Computer System Architecture

Subject Code:CS405

MODULE - 1

PARALLEL COMPUTER MODELS

COMPUTER ARCHITECTURE

- Computer architecture is concerned with the structure and behavior of the computer as seen by the user.
- It involves both hardware and programming requirements.
- It is a study of both Instruction Set Architecture(ISA) and machine implementation organization.

❖Lookahead

- It is used to prefetch instructions in order to overlap I/E (Instruction Fetch/Decode and Execution) operations and to enable functional parallelism.
- Functional parallelism can be achieved by two approaches
  1. Use multiple functional units simultaneously
  2. Use pipelining

❖Parallelism

- Parallel processing is a technique used to provide simultaneous data processing tasks for the purpose of increasing computational speed of a computer system.
- It perform concurrent data processing to achieve faster execution time.
- For implementing parallel processing , we use the following techniques

1. Pipelining
2. Multiplicity of functional units
3. DMA
4. Hierarchal memory system.
5. Multi-programming & Time sharing

Prepared By EBIN PM (AP, IESCE) 5

❖Pipelining

- Multiple instructions are overlapped in execution
- It increases the system throughput
- A digital computer involves 4 major steps (Instruction Cycle)

1. Instruction Fetch
2. Instruction Decoding
3. Operand Fetch
4. Execution

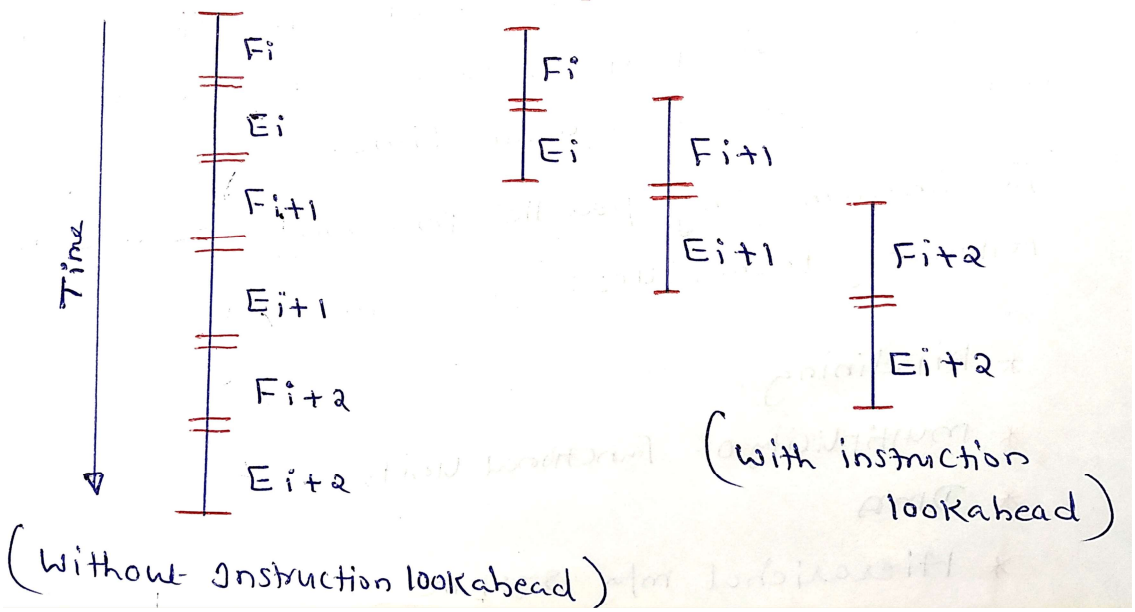
- In non-pipelined computer these 4 steps must be completed before the next instruction can be issued.

Prepared By EBIN PM (AP, IESCE) 6

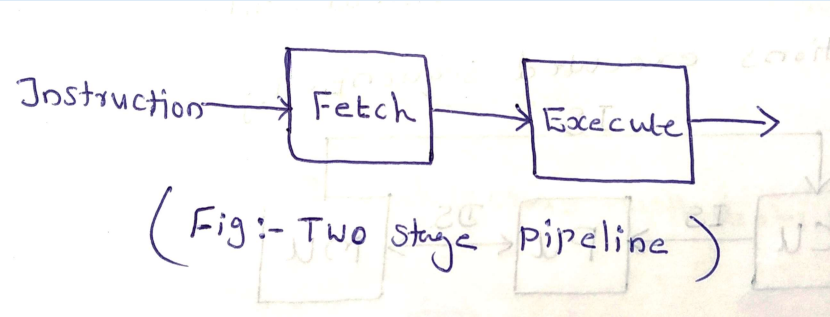
- In a pipelined computer , successive instructions are executed in an overlapped fashion.

- The two types of pipelines are
1. Arithmetic pipeline
2. Instruction pipeline

- The way to speedup the CPU is to overlap the fetching of the next instruction with the execution of the current instruction. This technique is called Instruction Lookahead.



- Without Look ahead, the three instructions take 6 time units to complete.
- With Look ahead, they take only 4 time units
- This overlapping of work is called a two stage pipeline



Prepared By EBIN PM (AP, IESCE) 9

FLYNN'S CLASSIFICATION OF COMPUTERS

- Flynn's classification scheme is based on how many streams of INSTRUCTIONS and DATA exist in a machine.
- A stream means a sequence of items (instructions or data) as executed or operated on by a processor.
- Some machines executes a single stream of instructions, while others execute multiple streams.
- Some machines reference single stream of data , and others reference many streams of data.

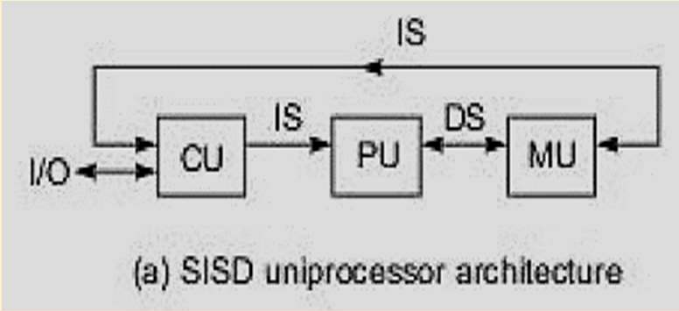
Prepared By EBIN PM (AP, IESCE) 10

❖Instruction Stream – Sequence of instructions

❖Data Stream – Sequence of data

1. SISD – Single Instruction stream single Data stream
2. SIMD – Single Instruction stream Multiple Data stream
3. MISD – Multiple Instruction stream Single Data stream
4. MIMD – Multiple Instruction stream Multiple Data stream

(1) SISD Computer



(a) SISD uniprocessor architecture

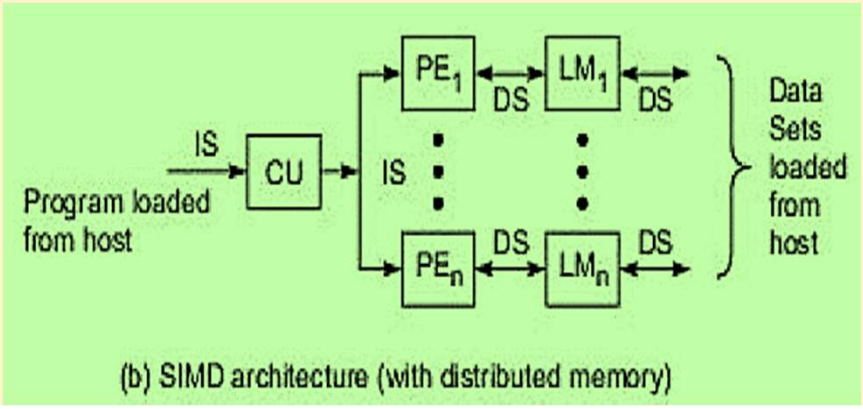
➢It is also known as Von Neumann architecture

➢Both instruction and data are fetched from the memory modules

➢Instructions are decoded by the Control Unit (CU),which sends the decoded instruction stream to the processor units for execution.

- All the serial computers are SISD
- It has more than one functional units
- Instructions executed sequentially

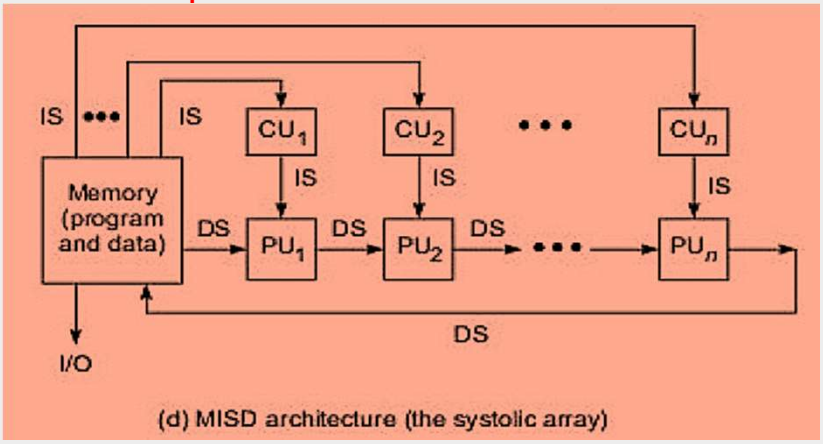
(2) SIMD Computer



(b) SIMD architecture (with distributed memory)

- SIMD computers are also known as Array processors
- A single instruction stream is broadcasted to multiple processors, each having its own data stream.
- Since each PE (Processing Element)operates on its own local data, there are multiple data streams.
- Simultaneous communication with processor is possible
- SIMD machine can be further divided in to Word-Slice Vs. Bit-Slice modes
- ILLIAC IV is the first SIMD machine

(3) MISD Computer

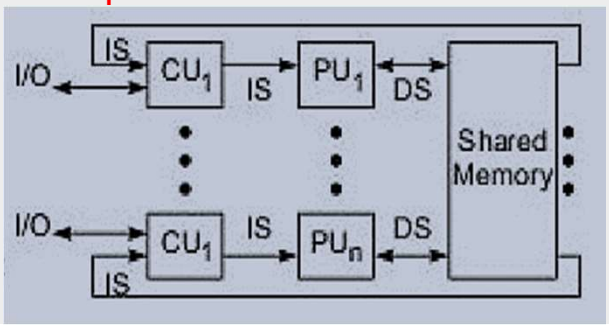


(d) MISD architecture (the systolic array)

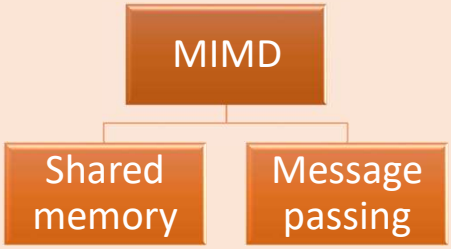
- Also known as Systolic array
- It has "n" processor units, each receiving distinct instructions

- The result of one processor become the input of the next processor
- No real existence, no practical system has been constructed
- Single data stream is fed in to multiple processing units.

(4) MIMD Computer

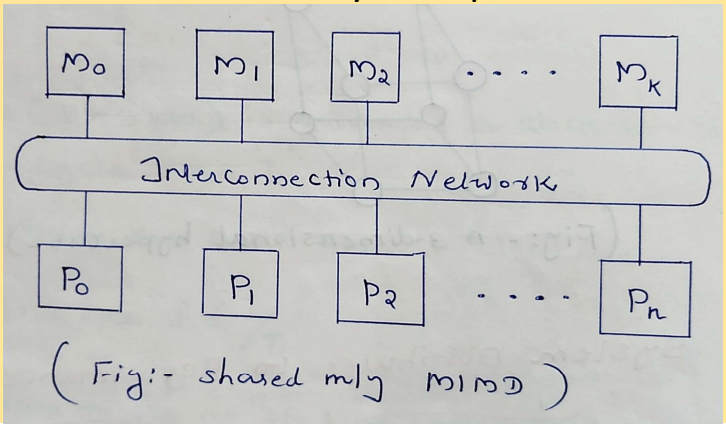


- Parallel computers are reserved for MIMD machines.
- Each processor has its own instruction stream and input data
- Capable of processing several programs at same time
- Eg: Supercomputers, Network parallel computer grids
- Two subclasses of MIMD are



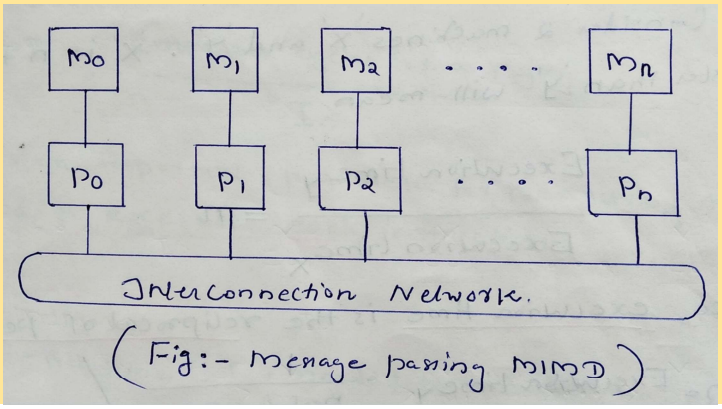
❖MIMD Shared Memory

- Here, any of the processors may read from or write to a shared memory address space.
- It is also called shared memory multiprocessor.



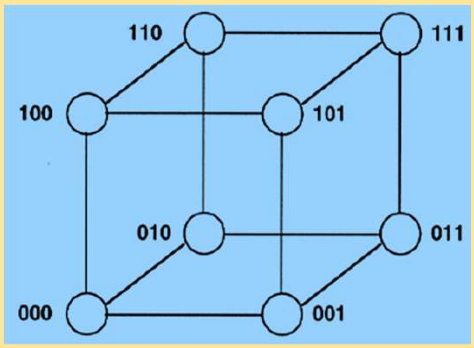
❖MIMD Message Passing

- Here, all the processors having their own local memory.
- In order to communicate, the processors send messages to each other by way of an interconnection network.



- The interconnection network may take different forms.
- A popular interconnection network for message passing mechanism is the "K-dimensional binary hypercube".
- For example, in a 3-dimensional hypercube , the processors are at the corner of a cube.

(Fig:3-dimensional hypercube)



SYSTEM ATTRIBUTES TO PERFORMANCE

- The computer user is interested in reducing response time (the time between the start and completion of an event) and increasing throughput (the total amount of work done in a given time)
- Consider two machines X and Y. "X is n times faster than Y" will mean

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

- Since, execution time is the reciprocal of performance, the following relationship holds:

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\frac{1}{\text{Performance}_Y}}{\frac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

- Because, performance and execution time are reciprocals, increasing performance decreases execution time.

➢Computer architects have come up with a variety of metrics to describe the computer performance.

Clock rate and CPI / IPC

- CPU is driven by a clock with a constant cycle time " **τ** "
- The inverse of the cycle time is "clock rate"

Clock rate $f = 1/\tau$

- The size of the program is determined by the instruction count (Ic)
- If we know the no.of clock cycles and the Ic, we can calculate the average number of clock cycles per instruction (CPI)

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

➢CPI is an important parameter for measuring the time needed to execute each instruction.

Performance factors

- The CPU time (T) needed to execute the program is estimated by

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

➢The execution of an instruction requires going through a cycle of events involving the instruction fetch, decode, operand fetch , execution and store the result.

- In this cycle only the instruction decode and execution phases are carried out in the CPU.
- The remaining three operations may require access to the memory.
- The processor cycles required for each instruction (CPI) can be attributed to cycles needed for instruction decode and execution (*p*), and cycles needed for memory references (*m* k*).
- The total time needed to execute a program can then be rewritten as

$$T = Ic^* (p + m^*k)^*\tau$$

- p➔ no: of processor cycles needed for inst decode & execution
- m➔ no: of memory reference needed
- k➔ ratio between memory and processor cycles
- τ➔ processor cycle time
- Memory cycle
  - Time needed to complete one memory reference
  - Denoted as m
  - k➔ depends on
    - speed of cache
    - memory technology
    - Processor memory interconnection scheme

- Let

C= total number of clock cycles needed to execute a given program(n instructions)

CPI = no.of clock cycles needed to execute single instruction

$$CPI = \frac{C}{Ic}$$

CPU time

$$\rightarrow T = C * \tau$$

$$\rightarrow T = \frac{C}{f}$$

Q: Machine A takes 15 seconds and Machine B runs the same program in 25 seconds. Which one is faster and by how much?

Ans: Assume, Machine A is n times faster than Machine B

Execution Time B / Execution Time A =n

Performance ratio = 25/15 = 1.67

So, Machine A is 1.67 times faster than machine B

MIPS rate

- Processor speed is measured in terms of Million Instructions Per Second (MIPS)

MIPS rate is based on following factors

- Clock rate f
- Instruction count Ic
- CPI of given machine

$$\text{MIPS rate} = \frac{Ic}{T * 10^6} = \frac{Ic}{Ic * CPI * \tau * 10^6} \rightarrow \frac{f}{CPI * 10^6} = \frac{f}{C/Ic * 10^6}$$

$$\text{MIPS Rate} = \frac{Ic * f}{C * 10^6}$$

Throughput Rate

- CPU throughput Wp
- It is the measure of how many programs can be executed per second, based on the MIPS rate & average program length

- $Wp = \frac{f}{Ic * CPI}$

PROBLEM

A benchmark program is run on a 40 MHz processor. The executed program consists of 100,000 instruction executions, with the following instruction mix and clock cycle count:

Instruction Type	Instruction Count	Cycles per Instruction
Integer arithmetic	45.000	1
Data transfer	32.000	2
Floating point	15.000	2
Control transfer	8000	2

Determine the effective CPI, MIPS rate, and execution time for this program.

Prepared By EBIN PM (AP, IESCE) 31

Solution

Clock speed of the Processor = 40 MHz

Instruction Type	Instruction Count	Cycles per Instruction	cycles
Integer arithmetic	45,000	1	45000
Data transfer	32,000	2	64000
Floating point	15,000	2	30000
Control transfer	8000	2	16000

Prepared By EBIN PM (AP, IESCE) 32

- Total no: of cycles required to execute complete program
  - ➔ 45000+64000+30000+16000
  - ➔ 155000 cycles
- C=155000 cycles
- Effective CPI= C/Ic
  - ➔ 155000/100000

  CPI = 1.55

Prepared By EBIN PM (AP, IESCE) 33

- MIPS rate= $f/CPI*10^6$
  - $= 40/ 1.55*10^6$
  - $=40*10^6/1.55*10^6$
  - $= 25.8$

- Given f=40 MHz ➔ $\tau = 1/40$
- $T = Ic * CPI * \tau$
- $= 100000*1.55*1/40$
- $=100000*1.55*0.025$
- $=3875$
- $= 3.875$ ms

Prepared By EBIN PM (AP, IESCE) 34

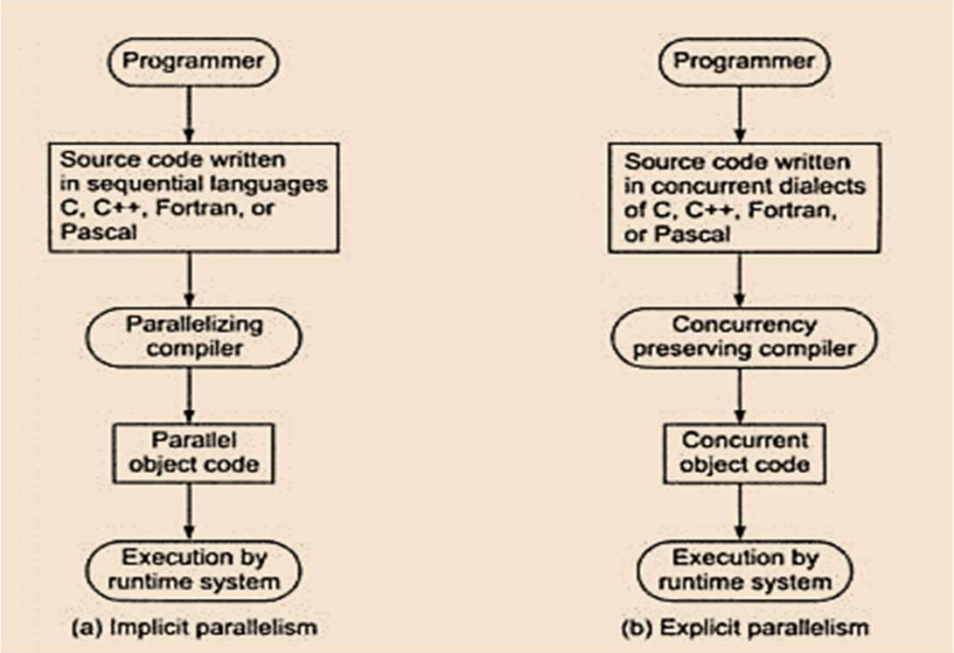
PARALLEL PROGRAMMING APPROACHES

Implicit Parallelism

- This approach uses C, C++, Fortran or Pascal to write the source program.
- Sequentially coded source program is translated in to parallel object code by a parallelizing compiler.
- The system(the libraries, compiler, OS)will do the parallelism in our code.

Explicit Parallelism

- The programmer requires more effort to develop a source program using C, C++, Fortran or Pascal
- Parallelism is explicitly specified in the user programs.
- This reduces the burden on the compiler to detect parallelism. The compiler needs to preserve the parallelism and assign target machine resources.
- The programmer decompose the work in to task and coordinate between them.



AMDAHL'S LAW

- It is named after Computer Scientist Gene Amdahl(a computer architect from IBM). It is also known as Amdahl's Argument.
- It is a formula used to find the maximum improvement possible by just improving a particular part of a system.
- It is often used in parallel computing to predict the theoretical speedup when using multiple processors.
- The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's Law.

- Amdahl's Law defines the speedup that can be gained by using a particular feature.

Speedup is the Ratio

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

Alternatively,

$$\text{Speedup} = \frac{\text{Execution Time for entire task without using enhancement}}{\text{Execution Time for entire task using enhancement when possible}}$$

- Speedup tells us how much faster a task will run using the machine with the enhancement as opposed to the original machine
- Amdahl's Law gives us a quick way to find the speedup from some enhancement, which depends on two factors:

(1) Fraction Enhanced

- The fraction of the computation time in the original machine that can be converted to take advantage of the enhancement
- For example, if 20 seconds of the execution time of a program that takes 60 seconds in total can use an enhancement, the fraction is 20/60.
- This value is called Fraction enhanced and is always less than or equal to 1.

(2) Speed-up Enhanced

- The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program
- If the enhanced mode takes 2 seconds for some portion of the program that can completely use the mode, while the original mode took 5 seconds for the same portion, the improvement is 5/2.
- This value is always greater than 1

➢ Amdahl's Law can serve as a guide to how much an enhancement will improve performance and how to distribute resources to improve cost/performance.

$$\text{Execution time}_{new} = \text{Execution time}_{old} \times \left((1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right)$$

❖ The overall speedup is the ratio of the execution times:

$$\text{Speedup}_{overall} = \frac{\text{Execution time}_{old}}{\text{Execution time}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Suppose that we are considering an enhancement that runs 10 times faster than the original machine but is usable only 40% of the time. What is the overall speedup gained by incorporating the enhancement?

Se = 10

F = 40/100 = 0.4

So = 1 / ((1 - F) + F/Se)

= 1/ (0.6 + 0.4/10)

= 1/0.64 = 1.56.

- Fraction = 0.1, Speedup = 10

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1-\text{Fraction}_{\text{enhanced}})+\frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{(1-0.1)+\frac{0.1}{10}} = \frac{1}{0.91} = 1.1$$

- Fraction = 0.9, Speedup = 10

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1-0.9)+\frac{0.9}{10}} = \frac{1}{0.19} = 5.3$$

SHARED MEMORY MULTIPROCESSORS

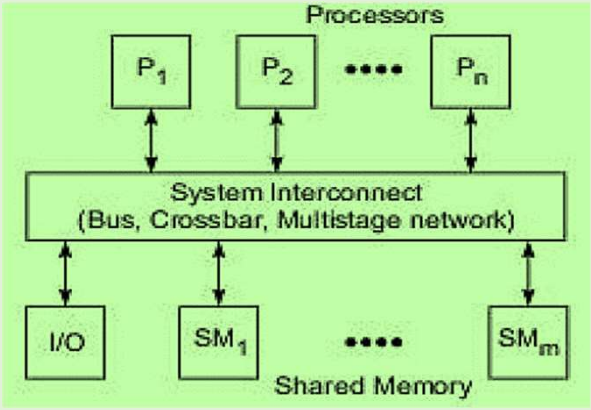
- Shared memory machines can be divided into three main classes based upon memory access times:

1. UMA - Uniform Memory Access model

2. NUMA – Non-Uniform Memory Access model

3. COMA – Cache-Only Memory Architecture model

(1) UMA Model

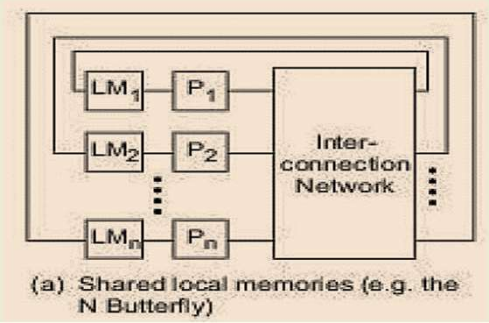


- All the processors share the physical memory uniformly
- All the processors have equal access time to all memory words, that is why it is called uniform memory access.

- Each processor may use private cache
- Peripherals are also shared in some fashion
- It is suitable for general purpose and time sharing applications by multiple users
- It can be used to speed up the execution of a single large program in time critical applications.
- When all processors have equal access to all peripheral devices, the system is called Symmetric Multiprocessors.
- In symmetric multiprocessors all the processors are equally capable of running the executive programs.

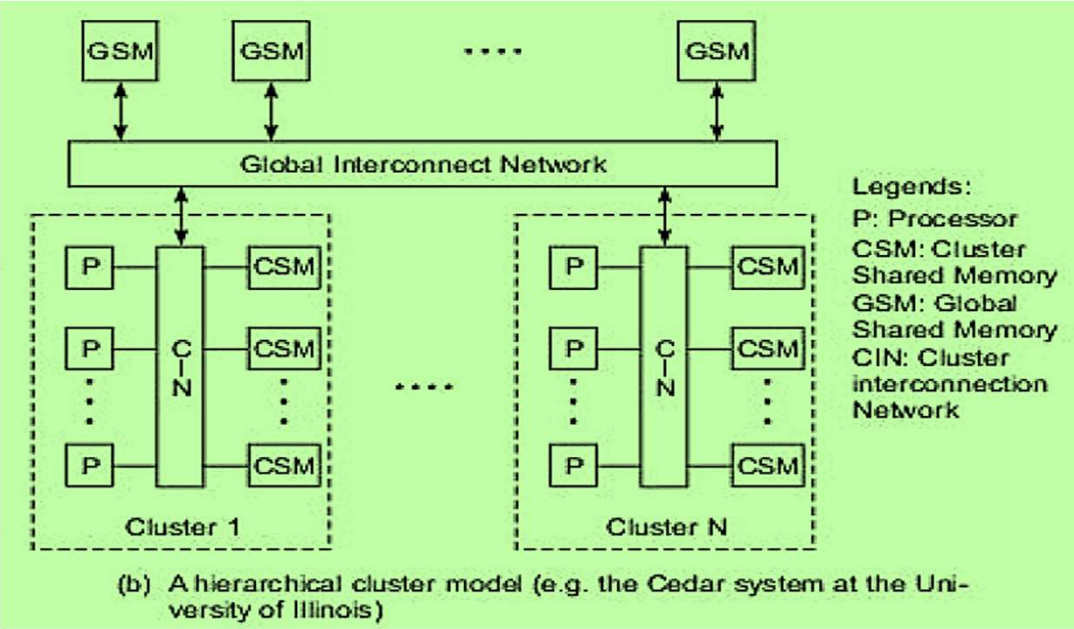
- In an Asymmetric Multiprocessor, only one or a subset of processors are executive-capable. The master processor can execute the OS and handle I/O. The remaining processors have no I/O capability; and thus are called attached processors(AP)
- Attached processors execute user codes under the supervision of the master processor.

(2) NUMA Model



(a) Shared local memories (e.g. the N Butterfly)

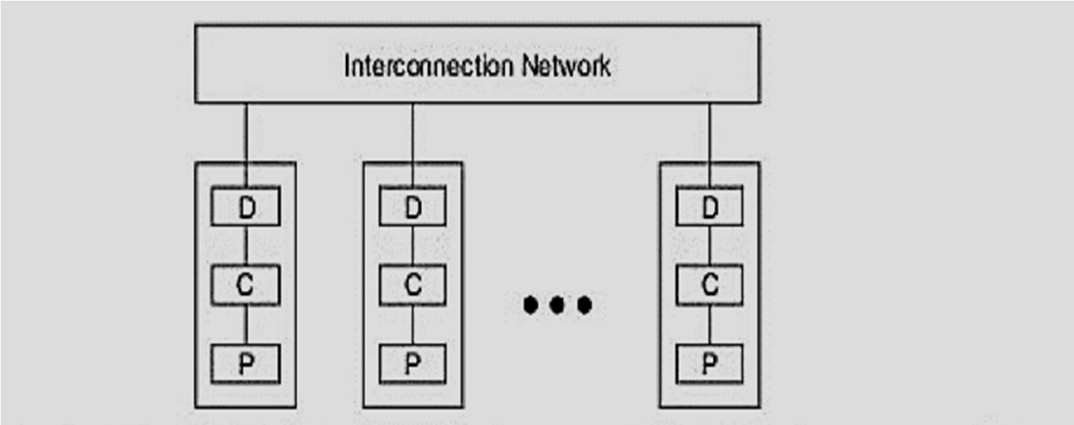
- It is a shared memory system in which the access time varies with location of the memory word.
- The shared memory is physically distributed to all processors, called local memories.
- The collection of all local memories forms a global address space accessible by all processors.
- In hierarchically structured multiprocessor, the processors are divided into several clusters. Each cluster is itself an UMA or NUMA multiprocessor.
- The clusters are connected to global shared memory modules. The entire system is considered a NUMA multiprocessor.
- All clusters have equal access to the global memory.



(b) A hierarchical cluster model (e.g. the Cedar system at the University of Illinois)

(3) COMA Model

- A multiprocessor using cache only memory assumes the COMA model
- It is a special case of NUMA machine, in which the distributed main memories are converted to caches
- There is no memory hierarchy and all the caches form a global address space.
- Remote cache access is assisted by the distributed cache directories.



The COMA model of a multiprocessor (P: Processor, C: Cache, D: Directory; e.g. the KSR-1)

Sr. No.	Key	UMA	NUMA
1	Definition	UMA stands for Uniform Memory Access.	NUMA stands for Non Uniform Memory Access.
2	Memory Controller	UMA has single memory controller.	NUMA has multiple memory controllers.
3	Memory Access	UMA memory access is slow.	NUMA memory accsss is faster than UMA memory.
4	Bandwidth	UMA has limited bandwidth.	NUMA has more bandwidth than UMA.
5	Suitablity	UMA is used in general purpose and time sharing applications.	NUMA is used in real time and time critical applications.
6	Memory Access time	UMA has equal memory access time.	NUMA has varying memory access time.
7	Bus types	3 types of Buses supported: Single, Multiple and Crossbar.	2 types of Buses supported: Tree, hiearchical.

Prepared By EBIN PM (AP, IESCE) 53

DISTRIBUTED - MEMORY MULTICOMPUTERS

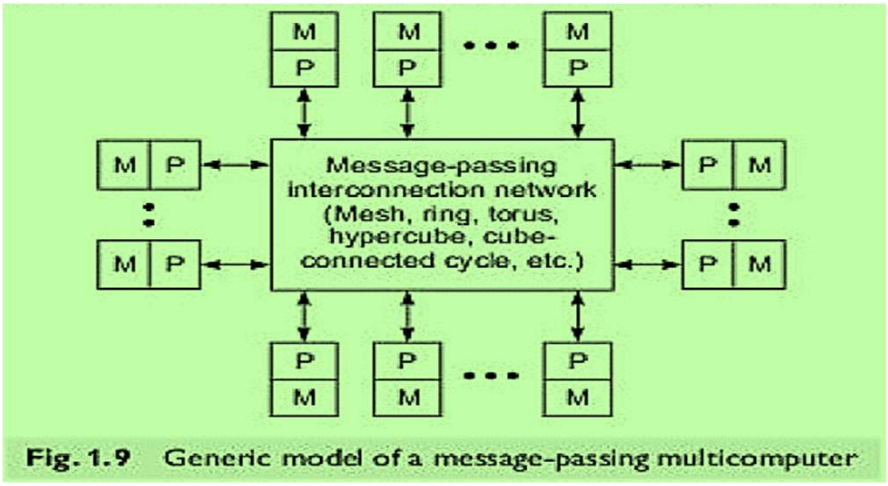


Fig. 1.9 Generic model of a message-passing multicomputer

Prepared By EBIN PM (AP, IESCE) 54

- The system consist of multiple computers called nodes, interconnected by a message passing network
- Each node is an autonomous computer consisting of a processor, local memory and sometimes attached disk or I/O peripherals.
- The message passing network provides point-to-point static connections among the nodes.
- All local memories are private and are accessible only by local processors. For this reason, traditional multicomputer have also called “no-remote-memory-access”(NORMA) machines.

- Modern multicomputers use hardware routers to pass message. A computer node is attached to each router.
- The boundary router may be connected to I/O and peripheral devices.
- Message passing between any two nodes involves a sequence of routers and channels
- Mixed type of nodes are allowed in a heterogeneous multicomputers
- The various network topologies used in multicomputers are

  Ring, Tree, Mesh, Torus, Hypercube, Cube-connected cycle
- Various communication patterns among nodes are

  One to One, Broadcasting, Permutations , Multicast patterns

VECTOR SUPER COMPUTER

- A vector processor is attached to a scalar processor.
- Host computer loads the program and data to the main memory.
- All instructions are first decoded by the scalar control unit.
- If the decoded instruction is a scalar operation or a program control operation, then it will be directly executed by the scalar processor using scalar functional pipelines

- If the instruction is decoded as a vector operation, it will be sent to the vector control unit. The vector data flow is coordinated by the control unit.

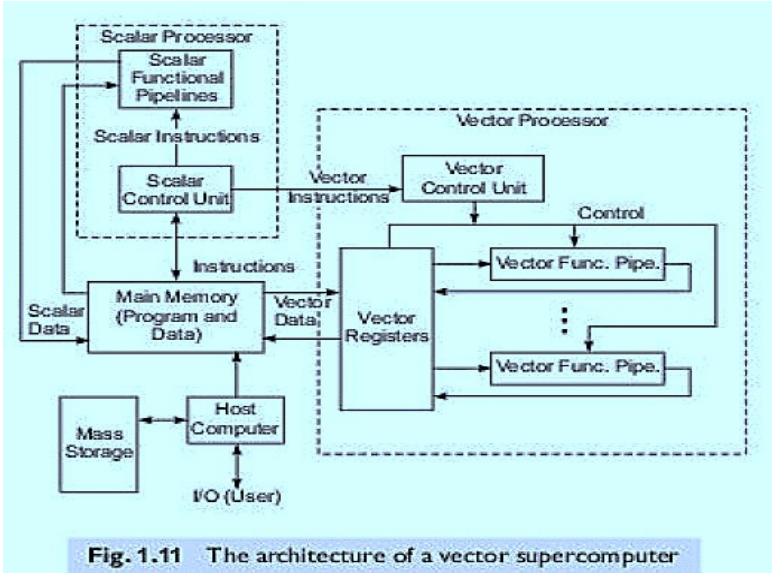


Fig. 1.11 The architecture of a vector supercomputer

➢The two pipeline vector supercomputer(vector processor) models are

1. Register-to-register architecture

- Vector registers are used to hold vector operands, intermediate and final vector result.
- There are fixed number of vector registers and functional pipelines in a vector processor

2. Memory-to-memory architecture

- Vector operands and results are directly retrieved from and stored into the main memory in super words (512 bits)

SIMD SUPERCOMPUTERS

➢Computers with multiple processing elements

➢SIMD Supercomputer is specified by a 5-tuple:

M = (N, C, I, M, R)

N – number of processing elements(PE) in the machine

C – Set of instructions directly executed by the control unit (CU)

I- Set of instructions broadcast by the CU to all PEs for parallel execution

M- Set of masking schemes

R – Set of data routing functions

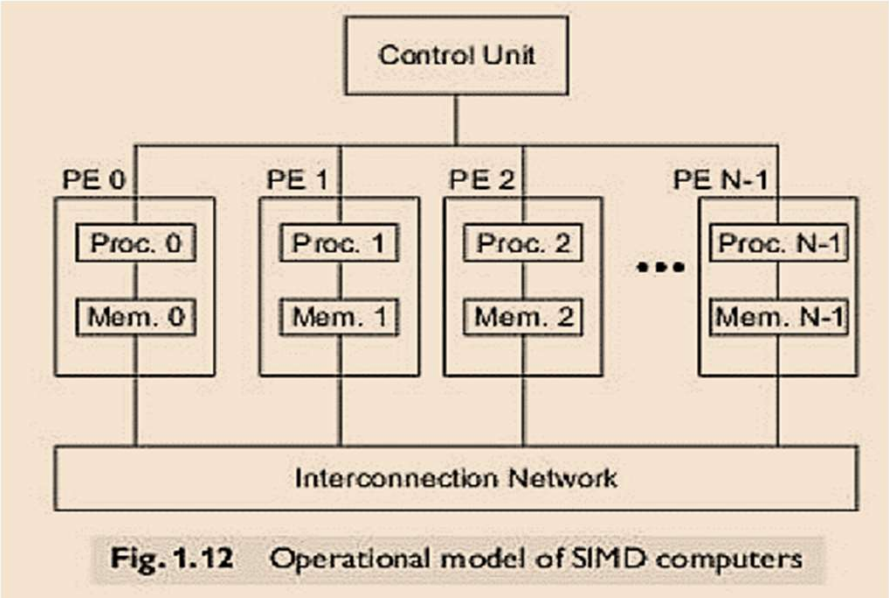


Fig. 1.12 Operational model of SIMD computers

CONDITIONS OF PARALLELISM

- To execute several programs in parallel, each segment has to be independent of the other segment.
- Dependency is the main challenge of parallelism.
- Dependence graph shows the relation between program statements.
  - Nodes of dependence graph➔ program statements
  - Directed edges with labels➔ relations among the statements

TYPES OF DEPENDENCES

- Data Dependence
- Control Dependence
- Resource Dependence

(1) Data Dependence

- Ordering relationship between statements are indicated by data dependence
- Determining how one instruction depends on another
- If two instructions are parallel, they can execute simultaneously in a pipeline
- If two instructions are dependent, they are not parallel and must be executed in order.

Prepared By EBIN PM (AP, IESCE) 63

- 5 types of data dependences are
  - (a) Flow dependence
  - (b) Anti-dependence
  - (c) Output dependence
  - (d) I/O dependence
  - (e) Unknown dependence

Prepared By EBIN PM (AP, IESCE) 64

(a) Flow Dependence

- A statement S2 is flow dependent on statement S1,
- if an execution path exists from S1 to S2, and if at least one output of S1 is fed as an input to S2
- Denoted as S1➔S2
  - Eg: consider following instruction
- S1: LOAD R1,A
- S2: ADD R2,R1

- S2 is flow dependent on S1
  - Coz o/p of S1 is fed as i/p of S2
  - Ie variable A is passed into register R1

(b) Anti-Dependence

- Statement S2 is anti-dependent on statement S1 if,
- S2 follows S1 in program order and if o/p of S2 overlaps the input to S1
- Denoted using a direct arrow crossed with a bar
- S1 $\not\longrightarrow$ S2
  - Eg: consider the following statements
  - S2: ADD R2,R1
  - S3: move R1,R3

- S3 is anti-dependent on S2 since S3 is overlapping the input to S2
- Ie conflicts in the register content of R1

(3) Output Dependence

- Two statements are output dependent if they produce the same output variable
- Denoted as S1 $\overset{o}{\longrightarrow}$ S2
- Eg:
- S1: LOAD R1,A
- S3: MOVE R1,R3

- S3 is output dependent on S1 coz they both modify the same register R1

(4) I/O Dependence

- I/O dependence occurs if same file is referenced by both I/O statements
- Read and write are the I/O statements
- Eg:

- S1: READ(4),A(i)
- S2: PROCESS
- S3: WRITE(4),B(I)
- S4: CLOSE(4)

- S1 and S3 are I/O dependent since both access the same file

(5) Unknown Dependence

- When the dependence relation cannot be determined between two statements, then it is known as unknown dependence.

Example of Data dependencies

Consider the following code

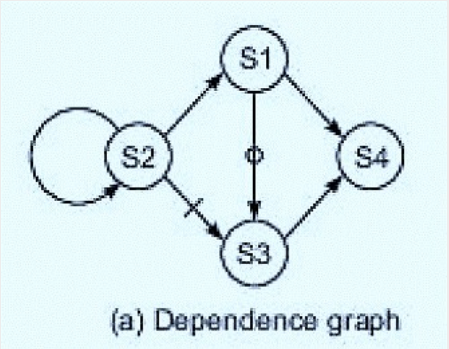
S1: Load R1, A / R1 ⟵ memory(A) /

S2: Add R2, R1 / R2 ⟵ (R1) + (R2) /

S3: Move R1, R3 / R1 ⟵ (R3) /

S4: Store B, R1 / memory(B) ⟵ (R1) /

- S2 is flow dependent on S1 because the variable A passed via the register R1
- S3 is anti-dependent on S2 because on potential conflicts in register content in R1
- S3 is output dependent on S1 because they both modify the same register R1



(a) Dependence graph

(2) Control Dependence

- An instruction is control dependent on a preceding instruction, if the outcome of later determines whether former should be executed or not.
- In the following example, instruction S2 control dependent on instruction S1
- However S3 is not control dependent up on S1 because S3 is always executed irrespective of outcome of S1.

```
S1: if(a == b)
S2: a= a + b
S3: b= a + b
```


- There is a control dependence between two statements S1 and S2 if,

(a) S1 could be possibly executed before S2

(b) The outcome of S1 execution will determine whether S2 will be executed.

(3)Resource Dependence

- It is concerned with the conflicts in using shared resources such as integer units, floating point units, registers and memory areas
- When the conflicting resource is an ALU, we call it ALU dependence
- If the conflicts involve workplace storage, it is called storage dependence

BERNSTEIN'S CONDITIONS

- Bernstein revealed a set of conditions based on which two processes can execute in parallel.
- Consider two processes P_1, and P_2 with their input sets I_1 and I_2 output sets O_1 and O_2, respectively.
- These two processes can execute in parallel and are denoted P1|| P2; if they are independent and therefore create deterministic results

- The following are the Bernstein's conditions

$$I_1 \cap O_2 = \varnothing$$
$$I_2 \cap O_1 = \varnothing$$
$$O_1 \cap O_2 = \varnothing$$

$I_1 \cap O_2 = \varnothing$ Input of the first program should not have any relation with the output of the second program (Anti-independent)

$I_2 \cap O_1 = \varnothing$ The output of the first program should not have any dependency on the input of the second program (Flow independent)

$O_1 \cap O_2 = \varnothing$ The output of the two program should not be same (output independent)

➢ Bernstein's conditions simply imply that two processes can execute in parallel if they are

- Flow-independent
- Anti-independent
- Output-independent

Prepared By EBIN PM (AP, IESCE) 75