MODULE I

Parallel computer models – Evolution of Computer Architecture, System attributes to performance, Amdahl's law for a fixed workload. Multiprocessors and Multicomputers, Multivector and SIMD computers, Architectural development tracks, Conditions of parallelism.

1.PARALLEL COMPUTER MODELS

- Parallel processing has emerged as a key enabling technology in modern computers, driven by the ever-increasing demand for higher performance, lower costs, and sustained productivity in real-life applications.
- Concurrent events are taking place in today's high-performance computers due to the common practice of multiprogramming, multiprocessing, or multicomputing.

- Parallelism appears in various forms, such as pipelining, vectorization, concurrency, simultaneity, data parallelism, partitioning, interleaving, overlapping, multiplicity, replication, time sharing, space sharing, multitasking, multiprogramming, multithreading, and distributed computing at different processing levels.

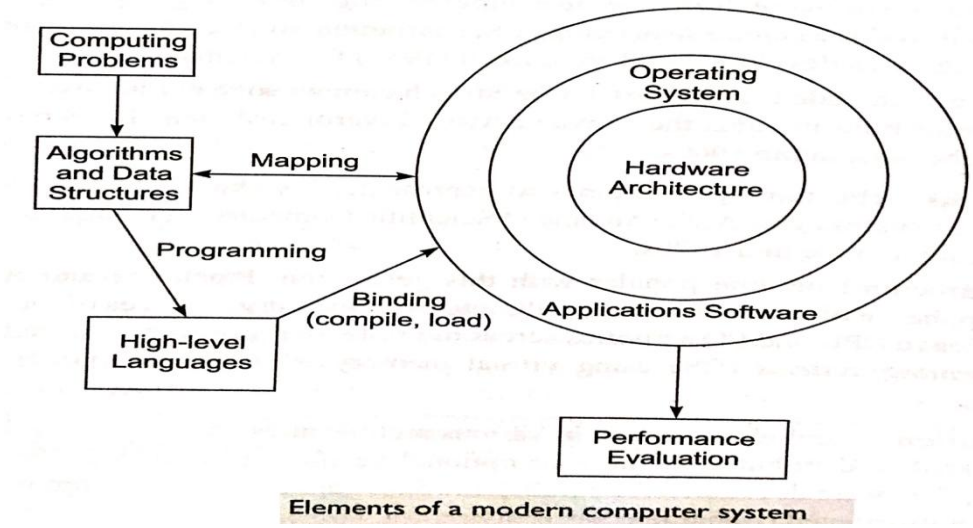
1.1THE STATE OF COMPUTING
1.1.1 Five Generation of Computers

Qn:Explain the five generations of computers?

Five Generations of Electronic Computers

Generation	*Technology and Architecture*	*Software and Applications*	*Representative Systems*
First (1945–54)	Vacuum tubes and relay memories, CPU driven by PC and accumulator, fixed-point arithmetic.	Machine/assembly languages, single user, no subroutine linkage, programmed I/O using CPU.	ENIAC, Princeton IAS, IBM 701.
Second (1955–64)	Discrete transistors and core memories, floating-point arithmetic, I/O processors, multiplexed memory access.	HLL used with compilers, subroutine libraries, batch processing monitor.	IBM 7090, CDC 1604, Univac LARC.
Third (1965–74)	Integrated circuits (SSI/-MSI), microprogramming, pipelining, cache, and lookahead processors.	Multiprogramming and time-sharing OS, multiuser applications.	IBM 360/370, CDC 6600, TI-ASC, PDP-8.
Fourth (1975–90)	LSI/VLSI and semiconductor memory, multiprocessors, vector supercomputers, multicomputers.	Multiprocessor OS, languages, compilers, and environments for parallel processing.	VAX 9000, Cray X-MP, IBM 3090, BBN TC2000.
Fifth (1991–present)	Advanced VLSI processors, memory, and switches, high-density packaging, scalable architectures.	Superscalar processors, systems on a chip, massively parallel processing, grand challenge applications, heterogeneous processing.	

1.1.2 Elements of Modern Computer



Elements of a modern computer system

1.1.3 Evolution of Computer Architecture

Qn:Describe the evolution of parallel computer architecture?

Qn:Explain the term look ahead parallelism?

The study of computer architecture involves both programming/software requirements and hardware organization. Therefore the study of architecture covers both **instruction set architectures** and **machine implementation organizations.**

As shown in figure below, Evolution Started with the **von Neumann architecture** built as a **sequential machine executing scalar data** . The sequential computer was improved from bit-serial to word—parallel operations, and from fixed—point to floating point operations. The von Neumann architecture is slow due to sequential execution of instructions in programs.

Lookahead , parallelism, and pipelining: Lookahead techniques were introduced to prefetch instructions in order to **overlap I/E** (instruction fetch/ decode and execution) operations and to enable **functional parallelism** .

Functional parallelism was supported by two approaches: One is to use **multiple functional units** simultaneously, and the other is to practice **pipelining** at various processing levels.

The latter includes pipelined instruction execution, pipelined arithmetic computations, and memory-access operations. Pipelining has proven especially attractive in performing identical operations repeatedly over **vector data strings**.

A vector is one dimensional array of numbers. A vector processor is CPU that implements an instruction set containing instructions that operate on one dimensional arrays of data called vectors.

Vector operations were originally carried out **implicitly** by software-controlled looping using scalar pipeline processors.

Explicit vector instructions were introduced with the appearance of vector processors. A vector processors equipped with multiple vector pipelines that can be concurrently used under hardware or firmware control.

There are two Families of pipelined vector processors:

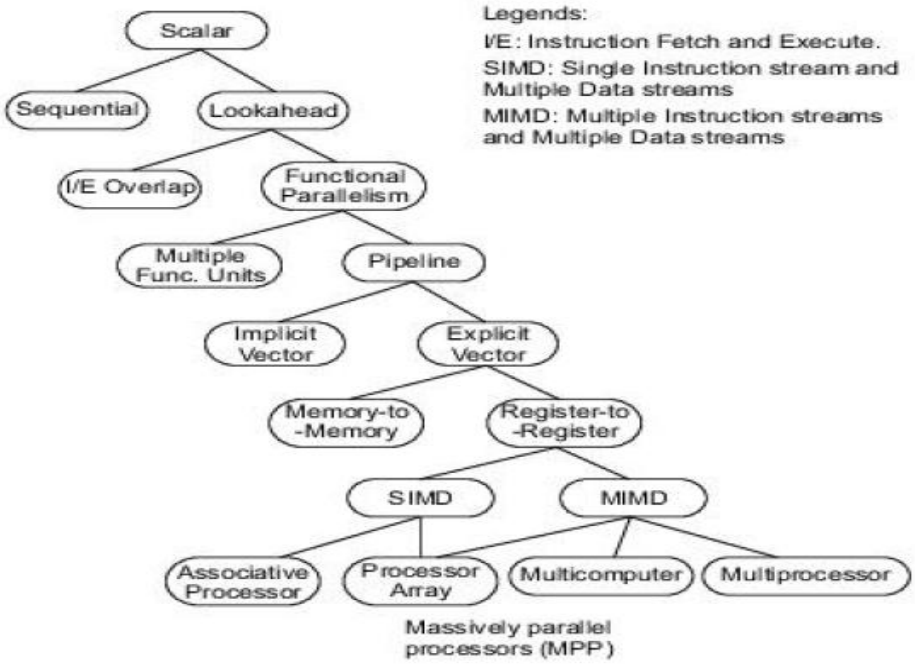
- **Memory –to-memory- architecture** supports the pipelined flow of vector operands directly from the memory to pipelines and then back to the memory.
- **Register-to register** architecture uses vector registers to interface between the memory and functional pipelines.

Another important branch of the architecture tree consists of the **SIMD** computers for synchronized vector processing. An SIMD computer exploits spatial parallelism rather than temporal parallelism as in a pipelined computer .SIMD computing is achieved through the use of an **array of processing elements [PEs]** synchronised by the same controller. Associative memory can be used to build SIMD **associative processors.**

Intrinsic parallel computers are those that execute programs in **MIMD mode**.
There are two major classes of parallel computers, namely, **Shared memory multiprocessors** and **message passing multicomputer**. The major distinction between multiprocessors and multicomputer lies in memory sharing and the mechanisms used for interprocessor communication.

The processors in a **multiprocessor system** communicate with each other through **shared variables in a common memory**.

Each computer node in a **multicomputer system** has a local memory, unshared with other nodes. lnterprocessor communication is done through **message passing** among the nodes.



Tree showing architectural evolution from sequential scalar computers to vector processors and parallel computers

Flynn's Classification (classified into 4)

Qn:Describe briefly about the operational model of SIMD computer with an example?

Qn: Characterize the architectural operations of SIMD and MIMD computers?

Describe briefly about Flynn's classification?

- Michael Flynn (1972) introduced a classification of various computer architectures based on notions of instruction and data streams. **Stream** denote a sequence of items (instructions or data) as executed or operated upon by a single processor. Two types of information flow into a processor: **instructions and data**. The **instruction stream** is defined as the sequence of instructions performed by the processing unit. The **data stream** is defined as the data traffic exchanged between the memory and the processing unit.

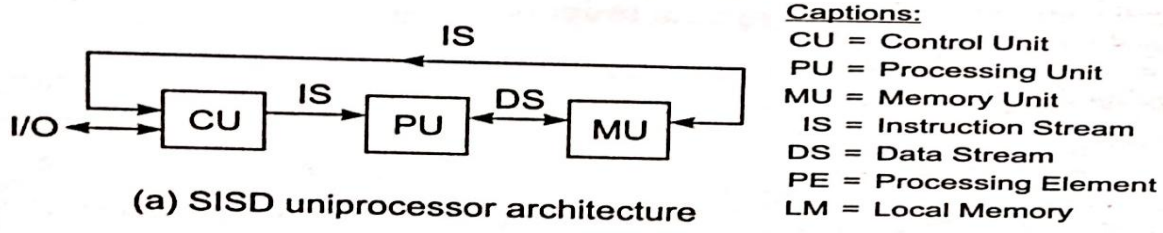
- Both instructions and data are fetched from the memory modules. Instructions are decoded by the control unit, which sends the decoded instruction to the processor units for execution. Data streams flow between the processors and the memory bidirectionally. Each instruction stream is generated by an independent control unit.

According to Flynn's classification, either of the instruction or data streams can be single or multiple. Computer architecture can be classified
into the

- **single-instruction single-data streams (SISD);**
- **single-instruction multiple-data streams (SIMD);**
- **multiple-instruction single-data streams (MISD); and**
- **multiple-instruction multiple-data streams (MIMD).**

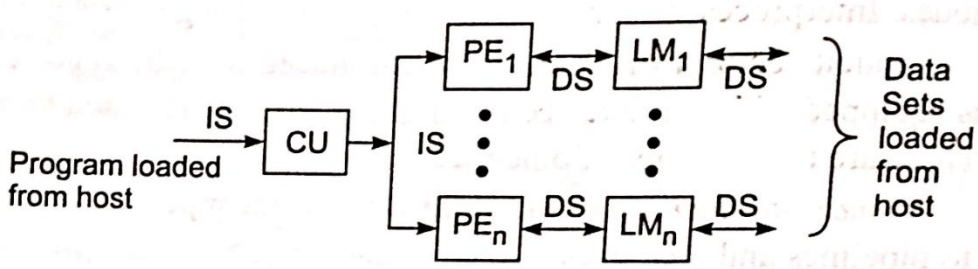
1. **SISD(Single Instruction Single Data Stream)**

- Conventional sequential machines are called SISD -[single instruction stream over single data stream] computers. Instructions are executed sequentially but may be overlapped in their execution stages (pipelining).

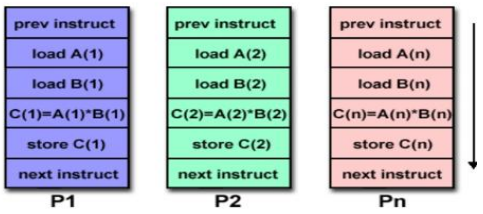


(a) SISD uniprocessor architecture

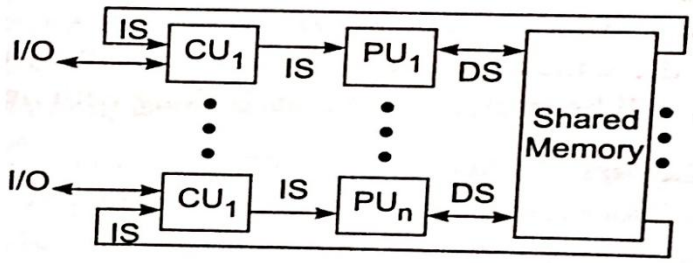
2. **SIMD(Single Instruction Multiple Data Stream)** – Represents vector computers/array processors equipped with scalar and vector hardware.There are multiple processing elements supervised by the same control unit. All PEs receive the same instruction broadcast from the control unit but operate on different data sets from distinct data streams.



(b) SIMD architecture (with distributed memory)



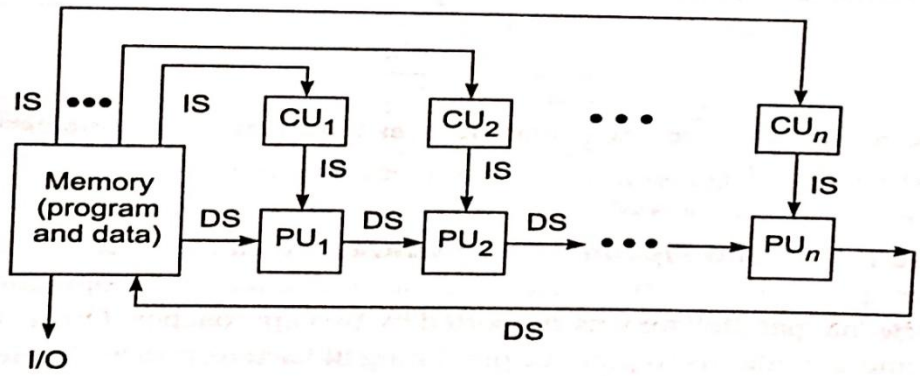
3. **<u>MIMD(multiple instructions over multiple data stream)</u>** – most popular model. parallel computers ters are reserved for MIMD



(c) MIMD architecture (with shared memory)

4. **<u>MISD(multiple instruction over single data stream)</u>**

The same data stream flows through a linear array of processors executing different instruction streams. This architecture is also known as systolic arrays For pipelined execution of specific algorithms.



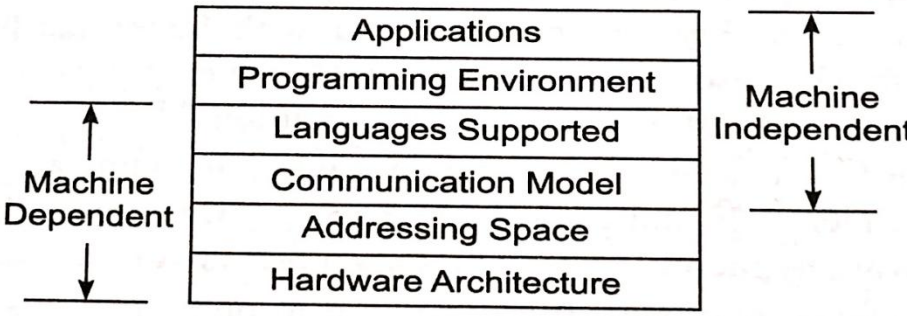
(d) MISD architecture (the systolic array)

Of the four machine models, most parallel computers built in the past assumed the MIMD model for general purpose computations. The SIMD and MISD models are more suitable for special-purpose

computations. For this reason, MIMD is the most popular model, SIMD next, and MISD the least popular model being applied in commercial machines.

Six Layers for Computer System Development

Qn:describe Six layers of computer system development?



Six layers for computer system development (

A layered development of parallel computers is illustrated in Fig. above, based on a classification by Lionel Ni [I990].

- **Hardware configurations** differ from machine to machine, even those of the same model.

- The **address space** of a processor in a computer system varies among different architectures. It depends on the memory organization, which is **machine—dependent.** These features are up to the designer and should match the target application domains.

- On the other hand, we want to develop application **programs and programming environments** which are **machine-independent.** Independent of machine architecture, the user programs can be ported to many computers with minimum conversion costs.

- **High- level languages and communication models** depend on the Architectural choices made in a computer system. From a programmer's viewpoint, these **two layers should be architecture-transparent.**

- **Programming languages** such as Fortran, C, C++, Pascal, Ada, Lisp and others can be supported by most computers. However, the **communication models**, shared variables versus message passing, are mostly machine-dependent.

Challenges in Parallel Processing: Major challenge is on the software and application side.It is still difficult to program parallel and vector computers.High performance computers should provide fast and accurate solutions to scientific ,engineering , business,social and defense problems.

2.SYSTEM ATTRIBUTES TO PERFORMANCE

Qn:Define the terms a)clock rate, b)CPI, MIPS rate, Throughput rate?

Qn:List out the matrics affecting scalability of a computer system for a given application?(Ic,p,m,k t)

Qn:List and explain four system attributes affecting the performance of CPU?(

instruction-set architecture, compiler technology, CPU implementation and control, and cache and memory hierarchy)

System Attributes versus Performance Factors

The ideal performance of a computer system requires a perfect match between machine capability and program behaviour.

Machine capability can be enhanced with better hardware technology, however program behaviour is difficult to predict due to its dependence on application and run-time conditions.

Below are the five fundamental factors for projecting the performance of a computer.

CPU is driven by a clock of constant clock with a **cycle time (τ). T**he inverse of cycle time is the **clock rate (f=1/ τ)**

Size of the program is determined by the ***Instruction Count(Ic).*** Different instructions in a particular program may require different number of clock cycles to execute. So,

Cycles per instruction (CPI):-is an important parameter for measuring the time needed to execute an instruction .

Execution Time/CPU Time (T): Let **Ic** be Instruction Count or total number of instructions in the program. The Execution Time or CPU time (T) will be:

$$T = Ic \times CPI \times \tau \qquad \text{Eq.1.1}$$

- The execution of an instruction involves the instruction fetch, decode, operand fetch, execution and store results.

Only instruction decode and execution phases are carried out in CPU. The remaining three operations may require access to memory

The CPI of an instruction type can be divided into two component terms corresponding to the total processor cycles and memory cycles needed to complete the execution of the instruction. That is :-

CPI = Instruction Cycle = Processor Cycles +Memory Cycles. ie

CPI = Instruction cycle = p + m + k

where

m = number of memory references

P = number of processor cycles

k = latency factor (how much the memory is slow w.r.t to CPU)

Therefore, Equation 1.1 can be rewrite as follows:-

$$T = Ic \times (p+m+k) \times \tau \qquad \text{Eq.1.2}$$

From the above equation the five factors affecting performance are:-Ic ,p,m,k, τ

System Attributes: The above five performance factors (**Ic ,p,m,k, τ**) are influenced by four system attributes**: instruction-set architecture, compiler technology, CPU implementation and control, and cache and memory hierarchy**, as specified in Table 1.2 below.

The instruction-set architecture affects the program length (1,) and processor cycles needed (p). The compiler technology affects the values of Ic, p, and the memory reference count (m). The CPU implementation and control determine the total processor time (p * τ) needed. Finally, the memory technology and hierarchy design affect the memory access latency (k * τ). The above CPU time can be used as a basis in estimating the execution rate of a processor.

Table 1.2 *Performance Factors versus System Attributes*

System Attributes	Performance Factors				
	Instr. Count, I_c	Average Cycles per Instruction, CPI			Processor Cycle Time, τ
		Processor Cycles per Instruction, p	Memory References per Instruction, m	Memory-Access Latency, k	
Instruction-set Architecture	✓	✓			
Compiler Technology	✓	✓	✓		
Processor Implementation and Control		✓			✓
Cache and Memory Hierarchy				✓	✓

MIPS Rate. The processor speed is often measured in terms of million instructions per second (MIPS). We simply call it the MIPS rate of a given processor.

Let C be the total number of cycles needed to execute a given program (ie **C=Ic * CPI**).
Then the **CPU time** in Eq. 1.2 can be estimated as

$$T = C * \tau$$
$$= C/f .$$

Furthermore, **CPI = C/Ic and**

$$T = Ic * CPI * \tau$$
$$= Ic * CPI/f.$$

MIPS rate $= Ic/(T*10^6)$
$= f/(CPI*10^6)$
$= (f*Ic)/(C*10^6)$

Eq.1.3

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{\text{CPI} \times 10^6} = \frac{f \times I_c}{C \times 10^6}$$

Or

Now Based on Equation 1.2 and 1.3

$$\textbf{CPU Time ,T} = (IC*10^{-6})/MIPS$$

MFLOPS:

Most compute intensive applications in science and engineering make heavy use of floating point operations. For such applications a more relevant measure of performance is floating point operations per second abbreviated as mflops. With prefix mega(10^6),giga(10^9) tera(10^12) or peta(10^15). Floating-point performance is expressed as millions of floating-point operations per second (MFLOPS), defined as follows ,only with floating-point instructions.

Number of executed floating-point operations in a program

$$\text{MFLOPS} = \frac{}{\text{execution time} * 10^6}$$

Throughput Rate:- Number of programs executed per unit time is called system throughput w_s(in programs per second).In a multiprogrammed system, the system throughput is often lower than CPU throughput Wp defined by

$$\mathbf{Wp} = \frac{\mathbf{f}}{\mathbf{Ic * CPI}} \qquad \textbf{Eq.1.4}$$

W = 1 / T
OR
W =(MIPS*10^6)/Ic

Problems:-

1 **A benchmark program is executed on a 40MHz processor. The benchmark program has the following statistics.**

Instruction Type	Instruction Count	Clock Cycle Count
Arithmetic	45000	1
Branch	32000	2
Load/Store	15000	2
Floating Point	8000	2

Calculate average CPI, MIPS rate & execution time for the above benchmark program

Given Clock speed of the processor= 40 MHz=40*10^6 Hz

Average CPI= C/Ic

=Total cycles to execute the program/Instruction count

=(45000 *1 + 32000*2 + 1500*2 + 8000*2) /(45000 + 3200 + 15000 + 8000)

=155000/100000

=1.55

Execution Time ,T= C / f

T = 150000 / 40 *10^6

T = 0.155 / 40

T = .003875 s

T = 3.875 ms (since 1s=1000ms)

MIPS rate = Ic / T * 10^6

MIPS rate = 100000/(0.003875*$10^{6)}$

= 25.8

2. Consider the execution of an object code with 2 *10^6 instructions on a 400 MHz processor. The program consists of four major types of instructions. The instruction mix and the number of cycles [CPI] needed for each instruction type are given below based on the result of a program trace experiment:

Instruction type	CPI	Instruction mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

(a) Calculate the average CPl when the program is executed on a uniprocessor with the above trace results.
(b) Calculate the corresponding MIPS rate based on the CPI obtained in part (a).

Answer:
average CPI = 0.6 +(2 *0.18) +(4* 0.12) +(8 *0.1) =2.24.
MIPS rate = f/(CPI*$10^{6)}$
=(400* 10^6) /(2.24 *10^6) =178

Programming Environments:

Qn: Difference between Implicit Parallelism and Explicit forms of Parallelism?

The programmability of a computer depends on the programming environment provided to the users. conventional uniprocessor computers are programmed in a ***sequential environment*** in which instructions are executed one after another in a sequential manner. Parallel computers employs parallel environment where parallelism is automatically exploited.. Based on the programming environments required parallelism can be of two types:

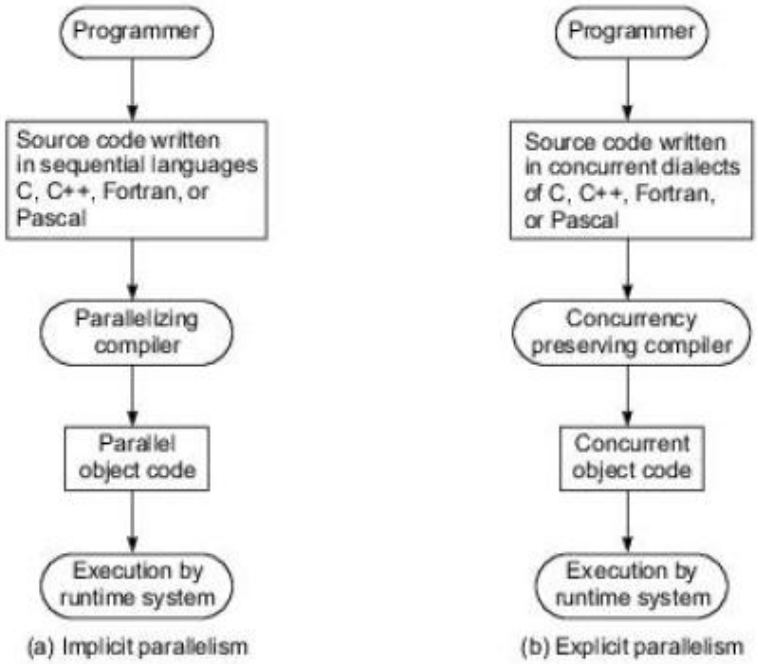


Fig. 1.5 Two approaches to parallel programming (Courtesy of Charles Seitz; adapted with permission from "Concurrent Architectures", p. 51 and p. 53, *VLSI and Parallel Computation*, edited by Suaya and Birtwistle, Morgan Kaufmann Publishers, 1990)

IMPLICIT PARALLELISM	EXPLICIT PARALLELISM
1. In computer science, implicit parallelism is a characteristic of a programming language that allows a **compiler or interpreter** to automatically exploit the parallelism inherent to the computations expressed by some of the language's constructs.	1. In computer programming, explicit parallelism is the representation of concurrent computations by means of primitives in the form of special-purpose directives or function calls. Most parallel primitives are related to process synchronization, communication or task partitioning.
2. Uses conventional languages such as C, C++, Fortran or Pascal to write source program	2. Requires more effort by programmers to develop a source program using parallel dialects like C, C++, Fortran and Pascal.
3. The **sequentially coded** source program is translated into parallel object code by a **parallelizing compiler.**	3. Parallelism is explicitly specified in the user programs.
4. Compiler detects parallelism and assigns target machine resources.	4. Burden on compiler is reduced as parallelism specified explicitly.
5. Success relies on intelligence of parallelizing compiler. Requires less effort from programmers.	5. Programmer's effort is more- special s/w tools needed to make environment more friendly to user groups.
6. Applied in shared memory multiprocessors.	6. Applied in Loosely coupled Multiprocessors to tightly coupled VLIW

3. MULTIPROCESSORS AND MULTICOMPUTERS

Qn. Distinguish between multiprocessor and multicomputers based on their structure, resource sharing, and interprocessor communication)?

Qn:List differences between UMA, NUMA,COMA Models?

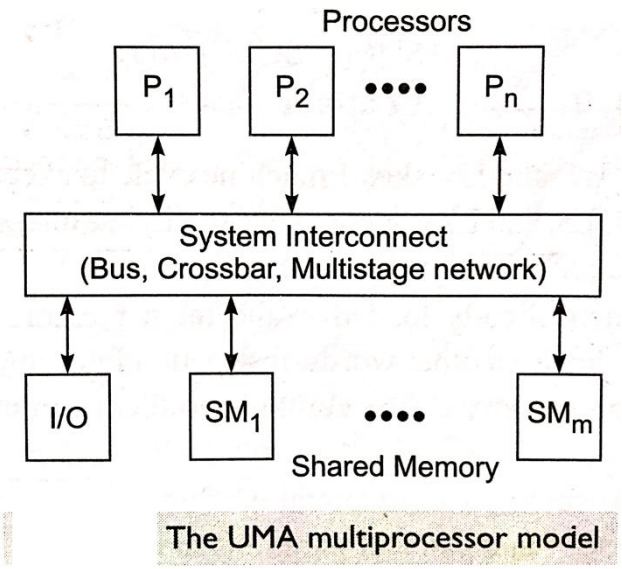
Qn:Describe in detail the types of shared memory multiprocessors models?

- 2 categories of parallel computer
- Distinguished by having shared memory(multiprocessors) or unshared distributed memory(multi computers)

Multiprocessors	Multicomputer
1. Single computer with multiple processors	1. Multiple autonomous computers
2. Each PE's(CPU/processing elements) do not have their own individual memories – memory and I/O resources are shared – Thus called **Shared Memory Multiprocessors**	2. Each PE's has its own memory and resources – no sharing – Thus called **Distributed Memory Multicomputers**
	3. Communication between PE's not

3. Communication between PE's a must 4. Tightly coupled – due to high degree of resource sharing 5. Use Dynamic Network – thus communication links can be reconfigured 6. Ex: Sequent Symmetry S-81 7. 3 Types - **UMA model** - **NUMA model** **COMA model**	mandatory 4. Loosely coupled as there is no resource sharing 5. Use Static Network – connection of switching units is fixed 6. Ex: Message Passing Multicomputer **7. NORMA model/ Distributed-Memory Multicomputer**

THE UMA MODEL



The UMA multiprocessor model

- Physical memory is uniformly shared by all processors
- All processors (PE1….PEn) take equal access time to memory – Thus its termed as **Uniform Memory Access Computers**
- Each PE can have its own private Cache
- High degree of resource sharing(memory and I/O) – Tightly Coupled
- Interconnection Network can be – Common bus, cross bar switch or Multistage n/w (discussed later)
- When all PE's have equal access to all peripheral devices – **Symmetric Multiprocessor**
- In **Asymmetric multiprocessor** only one subset of processors have peripheral access. Master Processors control Slave (attached) processors.

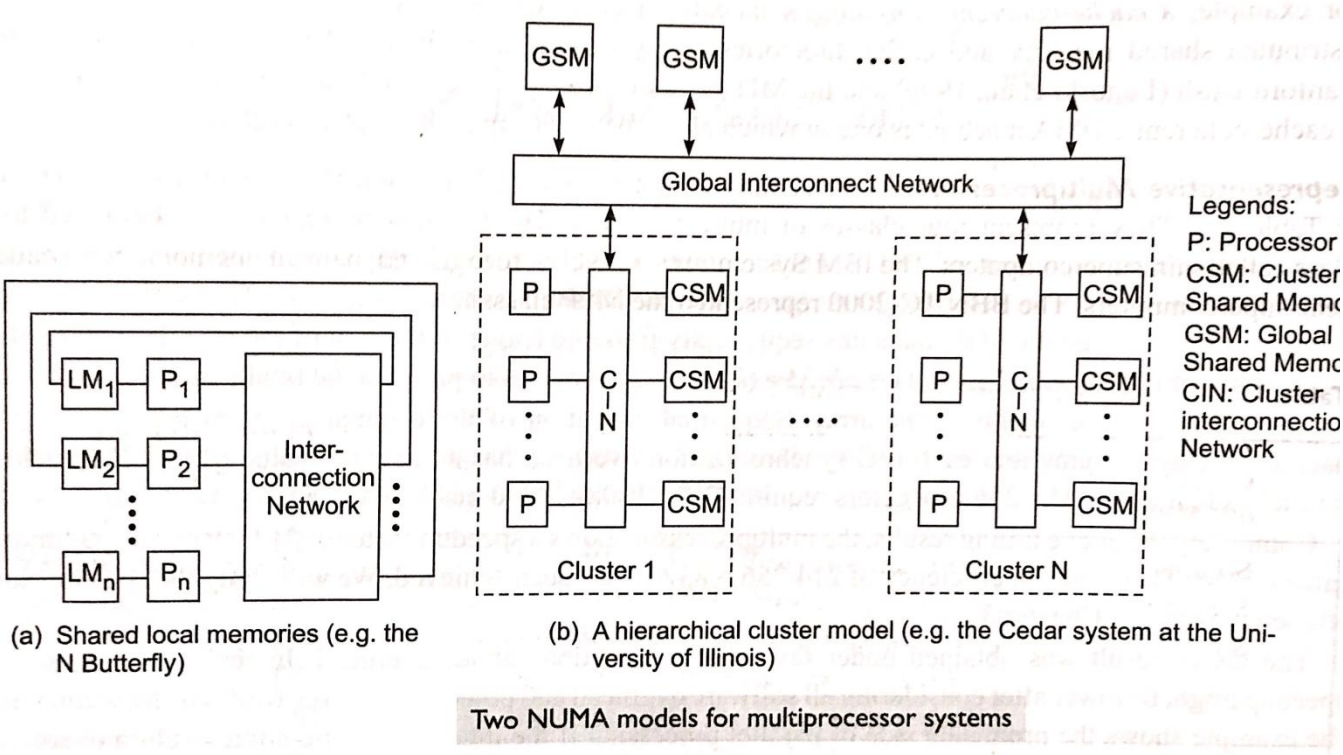
Applications of UMA Model

- Suitable for general purpose and time sharing application by multiple users
- Can be used to speed up execution of a single program in time critical application

DISADVANTAGES

- Interacting process cause simultaneous access to same locations – cause problem when an update is followed by read operation (old value will be read)
- Poor Scalability – as no: of processors increase –shared memory area increase-thus n/w becomes bottleneck.
- No: of processors usually in range(10-100)

THE NUMA MODEL (Ex: BBN Butterfly)



Two NUMA models for multiprocessor systems

- Access time varies with location of memory
- Shared memory is distributed to all processors – Local memories
- Collection of all local memories forms global memory space accessible by all processors.
- Its faster to access content within local memory of a processor than to access remote memory attached to another processor (delay through interconnection) – Thus named as **NON-Uniform Memory access** – because access time depends on whether data available in local memory of the processor itself or not.

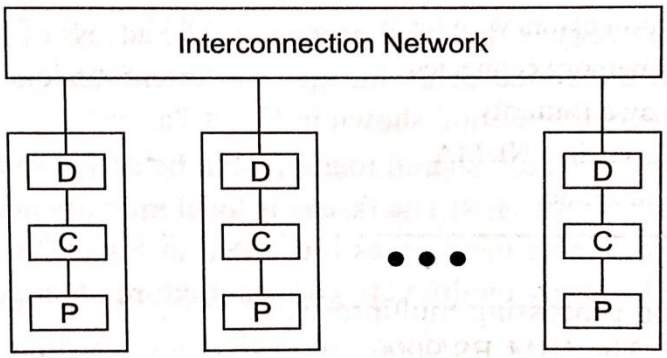
Advantage

- – reduces n/w bottleneck issue that occurs in UMA as prcessors have a direct access path to attached local memory

3 types of Memory access pattern

I. Fastest to access – local memory of PE itself
II. Next fastest – global memory shared by PE/Cluster
III. Slowest to access – remote memory(local memory of PE from another Cluster)

THE COMA MODEL



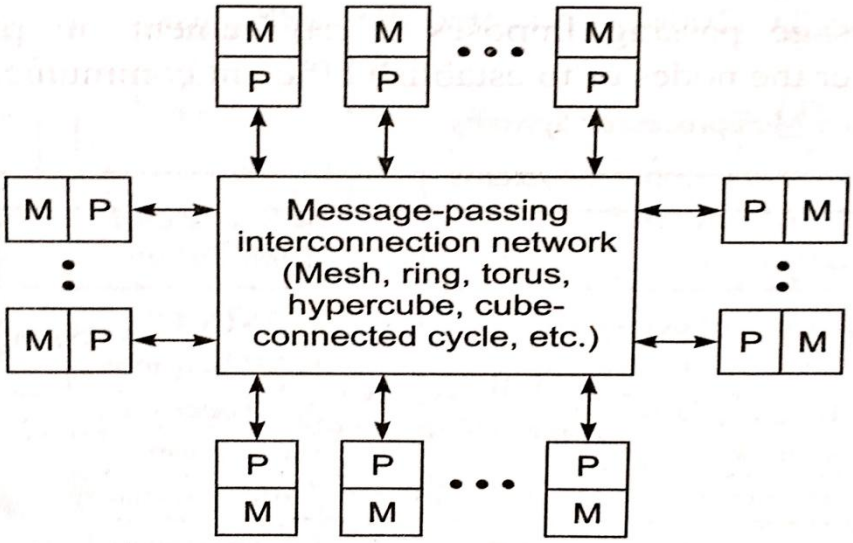
The COMA model of a multiprocessor (P: Processor, C: Cache, D: Directory;

- Multiprocessor + Cache Memory = COMA Model
- **Multiprocessor using cache only memory**
- Ex: Data diffusion Machine , KSR-1 machine
- Special case of NUMA machine in which distributed main memories are converted to caches – all caches together form a global address space
- Remote cache access is assisted by Distributed Cache directories (D in fig abv)

Application of COMA

- General purpose multiuser applications

DISTRIBUTED MEMORY / NORMA / MULTICOMPUTERS (Intel Paragon, nCUBE, SuperNode1000)



Generic model of a message-passing multicomputer

- Consists of multiple computers(called nodes)
- A node is an autonomous computer consisting of processor, local memory attached disks or I/O peripherals.
- Nodes interconnected by a message passing network – can be Mesh, Ring, Torus, Hypercube etc (discussed later)
- All interconnection provide point to point static connection among nodes

- Local memories are private and accessible only by processor – Thus Multicomputer are also called **No-remote-Memory –Access(NORMA)**(difference with UMA and NUMA)
- Communication between nodes if required is carried out by passing messages through static connection network.

Advantages over Shared Memory

- Scalable and Flexible : we can add CPU's
- Reliable and accessible : since with Shared memory a failure can bring the whole system down

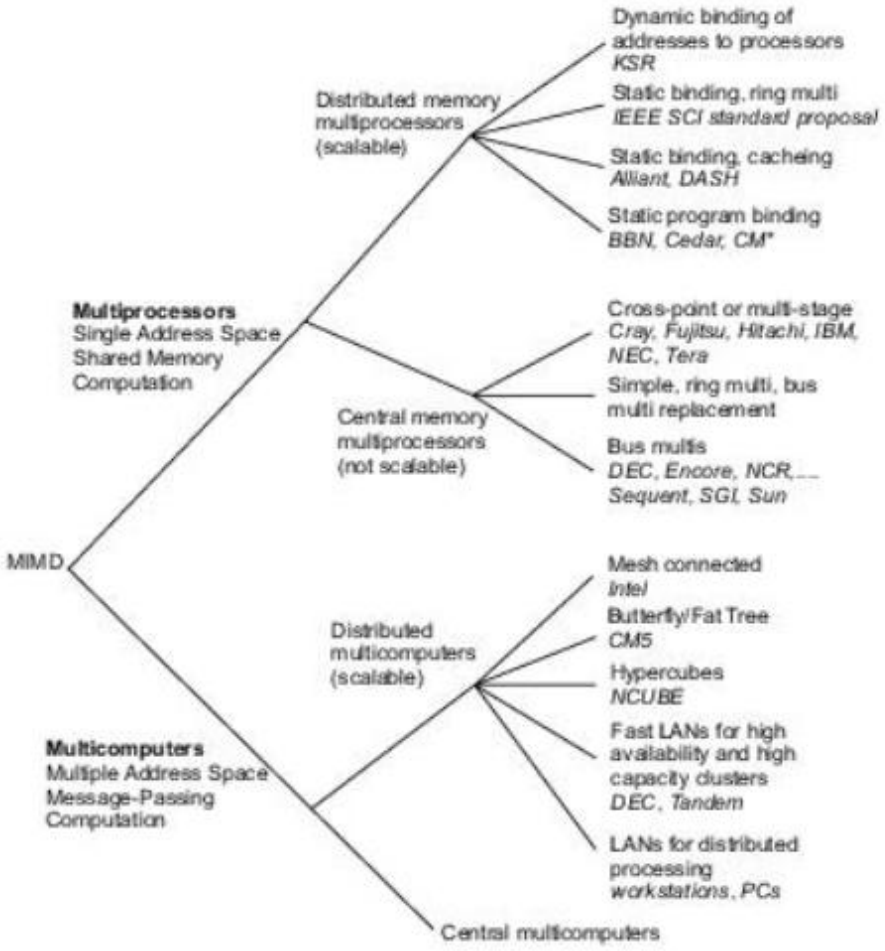
Disadvantage

- Considered harder to program because we are used to programming on common memory systems.

ATaxonomy of MIMD Computers

The architectural trend for general-purpose parallel computers is in favor of MIMD configurations with various memory configurations. Gordon Bell (1992) has provided a taxonomy of MIMD machines. He considers shared-memory multiproccssors as having a single address space. Scalable multiproccssors or multicomputers must use distributed memory. Multiprocessors using centrally shared memory have limited scalability.

Bell's taxonomy of MIMD computers



4. MULTIVECTOR and SIMD COMPUTERS

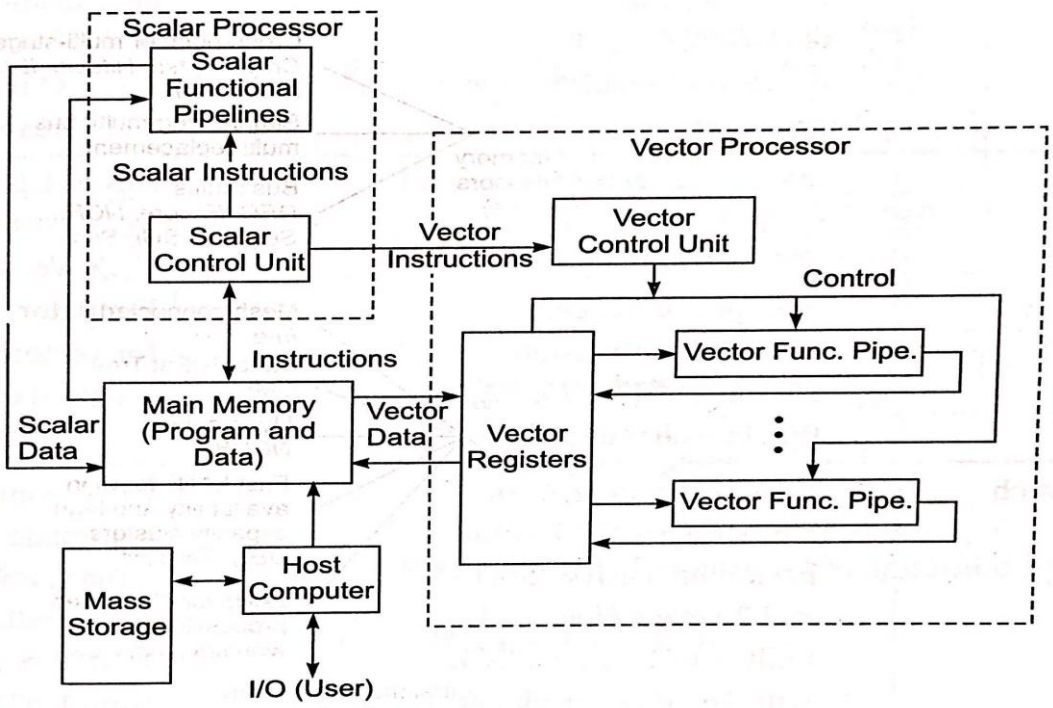
Qn:Write on the structure and functioning of vector supercomputers?

Qn:Differentiate multivector and SIMD computers?

We can classify supercomputers into 2

- Pipelined Vector machines using powerful processors equipped with vector hardware
- SIMD computers emphasizing on massive data parallelism

VECTOR SUPERCOMPUTERS



- A vector computer is usually built on top of scalar processor – its attached to scalar processor as an optional feature
- Program and data are first loaded into main memory through host computer
- Instructions are first decoded by scalar Control Unit

– if it's a scalar operation or program control operation, it will be directly executed using scalar functional pipelines.

-If its vector operation it will be send to the vector control unit. The control unit supervises the flow of vector data between main memory and vector functional pipelines. Vector data flow is coordinated by the control unit. A number of vector functional pipelines may be built into a vector processor

VECTOR PROCESSOR MODELS

Qn:Distinguish between Register to Register and Memory to memory architecture for building conventional multivector supercomputers?

2 types :

- **Register to Register architecture**
- **Memory to memory architecture**

REGISTER to REGISTER architecture

- The fig above shows a **register to register architecture**. Vector registers are used to hold vector operands, intermediate and final vector results.
- All vector registers are programmable
- Length of vector register is usually fixed (ex 64 bit for CRAY Series Supercomputer). Some machines use reconfigurable vector registers to dynamically match register length(ex: Fujitsu VP2000)
- Generally there are fixed no: of vector registers and functional pipilines in vector processors – hence they must be reserved in advance to avoid conflicts

MEMORY to MEMORY architecture

- differs from register to register architecture in use of vector stream unit in place of vector registers.
- Vector operands and results are directly retrieved from and stored into main memory in superwords (ex: 512 bits in Cyber 205)

Representative Supercomputers – Stardent 3000, Convex C3, IBM 390Cray Research Y-MP family, Fujitsu VP2000)

SIMD SUPERCOMPUTERS

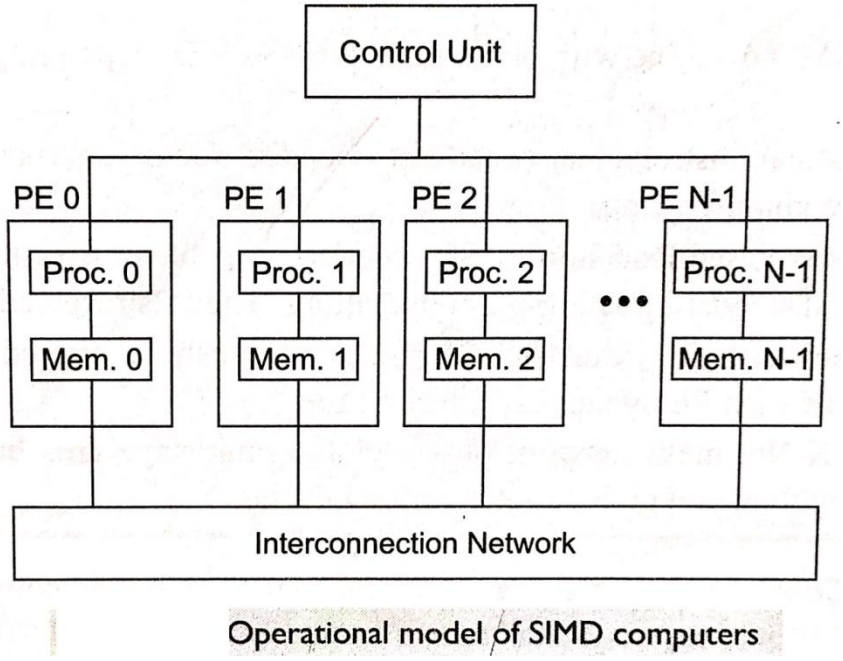
SIMD computer have One Control Processor and several Processing Elements.All Processing Elements execute the same instruction at the same time. Interconnection network between PEs determines memory access and PE Interaction.

SIMD computer (array processor) is normally interfaced to a host computer through the control unit. The host computer is a general purpose machine which serves as the operating manager of the entire system.

Each PE_i is essentially an ALU with attached working registers and local memory PEM_i for the storage of distributed data. The CU also has its own main memory for storage of programs. The function of CU is to decode all instruction and determine where the decoded instructions should be executed. Scalar or control type instructions are directly executed inside the CU. Vector instructions are broadcasted to the PEs for distributed execution.

Masking schemes are used to control the status of each PE during the execution of a vector instruction. Each PE may be either active or disabled during an instruction cycle. A masking vector is used to control the status of all PEs. Only enabled PEs perform computation. Data exchanges among the PEs are done via an inter-PE communication network, which performs all necessary data routing and manipulation functions. This interconnection network is under the control of the control unit.

Fig below shows an abstract model of SIMD computer having single instruction over multiple data streams



Operational model of SIMD computers

SIMD Machine Model

An operational model of SIMD computer is specified by a 5-tupe :

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

Where

1. **N** is the number of Processing Element's(PE's) in m/c (ex: Illiac IV has 64 PE's)
2. **C** is the set of instructions directly executed by Control Unit(CU)- including scalar and program flow control instructions.
3. **I** is the set of instruction broadcast by CU to all PE's for parallel execution(Ex: arithmetic, logic , data routing, masking etc)
4. **M** is the set of masking schemes which sets each PE's into enabled and disabled mode
5. **R** is the data-routing function, specifies the pattern to be set up in the interconnection n/w for inter-PE communications.

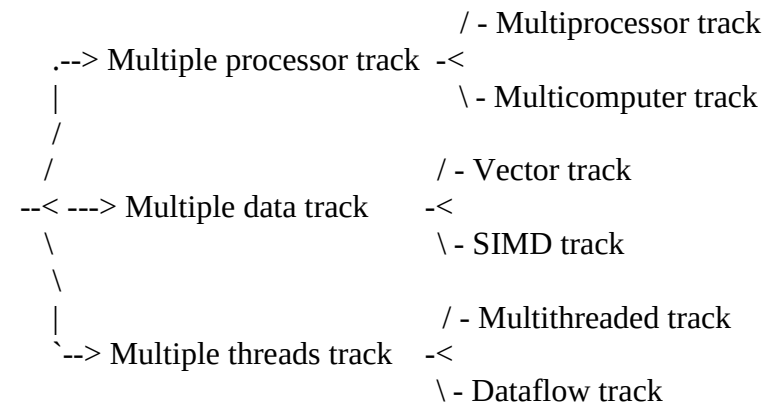
Representative Systems:

• MasPar MP-1 (1024 to 16384 PEs), CM-2 (65536 PEs), DAP600 Family (up to 4096 PEs), Illiac-IV (64 PEs)

5.ARCHITECTURAL DEVELOPMENT TRACKS

Qn: Explain the three tracks of evolution of parallel computers?

The evolution of parallel computers sprang along three tracks. These tracks are distinguished by similarity in the underlying parallel computational models.



1.4.1 Multiple Processor track

- In the multiple processor track, the source of parallelism is assumed to be the concurrent execution of different threads on different processors, with communication occurring either through shared memory (multiprocessor track) or via message passing (multicomputer track).

- In the multiple data track, the source of parallelism is assumed to be the opportunity to execute the same code on massive amounts of data. This could be through the execution of the same instruction on a sequence of data elements (vector track) or through the execution of the same sequence of instructions on similar data sets (SIMD track).

- In the multiple threads track, the source of parallelism is assumed to be the interleaved execution of different threads on the same processor so as to hide synchronization delays between threads executing on different processors. Thread interleaving could be coarse (multithreaded track) or fine (dataflow track).

Architecture of todays systems pursue development tracks. There are mainly 3 tracks. These tracks are illustrious by likeness in computational model & technological bases.

1. **Multiple Processor tracks:** multiple processor system can be shared memory multiprocessor or a distributed memory multicomputer.

(a) Shared Memory track:

Shared memory track shows a track of multiprocessor development employing a single address space in the entire system

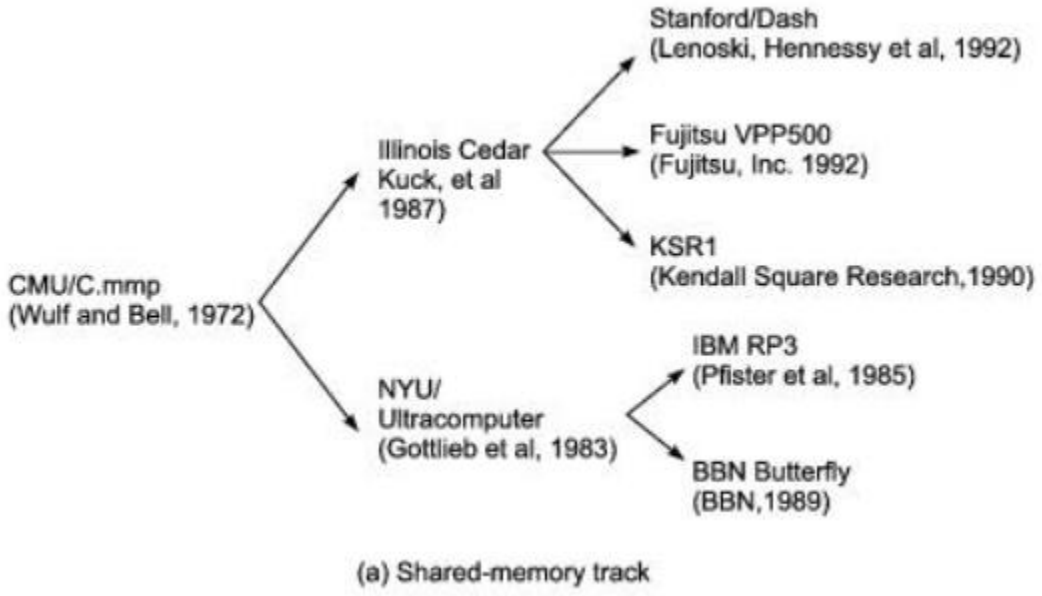


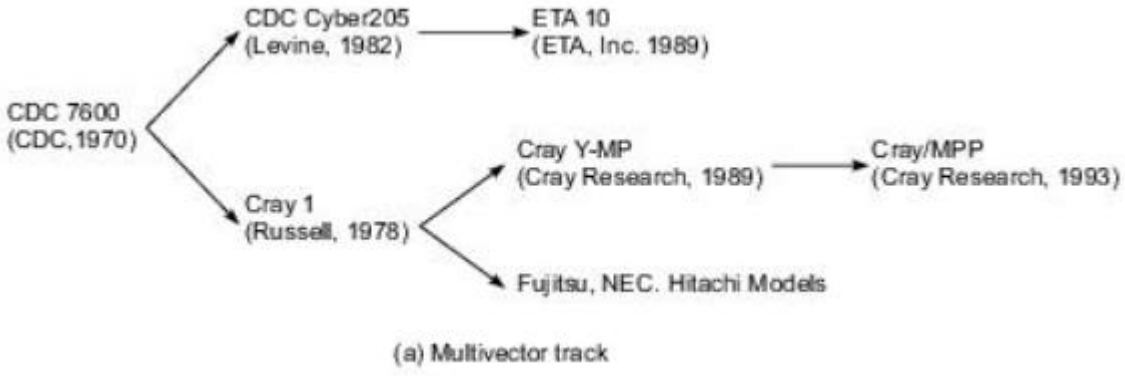
Fig. 1.17 Two multiple-processor tracks with and without shared memory

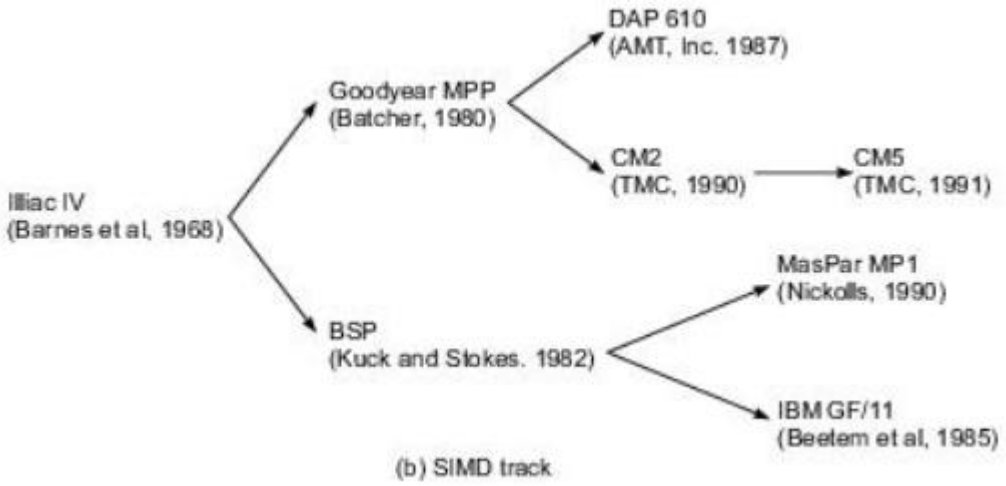
Message Passing Track

Th Cosmic Cube pioneered the development of message passing multicomputers.

2.Multivector and SIMD tracks

Multivector and SIMD tracks are useful for concurrent scalar/vector processing





(b) SIMD track

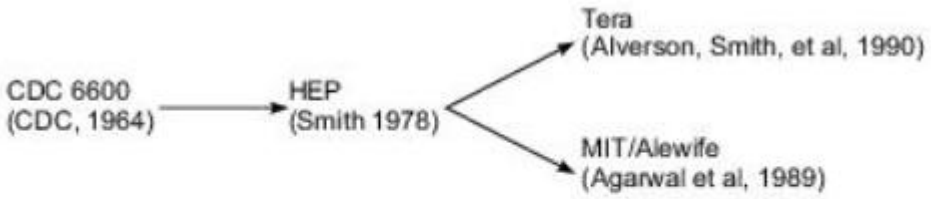
Multivector Track

These are traditional vector super computers.The CDC 7600 was first vector dual processor system.Two subtracks were derived from CDC 7600. The Cray and Japanese supercomputers are followed the register-to-register architecture. The other subtrack used memory-to-memory architecture in building vector supercomputers. We have identified only the CDC Cyber 205 and its successor the ETAICI here, for completeness in tracking different supercomputer architectures.

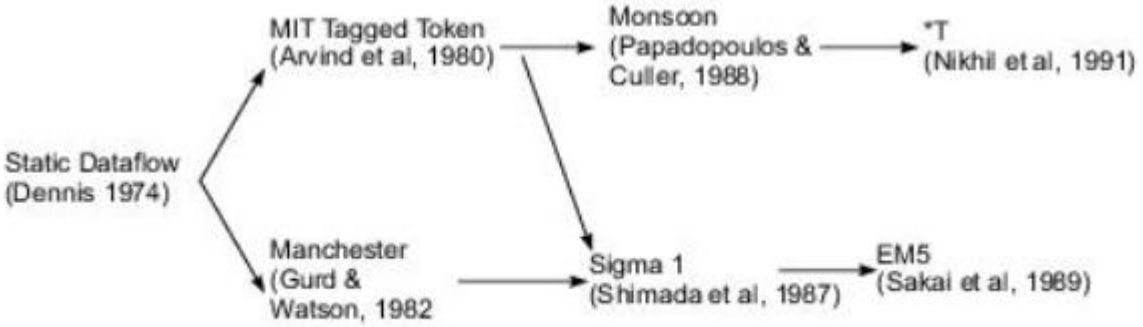
SIMD Track

The llliac IV pioneerd the construction of SIMD computers.

3.Multithreaded Track and Dataflow Track



(a) Multithreaded track



(b) Dataflow track

The term multithreading implies that there are multiple threads of control in each processor. Multithreading offers an effective mechanism for hiding long latency in building large-scale multiprocessors.. As shown in Fig. the multithreading idea was pioneered by Burton Smith [1978] in the HEP system which extended the concept of scoreboarding of multiple functional units in the CDC 6600.

Dataflow Track

The key idea is to use a dataflow mechanism,instead of a control-flow mechanism as in von Neumann machines, to direct the program flow. Fine grain instruction-level parallelism is exploited in dataflow computers.

6. CONDITIONS of PARALLELISM

Qn:Explain three types of dependencies?

The ability to execute several program segments in parallel requires each segment to be independent of the other segments. We use a **dependence graph** to describe the relation between statements The **nodes** of a dependence graph correspond to the program statement (instructions), and **directed edges** with different labels are used to represent the ordered relations among the statements. The analysis of dependence graphs shows where opportunity exists for parallelization and vectorization.

Program segments cannot be executed in parallel unless they are independent. Independence comes in several forms.

3 main types of dependencies

1. **Data dependence: situation in which a program segment (instruction) refers to the preceding statement.**
2. **Control dependence:**This refers to the situation where the order of the execution of statements cannot be determined before run time..**It occurs with branches. On many instruction pipeline architectures, the processor will not know the outcome of the branch in the fetch stage.**
3. **Resource Dependence: even if several segments are independent in other ways, they cannot be executed in parallel if there aren't sufficient processing resources (eg. Functional units)**

Data dependence: The ordering of relationship between statements is indicated by the data dependence.

Five type of data dependence are defined below:

Qn:Describe the possible hazards between read and write operations in an instruction pipeline?

1. Flow dependence: A statement S2 is flow dependent on S1 if an execution path exists from S1 to S2 and if at least one output (variables assigned) of S1 is used as input (operands to be used) to S2 .Also called RAW hazard and denoted as $S_1 \rightarrow S_2$

S1**. R2**<-R1+R3

S2. R4<-**R2**+R3

A data dependency occurs with instruction S2 as it is dependent on the completion of instruction S1.

5. **Antidependence**: Statement S2 is antidependent on the statement S1 if S2 follows S1 in program order and instruction S2 tries to writes a register or memory location before instruction S1 reads. The original order must be preserved to ensure that S1 reads correct

value. It also called WAR hazard and denoted as $S_1 \mapsto S_2$

S1. R4<-R1+**R5**

S2.**R5**<-R1+R2

6. **Output dependence** : two statements S1 and S2 are output dependent if they write to the same memory location(S1 tries to write an operand before it is written by S1). Also called WAW hazard and denoted as $S_1 \mapsto S_2$ .A WAW hazard may occur in concurrent execution environment.

S1. **R2**<-R4+R7
S2.**R2**<-R1+R3

4**. I/O dependence**: Read and write are I/O statements. I/O dependence occurs not because the same variable is involved but because the same file referenced by both I/O statement

5. **Unknown dependence**: The dependence relation between two statements cannot be determined in the following situations:

- The subscript of a variable is itself subscribed (ex:a(I(J)))
- The subscript does not contain the loop index variable. (ex: a[])
- A variable appears more than once with subscripts having different coefficients of the loop variable.
- The subscript is non linear in the loop index variable.

Thus Parallel execution of program segments which do not have total data independence can produce non-deterministic results.

Consider the following fragment of four instructions
program:

S1: Load R1, A /R1<-Memory(A)/

S2 : Add R2, R1 /R2<-(R1)+(R2)/

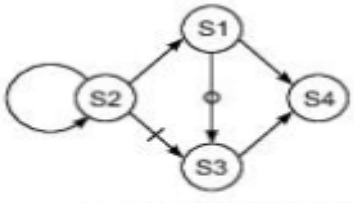
S3: Move R1, R3 /R1<-(R3)/

S4: Store B, R1 /Memory(B)<-(R1)/

• here the flow dependency S1 to S2, S3 to S4, S2 to S2

• Anti-dependency from S2to S3

• Output dependency S1 toS3



(a) Dependence graph

Consider a code fragment involving I/O opeerations

S1:	Read (4), A(I)	/Read array A from file 4/
S2:	Process	/Process data/
S3:	Write (4), B(I)	/Write array B into file 4/
S4:	Close (4)	/Close file 4/

(b) I/O dependence caused by accessing the same file by the read and write statements

2. **Control Dependence**: This refers to the situation where the order of the execution of statements cannot be determined before run time.

For example conditional statement(IF), will not be resolved until run time, where the flow of statement depends on the output of the conditional statement. Different paths taken after a conditional branch may introduce or eliminate data dependence among instructions.depend on the data hence we need to eliminate this data dependence among the instructions. This dependence also exists between operations performed in successive iterations of looping procedure.

Control-independent example:

```
for (i=0;i<n;i++)
{
a[i] = c[i];
if (a[i] < 0)
a[i] = 1;
}
```

Control-dependent ex:

```
for (i=1;i<n;i++)
{
if (a[i-1] < 0)
a[i] = 1;
}
```

Control dependence also avoids parallelism to being exploited. Compiler techniques or hardware branch prediction techniques are needed to get around the control dependence in order to exploit more parallelism.

3.Resource dependence:

Data and control dependencies are based on the independence of the work to be done. Even if several segments are independent in other ways, they cannot be executed in parallel if there aren't sufficient processing resources. Resource dependence is concerned with conflicts in using shared resources, such as registers, integer and floating point units, ALUs and memory areas among parallel events .ALU conflicts are called ALU dependence. Memory (storage) conflicts are called storage dependence.

Bernstein's Conditions –

Qn: What is the significance of Bernstein's conditions in detecting parallelism in a program?

Bernstein's conditions are a set of conditions which must exist if two processes can execute in parallel.

Bernstein's condition -1

Notation

- Let P1 and P2 be two processes.
- *Input set Ii* is the set of all input variables for a process *Pi* . *Ii* is also called the **read set** or **domain of *Pi.*** We define the input set Ii of a process Pi as the set of all input variables needed to execute the process.
- *Output set Oi* is the set of all output variables generated after execution for a process *Pi* .Oi is also called write set.

Input variables.are essentially operands which can be fetched from the memory or registers and output variables are the results to be stored in working registers or memory locations.

If P*1* and P*2* can execute in parallel (which is written as P*1* || P*2*), then:

$$\mathbf{I}_1 \cap \mathbf{O}_2 = \varnothing$$
$$\mathbf{I}_2 \cap \mathbf{O}_1 = \varnothing$$
$$\mathbf{O}_1 \cap \mathbf{O}_2 = \varnothing$$

Bernstein's condition -2

In terms of **data dependencies, Bernstein's conditions imply that two processes can execute in parallel if they are flow-independent, antindependent, and output-independent.** In general, a set of processes P_1, P_2 ,…,Pk, can execute in parallel if Bernstein's conditions are satisfied on a pairwise basis.That is $P_1||P_2||P_3$….||Pk if and only if Pi||Pj for all i≠j.

The parallelism relation || is **commutative** ie (P*i* || P*j* implies P*j* || P*i*), but **not transitive** (P*i* || P*j* and P*j* || P*k* does not guarantee P*i* || P*k*) . Therefore, || is not an equivalence relation. P*i* || P*j* || P*k* implies **associativity**.ie

(P*i* || P*j*)|| P*k* =P*i* || (P*j* || P*k*) . Since the order in which parallel executable processes are executed should not make any difference in the output sets.

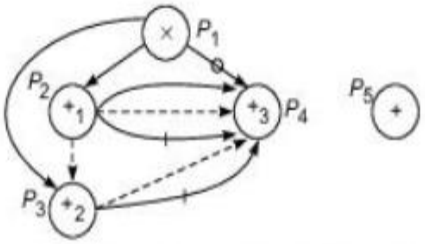
Example

Detection of parallelism in a program using Bernstein's conditions

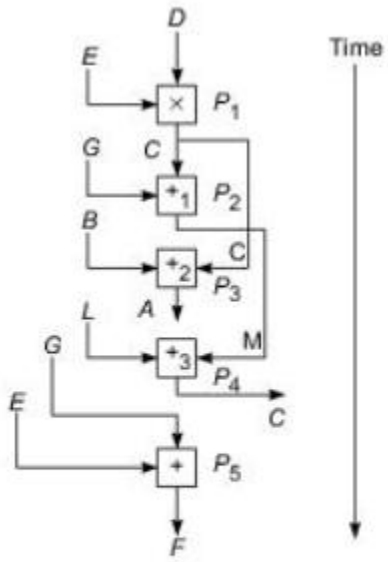
Consider the simple case in which each process is a single HLL statement. We want to detect the parallelism embedded in the following five statements labeled P I, P2, P3, P4, and P5, in program order.

$$\left.\begin{aligned} P_1 &: C = D \times E \\ P_2 &: M = G + C \\ P_3 &: A = B + C \\ P_4 &: C = L + M \\ P_5 &: F = G \div E \end{aligned}\right\}$$

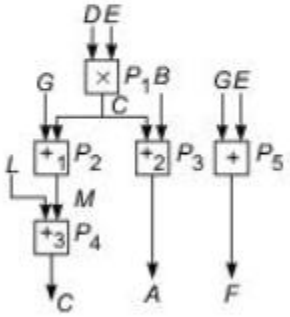
Assume that each statement requires one step to execute. No pipelining is considered here. The dependence graph shown in Fig. 2.2a demonstrates flow dependence as well as resource dependence. In sequential execution. five steps are needed (Flg. 2.2b).



(a) A dependence graph showing both data dependence (solid arrows) and resource dependence (dashed arrows)



(b) Sequential execution in five steps, assuming one step per statement (no pipelining)



(c) Parallel execution in three steps, assuming two adders are available per step

Fig. 2.2 Detection of parallelism in the program of Example 2.2

If two adders are available simultaneously, the parallel execution requires only three steps as shown in Fig. 2.2c. Pairwise, there are 10 pairs of statements to check against Bernstein's conditions. Only 5 pairs, P1,||P5, P2|| P3, P2| | P5, P5 ||P3, and P4||P5, can execute in parallel as revealed in Fig 2.2a if there are no resource conflicts. Collectively, only P2||P3||P5, is possible (Fig. 2.2c] because P2||P3, P3||P5, and P5||P2 are all possible.

Violations of any one or more of the three conditions in 2.1 prohibits parallelism between two processes. ln general, data dependence, control dependence, and resource dependence all prevent parallelism from being exploitable.

Hardware and Software Parallelism

Qn:Distinguish between hardware and software parallelism?

Hardware Parallelism	Software Parallelism
1. Its build into machines architecture and hardware multiplicity. Also known as machine parallelism	1. Its exploited by the concurrent execution of machine language instructions in a program
2. It's a function of cost and performance trade off	2. It's a function of algorithm, programming style and compiler optimization.
3. It displays resource utilization patterns of simultaneously executable operations. It also indicates the peak performance of processor resources	3. It displays patterns of simultaneously executable operations.
4. Its characterized by no: of instruction issues per machine cycle	4. The program flow graph displays the patterns of simultaneously executable operations 5. 2 types – • **Control parallelism** – allows 2 or more operations to be performed simultaneously. • **Data parallelism** – atmost same operation is performed over many data elements by many processors simultaneously.

One way to characterize the parallelism in a processor is by number of instruction issues per machine cycle. If a processor issues k instructions per machine cycle, then it is called a k-issue processor. A conventional pipelined processor takes one machine cycle to to issue a single instruction.These type of processors are called one-issue machines,with a single instruction pipeline in the processor.

Mismatch between software parallelism and hardware parallelism

Qn:Expalin the process of finding out the Mismatch between software parallelism and hardware parallelism

Consider the example program graph in Fig. 2.3a. There are eight instructions (four loads and four arithmetic operations) to be executed in three consecutive machine cycles. Four load operations are performed in the first cycle, followed by two multiply operations in the second cycle and two add/subtract operations in the third cycle. Therefore. the parallelism varies from 4 to 2 in three cycles. The average software parallelism is equal to 8/3= 2.67 instructions per cycle in this example program.

Now consider execution of the same program by a two-issue processor which can execute one memory access (load or write) and one arithmetic (add, subtract, multiply etc.) operation simultaneously. With this hardware restriction, the program must execute in seven machine cycles as shown in Fig. 1.3b.

Therefore. the hardware parallelism displays an average value of 8/7= 1.14 instmctions executed per cycle. This demonstrates a mismatch between the software parallelism and the hardware parallelism.

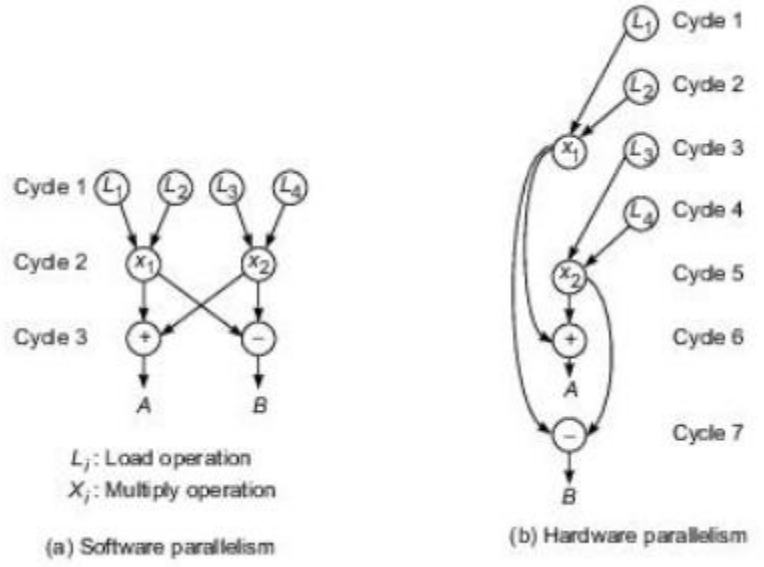


Fig 1.3 Executing an example program by a two-issue superscalar processor

Let us try to match the software parallelism shown in Fig. 2.3a in a hardware platform of a dual processor system, where single-issue processors are used. The achievable hardware parallelism is shown in Fig. 1.4.where L/S stands for load/store operations. Note that six processor cycles are needed to execute the I2 instructions by two processors. .S1 and S2 are two inserted store operations, and I5 and l6 are two inserted load operations. These added instructions are needed for interprocessor communication through the shared memory.

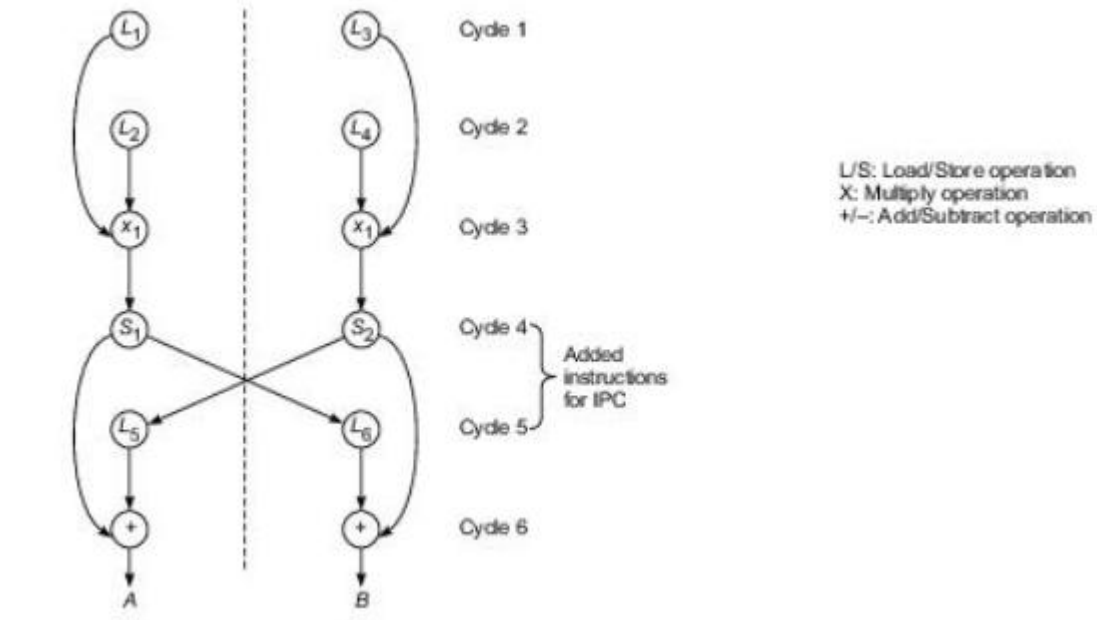


Fig 1.4 :Dual-processor execution of the program in fig 1.3 a

Of the many types of software parallelism, two are most frequently cited as important to parallel programming: The first is *control parallelism* which allows two or more operations to be performed simultaneously. The second type has been called *data parallelism*, in which almost the same operation is

performed over many data elements by many processors simultaneously.

Control parallelism, appearing in the form of pipelining or multiple functional units, is limited by the pipeline length and by the multiplicity of functional units. Both pipelining and functional parallelism are handled by the hardware; programmers need take no special actions to invoke them. Data parallelism offers the highest potential for concurrency. It is practiced in both SIMD and MIMD modes on MPP systems. Data parallel code is easier to write and to debug than control parallel code. Synchronization in SIMD data parallelism is handled by the hardware. Data parallelism exploits parallelism in proportion to the quantity of data involved.

To solve the mismatch problem between software parallelism and hardware parallelism, one approach is to develop compilation support, and the other is through hardware redesign for more efficient exploitation of parallelism. These two approaches must cooperate with each other to produce the best result.

ROLE OF COMPILERS - Hardware processors can be better designed to exploit parallelism by an **optimizing compiler.**That is compiler techniques are used to exploit hardware features to improve performance. Such processors use large register file and sustained instruction pipelining to execute nearly one instruction per cycle. The large register file supports fast access to temporary values generated by an optimizing compiler. The registers are exploited by code optimizer and global register allocator in such a compiler.

7.AMDAHL'S LAW FOR FIXED WORKLOAD

Qn:State amdahl's law and describe its significance:?

BASICS OF PERFORMANCE EVALUATION

A **sequential algorithm** is evaluated in terms of its execution time which is expressed as a function of its input size.

For a **parallel algorithm,** the execution time depends not only on input size but also on factors such as parallel architecture, no. of processors, etc.

Important Performance Metrics are:

- Parallel Run Time
- Speedup
- Efficiency

Parallel Runtime

The parallel run time **T(n)** of a program or application is the time required to run the program on an **n-processor** parallel computer.

When **n = 1**, **T(1)** denotes sequential runtime of the program on single processor.

Speedup

Speedup **S(n)** is defined as the ratio of time taken to run a program on a singleprocessor to the time taken to run the program on a parallel computer with identical processors.

$$S(n) = \frac{T(1)}{T(n)}$$

It measures how faster the program runs on a parallel computer rather than on a single processor.

Efficiency

The Efficiency **E(n)** of a program on **n processors is** defined as the ratio of speedup achieved and the number of processor used to achieve it.

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{n.T(n)}$$

Speedup Performance Laws

- **Amdahl's Law** *[based on fixed problem size or fixed work load]*
- **Gustafson's Law***[for scaled problems, where problem size increases with machine size i.e. the number of processors]*
- **Sun & Ni's Law** *[applied to scaled problems bounded by memory capacity]*

Amdahl's Law (1967)

Amdahl's law is used to find the maximum improvement possible by improving a particular part of a system. In parallel computing, Amdahl's law is mainly used to predict the theoretical maximum speedup for program processing using multiple processors

For a given problem size, the speedup does not increase linearly as the number of processors increases. In fact, the speedup tends to become saturated. This is a consequence of Amdahl's Law.

According to Amdahl's Law, a program contains two types of operations:

- ***Completely sequential***
- ***Completely parallel***

Let, the time **Ts** taken to perform sequential operations be a fraction **α** ($0<\alpha\leq1$) of the total execution time **T(1)** of the program, then the time **Tp to perform parallel operations shall be (1-α) of T(1).**

Thus, **Ts = α.T(1)** and **Tp = (1-α).T(1)**

Assuming that the parallel operations achieve linear speedup (i.e. these operations use 1/n of the time taken to perform on each processor), then

$$T(n) = Ts + Tp/n = \alpha . T(1) + \frac{(1-\alpha).T(1)}{n}$$

Thus, the speedup with **n** processors will be:

$$S(n) = \frac{T(1)}{T(n)}$$

$$= \frac{1}{\alpha + \frac{(1-\alpha)}{n}}$$

$$S_n = \frac{n}{1+(n-1)\alpha}$$

EQ: 1.1

Sequential operations will tend to dominate the speedup as **n** becomes very large.

$$\text{As } n \rightarrow \infty, S(n) \rightarrow 1/\alpha$$

(Division of any number by infinity=0)

This means, no matter how many processors are employed, the speedup in this problem is limited to 1/*α*. This is known as **sequential bottleneck** of the problem.

Note*: Sequential bottleneck* **cannot** *be removed just by increasing the no. of processors***.**

Example:

- Suppose that a calculation has a 4% serial portion, what is the limit of speedup on 16 processors?

$$S_n = \frac{n}{1+(n-1)\alpha}$$

16/(1 + (16 – 1)*.04) = 10

What is the maximum speedup?(ie **1/α)**

1/0.04 = 25

- *If 90% of a calculation can be parallelized (i.e. 10% is sequential) then the maximum speed-up which can be achieved on 5 processors is 1/(0.1+(1-0.1)/5) or roughly 3.6 (i.e. the program can theoratically run 3.6 times faster on five processors than on one)*

$$S(n) = \frac{1}{\alpha + \frac{(1-\alpha)}{n}}$$

=1/(0.1+(1-0.1)/5)
= 3.6

(i.e. the program can theoretically run 3.6 times faster on five processors than on one)

Amdahl's law for fixed workload

Qn:Describe amdahl's law for fixedworkload?

Qn:Describe the term asymptotic speedup?

Δ =Computing capacity of a single processor

***W* =Total amount of work(instructions or computations)**

***DOP*=Degree of parallelism(The number of processors used to execute a program).**

***n* = Machine size**

***w* =Workload**

***m* =Maximum parallelism in a profile**

Asymptotic Speedup:

Asympnatic Speedup Denote the amount of work executed with DOP = i as $W_i = i\Delta t_i$ or we can write $W = \sum_{i=1}^{m} W_i$.

The execution time of W_i on a single processor [(sequentially) is $t_i(1) = W_i/\Delta$. ..

The execution time of W_i on K processors is $t_i(k) = W_i/k\Delta$. .

The execution time with infinite number of processors is $t_i(\infty) = W_i/i\Delta$

Thus we can write the **response time** as:

$$T(1) = \sum_{i=1}^{m} t_i\ (1) = \sum_{i=1}^{m} \frac{W_i}{\Delta}$$

$$T(\infty) = \sum_{i=1}^{m} t_i\ (\infty) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}$$

The *asymptotic speedup* S_∞ is defined as the ratio of $T(1)$ to $T(\infty)$:

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} W_i/i}$$

EQ-1.2

Fixed Load Speedup

The speed up formulae given in EQ-1.2 is based on fixed workload, regardless of machine size. Speed up equation given below do give consideration to machine size also.

As the number of processors increases in a parallel computer, the fixed load is distributed to more processors for parallel exccution. Therefore the main objective is to produce the results as soon as possible. In other words, minimal turnaround time is the primary goal. Speedup obtained for time critical applications is called **fixed—load speedup.**

We consider below both the cases of DOP $< n$ and of DOP $\geq n$. We use the ceiling function $\lceil x \rceil$ to represent the smallest integer that is greater than or equal to the positive real number x. When x is a fraction, $\lceil x \rceil$ equals 1. Consider the case where DOP $= i > n$. Assume all n processors are used to execute W_i exclusively. The execution time of W_i is

$$t_i(n) = \frac{W_i}{i\Delta}\left\lceil \frac{i}{n} \right\rceil$$

Thus the response time is

$$T(n) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}\left\lceil \frac{i}{n} \right\rceil$$

Note that if $i < n$, then $t_i(n) = t_i(\infty) = W_i/i\Delta$. Now, we define the *fixed-load speedup factor* as the ratio of $T(1)$ to $T(n)$:

$$S_n = \frac{T(1)}{T(n)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i}\left\lceil \frac{i}{n} \right\rceil}$$

EQ-1.3

Amdahl's Law Revisited

Gene Amdahl derived a fixed-load speedup For the special case where the computer operates either in sequential mode (with DOP = 1) or in perfectly parallel mode (with DOP =n). That is, Equation 1.3 is then simplified to,

$$S_n = \frac{W_1 + W_n}{W_1 + W_n/n}$$

Amdahl's law implies that the sequential portion of the program w_i does not change with respect to the machine size n. However, the parallel portion is evenly executed by n processors, resulting in a reduced time.

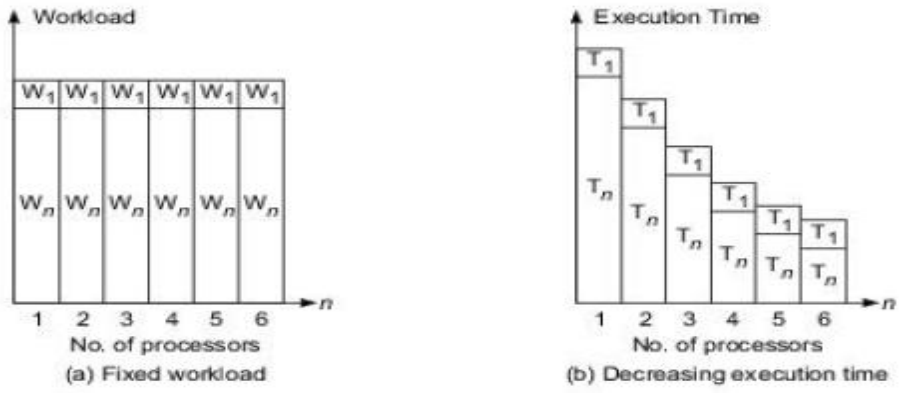


Fig:Fixed load speedup model and amdahl's law

As shown above, when the number of processors increases, the load on each processor decreases. However. the total amount of work (workload) $w_1 + w_n$ is kept constant as shown in Fig. a.

In Fig. b, the total execution time decreases because $T_n = w_n/n$. Eventually, the sequential part will dominate the performance because $T_n=0$ as n becomes very large and T_1 is kept unchanged.

More Solved Problems

1 .A workstation uses a 15 MHz processor with a claimed 1000 MIPS rating to execute a given program mix. Assume one cycle delay for each memory access.

a. What is the effective CPI of this computer?

b. Suppose the processor is upgraded with a 3.0 MHz clock. However, the speed of the memory subsystem remains unchanged, and consequently two clock cycles are needed per memory access. If 30% of the instructions require one memory access and another 5% require two memory accesses per instruction, what is the performance (new MIPS rating) of the upgraded processor with a compatible instruction set and equal instruction counts in the given program mix*?(Kerala university QP)*

Answer:

a)MIPS=f/(CPI*10^6)

CPI=f/(MIPS*10^6)

=(1.5*$10^{9)}$ /(1000*10^6)

=1.5

b)30% instructions take one memory(hence 2 clock cycles as two clock cycles are needed per memory access as per qn).Similarly 5% instructions take 2 memory accesses(ie..4 cycles).remaining 65% takes 1 clock cycle.

Total number of cycles=(30*2)+(5*4)+(65*1)= 145 cycles

CPI=total no of clock cycles/total no of instructions

=145/100

=1.45

MIPS=f/(CPI*10^6)

=(3*$10^{9)}$ /(1.45*10^6)

=2068.9

2

Draw a dependence graph to show all the dependences among the following statements.

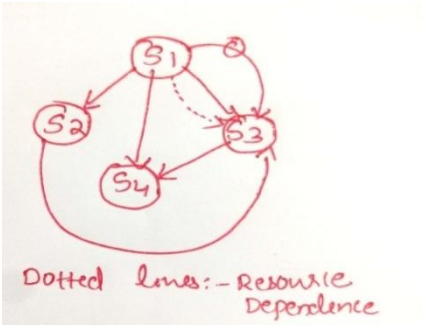
$S_1 : A = B + D$

$S_2 : C = A \times 3$

$S_3 : A = A + C$

$S_4 : E = A/2$

Answer:



3

Problem 2.5 Analyze the data dependences among the following statements in a given program:

S1:	Load R1, 1024	/R1 ← 1024/
S2:	Load R2, M(10)	/R2 ← Memory(10)/
S3:	Add R1, R2	/R1 ← (R1) + (R2)/
S4:	Store M(1024), R1	/Memory(1024) ← (R1)/
S5:	Store M((R2)), 1024	/Memory(64) ← 1024/

where (Ri) means the content of register Ri and Memory(10) contains 64 initially.

(a) Draw a dependence graph to show all the dependences.

(b) Are there any resource dependences if only one copy of each functional unit is available in the CPU?

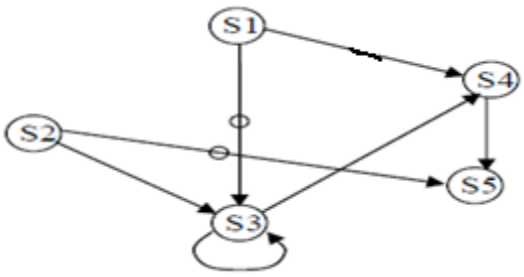
(c) Repeat the above for the following program statements:

S1:	Load R1, M(100)	/R1 ← Memory(100)/
S2:	Move R2, R1	/R2 ← (R1)/
S3:	Inc R1	/R1 ← (R1) + 1/
S4:	Add R2, R1	/R2 ← (R2) + (R1)/
S5:	Store M(100), R1	/Memory(100) ← (R1)/

Answers:

(a) Dependence graph:

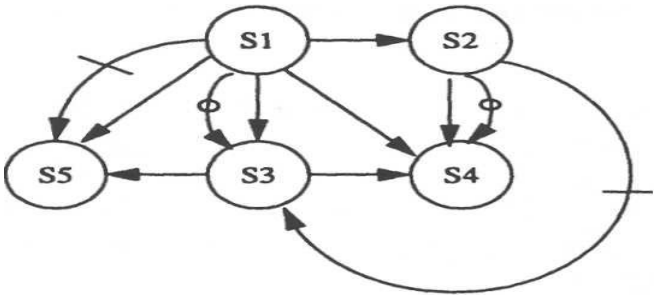
a)



(b) There are storage dependences between instruction pairs (S2, S5) and (S4, S5).

There is a resource dependence between Sl and S2 on the load unit, and another between S4 and S5 on the store unit.

(c) There is an ALU dependence between S3 and S4, and a storage dependence between S1 and S5.



4.

Problem 2.6 A sequential program consists of the following five statements, S1 through S5. Considering each statement as a separate process, clearly identify *input* set I_i and *output* set O_i of each process. Restructure the program using Bernstein's conditions in order to achieve maximum parallelism between processes. If any pair of processes cannot be executed concurrently, specify which of the three conditions is not satisfied.

```
S1:   A = B + C
S2:   C = B × D
S3:   S = 0
S4:   Do I = A, 100
          S = S + X(I)
      End Do
S5:   IF (S .GT. 1000) C = C × 2
```

Answer:

The input and output sets for the instructions are enumerated below:

$I1 = \{B, C\}$, $O_1 = \{A\}$,

$I_2 = \{B,D\}$, $O_2 = \{C\}$,

$I3 = \varnothing$, $O_3 = \{S\}$,

$I4 = \{S,A,X(I)\}$, $O_4 = \{S\}$,

$I5 = \{S,C\}$, $O5 = \{C\}$.

Using Bernstein's conditions, we find that

S1 and S3 can be executed concurrently, because $I1 \cap O3 = \varnothing$, $I3 \cap O1 = \varnothing$, and $O1 \cap O3 = \varnothing$.

S2 and S3 can be executed concurrently, because $I2 \cap O3 = \varnothing$, $I3 \cap O2 = \varnothing$, and $O2 \cap O3 = \varnothing$ ..

S2 and S4 can be executed concurrently, because $I2 \cap O4 = \varnothing$, $I4 \cap O2 = \varnothing$, and $O_2 \cap O_4 = \varnothing$.

S1 and S5 cannot be executed concurrently, because $I1 \cap O5 = \{C\}$.

S1 and S2 cannot be executed concurrently, because $I1 \cap O_2 = \{C\}$.

S1 and S4 cannot be executed concurrently, because $I4 \cap O_1 = \{A\}$.

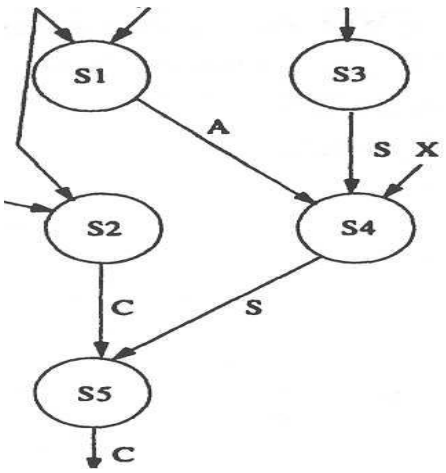
S2 and S5 cannot be executed concurrently, because $I5 \cap O_2 = O5 \cap O_2 = \{C\}$.

S3 and S4 cannot be executed concurrently, because $I4 \cap O_3 = O_4 \cap O_3 = \{S\}$.

S3 and S5 cannot be executed concurrently, because $I5 \cap O_3 = \{S\}$.

S4 and S5 cannot be executed concurrently, because $I5 \cap I4 = I5 \cap 0_4 = \{S\}$.

The program can be reconstructed as shown in the flow graph below:



5

Problem 2.4 Perform a data dependence analysis on each of the following Fortran program fragments. Show the dependence graphs among the statements with justification.

(a)
S1: A = B + D
S2: C = A × 3
S3: A = A + C
S4: E = A / 2

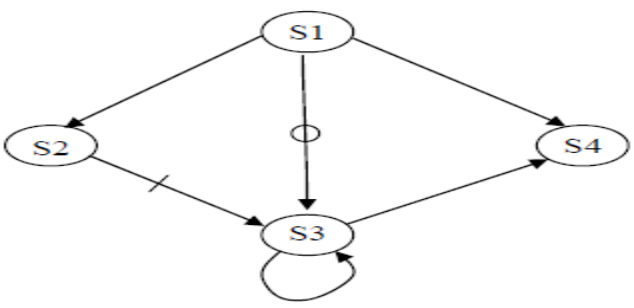
(b)
S1: X = SIN(Y)
S2: Z = X + W
S3: Y = −2.5 × W
S4: X = COS(Z)

(c) Determine the data dependences in the same and adjacent iterations of the following Do-loop.

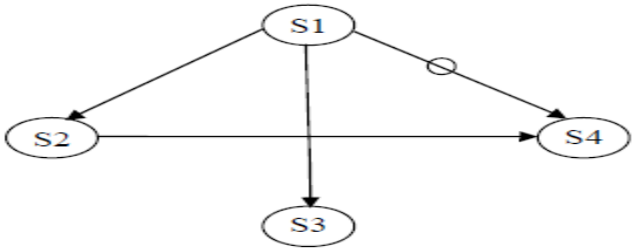
```
      Do 10 I = 1, N
S1:       A(I + 1) = B(I − 1) + C(I)
S2:       B(I) = A(I) × K
S3:       C(I) = B(I) − 1
   10 Continue
```

Answer:

a)



b)



c)

6:Define pipeline throughput and efficiency?

Throughput Rate:- Number of programs executed per unit time is called system throughput w_s(in programs per second).In a multiprogrammed system, the system throughput is often lower than CPU throughput Wp defined by

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

W = 1 / T
OR
W =(MIPS*10^6)/Ic

Efficiency

The Efficiency **E(n)** of a program on **n processors is** defined as the ratio of speedup achieved and the number of processor used to achieve it.

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{n.T(n)}$$