state or using the snoopy bus for invalidation.

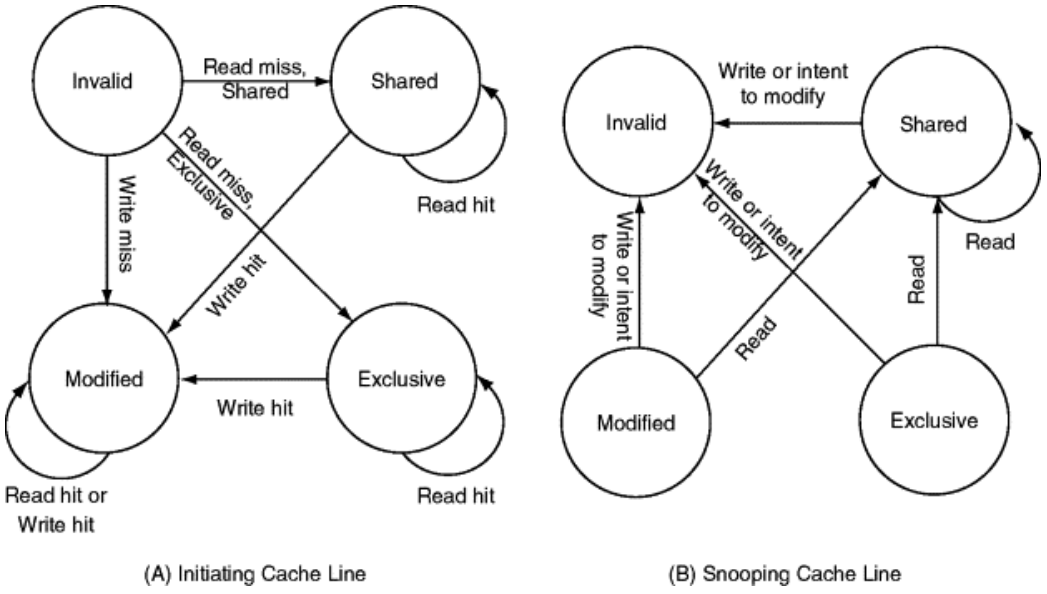
- **Block replacement** − When a copy is dirty, it is to be written back to the main memory by block replacement method. However, when the copy is either in valid or reserved or invalid state, no replacement will take place.

MESI Protocol

It is the most widely used cache coherence protocol. Every cache line is marked with one the following states:

- **Modified**
  This indicates that the cache line is present in current cache only and is dirty i.e its value is different from the main memory. The cache is required to write the data back to main memory in future, before permitting any other read of invalid main memory state.
- **Exclusive**
  This indicates that the cache line is present in current cache only and is clean i.e its value matches the main memory value.

- **Shared**
  It indicates that this cache line may be stored in other caches of the machine.
- **Invalid**
  It indicates that this cache line is invalid.

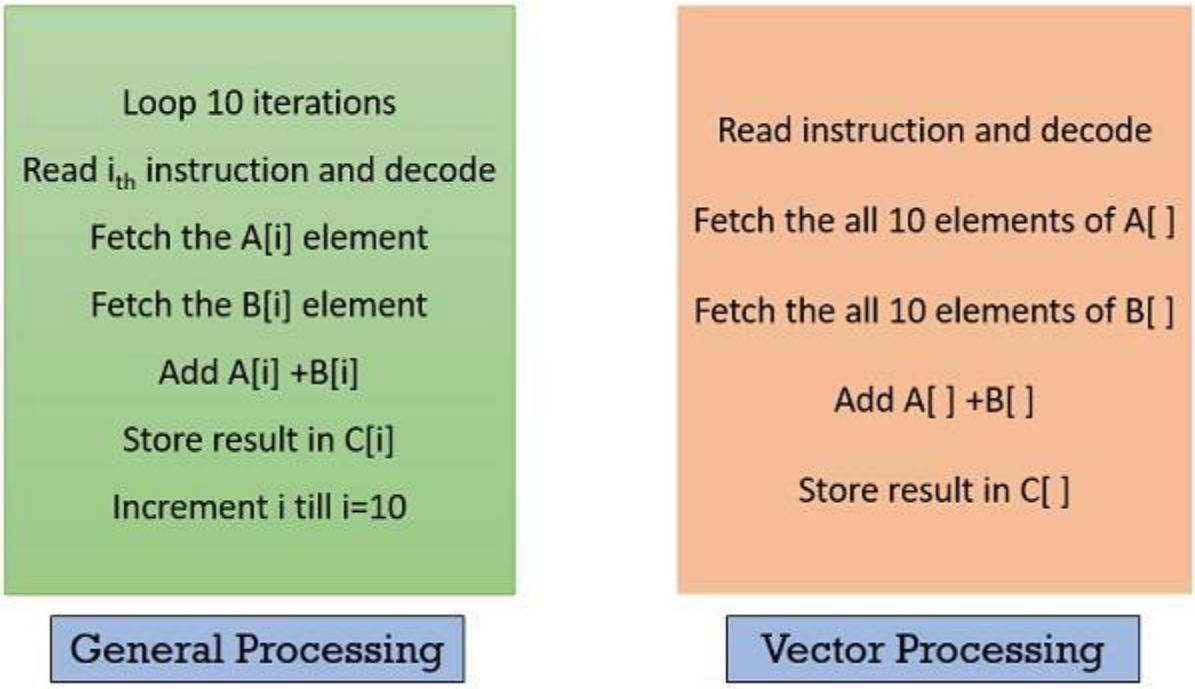


Introduction

We need computers that can solve mathematical problems for us which include, arithmetic operations on the large arrays of integers or floating-point numbers quickly. The general-purpose computer would use loops to operate on an array of integers or floating-point numbers. But, for large array using loop would cause overhead to the processor.

To avoid the overhead of processing loops and fasten the computation, some kind of parallelism must be introduced. **Vector processing** operates on the entire array in just one operation i.e. it operates on elements of the array in **parallel**. But, vector processing is possible only if the operations performed in parallel are **independent**.

Look at the figure below, and compare the vector processing with the general computer processing, you will notice the difference. Below, instructions in both the blocks are set to add two arrays and store the result in the third array. Vector processing adds both the array in parallel by avoiding the use of the loop.



Operating on multiple data in just one instruction is also called **Single Instruction Multiple Data** (SIMD) or they are also termed as **Vector instructions**. Now, the data for vector instruction are stored in **vector registers**.

Each vector register is capable of storing several data elements at a time. These several data elements in a vector register is termed as a **vector operand**. So, if there are n number of elements in a vector operand then n is the **length of the vector**.

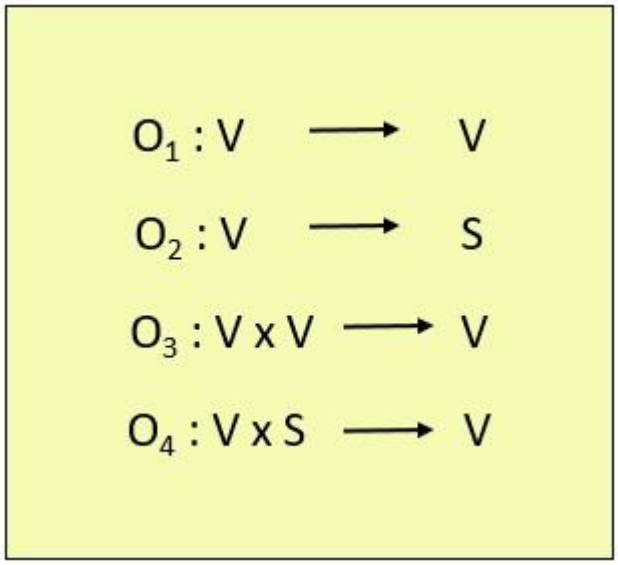
Supercomputers were evolved to deal with billions of floating-point operations/second. Supercomputer optimizes numerical computations (vector computations).

But, along with vector processing supercomputers are also capable of doing scalar processing.
Later, **Array processor** was introduced which particularly deals with vector processing, they do not indulge in scalar processing.

Characteristics of Vector Processing

Each element of the vector operand is a **scalar quantity** which can either be an integer, floating-point number, logical value or a character. Below we have classified the vector instructions in four types.

Here, V is representing the vector operands and S represents the scalar operands. In the figure below, O1 and O2 are the unary operations and O3 and O4 are the binary operations.



Most of the vector instructions are **pipelined** as vector instruction performs the same operation on the different data sets repeatedly. Now, the pipelining has start-up delay, so longer vectors would perform better here.

The pipelined vector processors can be classified into two types based on from where the operand is being **fetched** for vector processing. The two architectural classifications are Memory-to-Memory and Register-to-Register.

In **Memory-to-Memory** vector processor the operands for instruction, the intermediate result and the final result all these are retrieved from the **main memory**. TI-ASC, CDC STAR-100, and Cyber-205 use memory-to-memory format for vector instructions.

In **Register-to-Register** vector processor the source operands for instruction, the intermediate result, and the final result all are retrieved from **vector or scalar registers**. Cray-1 and Fujitsu VP-200 use register-to-register format for vector instructions.

Vector Instruction

A vector instruction has the following fields:

1. Operation Code

Operation code indicates the **operation that** has to be performed in the given instruction. It decides the functional unit for the specified operation or reconfigures the multifunction unit.

2. Base Address

Base address field refers to the **memory location** from where the operands are to be fetched or to where the result has to be stored. The base address is found in the memory reference instructions. In the vector instruction, the operand and the result both are stored in the vector registers. Here, the **base address** refers to the designated **vector register**.

3. Address Increment

A vector operand has several data elements and address increment specifies the **address of the next element in the operand**. Some computer stores the data element consecutively in main memory for which the increment is always 1. But, some computers that do not store the data elements consecutively requires the variable address increment.

4. Address Offset

Address Offset is always specified related to the base address. The effective **memory address** is calculated using the address offset.

5. Vector Length

Vector length specifies the **number of elements in a vector operand**. It identifies the **termination** of a vector instruction.

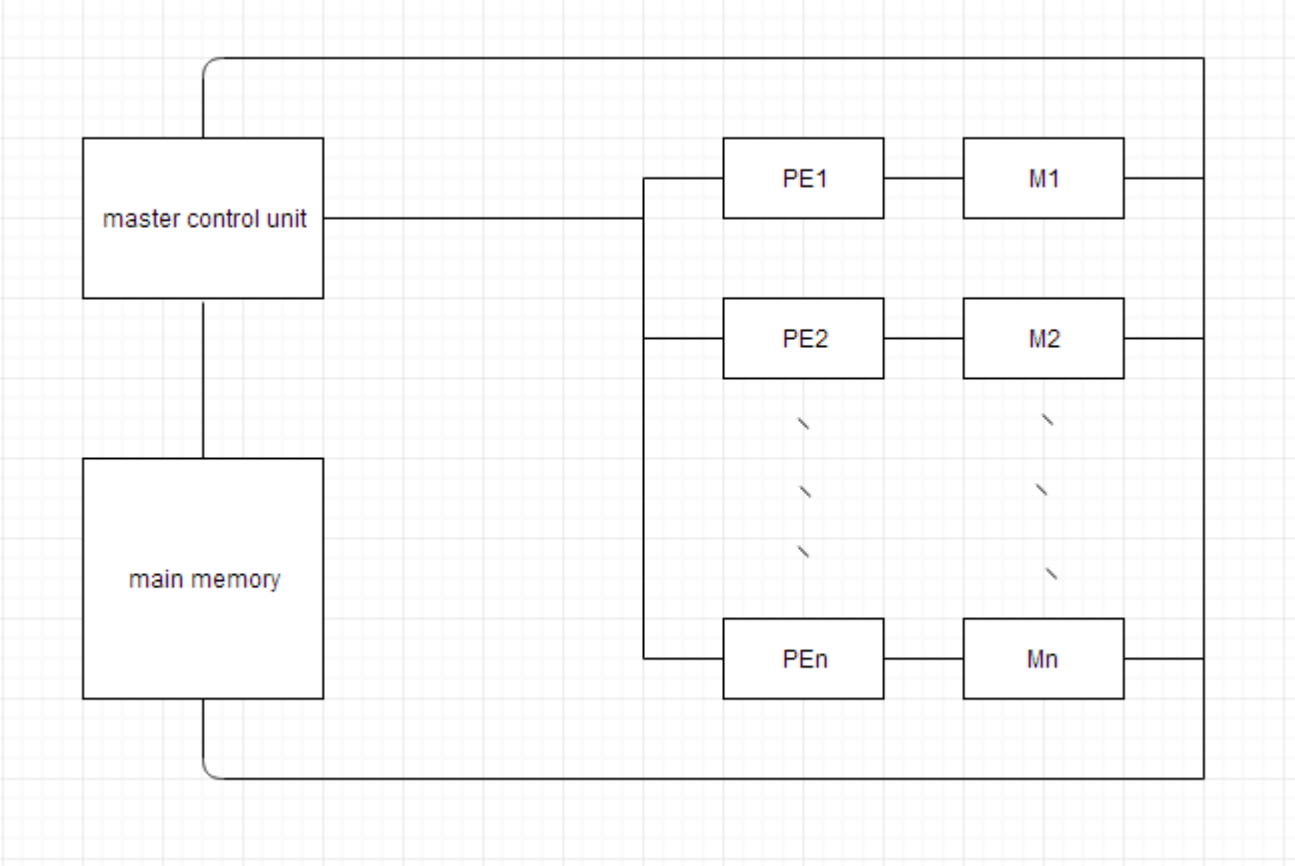
SIMD Array Processors

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an **ALU** and **registers**. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the **ILLIAC IV** computer developed by the **Burroughs corps**. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.



Why use the Array Processor

- Array processors increases the overall instruction processing speed.
- As most of the Array processors operates asynchronously from the host CPU, hence it improves the overall capacity of the system.
- Array Processors has its own local memory, hence providing extra memory for systems with low memory.

Multithreading

Threading can be useful in a single-processor system by allowing the main execution thread to be responsive to user input, while the additional worker thread can execute long-running tasks that do not need user intervention in the background. Threading in a multiprocessor system results in true concurrent execution of threads across multiple processors and is therefore faster. However, it requires more careful programming to avoid non-intuitive behavior such as racing conditions, deadlocks, etc.

Operating systems use threading in two ways:

- Pre-emptive multithreading, in which the context switch is controlled by the operating system. Context switching might be performed at an inappropriate time, hence, a high priority thread could be indirectly pre-empted by a low priority thread.
- Cooperative multithreading, in which context switching is controlled by the thread. This could lead to problems, such as deadlocks, if a thread is blocked waiting for a resource to become free.

The 32- and 64-bit versions of Windows use pre-emptive multithreading in which the available processor time is shared such that all the threads get an equal time slice and are serviced in a queue-based mode. During thread switching, the context of a pre-empted thread is stored and reloaded in the next thread in the queue. The time slice is so short that the running threads seem to be executing in parallel.