

# CS405 Computer System Architecture

Dr CK Raju

Dept of CSE

September 5, 2018



# Presentation Outline

- 1 Module I: Parallel Computer Models
  - Evolution of Computer Architecture
  - System Attributes to Performance
  - Multiprocessors and Multicomputers
  - Multivector and SIMD



# Contents of Module I



# Contents of Module I

- Architecture of a Simple Processor



# Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models



# Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture



# Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance



# Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers





# Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers
- Multivector and SIMD Computers

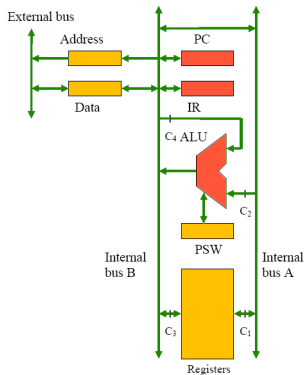


# Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers
- Multivector and SIMD Computers
- Conditions of Parallelism



# Hardware Architecture



ALU (arithmetic, logical and shift),  
FPU (Floating Point Unit),  
IR (Instruction Register),  
PC (Program Counter),  
PSW (Program Status Word)

# Algorithm of a Simple Processor

repeat forever

begin

- fetch the next instruction from main memory - almost same for all instructions
- execute - i.e. carry out - the instruction fetched - might vary with every instruction



# Algorithm of a Simple Processor

repeat forever

begin

- fetch the next instruction from main memory - almost same for all instructions
- execute - i.e. carry out - the instruction fetched - might vary with every instruction

Note: There are two views for the processor

- (a) In one view, machine instruction set architecture (ISA) defines microprocessor
- (b) In other view, hardware design and circuitry defines the microprocessor end



# Evolution of Computer Architecture

Generation	Technology	Systems
First (1945-54)	Vacuum tubes, relays	Eniac, IBM 701
Second (1955-64)	Transistors	Univac, IBM 7090,
Third (1965-74)	Integrated Circuits	IBM 360, PDP-8
Fourth (1975-90)	Microprocessors	VAX 9000, Cray X-MP
Fifth (1991 onwards)	Artificial Intelligence	IBM ES 9000, Xeon PHI



# Evolution of Computer Architecture

Generation	Architecture	Systems
First	PC, ALU, FP Arithmetic	Eniac, IBM 701
Second	Multiplexed Memory Access	Univac, IBM 7090,
Third	SSI, MSI, Cache	IBM 360, PDP-8
Fourth	LSI, VLSI, multiprocessors	VAX 9000, Cray X-MP
Fifth	Scalable Multicomputers	IBM ES 9000, Xeon PHI



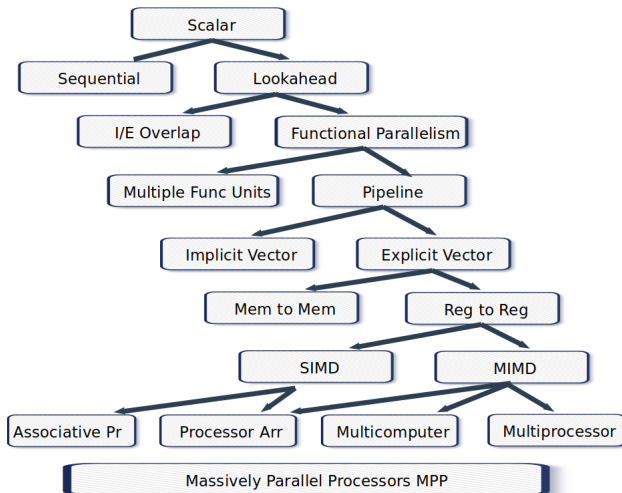
# Evolution of Computer Architecture

Generation	Software	Systems
First	Assembly, Machine	Eniac, IBM 701
Second	HLL with compilers	Univac, IBM 7090,
Third	Multiprogramming OS	IBM 360, PDP-8
Fourth	Multiprocessor OS, Parallel	VAX 9000, Cray X-MP
Fifth	Superscalar, massively parallel	IBM ES 9000, Xeon PHI





# Evolution of Computer Architecture



# Lookahead, Parallelism and Pipelining



# Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism



# Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining



# Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly



# Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly

Your smartphone could be an octa-core processor-based ...



# Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly

Your smartphone could be an octa-core processor-based ... What should be its parallel architecture ...



# Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly

Your smartphone could be an octa-core processor-based ... What should be its parallel architecture ... think, refer and give a feedback in the next session





# Flynn's Classification



# Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams



# Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequential machines were SISD - single instruction stream over single data stream



# Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequential machines were SISD - single instruction stream over single data stream
- Vector computers equipped with scalar or vector hardware appear as SIMD



# Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequential machines were SISD - single instruction stream over single data stream
- Vector computers equipped with scalar or vector hardware appear as SIMD
- Parallel computers are reserved for MIMD machines



# Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequential machines were SISD - single instruction stream over single data stream
- Vector computers equipped with scalar or vector hardware appear as SIMD
- Parallel computers are reserved for MIMD machines
- A MISD architecture called systolic arrays for pipelined execution of specific algorithms is also present



# Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like



# Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology





# Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features



# Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features
- efficient resource management



# Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features
- efficient resource management
- algorithm design



# Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features
- efficient resource management
- algorithm design
- data structures, language efficiency, programmer skills, compiler technology etc

Note: Generally its *turnaround time* that is considered that includes disk and memory access, input and output activities, compilation time, OS overhead and CPU time. Since CPU time includes system CPU time and user CPU time, we focus only on *user CPU time* for the purpose of benchmarking performance.



# Clock Rate and CPI - cycles per instruction



# Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time  $T$



# Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time  $T$
- Clock Rate is  $f = 1/T$



# Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time  $T$
- Clock Rate is  $f = 1/T$
- Size of program is determined by Instruction Count  $I_c$  - no of instructions to be executed in the program





# Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time  $T$
- Clock Rate is  $f = 1/T$
- Size of program is determined by Instruction Count  $I_c$  - no of instructions to be executed in the program
- Different instructions use different clock cycles



# Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time  $T$
- Clock Rate is  $f = 1/T$
- Size of program is determined by Instruction Count  $I_c$  - no of instructions to be executed in the program
- Different instructions use different clock cycles
- Cycles per instruction CPI is therefore used as a parameter to measure time



# Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time  $T$
- Clock Rate is  $f = 1/T$
- Size of program is determined by Instruction Count  $I_c$  - no of instructions to be executed in the program
- Different instructions use different clock cycles
- Cycles per instruction CPI is therefore used as a parameter to measure time
- Average CPI is found out for the program (taking into consideration all its instructions and the clock cycles needed for executing these)



# Performance Factors



# Performance Factors

- Let Instruction Count of a program be  $I_c$



# Performance Factors

- Let Instruction Count of a program be  $I_c$
- CPU time =  $I_c \times \text{CPI} \times T$  (const cycle time =  $1/f$ )



# Performance Factors

- Let Instruction Count of a program be  $I_c$
- CPU time =  $I_c \times \text{CPI} \times T$  (const cycle time =  $1/f$ )
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)



# Performance Factors

- Let Instruction Count of a program be  $I_c$
- CPU time =  $I_c \times \text{CPI} \times T$  (const cycle time =  $1/f$ )
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)
- Here instruction decode and execute are carried out by CPU. Other stages need access to memory.





# Performance Factors

- Let Instruction Count of a program be  $I_c$
- CPU time =  $I_c \times \text{CPI} \times T$  (const cycle time =  $1/f$ )
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)
- Here instruction decode and execute are carried out by CPU. Other stages need access to memory.
- A memory cycle is time needed to complete one memory reference - usually memory cycle is  $k$  times processor cycle (ratio)



# Performance Factors

- Let Instruction Count of a program be  $I_c$
- CPU time =  $I_c \times \text{CPI} \times T$  (const cycle time =  $1/f$ )
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)
- Here instruction decode and execute are carried out by CPU. Other stages need access to memory.
- A memory cycle is time needed to complete one memory reference - usually memory cycle is  $k$  times processor cycle (ratio)
- Therefore total time =  $I_c \times (p + m \times k) \times T$  where  $p$  - processor cycles needed for decode and execute,  $m$  is no of memory references needed and  $k$  is ratio between memory cycle and processor cycle,  $I_c$  is the instruction count and  $T$  is the processor cycle time



# System Attributes



# System Attributes

- The five performance factors -  $I_c$  ,  $p$ ,  $m$ ,  $k$ , and  $T$  are influenced by four system attributes



# System Attributes

- The five performance factors -  $I_c$  ,  $p$ ,  $m$ ,  $k$ , and  $T$  are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy



# System Attributes

- The five performance factors -  $I_c$ ,  $p$ ,  $m$ ,  $k$ , and  $T$  are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length ( $I_c$ ) and processor cycles needed ( $p$ )



# System Attributes

- The five performance factors -  $I_c$ ,  $p$ ,  $m$ ,  $k$ , and  $T$  are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length ( $I_c$ ) and processor cycles needed ( $p$ )
- Compiler technology affects program length ( $I_c$ ), processor cycles needed ( $p$ ) and memory reference count ( $m$ )



# System Attributes

- The five performance factors -  $I_c$ ,  $p$ ,  $m$ ,  $k$ , and  $T$  are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length ( $I_c$ ) and processor cycles needed ( $p$ )
- Compiler technology affects program length ( $I_c$ ), processor cycles needed ( $p$ ) and memory reference count ( $m$ )
- CPU implementation and control determine total processor time  $p \times T$  needed














# System Attributes

- The five performance factors -  $I_c$  ,  $p$ ,  $m$ ,  $k$ , and  $T$  are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length ( $I_c$ ) and processor cycles needed ( $p$ )
- Compiler technology affects program length ( $I_c$ ), processor cycles needed ( $p$ ) and memory reference count ( $m$ )
- CPU implementation and control determine total processor time  $p \times T$  needed
- Memory technology and hierarchy design influence memory access latency ( $k \times T$ )



# Performance Factors versus System Attributes

## THE STATE OF COMPUTING System Attributes to Performance

System Attributes	Performance Factors				
	Instruction Count (I)	Average Cycles per Instruction (CPI)			Processor Cycle Time ( $\tau$ )
		Processor Cycles per Instruction (CPI and p)	Memory References per Instruction (m)	Memory Access Latency (k)	
Instruction-set Architecture					
Compiler Technology					
Processor Implementation and Control					
Cache and Memory Hierarchy					



# Multiprocessors and Multicomputers

There are two main categories of parallel computer systems

- systems with common memory that are shared
- systems with distributed memory that are not shared



# Shared memory Multiprocessors

Shared memory multiprocessor models are of three types:

- Uniform-memory-access (UMA)
- Nonuniform-memory-access (NUMA)
- Cache-only memory architecture (COMA)

These systems differ in how the memory and peripheral resources are shared or distributed.



# UMA Model I

- Physical memory uniformly shared by all processors, with equal access time to all words
- Processors may have local cache memories
- Peripherals also shared in some fashion
- Tightly coupled systems use a common bus, crossbar, or multistage network to connect processors, peripherals and memories
- Many manufacturers have multiprocessor (MP) extensions of uniprocessor (UP) product lines

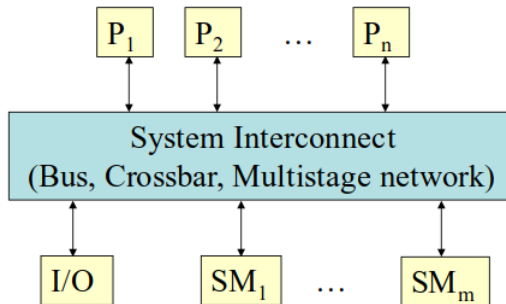


# UMA Model II

- Synchronization and communication among processors achieved through shared variables in common memory
- Symmetric MP systems - all processors have access to all peripherals, and any processor can run the OS and I/O device drivers
- Asymmetric MP systems - not all peripherals accessible by all processors; kernel runs only on selected processors (master node); others are called attached processors (AP)



# The UMA Multiprocessor Model



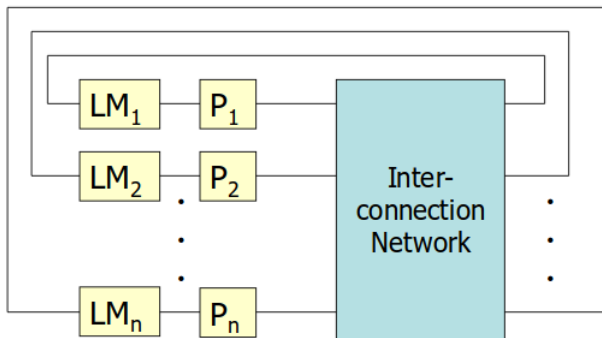
# NUMA Model I

- Shared memories, but access time depends on the location of the data item
- Shared memory is distributed among the processors as local memory, but each of these is still accessible by all processors (with varying access times)
- Memory access is fastest from the locally connected processor, with the interconnection network adding delays for other processor accesses.
- Additionally there may be global memory in a multiprocessor system, with two separate interconnection networks, one for clusters of processors and their cluster memories, and the other for the global shared memories

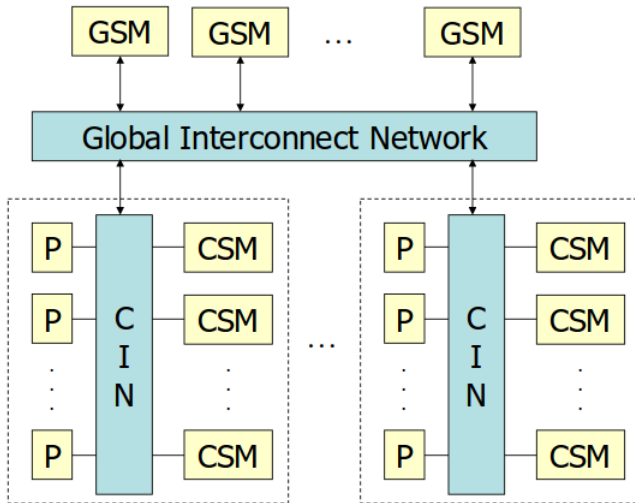




# Shared Local Memories



# Hierarchical Cluster Model

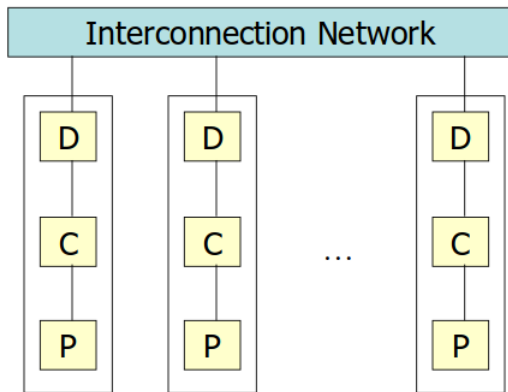


# COMA Model

- In the COMA model, processors only have cache memories; the caches, taken together, form a global address space
- Each cache has an associated directory that aids remote machines in the lookups; hierarchical directories may exist in machines based on this model
- Initial data placement is not critical, as cache blocks will eventually migrate to where they are needed



# Cache-Only Memory Architecture



# Other Models

There are also other models used in multiprocessor systems, based on a combination of models just described.

- Cache-coherent non-uniform memory access (each processor has a cache directory, and the system has a distributed shared memory)
- cache-coherent cache-only model (processors have caches, no shared memory, caches must be kept coherent)



# Some Early Commercial Multiprocessor Systems

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.

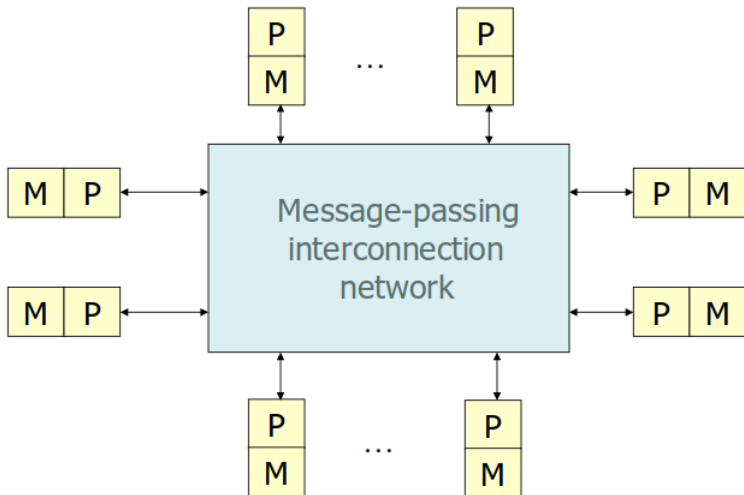


# Multicomputer Models

- Multicomputers consist of multiple computers, or nodes interconnected by a message passing network
- Each node is autonomous, with its own processor and local memory and sometimes local peripherals
- The message-passing network provides point-to-point static connections among the nodes
- Local memories are not shared, so traditional multicomputers are sometimes called no-remote-memory-access (or NORMA) machines
- Inter-node communication is achieved by passing messages through the static connection network



# Generic Message Passing Multicomputer





# Multicomputer Generations

- Each multicomputer uses routers and channels in its interconnection network, and heterogeneous systems may involve mixed mode types and uniform data representation and communication protocols
- First generation: hypercube architecture, software-controlled message switching, processor boards
- Second generation: mesh-connected architecture, hardware message switching, software for medium-grain distributed computing
- Third generation: Fine-grained distributed computing, with each VLSI chip containing the processor and communication resources.



# Some Early Commercial Multicomputer Systems

System Features	Intel Paragon XP/S	nCUBE/2 6480	Parsys 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.



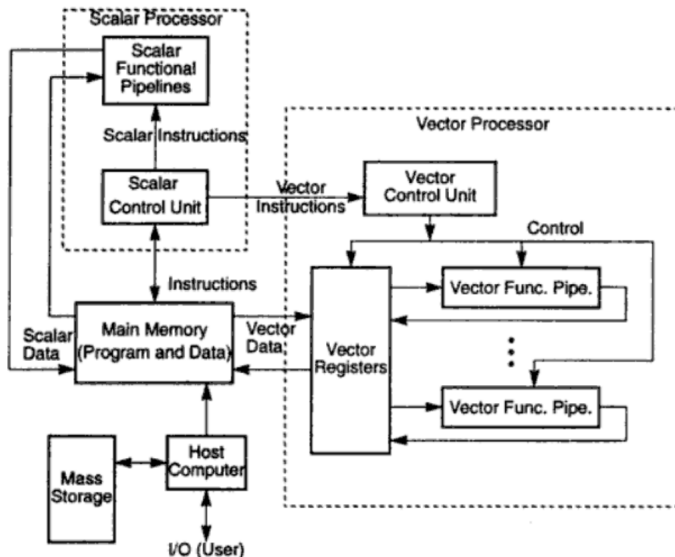
# Multivector and SIMD Computers

## Vector Processors

- Vector Processor Variants
  - Vector Superscomputers
  - Attached Processors
- Vector Processor Models / Architecture
  - Register-to-register architecture (vectors are retrieved from registers)
  - Memory-to-memory architecture (vectors are retrieved from memory)
- Representative Systems
  - Cray-I
  - Cray Y-MP (2, 4 or 8 processors with 16Gflops peak performance)
  - Convex C1, C2, C3 series (C3800 family with 8 processors, 4 GB main memory, 2 Gflops peak performance)
  - DEC VAX 9000 (pipeline chaining support)



# Architecture of Vector Supercomputer - register-to-register



# Representative Vector Supercomputers

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 support 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 x 64 x 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 runs with 2, 4, or 8 processors, 2.67 Gflop peak with Y-MP8256. C-90 has 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.



# Multivector and SIMD Computers

## SIMD Supercomputers

- SIMD Machine Model

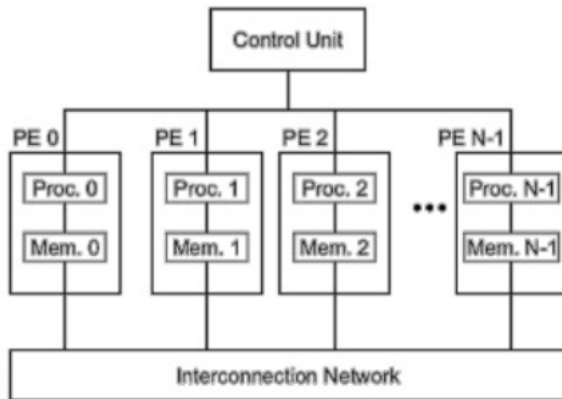
- $S = \langle N, C, I, M, R \rangle$
- N: No of PEs in the machine
- C: Set of instructions (scalar/program flow) directly executed by control unit
- I: Set of instructions broadcast by CU to all PEs for parallel execution
- M: Set of masking schemes (to know which processors are active)
- R: Set of data routing functions

- Representative Systems

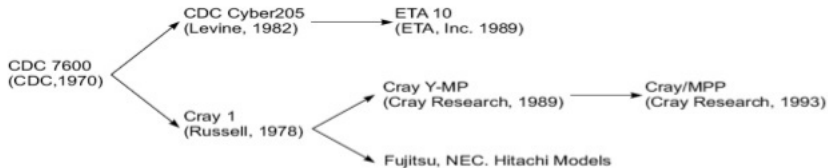
- MasPar MP-1 (1024 to 16384 PEs)
- CM-2 (65536 PEs)
- DAP600 Family (upto 4096 PEs)
- Illiac-IV (64 PEs)



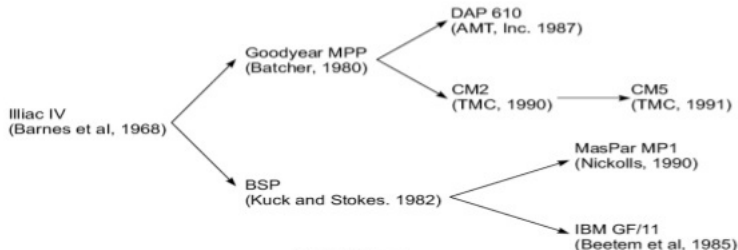
# Operational Model of SIMD Supercomputers



# History of Multivector and SIMD tracks



(a) Multivector track



(b) SIMD track



# Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers
- Multivector and SIMD Computers
- **Conditions of Parallelism**



# Conditions of Parallelism

- Data, Control and Resource Dependences
- Hardware and Software Parallelism
- Role of Compilers



# Conditions of Parallelism

- **Data, Control and Resource Dependences**
- Hardware and Software Parallelism
- Role of Compilers



# Data Dependences

- Flow Dependence
- Antidependence
- Output Dependence
- I/O Dependence
- Unknown Dependence



# Data Dependences

- Flow Dependence

S1: Load R1, A

S2: Add R2, R1

**S2 is flow dependent on S1**

- Antidependence
- Output Dependence
- I/O Dependence
- Unknown Dependence



# Data Dependences

- Flow Dependence
- Antidependence

S2: Add R2, R1

S3: Mov R1, R3

**S3 is anti-dependent on S2**

- Output Dependence
- I/O Dependence
- Unknown Dependence



# Data Dependences

- Flow Dependence
- Antidependence
- Output Dependence
- I/O Dependence

**If same I/O device (eg:- file) is involved in two statements, the statements are said to be I/O dependent.**

- Unknown Dependence



# Data Dependences

- Flow Dependence
- Antidependence
- Output Dependence
- I/O Dependence
- Unknown Dependence
  - The subscript of a variable is itself subscribed
  - Subscript does not contain the loop index variable
  - Variable appears more than once with subscripts having different coefficients of loop variable
  - Subscript is nonlinear in the loop index variable





# Control Dependence

- Loop constructs

- Control Independent

```
DO 20 K = 1,N  
    A(K) = C(K)  
    IF (A(K).LT.0) A(K) = 1  
20 CONTINUE
```

- Control Dependent

```
DO 30 K = 1, N  
    IF (A(K-1).EQ.0) A(K) = 0  
30 CONTINUE
```



# Resource Dependence

Concerned with conflicts in using shared resources

Shared resources - Integer units, Floating-point units, Registers, Memory areas, ALU etc

Eg:- In *ALU dependence*, conflicting resource is ALU



# Resource Dependence: Bernstein's Condition 1966

If  $P_1$  and  $P_2$  are to be parallelizable processes, whose inputs are  $I_1$  and  $I_2$  respectively and outputs are  $O_1$  and  $O_2$ ,

The following Bernstein conditions should be met

- $I_1 \cap O_2 = \emptyset$
- $I_2 \cap O_1 = \emptyset$
- $O_1 \cap O_2 = \emptyset$



# Resource Dependence: Hardware Parallelism and Software Parallelism

Hardware Parallelism: defined largely by machine architecture and hardware multiplicity

Involves cost and performance tradeoffs

Software Parallelism: function of algorithm, programming style and program design



# Resource Dependence : Compiler techniques

Compiler techniques used to exploit hardware features to improve performance.

Compiling for multiprocessors much more challenging than for uniprocessors.

Granularity and Communication latency significant in code optimization and scheduling process.

