CS405 COMPUTER SYSTEM ARCHITECTURE

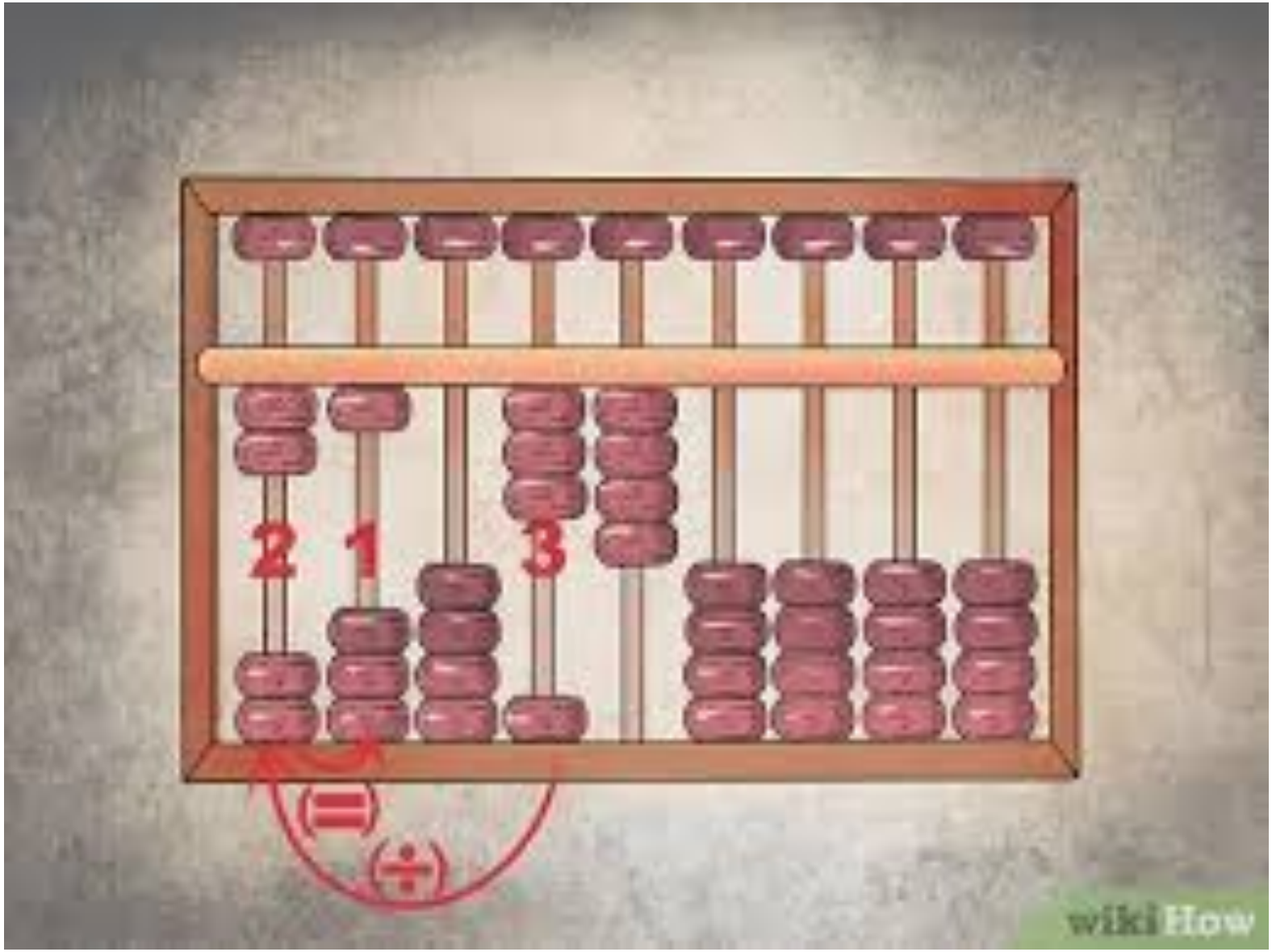
By Elizabeth Isaac
Department of Computer Science and engineering
M A College of Engineering

K. Hwang and Naresh Jotwani, Advanced Computer Architecture, Parallelism, Scalability, Programmability, TMH, 2010

Module 1

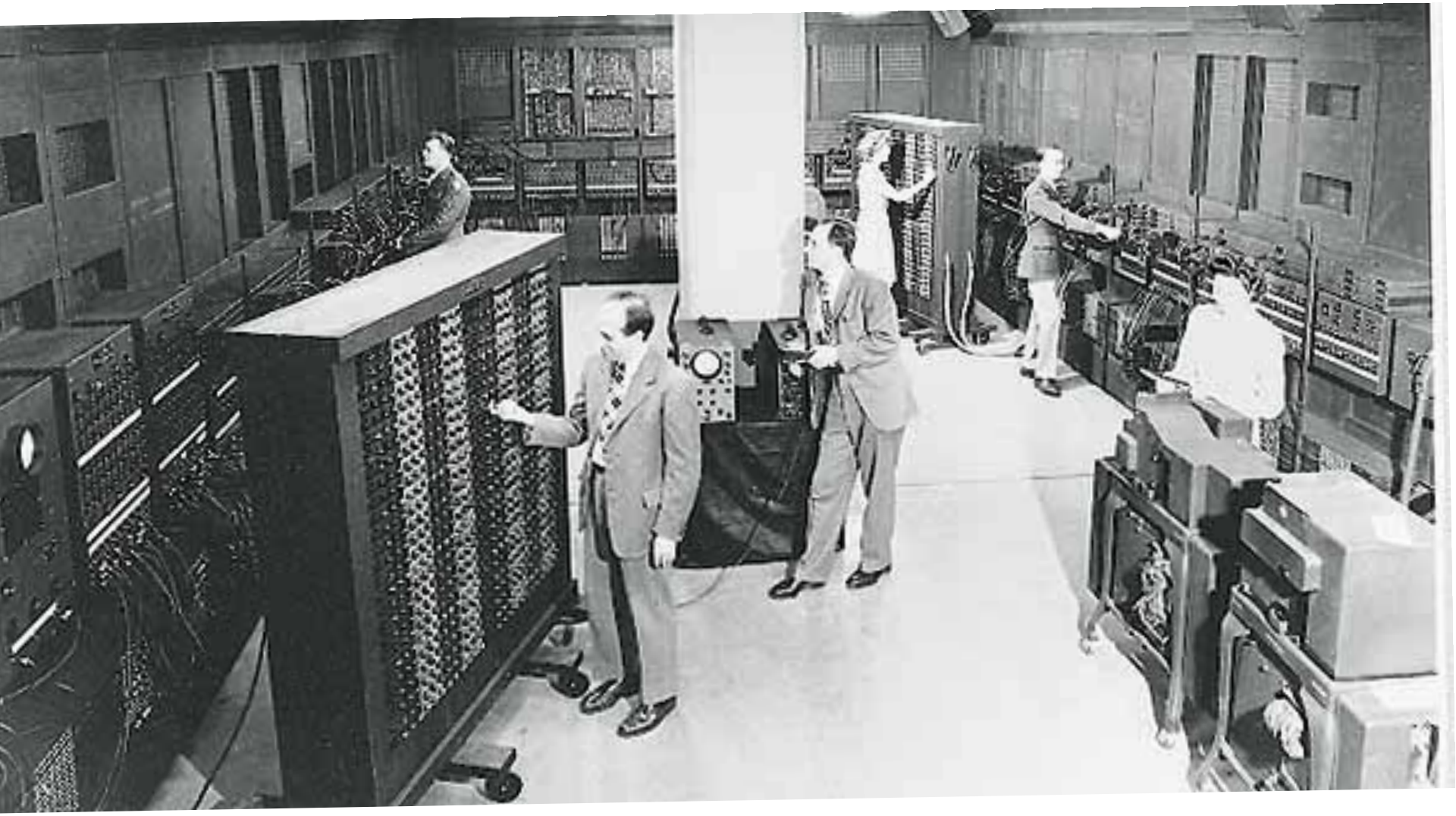
- **Parallel computer models**
  - Evolution of Computer Architecture
  - System Attributes to performance
  - Amdahl‘s law for a fixed workload
  - Multiprocessors and Multicomputers
  - Multivector and SIMD computers
  - Architectural development tracks
  - Conditions of parallelism

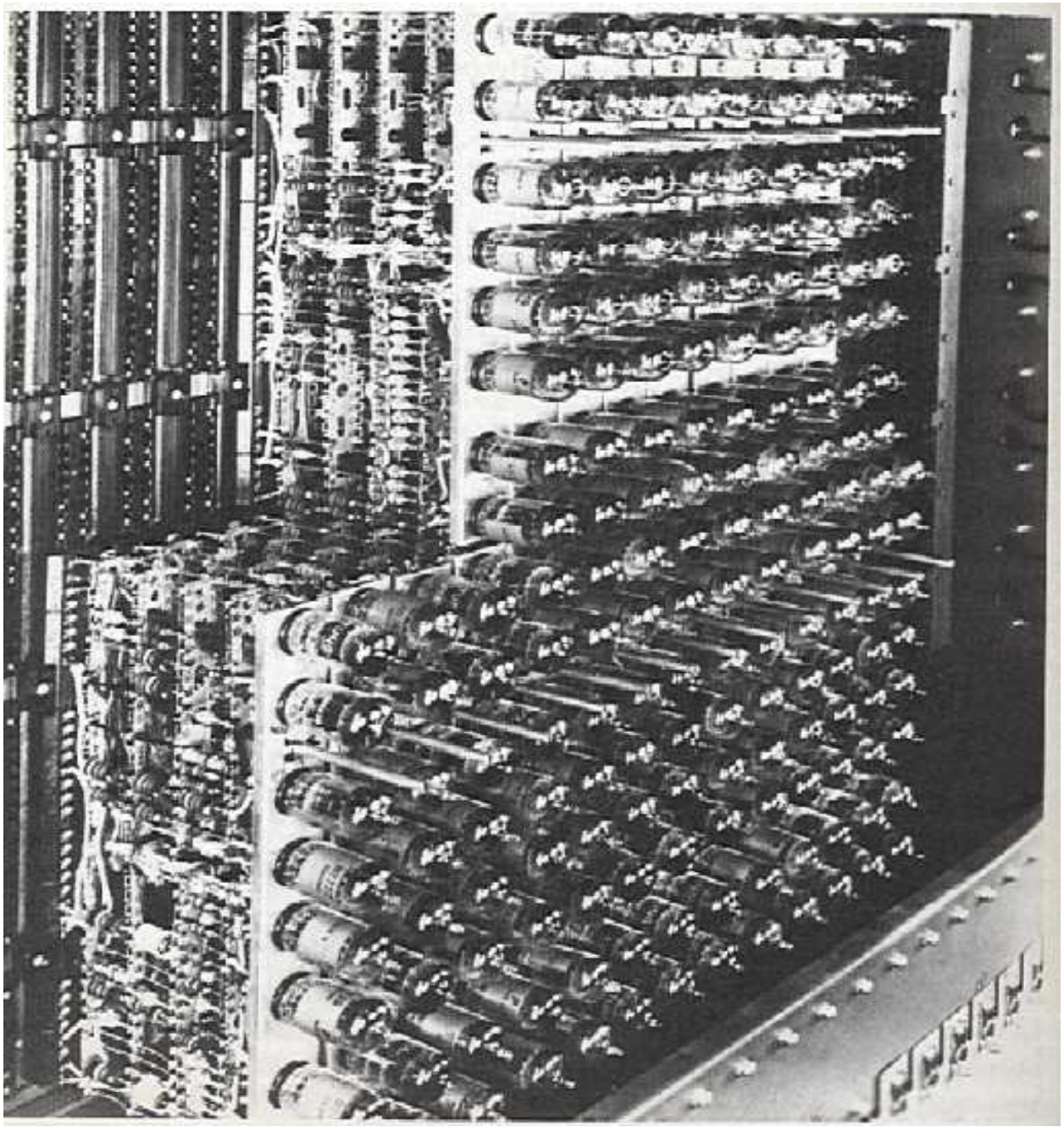
ABACUS



ROTATING WHEEL CALCULATOR

DIFFERENCE ENGINE





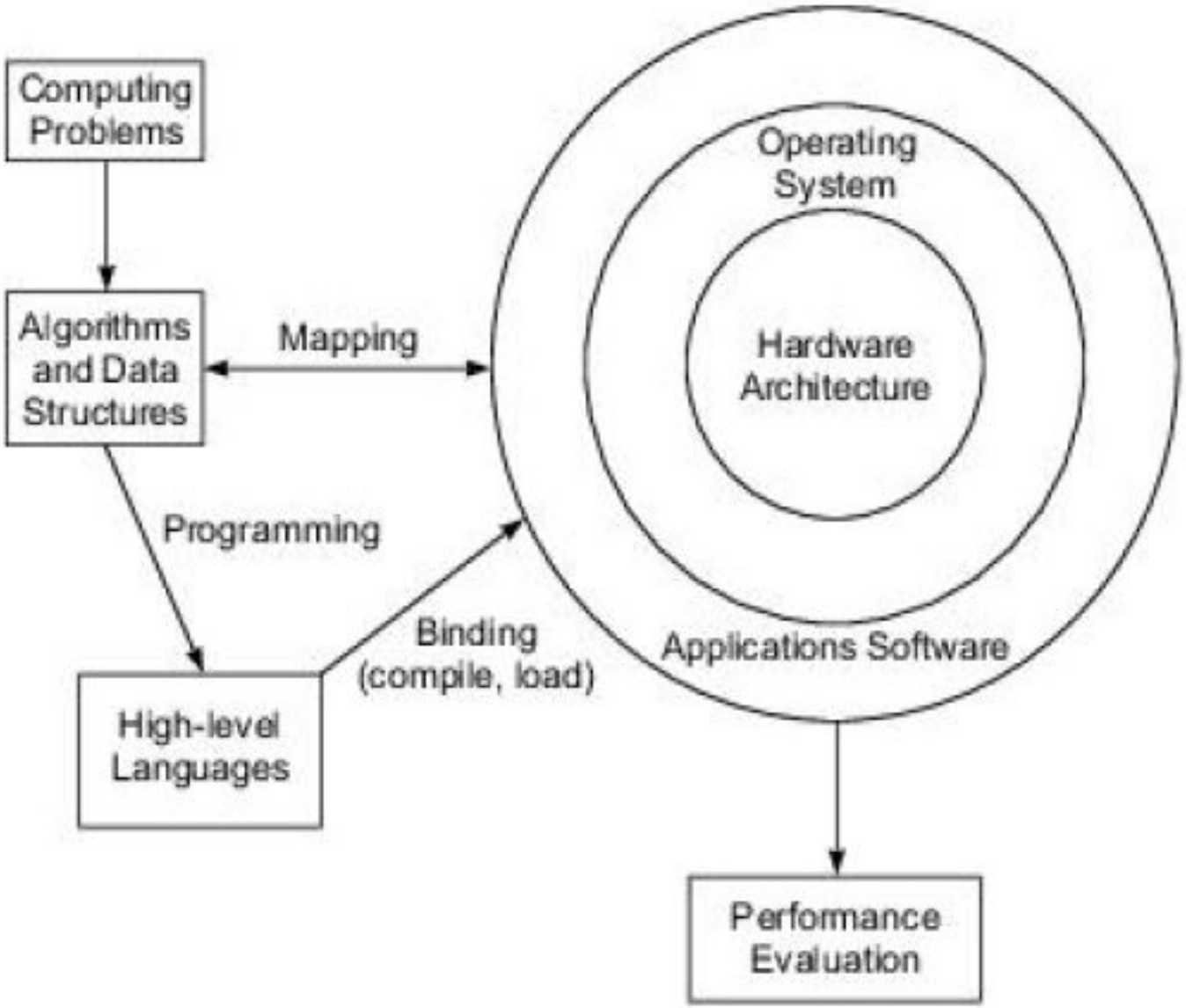
COMING UP TECHNOLOGY

Parallel computer models

Computer Generations : Electronic computers have gone through five generations of development.

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.	See Tables 1.3–1.6 and Chapter 13.

- **Elements of Modern Computers**



- **Computing Problem**
  - computer architecture concept is no longer restricted to the bare machine hardware structure
  - Consist of machine hardware, an instruction set, system software, application programs, and user interfaces
  - Cost effective solutions
  - Example:

(1) Alpha numerical problems in business and government (solutions demand efficient transaction processing, large database management and information retrieval operations)

(2) Artificial intelligence [Al] problems (solution demand logic inferences and symbolic manipulations)

- **Algorithm and Data Structures**
  - Special algorithms (deterministic and non-deterministic) and data structures
  - Problem formulation and the development of parallel algorithms
- **Hardware Resources**
  - Three nested circles in figure
  - Special hardware interfaces
  - Software interface programs

- **Operating Systems**
  - Allocation and Deallocation
  - Performance Evaluation
  - Mapping and Optimal mappings

- **System Software Support**
  - Efficient programs in high-level languages
  - **HLL** → Compiler → **Object Code**

- **Compiler Support**

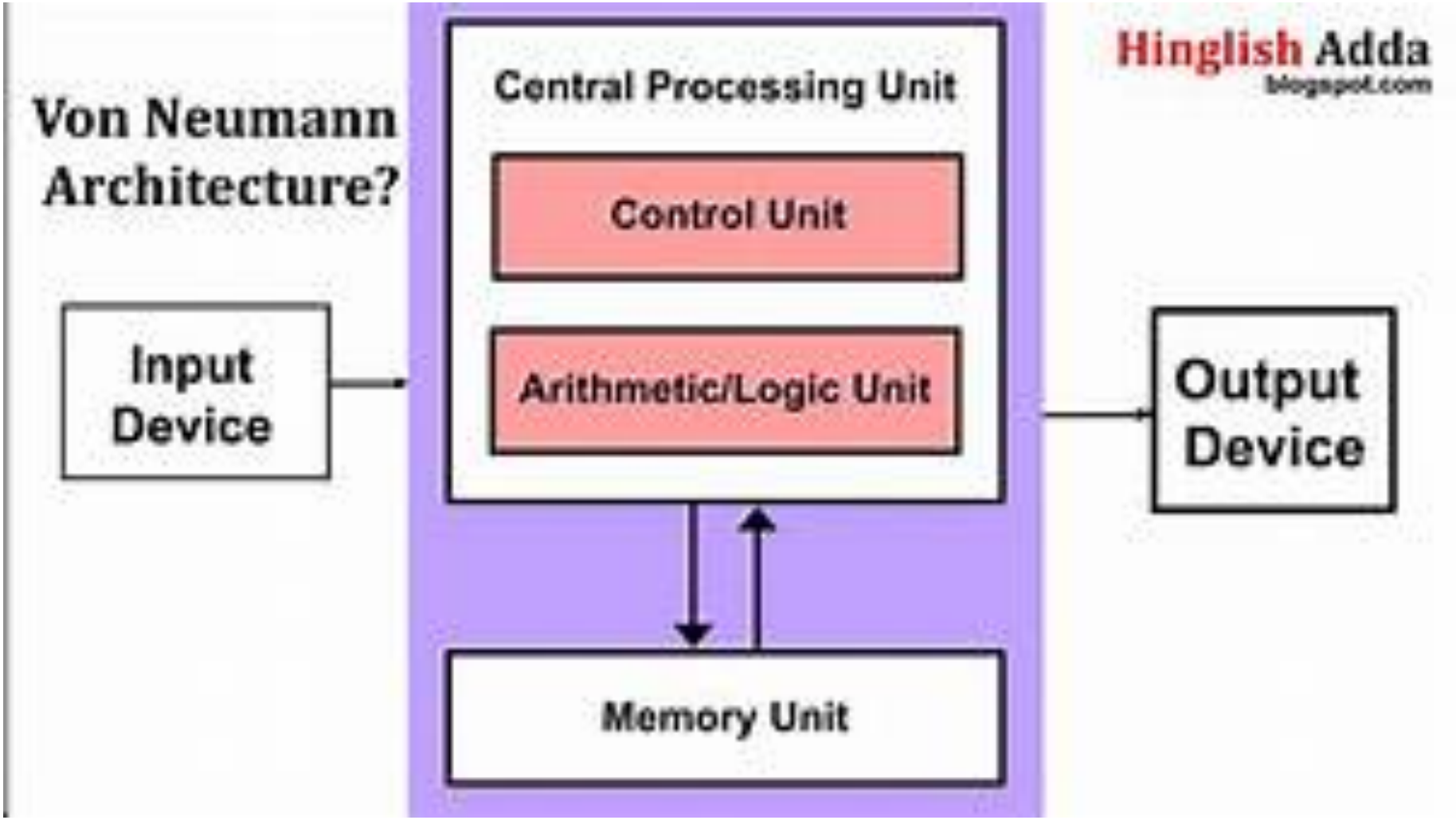
3 compiler upgrades:

- **Preproecssor, Precompiler and Parallelizing Compiler**
- **Preproecssor** : Uses a sequential compiler and a low-level library of the target computer
- **Precompiler :** Requires Program flow analysis, dependence checking and limited optimizations
- **Parallelizing Compiler :** Automatically detect parallelism in source code

Module 1

- **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

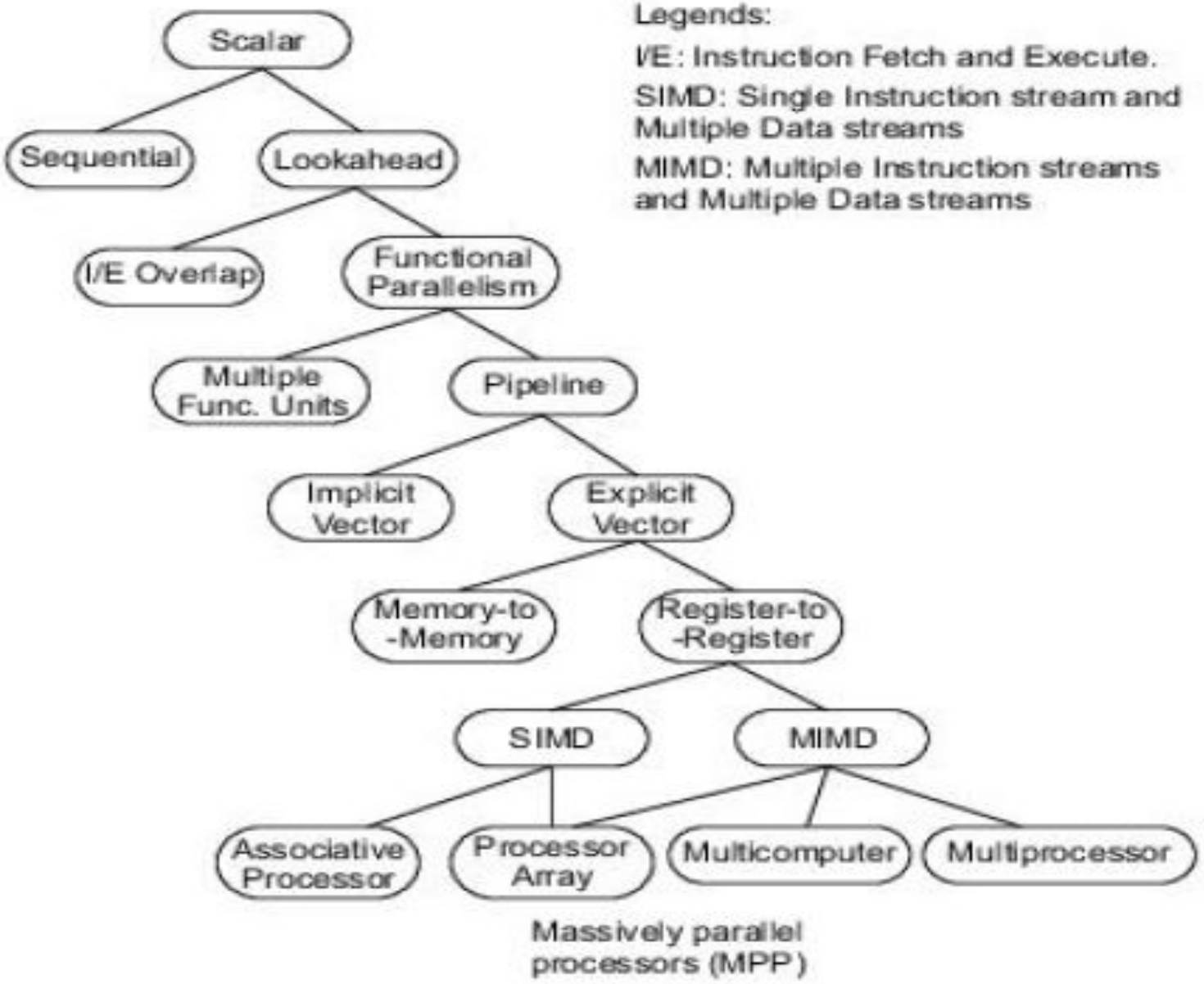
Von Neumann Architecture



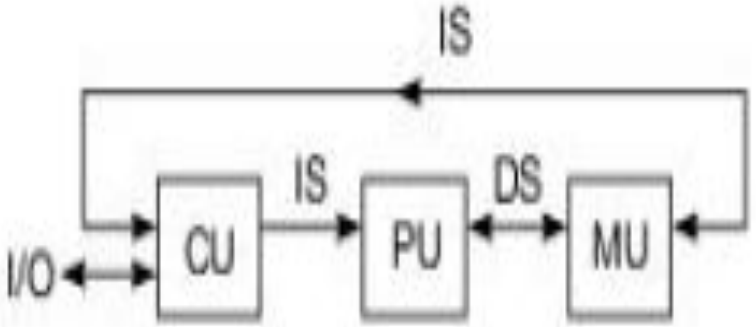
Evolution of Computer Architecture

- Hardware organization and programming/software requirements
- Abstracted by instruction set
- Architectural evolution from sequential scalar computers to vector processors and parallel computers

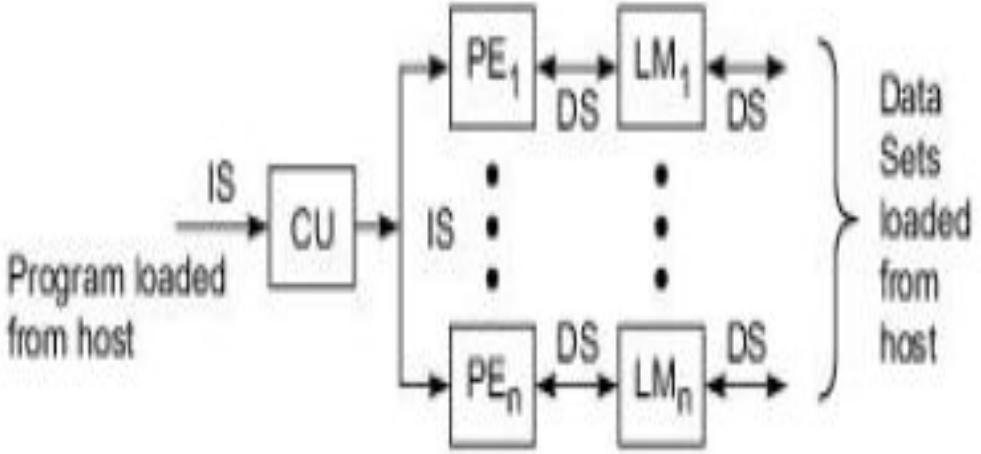
- **Architectural Evolution**



- **Flynn's Classification**
  - Michael Flynn (1972)
  - Based on notations of instructions and data streams
    - SISD (Single instruction streams over single data streams)
    - SIMD (Single instruction streams over multiple data streams)
    - MISD (Multiple instruction streams over single data streams)
    - MIMD (Multiple instruction streams over multiple data streams)



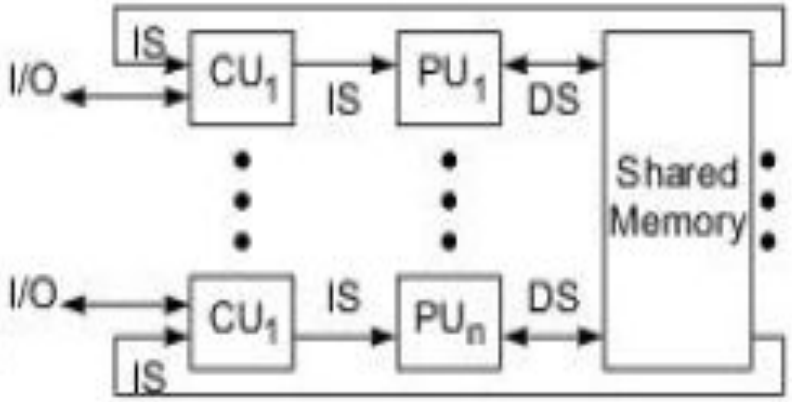
(a) SISD uniprocessor architecture



(b) SIMD architecture (with distributed memory)

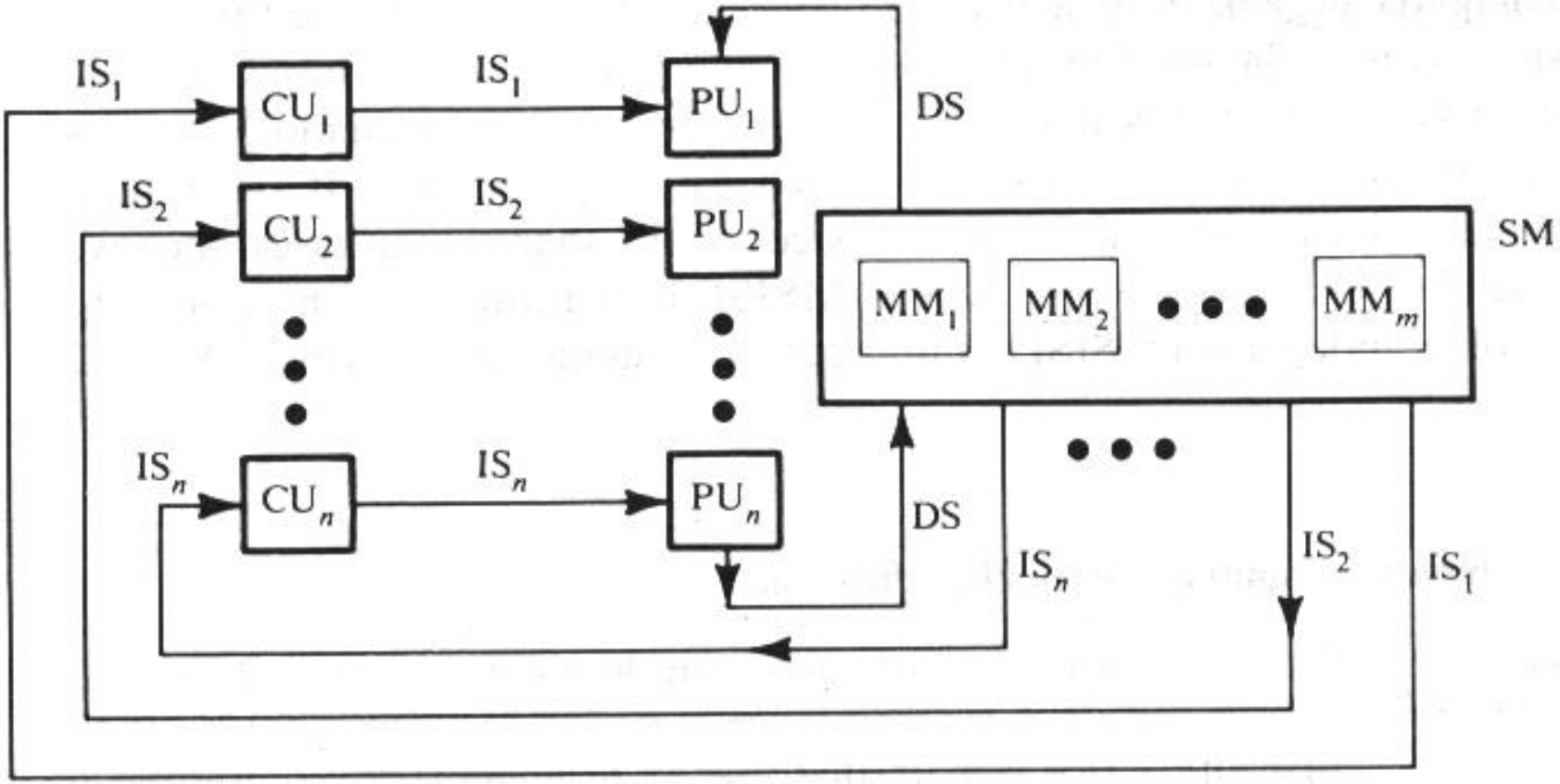
Captions:

CU = Control Unit
PU = Processing Unit
MU = Memory Unit
IS = Instruction Stream
DS = Data Stream
PE = Processing Element
LM = Local Memory

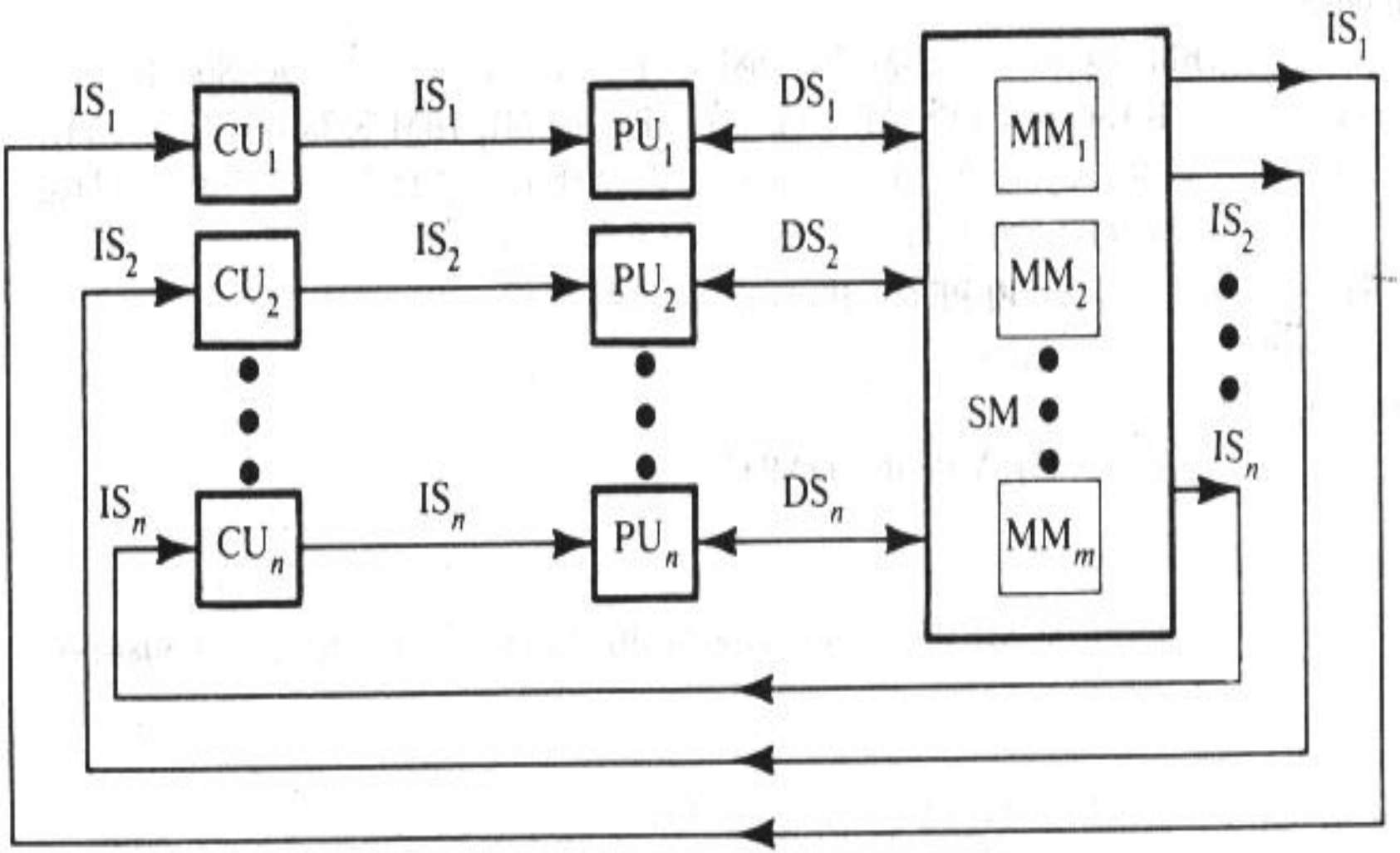


(c) MIMD architecture (with shared memory)

MISD

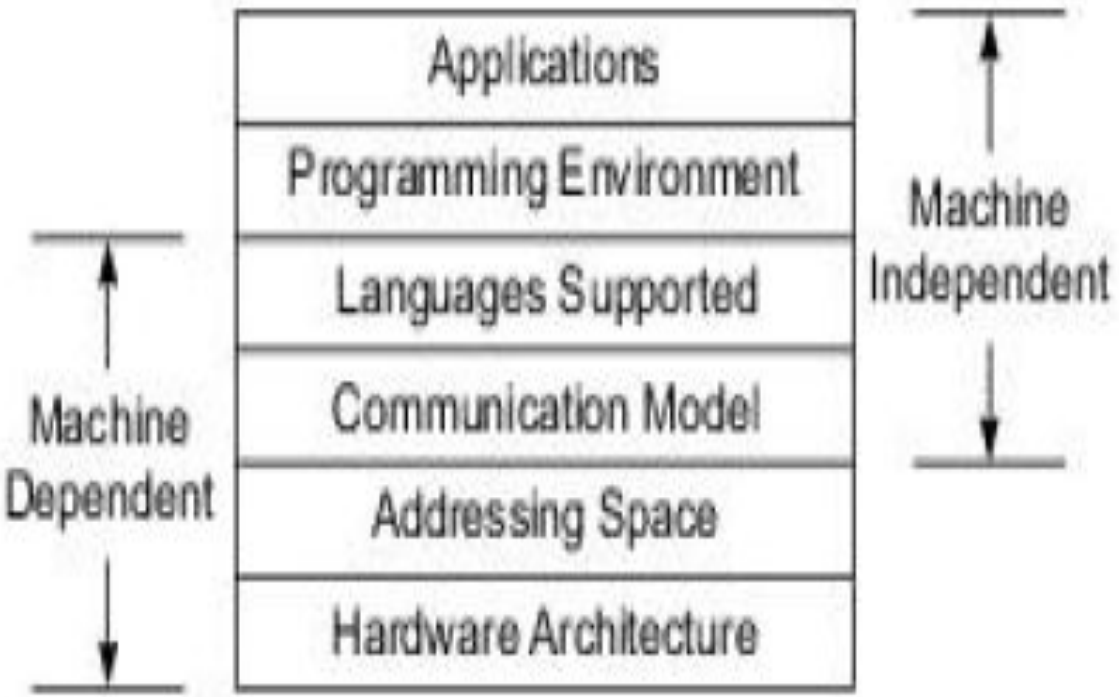

IS1
CU1
IS1
PU1
DS
IS2
CU2
IS2
PU2
SM
MM1
MM2
MMm
ISn
CUn
ISn
PUn
DS
ISn
IS2
IS1

MIMD


IS1
CU1
IS1
PU1
DS1
MM1
IS1
IS2
CU2
IS2
PU2
DS2
MM2
IS2
SM
ISn
ISn
CUn
ISn
PUn
DSn
MMm

- Parallel/ vector Computer:
  - Programs in MIMD mode
  - Two classes
    - Shared-Memory Multiprocessors
    - Message- passing multicomputers

- Six layers for computer system development



Module 1

- **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

System Attributes to Performance

- Machine capability and program behavior
- **Clock Rate and CPI:**
  - Processor is driven by a clock with a constant cycle time τ. The inverse of the cycle time is the clock rate (f= 1/ τ).
  - Size by program Instruction Count(Ic)
  - Cycles per instruction (CPI)

System Attributes to Performance

- **Performance Factors**

  I_c : number of instructions(Instruction count)

CPU Time, $T = I_c * \text{CPI} * \tau$ (T in Seconds/Program)

- we can rewrite as $T = I_c * (p + m * k) * \tau$

p : Number of processor cycles for ID and EX
m : Number of memory references needed
K : Ratio between memory cycle and processor cycle

System Attributes to Performance

- **System Attribute**: five performance factors (I_c, p, m, k, τ) are influenced by four system attributes:
  - Instruction-set architecture
  - Compiler technology
  - CPU implementation and control
  - Cache and memory hierarchy

- **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				✓	✓

System Attributes to Performance

- MIPS Rate

$$\text{MIPS rate} = \frac{Ic}{T * 10^6}$$

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

$$\text{MIPS rate} = \frac{Ic * f}{Ic * CPI * 10^6}$$

$$\text{MIPS rate} = \frac{f}{CPI * 10^6}$$

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

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

THROUGHPUT rate

$$W_p = \frac{f}{I_c * CPI} = \frac{MIPS * 10^6}{I_c}$$

- **Problem:**
  - Machine A runs a program in 20 seconds. Machine B runs the same program in 25 seconds. How many times faster is machine A?

$$n = \text{ExecTime(B)} / \text{ExecTime(A)}$$

$$= \frac{25 \text{ sec}}{20 \text{ sec}} = 1.25$$

Machine A is 1.25 times faster than Machine B

Numerical:- 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 for the above program
average CPI,MIPS rate & execution time

$$\text{Average CPI} = \frac{C}{I_c}$$

C = Total # cycles to execute a whole program

I_c = Total Instructions

$$= \frac{45000 \times 1 + 32000 \times 2 + 1500 \times 2 + 8000 \times 2}{45000 + 3200 + 15000 + 8000}$$

$$= \frac{155000}{100000}$$

CPI = 1.55

Execution Time = C / f

$$T = 155000 / 40 \times 10^6$$

$$T = 0.155 / 40$$

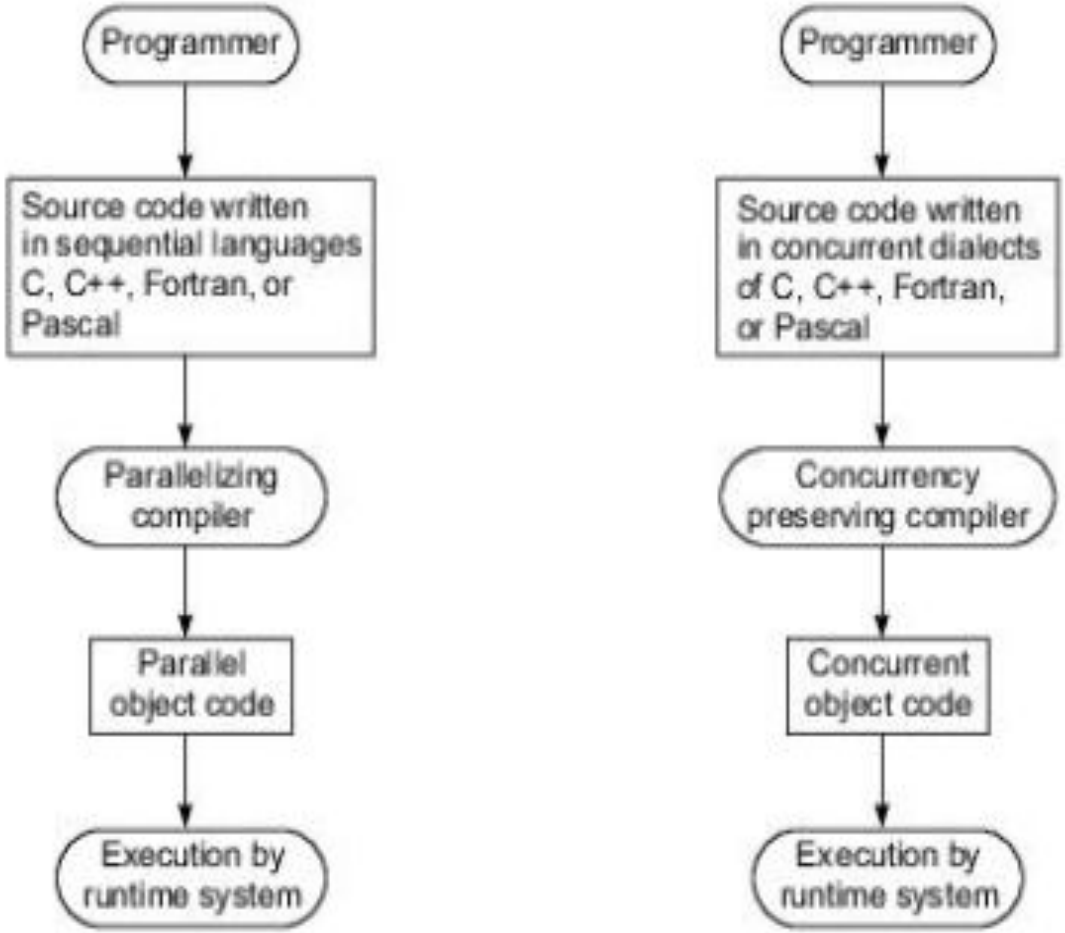
T = 3.875 ms

$$\text{MIPS rate} = I_c / T \times 10^6$$

$$= 100000 / 40 \times 10^6$$

MIPS rate = 25.81

- **Implicit parallelism** : a conventional language, such as C, C++, fortran, or pascal to write the source program
- **Explicit parallelism :** requires more effort by the programmer to develop a source program using parallel dialects of C, C++, fortran, or pascal
  - Parallelism is explicitly specified in the user programs
  - The compiler needs to preserve parallelism and, where possible, assigns target machine resources. New programming language chapel



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)

Module 1

- **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

Amdahl's Law for a Fixed Workload

- Number of processors increases in a parallel computer, the fixed load is distributed to more processors for parallel execution
- Minimal turnaround time is the primary goal.
- Speedup obtained for time-critical applications is called fixed—load speedup

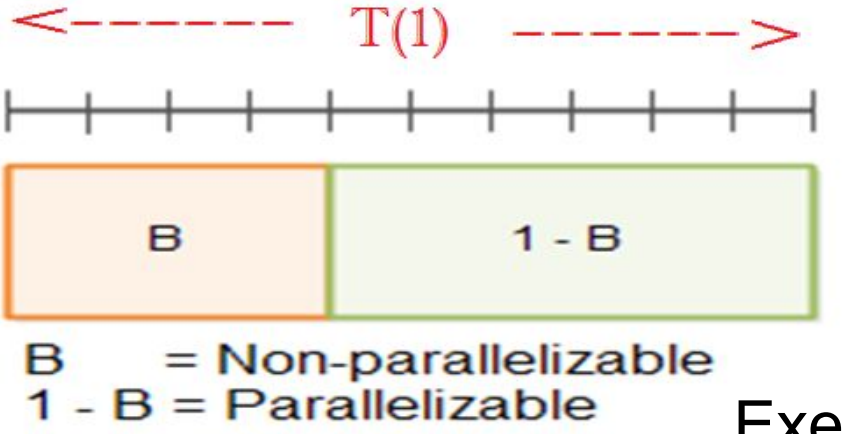
Amdahl's Law

- A program (or algorithm) which can be parallelized can be split up into two parts:

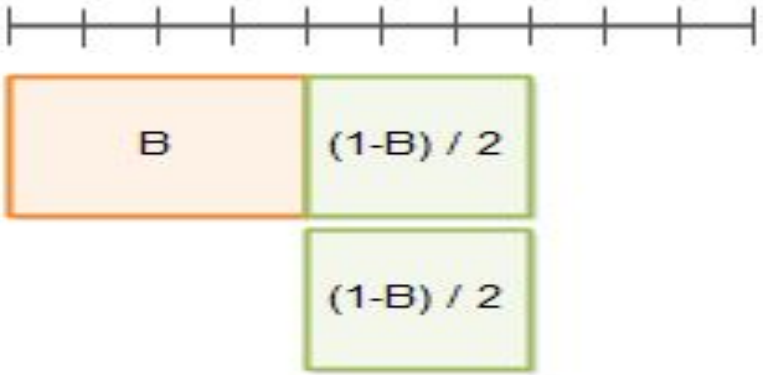
 A part which cannot be parallelized and

 A part which can be parallelized

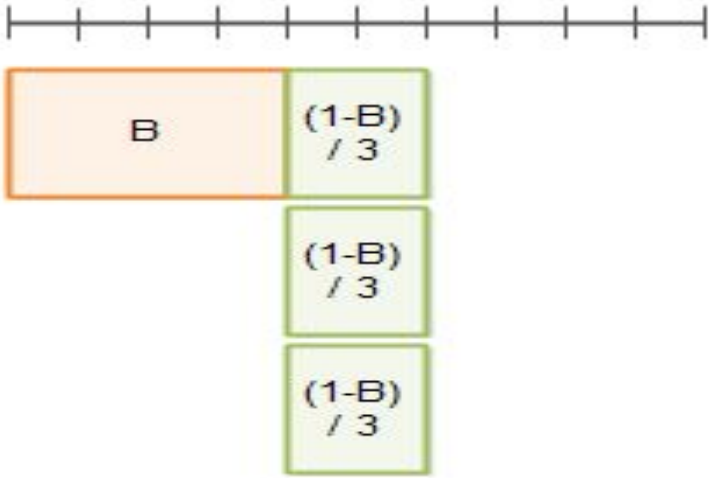
First of all, a program can be broken up into a non-parallelizable part B, and a parallelizable part 1-B, as illustrated by this diagram: total execution time T(1).



Execution time with a parallelization factor of 2:



Execution time with a parallelization factor of 3:



Amdahl's law

Amdahl's Law (1967)

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 ($0<\alpha\leq 1$) 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)**

Amdahl's Law

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

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

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

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

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

- **Example: 1**
- Suppose that a calculation has a 4% serial portion,
- a) What is the limit of speedup on 16 processors?
- b) What is the maximum speedup?

Ans:

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

a) Limit of speedup on 16 processors

=16/(1 + (16 – 1)*.04) = 10

b) The maximum speedup = **1/α**

= 1/0.04 = 25

Amdahl's Law

- The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's law.
- Amdahl's law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.
- Law defines the term 'speedup'.

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

Alternatively,

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

Amdahl's Law

- The execution time using the original computer with the enhanced mode will be the time spent using the unenhanced portion of the computer

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

The overall speedup is the ratio of the execution times:

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

3. We are considering an enhancement to the processor of a web server. The new CPU is 20 time faster on search queries than the old processor. The old processor is busy with search queries 70% of the time, what is the speedup gained by integrating the enhanced CPU?

$$Speedup = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$Fraction_{enhanced} = 70\ \% = 0.70$

$Speedup_{enhanced} = 20$

$$Speedup = \frac{1}{(1 - 0.70) + \frac{0.70}{20}} = \frac{1}{0.335} = 2.985$$

Parallel programs

- If 30% of the execution time may be the subject of a speedup, p will be 0.3; if the improvement makes the affected part twice as fast, s will be 2. Amdahl's law states that the overall speedup of applying the improvement will be:

$$S_{\text{latency}} = \frac{1}{1 - p + \frac{p}{s}} = \frac{1}{1 - 0.3 + \frac{0.3}{2}} = 1.18.$$

- For example, assume that we are given a serial task which is split into four consecutive parts, whose percentages of execution time are $p1 = 0.11$, $p2 = 0.18$, $p3 = 0.23$, and $p4 = 0.48$ respectively. The 1st part is not speed up, so $s1 = 1$, while the 2nd part is sped up 5 times, so $s2 = 5$, the 3rd part is sped up 20 times, so $s3 = 20$, and the 4th part is sped up 1.6 times, so $s4 = 1.6$.

- By using Amdahl's law, the overall speedup is

$$S_{\text{latency}} = \frac{1}{\frac{p1}{s1} + \frac{p2}{s2} + \frac{p3}{s3} + \frac{p4}{s4}} = \frac{1}{\frac{0.11}{1} + \frac{0.18}{5} + \frac{0.23}{20} + \frac{0.48}{1.6}} = 2.19.$$

- Notice how the 20 times and 5 times speedup on the 2nd and 3rd parts respectively don't have much effect on the overall speedup when the 4th part (48% of the execution time) is accelerated by only 1.6 times.

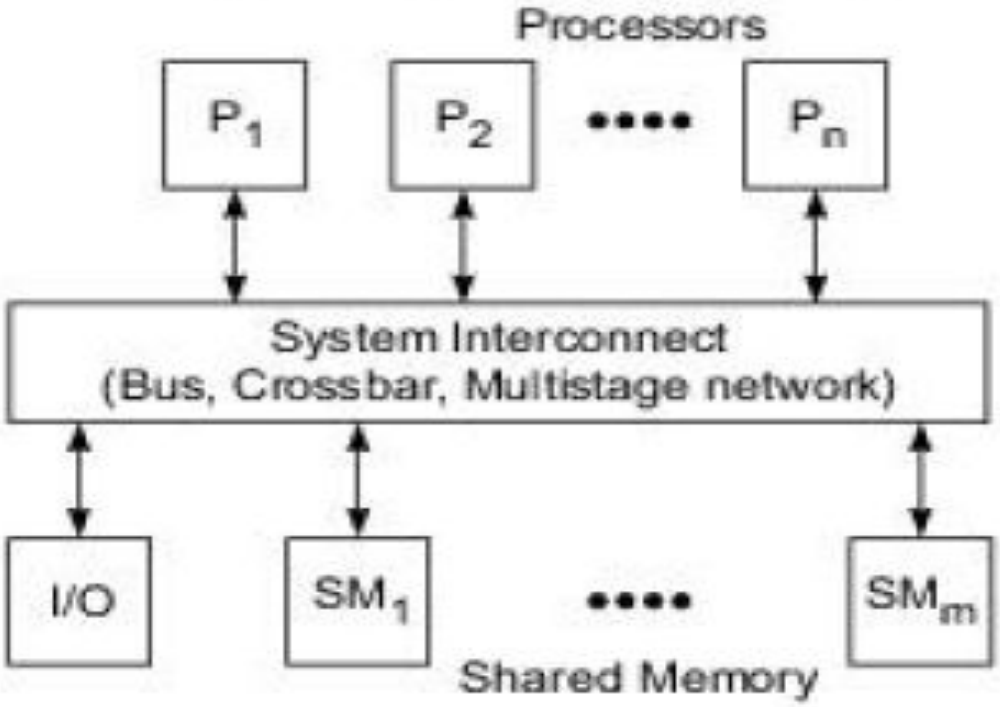
Module 1

- **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

Multiprocessors and Multicomputers

- 3 shared- memory multiprocessor models
    - Uniform memory-access(UMA) Model
    - Non-uniform memory-access(NUMA) Model
    - Cache-only memory architecture(COMA) Model

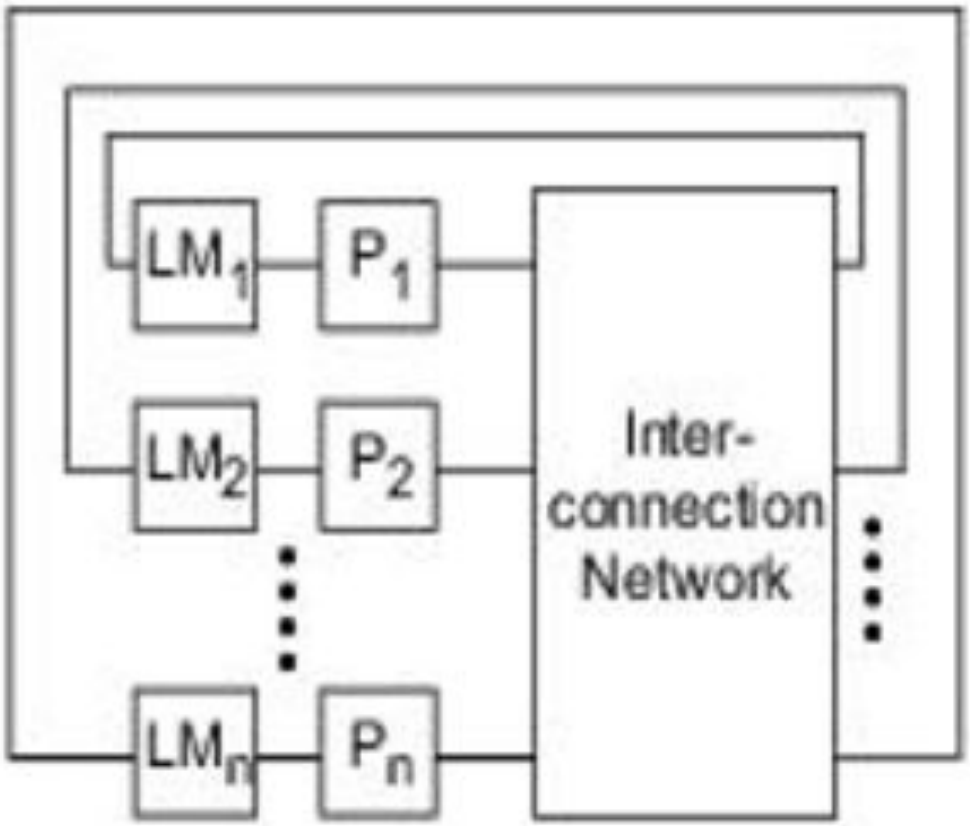
- **UMA Model**
  - Physical memory is uniformly shared by all the processors
  - Equal access time – so called uniform memory access
  - Private cache



Multiprocessors – Tightly coupled systems

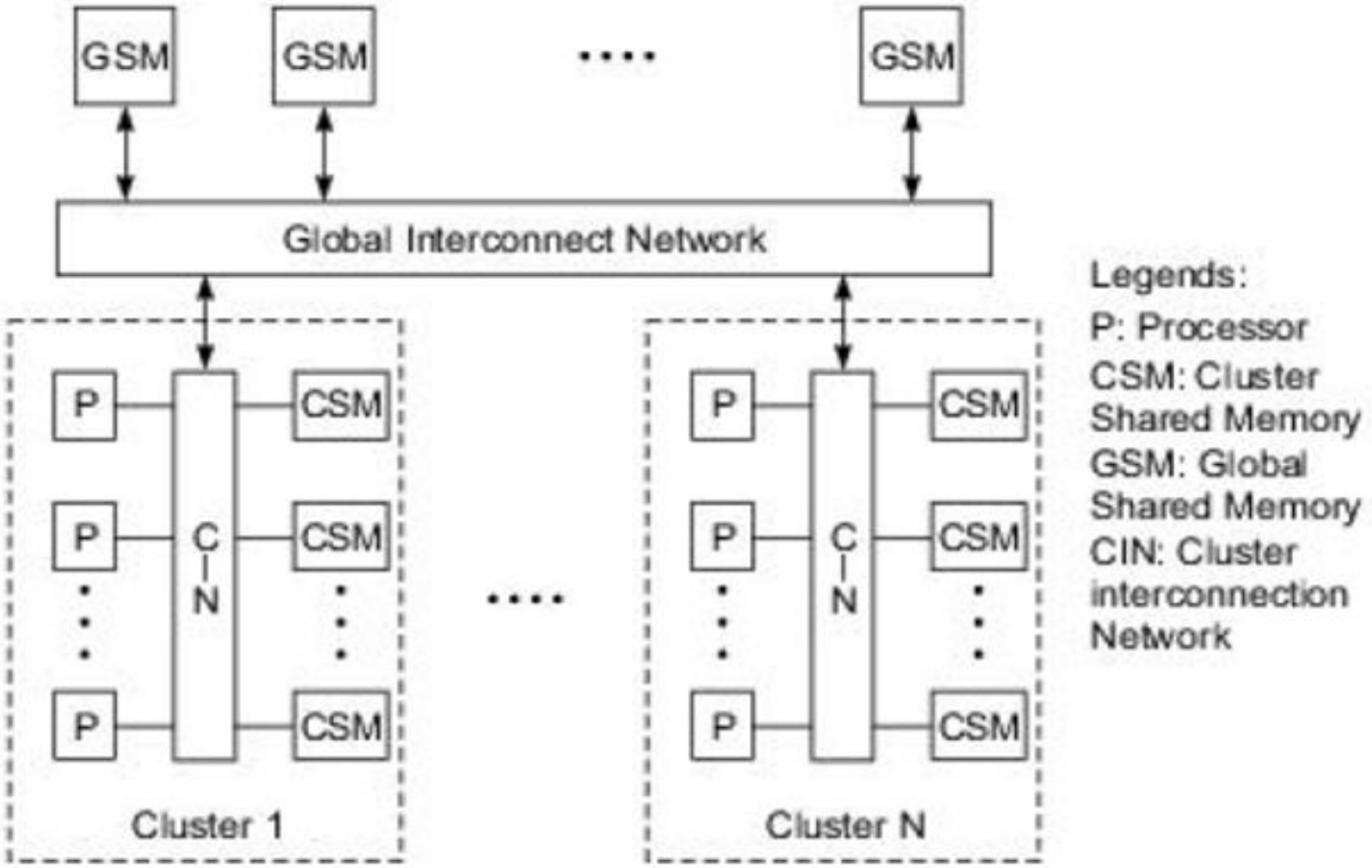
- General-purpose and time sharing applications
- Parallel events, synchronization and communication using shared variables
- Symmetric multiprocessor
- Asymmetric multiprocessor

- **NUMA Model**
  - Shared-memory system



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

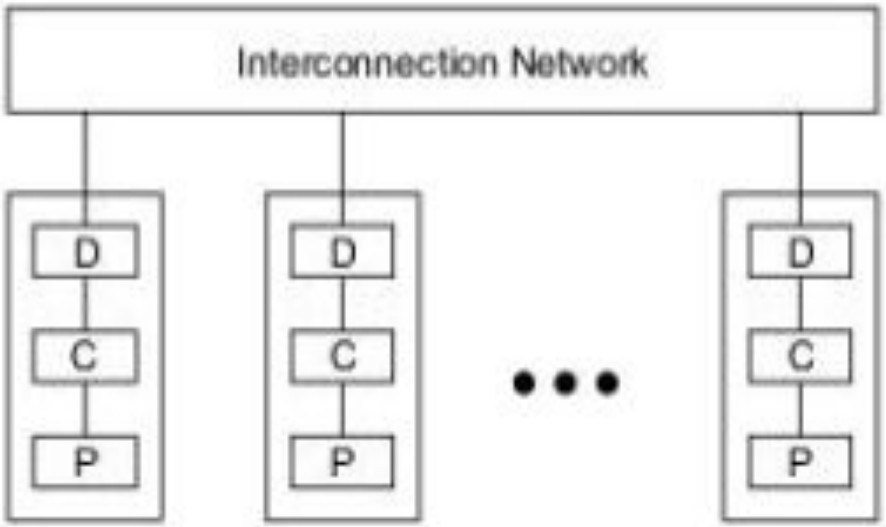
NUMA Model



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

- **COMA Model**

  - Special case of a NUMA machine
  - Distributed main memories are converted to caches
  - No memory hierarchy
  - Caches form a global address space
  - Data will eventually migrate to where it will be used.



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

Other Variations for multiprocessor

- Cache-coherent non-uniform memory access model (CC-NUMA)

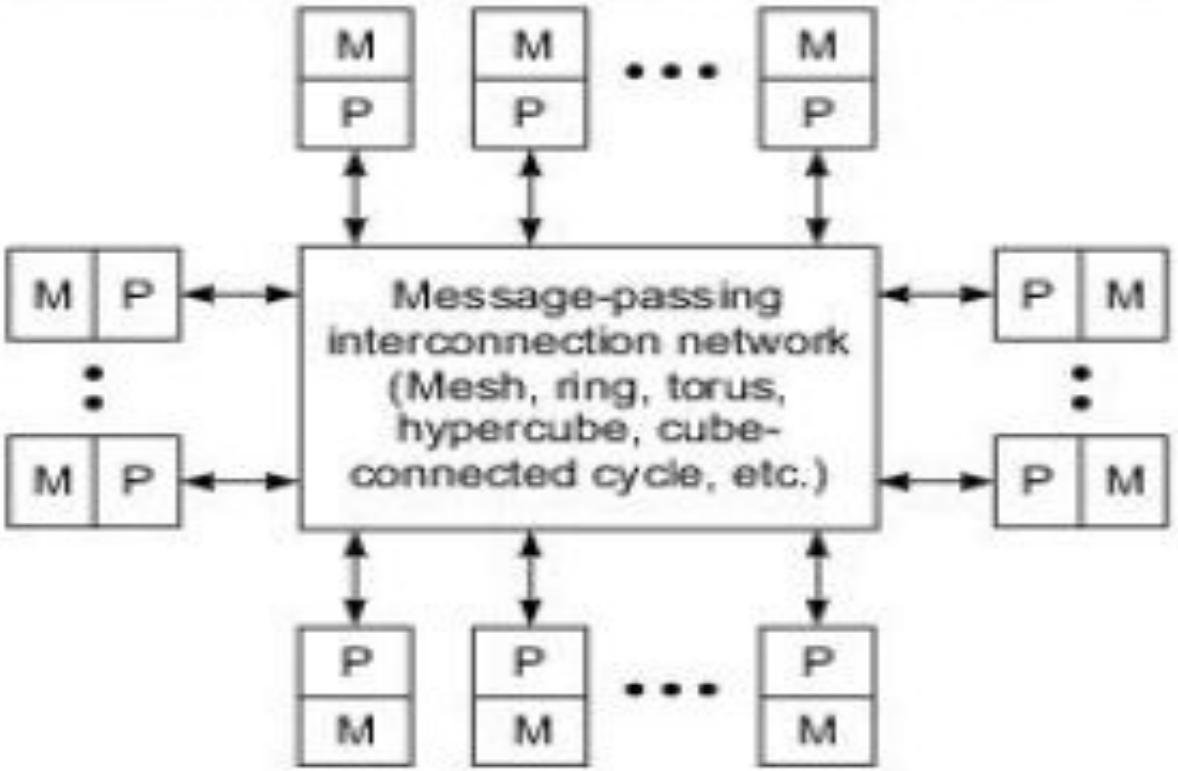
- Representative Multiprocessors

Company and Model	*Hardware and Architecture*	*Software and Applications*	*Remarks*
Sequent Symmetry S-81	Bus-connected with 30 i386 processors, IPC via SLIC bus; Weitek floating-point accelerator.	DYNIX/OS, KAP/Sequent preprocessor, transaction multiprocessing.	Latter models designed with faster processors of the family.
IBM ES/9000 Model 900/VF	6 ES/9000 processors with vector facilities, crossbar connected to I/O channels and shared memory.	OS support: MVS, VM KMS, AIX/370, parallel Fortran, VSF V2.5 compiler.	Fiber optic channels, integrated cryptographic architecture.
BBN TC-2000	512 M88100 processors with local memory connected by a Butterfly switch, a NUMA machine.	Ported Mach/OS with multiclustering, parallel Fortran, time-critical applications.	Latter models designed with faster processors of the family.

- **Distributed-Memory Multicomputers**

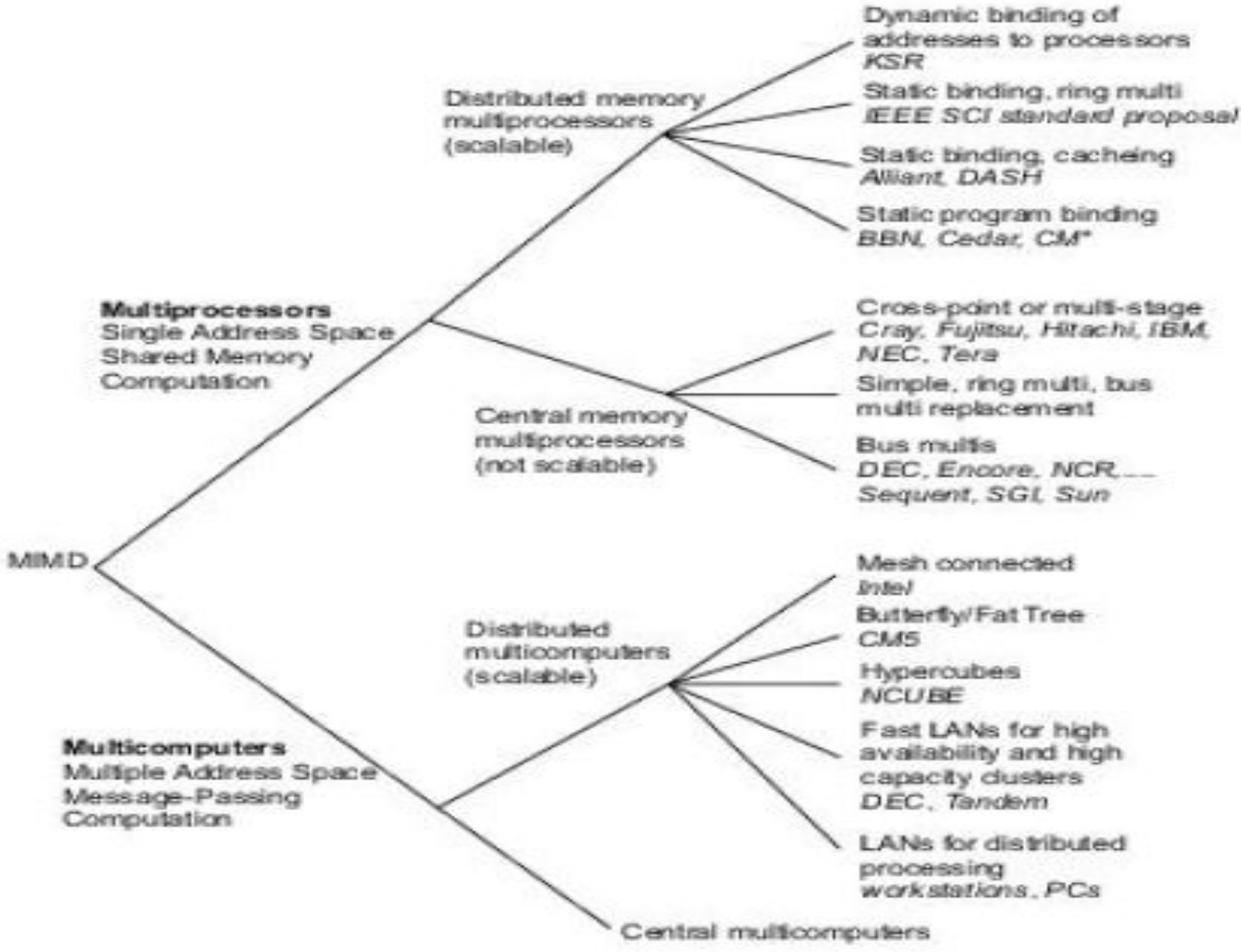
  - System consists of multiple computers – nodes
  - Interconnected by a message-passing network
  - Node is an autonomous computer
  - Consist of a processor, local memory, and disks or l/O peripherals
  - Point-to-point static connections among the nodes
  - Multicomputers also called no-remote-memory- access (NORMA)

- Accessible only by local processors



Generic model of a message passing multicomputers

System Features	*Intel Paragon XP/S*	*nCUBE/2 6480*	*Parsys Ltd. SuperNode1000*
Node Types and Memory	50 MHz i860 XP computing nodes with 16–128 Mbytes per node, special I/O service nodes.	Each node contains a CISC 64-bit CPU, with FPU, 14 DMA ports, with 1–64 Mbytes /node.	EC-funded Esprit supernode built with multiple T-800 Transputers per node.
Network and I/O	2-D mesh with SCSI, HIPPI, VME, Ethernet, and custom I/O.	13-dimensional hypercube of 8192 nodes, 512-Gbyte memory, 64 I/O boards.	Reconfigurable interconnect, expandable to have 1024 processors.
OS and Software Task Parallelism Support	OSF conformance with 4.3 BSD, visualization and programming support.	Vertex/OS or UNIX supporting message passing using wormhole routing.	IDRIS/OS UNIX-compatible.
Application Drivers	General sparse matrix methods, parallel data manipulation, strategic computing.	Scientific number crunching with scalar nodes, database processing.	Scientific and academic applications.
Performance Remarks	5–300 Gflops peak 64-bit results, 2.8–160 GIPS peak integer performance.	27 Gflops peak, 36 Gbytes/s I/O	200 MIPS to 13 GIPS peak.


MIMD
Multiprocessors
Single Address Space
Shared Memory
Computation
Distributed memory
multiprocessors
(scalable)
Dynamic binding of
addresses to processors
KSR
Static binding, ring multi
IEEE SCI standard proposal
Static binding, caching
Alliant, DASH
Static program binding
BBN, Cedar, CM*
Central memory
multiprocessors
(not scalable)
Cross-point or multi-stage
Cray, Fujitsu, Hitachi, IBM,
NEC, Tera
Simple, ring multi, bus
multi replacement
Bus multis
DEC, Encore, NCR,...
Sequent, SGI, Sun
Multicomputers
Multiple Address Space
Message-Passing
Computation
Distributed
multicomputers
(scalable)
Mesh connected
Intel
Butterfly/Fat Tree
CM5
Hypercubes
NCUBE
Fast LANs for high
availability and high
capacity clusters
DEC, Tandem
LANs for distributed
processing
workstations, PCs
Central multicomputers

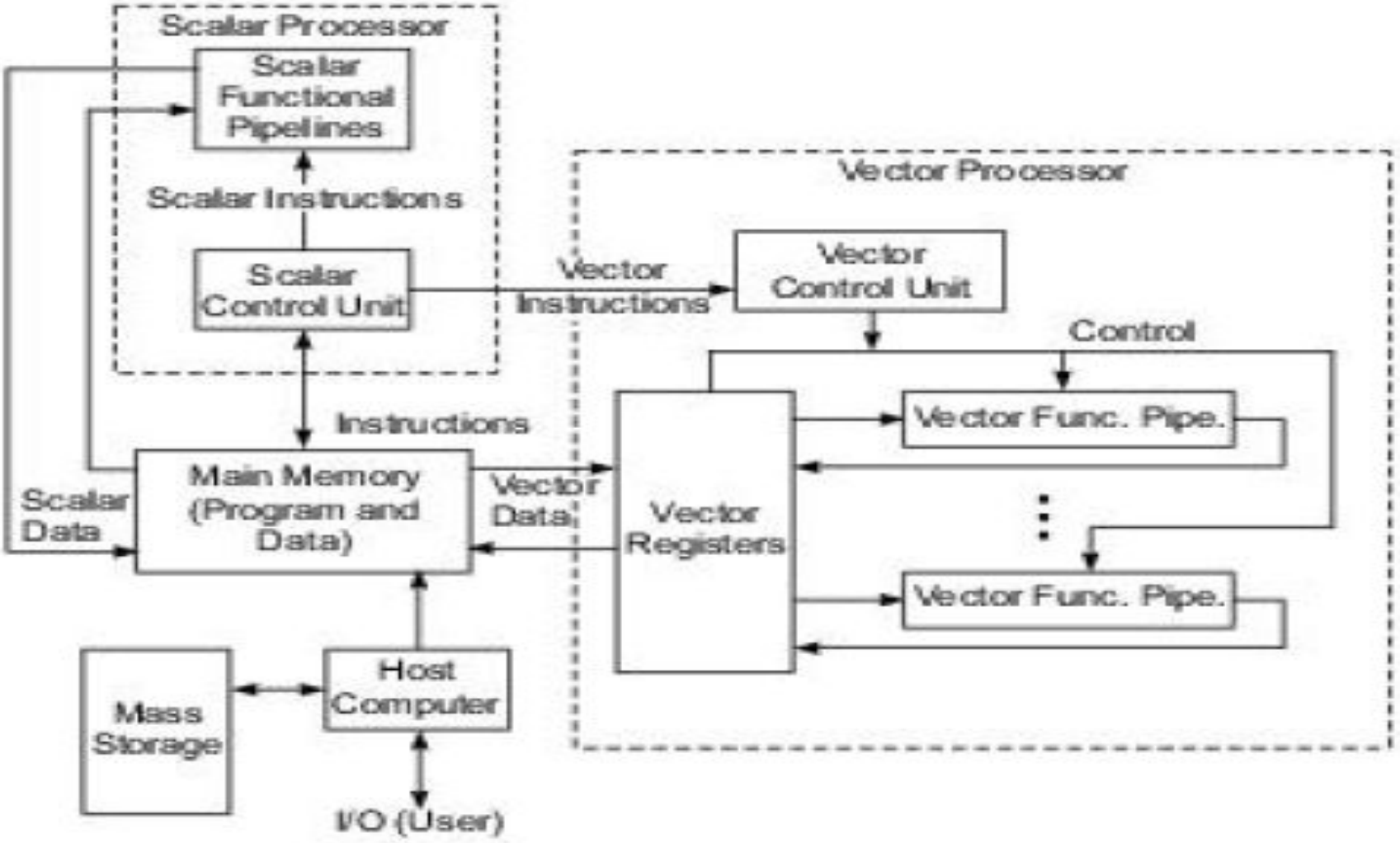
Module 1

- **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

Multivector and SIMD computers

- Classify supercomputers either as
    - Vector Supercomputers
    - SIMD Supercomputers
- **Vector Supercomputer** – built on top of supercomputer
- Vector processor attached to scalar processor
- All instructions are first decoded by the scalar control unit
- If decoded instruction either a scalar operation or a program control operation : executed by the scalar processor using the scalar functional pipelines

- The architecture of a vector supercomputer

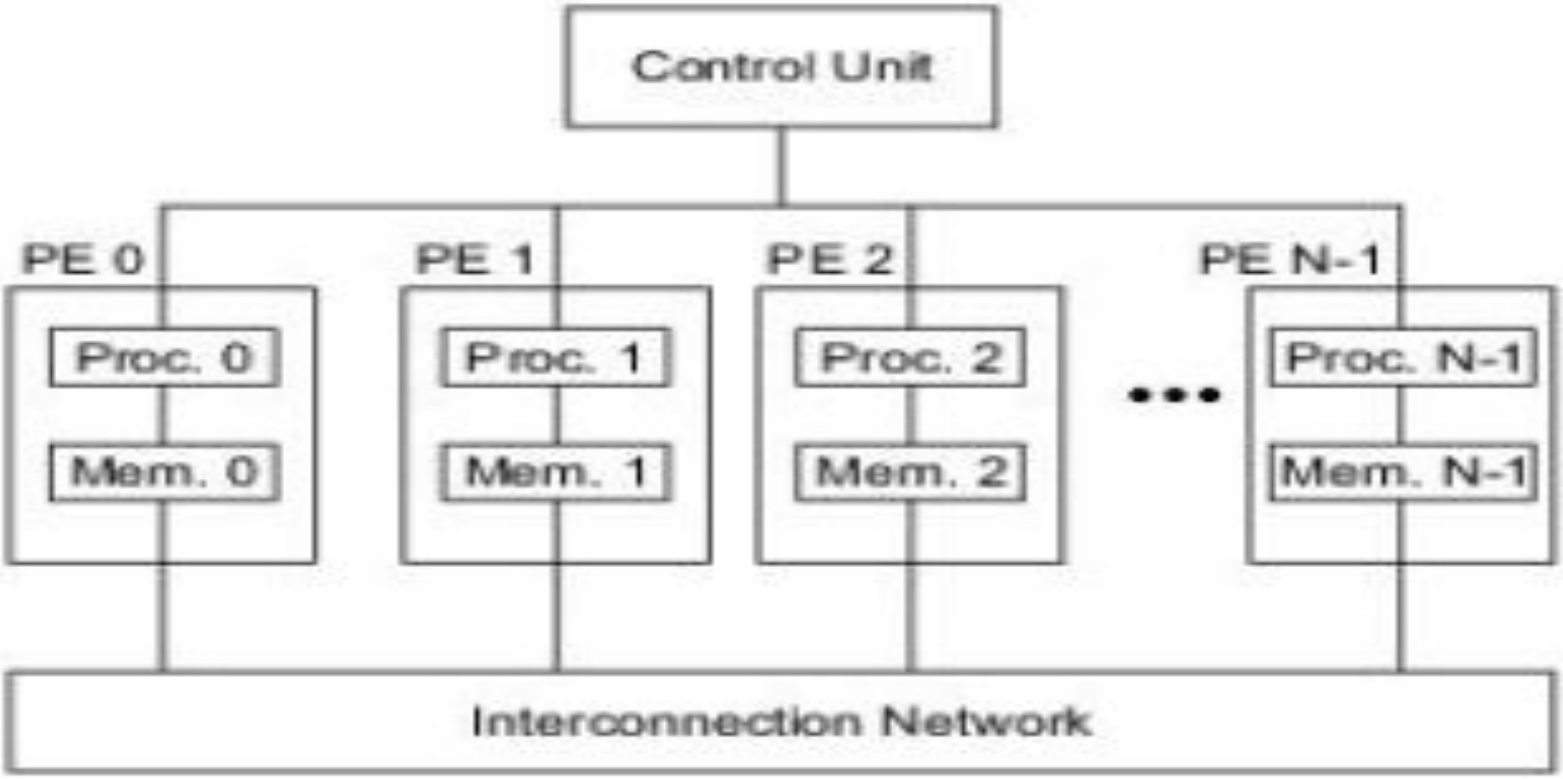


- Early Vector Processor Model:

System Model	*Vector Hardware Architecture and Capabilities*	*Compiler and Software Support*
Convex C3800 family	GaAs-based multiprocessor with 8 processors and 500-Mbyte/s access port. 4 Gbytes main memory. 2 Gflops peak performance with concurrent scalar/vector operations.	Advanced C, Fortran, and Ada vectorizing and parallelizing compilers. Also supported interprocedural optimization, POSIX 1003.1/OS plus I/O interfaces and visualization system
Digital VAX 9000 System	Integrated vector processing in the VAX environment, 125–500 Mflops peak performance. 63 vector instructions. $16 \times 64 \times 64$ vector registers. Pipeline chaining possible.	MS or ULTRIX/OS, VAX Fortran and VAX Vector Instruction Emulator (VVIEF) for vectorized program debugging.
Cray Research Y-MP and C-90	Y-MP ran with 2, 4, or 8 processors, 2.67 Gflop peak with Y-MP8256. C-90 had 2 vector pipes/CPU built with 10K gate ECL with 16 Gflops peak performance.	CF77 compiler for automatic vectorization, scalar optimization, and parallel processing. UNICOS improved from UNIX/V and Berkeley BSD/OS.

SIMD Supercomputers

- single instruction stream over multiple data streams



Operational model of SIMD computers

- An operational model of an SIMD computer

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

{1} N is the number of processing elements (PEs) in the machine

{2} C is the set of instructions directly executed by the control unit (CU)

{3} I is the set of instructions broadcast by the CU to all PEs for parallel execution

{4} M is the set of masking schemes

{5} R is the set of data-routing functions

- Some Early Commercial SIMD Supercomputers

System Model	*SIMD Machine Architecture and Capabilities*	*Languages, Compilers and Software Support*
MasPar Computer Corporation MP-1 Family	Designed for configurations from 1024 to 16,384 processors with 26,000 MIPS or 1.3 Gflops. Each PE was a RISC processor, with 16 Kbytes local memory. An X-Net mesh plus a multistage crossbar interconnect.	Fortran 77, MasPar Fortran (MPF), and MasPar Parallel Application Language; UNIX/OS with X-window, symbolic debugger, visualizers and animators.
Thinking Machines Corporation, CM-2	A bit-slice array of up to 65,536 PEs arranged as a 10-dimensional hypercube with 4 × 4 mesh on each vertex, up to 1M bits of memory per PE, with optional FPU shared between blocks of 32 PEs. 28 Gflops peak and 5.6 Gflops sustained.	Driven by a host of VAX, Sun, or Symbolics 3600, Lisp compiler, Fortran 90, C*, and *Lisp supported by PARIS
Active Memory Technology DAP600 Family	A fine-grain, bit-slice SIMD array of up to 4096 PEs interconnected by a square mesh with 1 K bits per PE, orthogonal and 4-neighbor links, 20 GIPS and 560 Mflops peak performance.	Provided by host VAX/VMS or UNIX Fortran-plus or APAL on DAP, Fortran 77 or C on host.

Module 1

- **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

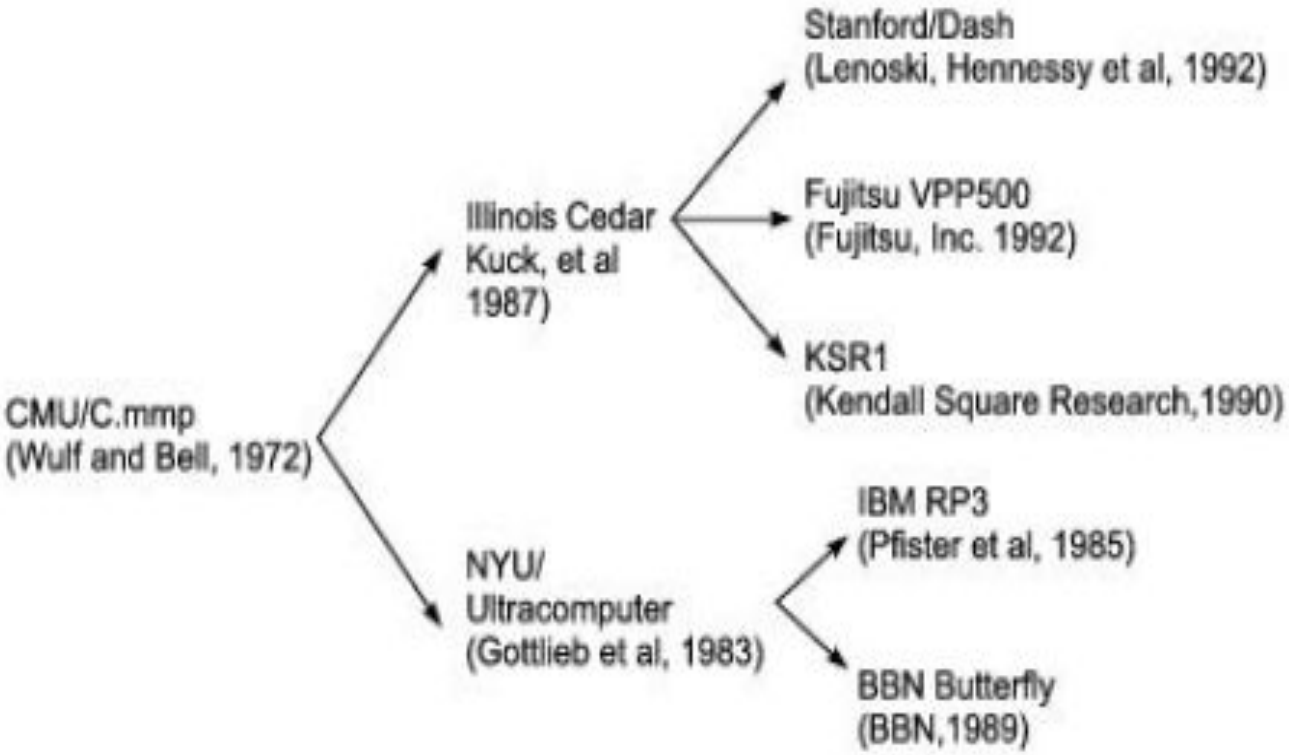
Architectural Development Tracks

1. Multiple-Processor Tracks
2. Multivector and SlMD Tracks
3. Multithreaded and Dataflow Tracks

1. Multiple-Processor Tracks(Shared-Memory Track and Message-passing Tracks)

- Multiple-processor system can be either a shared-memory multiprocessor or a distributed-memory multicomputer

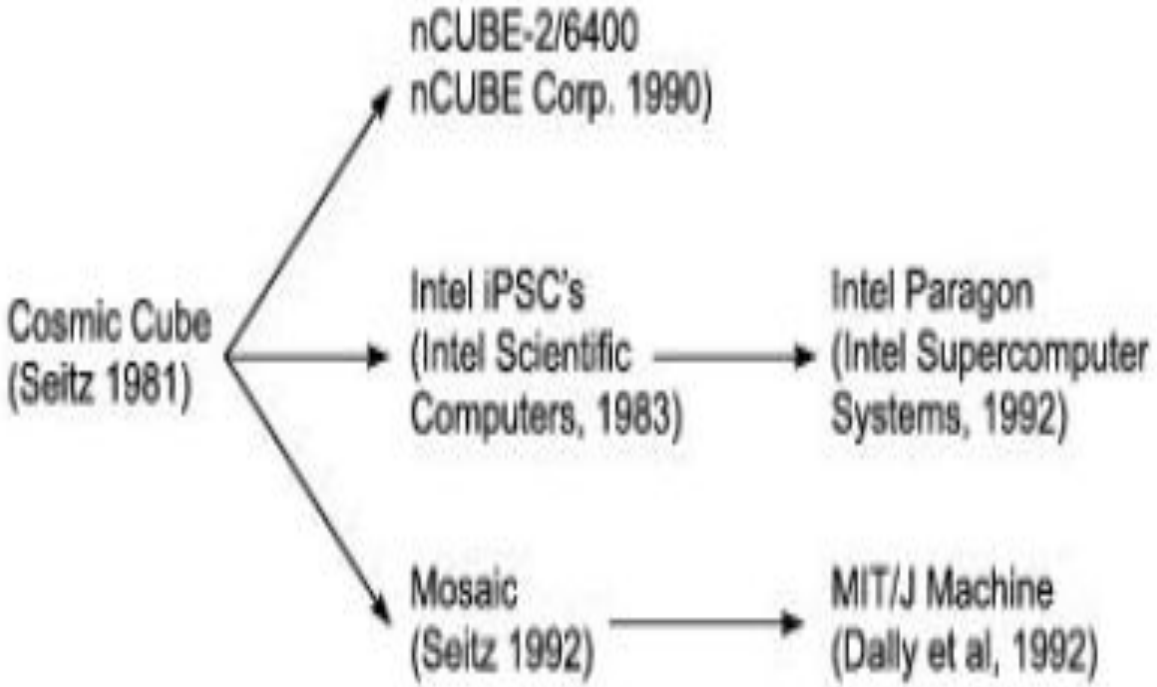
- **Shared-memory Track**



(a) Shared-memory track

- Employing a single address space in the entire system

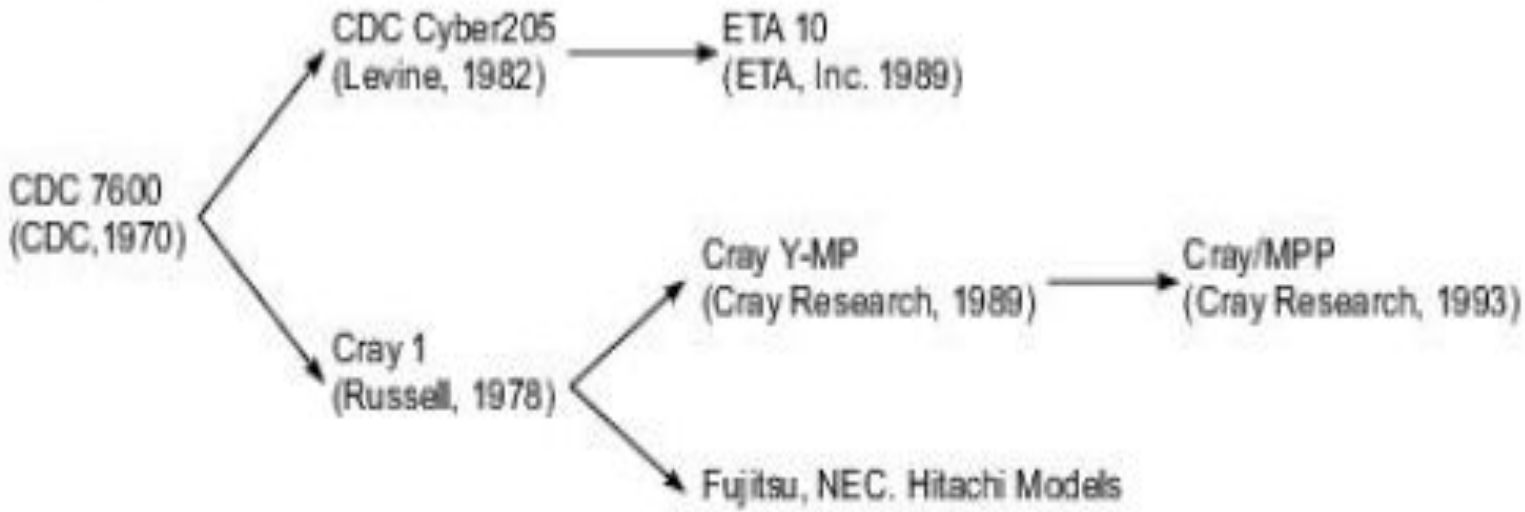
- **Message-passing Track**



(b) Message-passing track

2. Multivector and SIMD Tracks

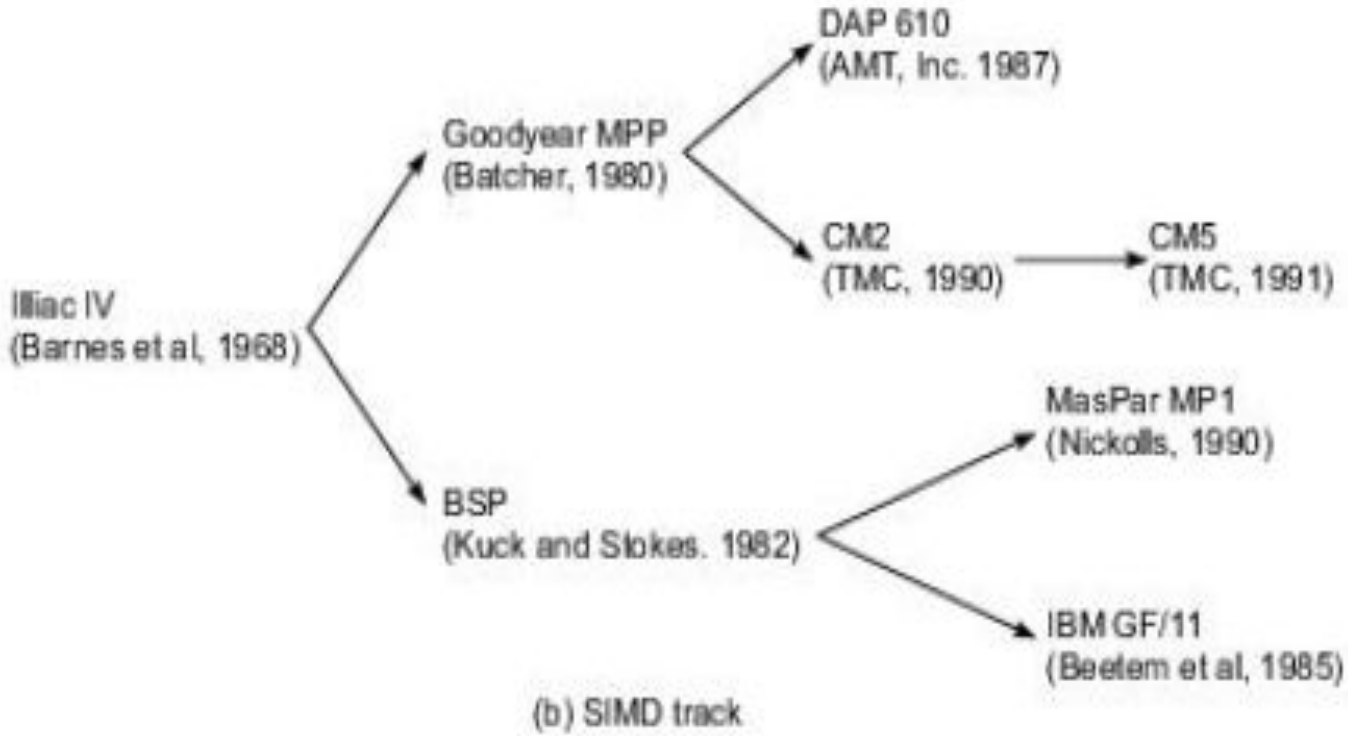
- **Multivector Track**
  - Traditional vector supercomputers



(a) Multivector track

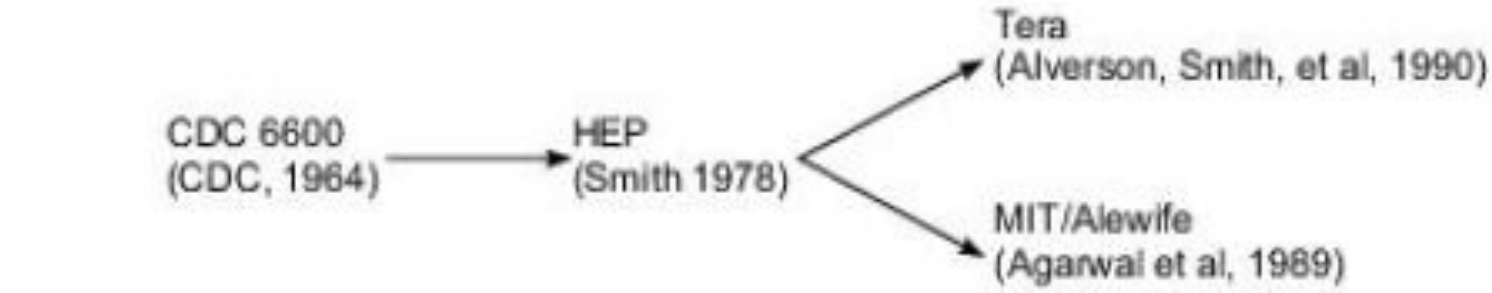
- **The SIMD Track**

  - Illiac IV pioneered the construction of SIMD computers

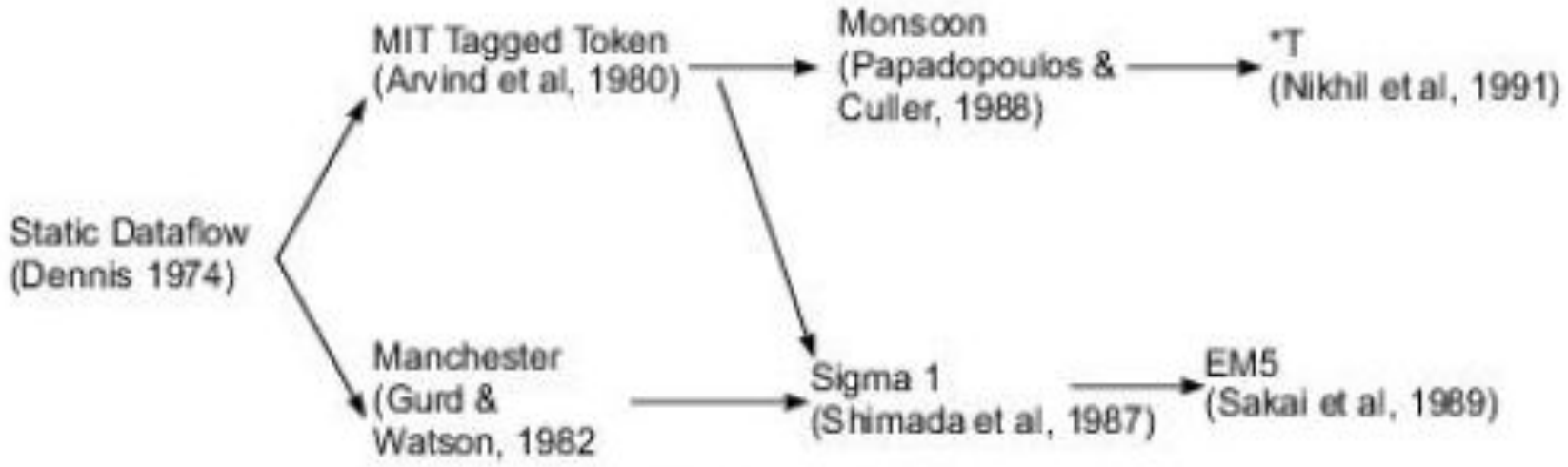


(b) SIMD track

3. Multithreaded and Dataflow Tracks



(a) Multithreaded track



(b) Dataflow track

- Each processor maintains a single thread of control with its hardware resources
- Multithreading - there are multiple threads of control in each processor
- Hiding long latency in building large-scale multiprocessors

- **The dataflow track**
  - To direct the program flow
  - Fine-grain instruction-level parallelism is exploited **(Fig(b))**

Module 1

- **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

Conditions Of Parallelism

- To move parallel processing into the mainstream of computing make significant progress in
  - Computation Models
  - Inter Process Communications
  - System Integration

1) Data and Resource Dependence

- Dependence graph to describe the relations
  - **Data Dependence**

Five types of data dependence:

[1] *Flow dependence*: if at least one output of S1 feeds in as input to S2

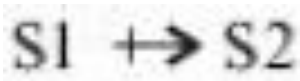
$$S1 \rightarrow S2.$$

Eg:S1: A=B+C

S2: D=F x A

1) Data and Resource Dependence

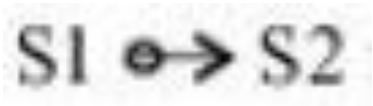
[2] *Antidependence*: if the output of S2 overlaps the input to S1



Eg:S1: R1=R3+R7

S2: R3=R4+R5

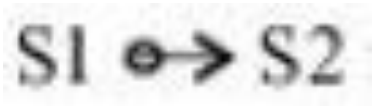
[3] *Output dependence*: Two statements are output dependence if they produce the same output variable.



Eg:S1:A=B+C

S2:A=E-F

[[4] *I/O dependence* : Read and write are I/O statements



S1: Read (4),A(I)

S2: Process

S3: Write (4),B(I)

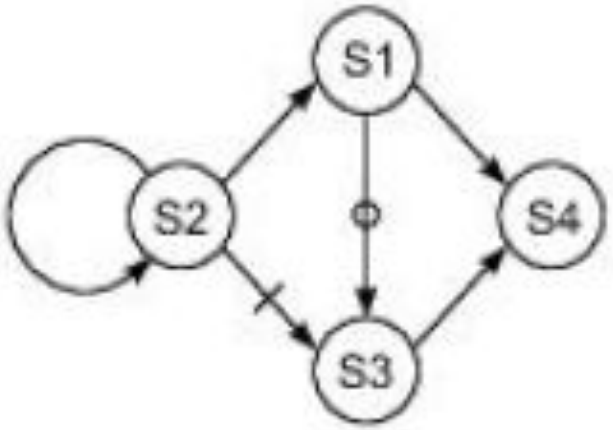
S4: Close(4)

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

- The subscript of a variable is itself subscribed.
- The subscript does not contain the loop index variable.
- A variable appears more than once with subscripts having different coefficients of the loop variable.
- The subscript is nonlinear in the loop index variable

Data dependence in programs

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)/



(a) Dependence graph

- **Control Dependence**

  - Refers to the situation where the order of execution of statements cannot be determined before run time
  - The successive iterations of the following loop are control-independent:

Control Dependence

```
      Do     10I = 1, N
             IF (A(I – 1) .EQ. 0) A(I) = 0
10    Continue
```

- **Resource Dependence**

  - Concerned with the conflicts in using shared resources such as integer units, floating point units, registers, and memory areas, among parallel events
  - When the conflicting resource is an ALU, call it ALU dependence

- **Bernstein's condition:**
  - Revealed a set of conditions based on which two processes can execute in parallel
- Define the input set i_i, of a process p_i, as the set of all input variables needed to execute the process

Bernstein's condition

- Consider two processes P1 and P2 with their input sets I1 and I2 and output sets O1 and O2 respectively
- 2 processes parallel P1|| P2

Bernstein's condition

$$\left.\begin{aligned} I_1 \cap O_2 &= \phi \\ I_2 \cap O_1 &= \phi \\ O_1 \cap O_2 &= \phi \end{aligned}\right\}$$

- Three conditions are known as Bernstein's Conditions

DETECTION OF PARALLELISM

P1: C= D x E

P2: M=G+C

P3: A=B+C

P4:C=L+M

P5: F=G/E

DETECTION OF PARALLELISM

P1: C= D x E

P2: M=G+C

CONSIDER P1 AND P2

$I_1 \cap O_2 = \phi$

$I_2 \cap O_1 = C \neq \phi$

P1 $\not\parallel$ P2

DETECTION OF PARALLELISM

P1: C= D x E

P3: A=B+C

CONSIDER P1 AND P3

$I_1 \cap O_3 = \phi$

$I_3 \cap O_1 = C \neq \phi$

P1 $\not\parallel$ P3

DETECTION OF PARALLELISM

P1: C= D x E

P4: C=L+M

CONSIDER P1 AND P4

$I_1 \cap O_4 = \phi$

$I_4 \cap O_1 = \phi$

$O_1 \cap O_4 = C \neq \phi$

P1 $\nparallel$ P4

DETECTION OF PARALLELISM

P1: C= D x E

P5: F=G/E

CONSIDER P1 AND P5

$$I_1 \cap O_5 = \phi$$
$$I_5 \cap O_1 = \phi$$
$$O_1 \cap O_5 = \phi$$
$$P1 \parallel P5$$

DETECTION OF PARALLELISM

P2: M= G+C

P3: A=B+C

CONSIDER P2 AND P3

$$I_2 \cap O_3 = \phi$$
$$I_3 \cap O_2 = \phi$$
$$O_2 \cap O_3 = \phi$$
$$\text{P2} \parallel \text{P3}$$

DETECTION OF PARALLELISM

P2: M= G+C

P4: C=L+M

CONSIDER P2 AND P4

$I_2 \cap O_4 =$C

P2 $\not\parallel$ P3

DETECTION OF PARALLELISM

P2: M= G+C

P5: F=G/E

CONSIDER P2 AND P5

$$I_2 \cap O_5 = \phi$$
$$I_5 \cap O_2 = \phi$$
$$O_2 \cap O_5 = \phi$$
$$\text{P2} \parallel \text{P5}$$

DETECTION OF PARALLELISM

P3: A= B+C

P4: C=L+M

CONSIDER P3 AND P4

$I_3 \cap O_4 = \mathrm{C}$

P3 $\not\parallel$ P4

DETECTION OF PARALLELISM

P3: A= B+C

P5: F=G/E

CONSIDER P3 AND P5

$$I_3 \cap O_5 = \phi$$
$$I_5 \cap O_3 = \phi$$
$$O_3 \cap O_5 = \phi$$
$$P3 \parallel P5$$

DETECTION OF PARALLELISM

P4: C= L+M

P5: F=G/E

CONSIDER P3 AND P5

$I_4 \cap O_5 = \phi$

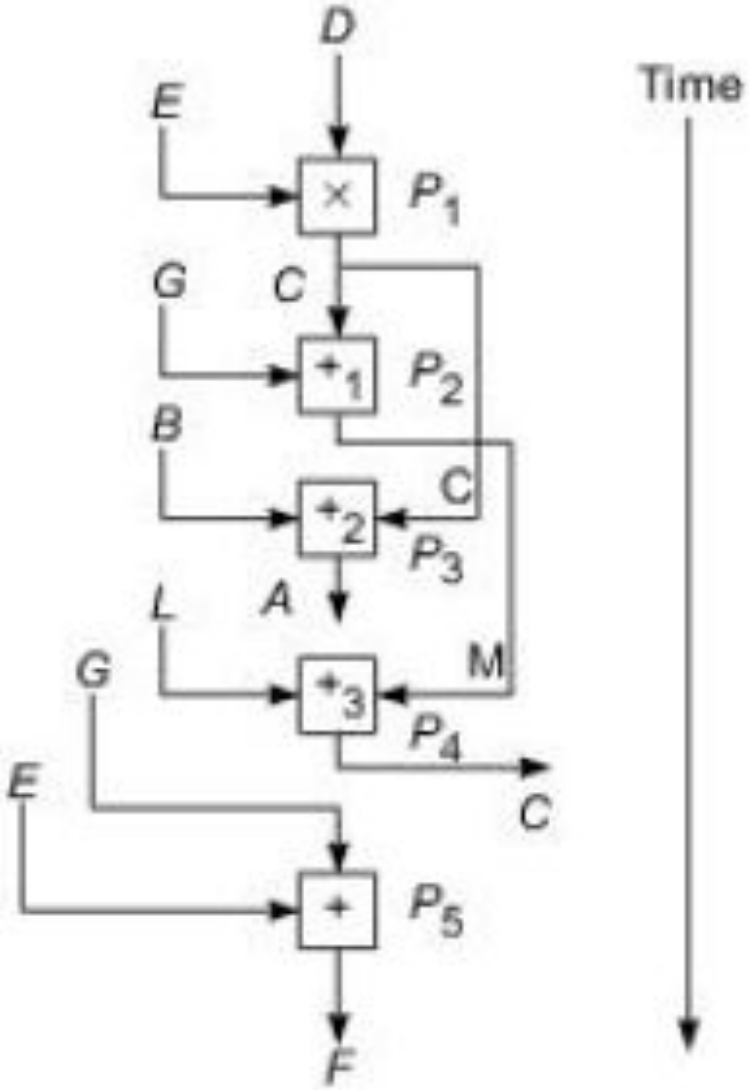
$I_5 \cap O_4 = \phi$

$O_4 \cap O_5 = \phi$

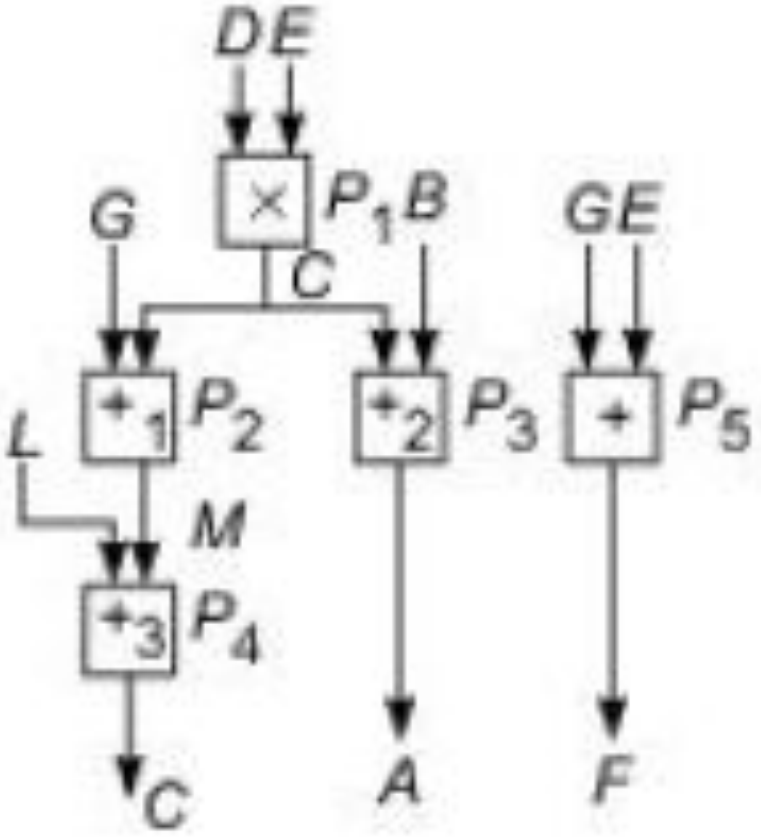
$\mathrm{P4} \parallel \mathrm{P5}$

DETECTION OF PARALLELISM

- ONLY 5 pairs
- P1 ‖ P5, P2 ‖ P3, P2 ‖ P5, P3 ‖ P5 and P4 ‖ P5 can execute in parallel if there is no resource conflict
- Collectively P2 ‖ P3 ‖ P5



(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

- **Hardware and Software Parallelism**

  - Need special hardware and software support
    - **Hardware Parallelism**

      Type of parallelism defined by the machine architecture and hardware multiplicity
    - **Software Parallelism**

      Revealed in the program profile or in the program flow graph

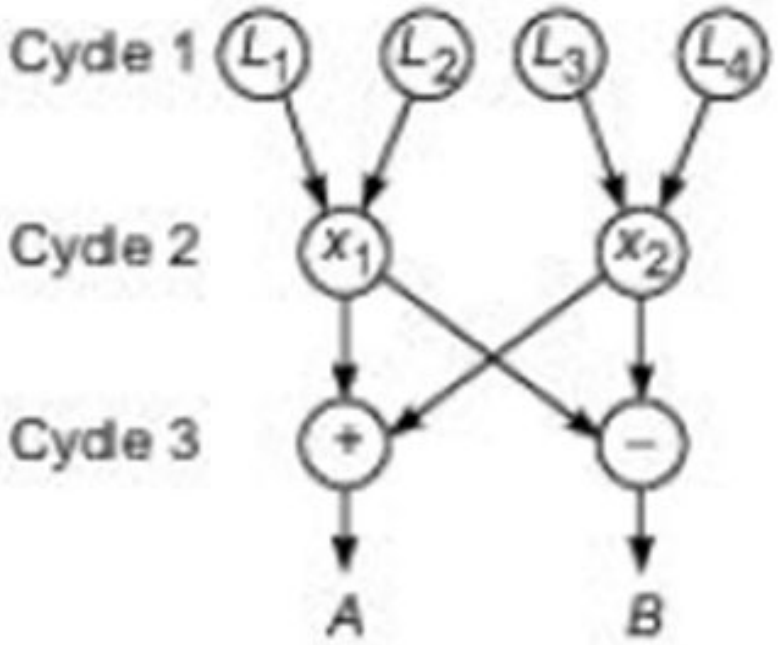
      Software parallelism is it function of algorithm, programming style, and program design

Miss match

- A=L1*L2+L3*L4
- B=L1*L2-L3*L4

8 instruction –four load, 2 multi, 2 add/sub

- 8/3=2.67

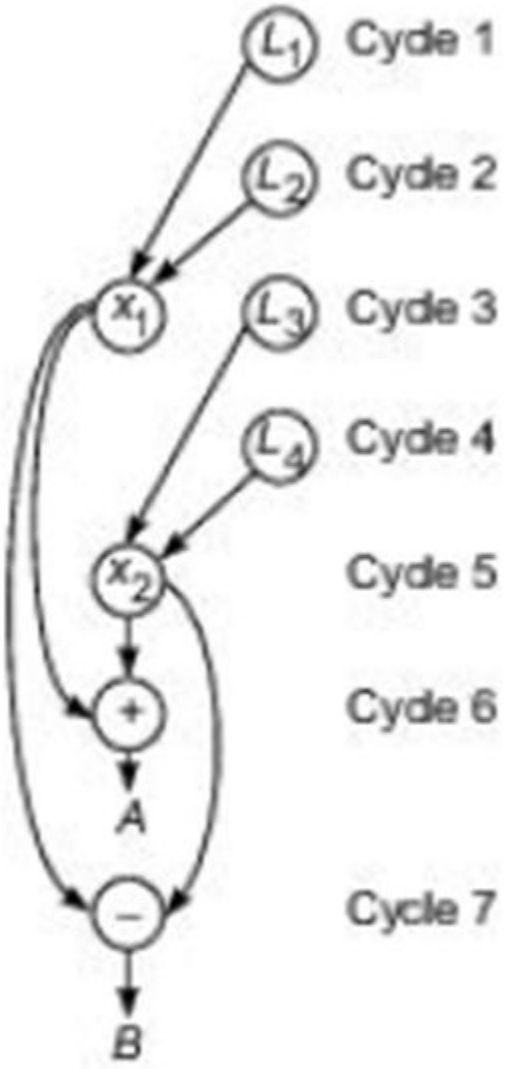


L_j: Load operation
X_j: Multiply operation

(a) Software parallelism

- 2-issue processor
- Execute one memory access(load & write) and one arithmetic (add,sub,mul)

- 8/7=1.14



(b) Hardware parallelism