Computer Organization & Architecture

Unit-5

Parallel Processors

Parallel processing challenges - Flynn's classification- SISD, MIMD, SIMD, SPMD and vector architectures.

* Multiprocessor - a computer system with at least 2 processors.

* Instruction Level Parallelism: many instructions in code that do not depend on each other. It is possible to execute those instructions in parallel (task-level parallelism)

* Parallel Processing Program: a single program that runs on multiple programs simultaneously.

* Parallel Processing Challenges

A. The difficulty lies not in the hardware side - but rather in the software side - It is difficult to write software that uses multiple processors to complete one task faster, becoming increasingly more difficult as the no. of processors increases.

→ Parallel processing programs are so much harder to develop than sequential programs, because one must get better performance/ energy efficiency by using the parallel processing program, otherwise one could simply use a much simpler sequential program.

B. Performance Issues

→ In some situations it is not get the parallel processing to run faster than sequential programs. This is because of:

(i) Scheduling – must balance load and share system resources effectively.

(ii) Portioning the work into parallel pieces : divide task equally to all the processas

(iii) Load Balancing : work load must be distributed uniformly and the execution time must also be equal

(iv) Synchronization Time & Overhead

C. Speed Up Challenges

Amdahl's Law : In a program with parallel processing, a relatively few instructions that have to be performed in sequence will have a limiting factor on the program speed up such that adding more processors may not make the program run faster.

(OR)

Challenge1 : Limited parallelism

→ Limited parallelism refers to the maximum no. of tasks or computations that can effectively be performed in parallel. When the no. of tasks exceeds the available parallelism, some tasks have to wait for the others to finish before they can be processed, leading to inefficient resource utilization.

→ This limited parallelism is directly defined by Amdahl's law, wherein the speedup of a task is limited by the proportion of the task that cannot be parallelized, i.e even if we parallelize a portion of the task, there will always be some sequential portion that cannot be parallelized, limiting the overall speedup.

According to Amdahl's law:

$$\text{Speedup} = \frac{1}{(1-P) + \left(\frac{P}{n}\right)}$$

P = proportion of task that can be parallelized.

n = no. of processors or computing units available

Challenge 2 : Long latency to remote memory

Latency = time it takes for a request / data transfer to travel from one point to another.

→ In parallel processing, it is crucial to share data efficiently among different processors.

→ However, when data needs to be accessed from remote memory, it can lead to long latencies

→ This delay impacts the overall performance, as processors have to wait for data to arrive ⇒ idle time ⇒ reduced efficiency.

Getting good speedup while keeping problem size fixed is harder than good speedup by increasing problem size

Strong scaling: speedup is measured, while keeping the problem size fixed.

Weak Scaling: problem size grows proportionally to the increase in the number of processors

* Flynn's Classification

→ a popular taxonomy of computer architecture

→ based on the notion of a stream of information. Two types of information flow into a processor: instructions and data

(i) Instruction Stream - defined as the sequence of instructions executed by the processing unit

(ii) Data Stream - defined as the sequence of data including inputs, partial or temporary results, called by the instruction stream.

→ According to Flynn's classification, either the instruction or data stream can be single or multiple, and the categorization is as

(i) Single Instruction Stream, Single-Data Streams (SISD)

(ii) Single Instruction Stream, Multiple Data Streams (SIMD)

(iii) Multiple Instruction Stream, Single Data Streams (MISD)

(iv) Multiple Instruction Stream, Multiple data Streams (MIMD)

	Instruction Stream	
	Single	Multiple
Data Stream: Single	SISD	MISD
Data Stream: Multiple	SIMD	MIMD

A. SISD Computers

→ a uniprocessor machine, capable of executing a single instruction, operating on a single data stream

single instruction - only one instruction is being acted on by the CPU during any one clock cycle

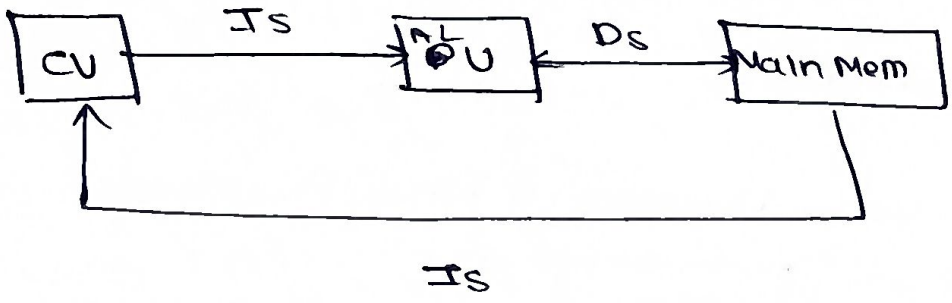
single data : only one data stream is being used as input during any one clock cycle

→ conventional single-processor Von-Neumann computers are classified as SISD systems.

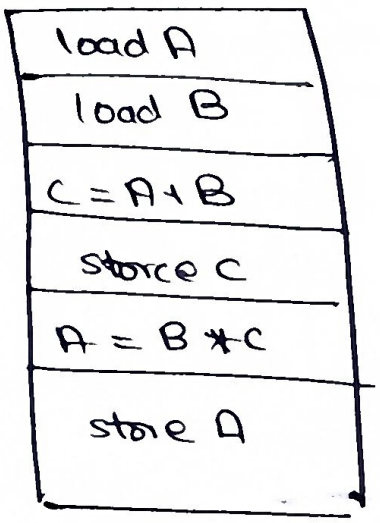
→ Instructions are executed sequentially, but may be overlapped in their execution stages (pipelining)

→ SISD computers may have more than one functional unit, all under the supervision of the control unit.

$I_S = D_S = 1$



Example



B. SIMD Computers

→ An SIMD system is capable of executing the same instruction on all the CPUs but operating on different data streams

Single Instruction: All processing units execute the same instruction at any given clock cycle.

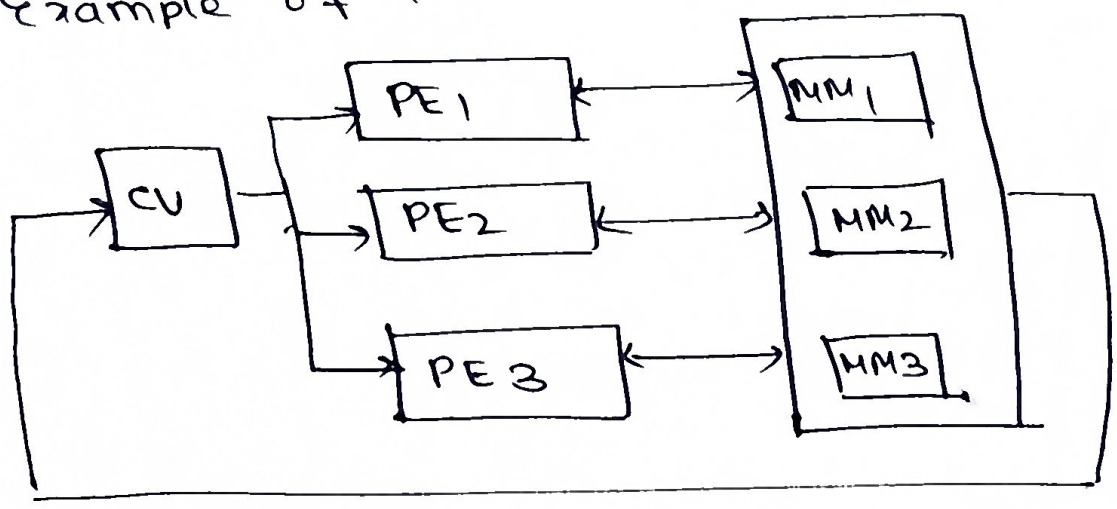
Multiple Data: Each processing unit can operate on a different data element.

→ SIMD's computer has a single CU which issues one instruction at a time but it has multiple ALU's to carry out on multiple data sets simultaneously.

→ ∴ All processing elements simultaneously execute the same instruction and are said to be "lock-stepped" together

→ Each processor must be allowed to complete its instruction before the next instruction is taken for execution. Thus, the execution of instructions is synchronous.

Example of SIMD = Vector Processor & Array Processor



Example

P1	P2	P3
load A(1)	load A(2)	load A(3)
load B(1)	load B(2)	load B(3)
C(1) = A(1) * B(1)	C(2) = A(2) * B(2)	C(3) = A(3) * B(3)
store C(1)	store C(2)	store C(3)

* Advantages of SIMD

→ amortize the cost of CU over dozens of execution units

→ reduced instruction bandwidth and space

→ SIMD needs only one copy of the code that is being simultaneously executed.

→ SIMDs works best when dealing w/ arrays in for loops because parallelism is achieved by performing the same operation on independent data.

* Disadvantage of SIMD

→ very weak in case of switch statements, where each execution unit must perform a different operation on its data.

c. MISD Computers.

→ In MISD, multiple instructions operate on a single data stream

→ Capable of executing different instructions on different PDs, but all of them operating on the same dataset

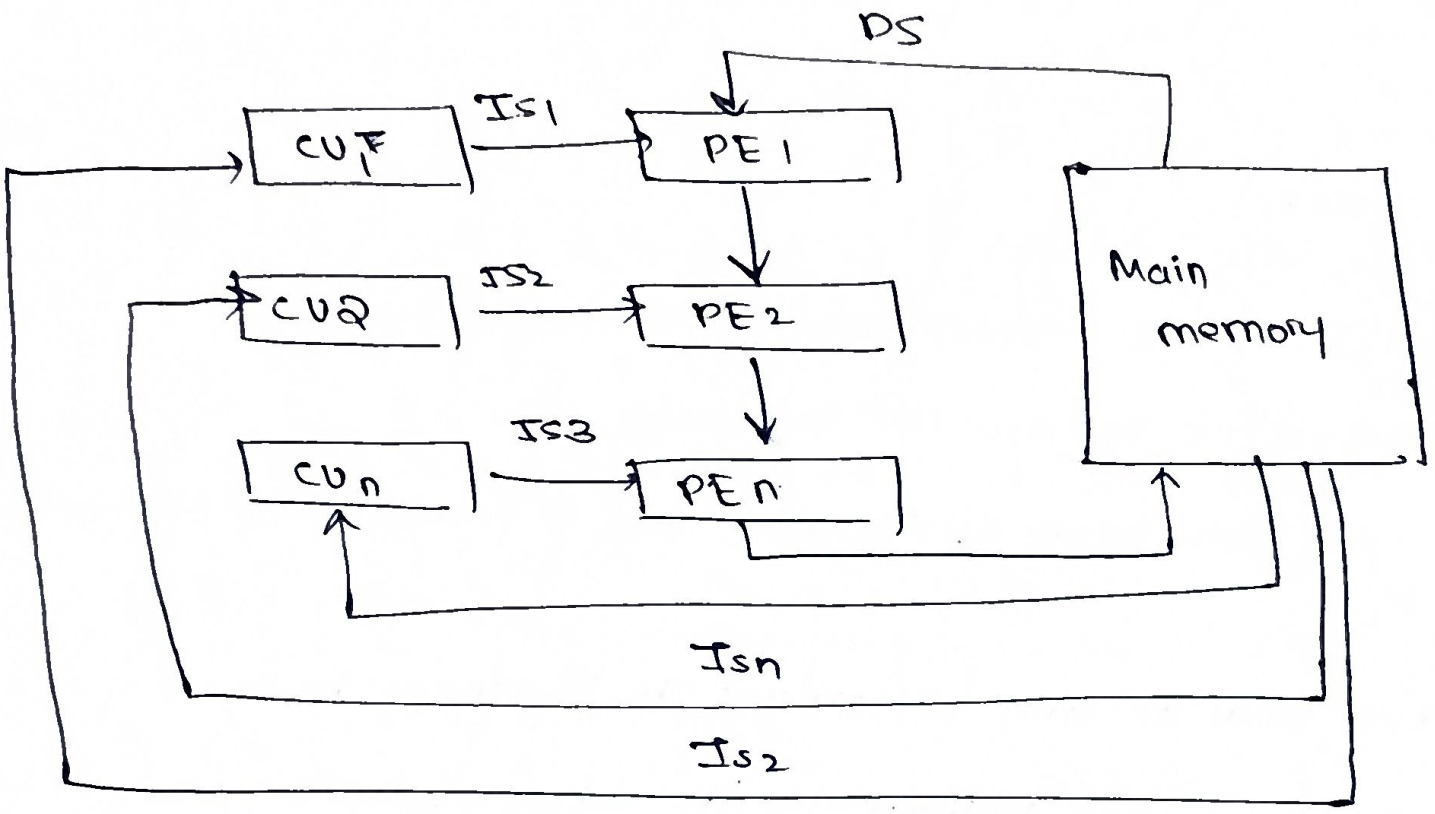
Multiple Instruction: Each processing unit may be executing different instruction stream.

SingleData: Every processing unit operates on the same data element

→ MISD models are not practically useful, non available commercially

→ All processing elements are interacting with the common shared memory for the organization of a single data stream.

→ The only known MISD computer is the C.mmp at CMU.



Example

load A(1)	load A(1)	load A(1)
C(1) = A(1) * 5	c(2) = A(1) * 5	c(n) = A(1) * 30
store c(1)	store c(2)	store c(n)
P_1	P_2	P_n

D. MIMD Computers

→ capable of executing multiple instructions on multiple data sets

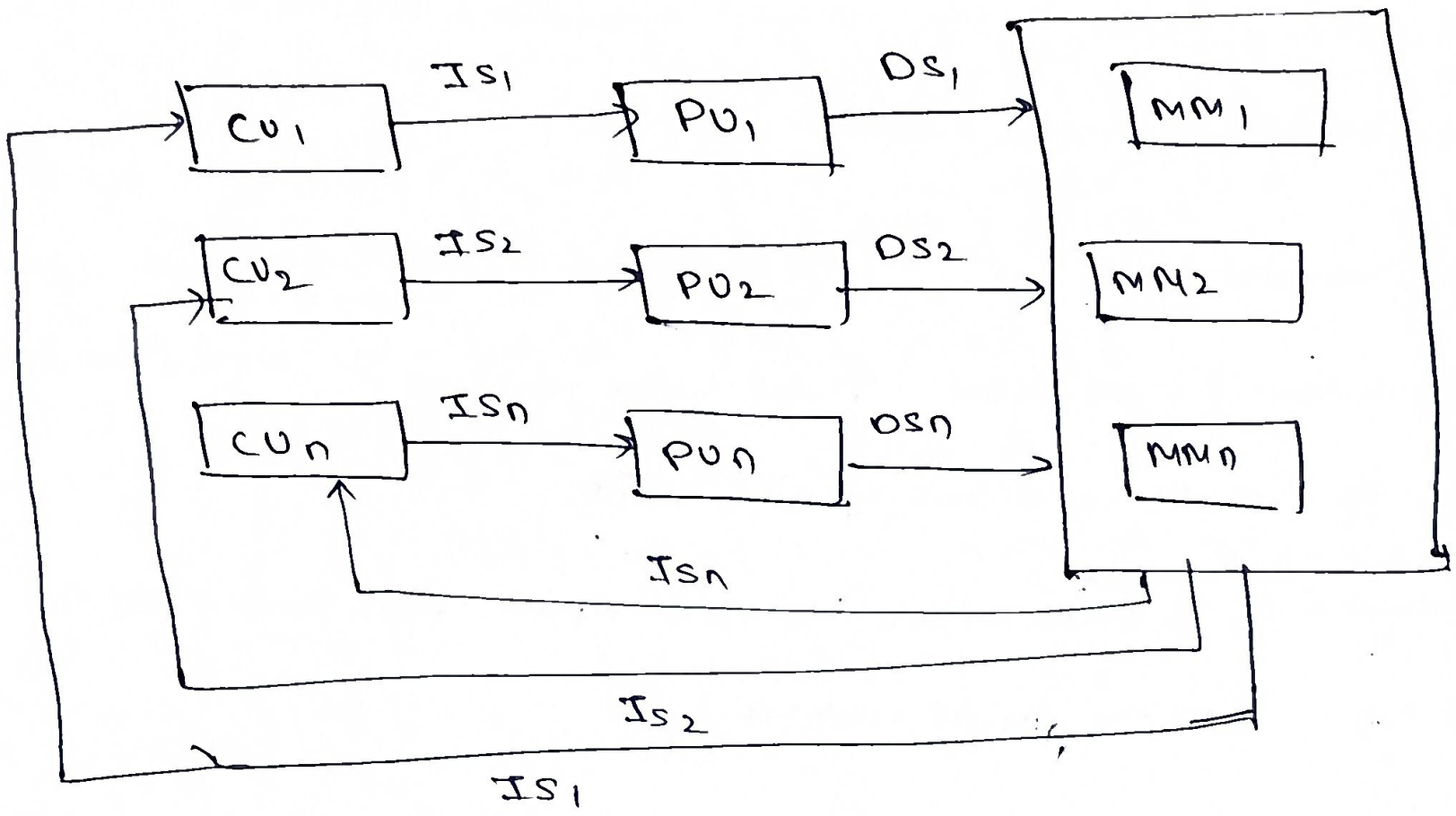
Multiple Instruction : Every processor may be executing a different instruction stream

Multiple Data : Every processor works w/ a diff. data stream.

→ Tasks executed by different processors can start a finish at different times ⇒ run asynchronously

→ This classification actually recoanizes the parallel computer.

MIMD = a parallel computer.



Example

P1	P2	Pn
load A(1)	call func(D)	load A(2)
load B(1)	x = 4 * 2	load B(2)
C(1) = A(1) * B(1)	sum = x * 2	C(2) = A(2) + B(2)
store C(1)	store sum	store C

* Vector Architecture

→ SIMD Processing – single instruction operates on multiple data elements

Time – Space Duality

Array Processor – instruction operates on multiple data elements at the same time using different spaces

Vector Processor - Instruction operates on multiple data elements in consecutive time steps using the same space.

* Working & Basic Requirements of a Vector Processor

→ A vector is a one-dimensional array of numbers

→ A vector processor is one whose instructions operate on vectors rather than single scalar values

Basic Requirements

→ Need to load / store vectors from vector registers

→ Need to operate on vectors of diff. length

→ Elements of a vector might be stored apart from each other in memory (vector stride register)

Stride = distance in memory between 2 elements of a vector

Working

→ best suited for problems w/ lots of data-level parallelism

→ Rather than having 64 ALUs perform 64 additions simultaneously (like array processors), the vector architecture pipelines the ALU to get a better performance.

→ The basic idea of vector architecture is to collect data elements from memory, put them in order into a large set of registers, operate on them sequentially in registers using pipelined execution, & then write results back into memory

* Comparison of Vector to Conventional MIPS Code

→ Vector operations use the same name as MIPS instructions, but with the letter V appended.

eg. addv.d adds 2 double precision vectors

lv, sv : load/store vector

addvs.d : add scalar to each element of vector of double

→ Most importantly, the vector processor greatly reduces the dynamic instruction bandwidth, executing only 6 instrs almost 600 for traditional MIPS architecture for DAXPY

↓

$Y = a \times X + Y$

X and Y are vectors of 64 double precision

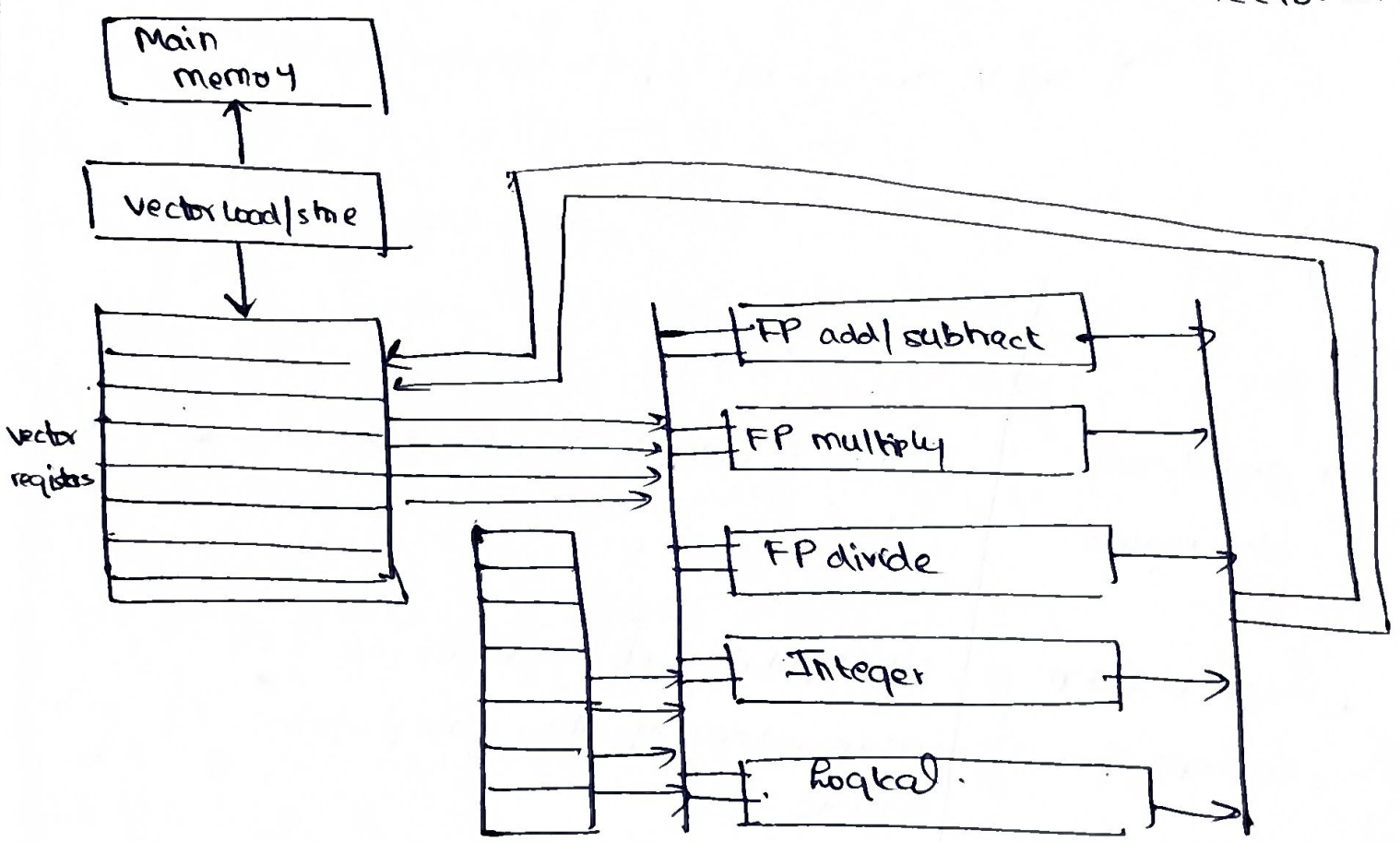
* Vector vs. Scalar

Vector	Scalar
(1) Instruction fetch and decode bandwidth drastically reduced	fetch & decode bandwidth does not reduce
(2) computation of each result in the vector is independent of the computation of other results in the same vector	computation of each result is not independent of the computation of other results
(3) h/w should check for hazards only between 2 vector inst	h/w should check for hazards between every element in array
(4) save energy because of reduced checking	does not save energy

(5) cost of latency to main memory is seen only once for the entire vector	cost of latency to memory is seen for each word of the scalar
(6) Efficient use of memory & inst bandwidth	no efficient use of memory & instruction bandwidth.
(7) Entire loop behaviour is predetermined.	entire loop behaviour is not predetermined.

VMIPs

vectors. multimedia ext & vector lane??



→ loosely based on cray-1

VMIPs instruction set

→ scalar portion is similar to MIPS

→ vector portion is a logical extension of MIPS

Registers

→ has 8 vector registers

→ each vector register holds 64 elements of 64 bits

→ has 16 read & 8 write ports to supplies operands to vector functional units

Vector Functional units

→ each unit is fully pipelined

→ starts a new operation on each clock cycle.

Control Unit detects hazards

The 5 functional units are

FP add/subtract

FP multiply

FP divide

Integer & logical units

Computer Organization and Architecture

Unit - 5

Introduction to Graphics Processing Units

→ Designed specifically for performing the complex mathematical & geometric calculations that are necessary for graphics rendering.

→ major focus for GPUs given by computer game industry - developed faster graphics hardware

→ also called a Visual Processing Unit (VPU)

* Characteristics of GPUs

→ GPUs are accelerators that supplement a CPU, so they need not be able to perform all the tasks of CPU ⇒ dedicate more resources to graphics

→ GPU problem sizes are typically hundreds of megabytes to gigabytes

* Differences between GPUs and CPUs

→ GPUs do not rely on multilevel caches, rather they use hardware multithreading

→ GPU memory is oriented towards bandwidth rather than latency
  → have special graphics DRAM chips
  → have smaller main memories

→ GPUs accommodate many parallel processors (MIMD), i.e it is much more highly multithreaded than a typical CPU.

→ GPUs have higher transistor counts

→ have a faster and more advanced memory interface., as they need to shift araind more data than CPUs.

* Hardware / Software Interface

→ programming Languages were developed to write programs directly for the GPUs.

→ eg. NVIDIA's CUDA (Compute Unified Device Architecture) (allows C programs to execute on GPUs)

→ All forms of parallelism are united through the CUDA thread.

→ The compiler and the hardware can group thousands of CUDA threads to utilize various styles of parallism – multithreading, MIMD, SIMD & ILP.

→ These threads are executed in groups of 32 at a time

* NVIDIA GPU Architecture

→ consists of a collection of multithreaded SIMD processors, i.e a GPU is a MIMD composed of multithreaded SIMD processors

→ A Thread Block Scheduler hardware assigns blocks of threads to the multithreaded SIMD processors

→ The SIMD Thread Scheduler within an SIMD processor schedules when SIMD threads should run.

* NVIDIA GPU Memory Structures

Local Memory: on chip memory that is local to each multithreaded SIMD processor

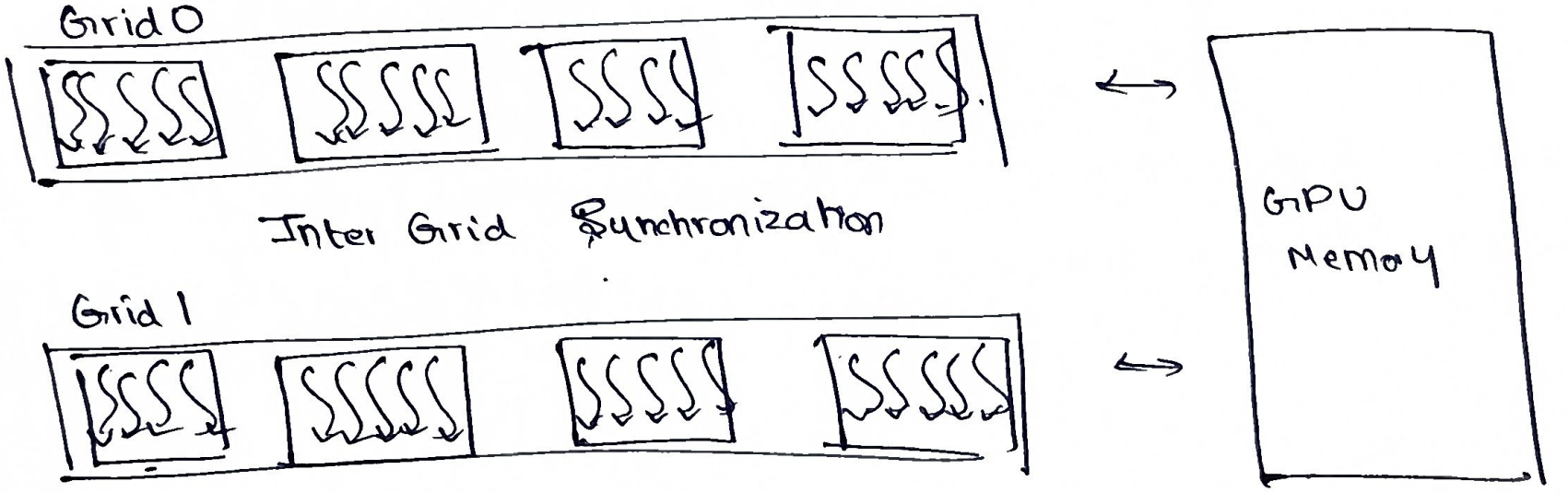
→ shared by all the SIMD lanes within a multithreaded SIMD processor.

GPU Memory : off-chip DRAM shared by the whole GPU and all thread blocks

CUDA Thread

↓ Per-CUDA Thread Private Memory

Thread Block



* Efficiency of GPUs

Efficient For

→ fast parallel floating point ops
→ SIMD ops
→ High Computation per memory access

Not Efficient For

→ double precision
→ Logical operations on integer data
→ branching intensive operations
→ random access, memory intensive operations

Unit - 5

Hardware Multithreading

* **Multithreading** : a mechanism by which the instruction stream is divided into several smaller streams, called threads, that can be executed in parallel

* **Threads** : → include program counter, register state & stack
  → a thread is a lightweight process
  → threads share a single address space

* **Thread Switch** : switching processor control from one thread to another within the same process

* **Process** : includes one or more threads, the address space & the OS state

* **Hardware Multithreading**

→ allows multiple threads to share the functional units of a single processor in an overlapping fashion, to ensure efficient hardware utilization

* **Types of Hardware Multithreading**

(i) Fine-Grained multithreading

(ii) Coarse-Grained Multithreading

A. **Fine-Grained Multithreading**

→ switch between threads on each execution, resulting in interleaved execution

→ interleaving is done in a round robin fashion, skipping any threads that are stalled at that clock cycle.

Advantage

→ can hide throughput losses arising from both short and long stalls, since instructions from other threads can be executed when one thread stalls.

Disadvantages

→ slows down the execution of individual threads, since a thread that is ready to execute without stalls will be delayed by instructions from other threads.

B. Coarse-Grained Multithreading

→ switches threads only on costly stalls.

→ relieves the need to have switching be extremely fast

Advantage

→ less likely to slow down an individual thread

Disadvantage

→ limited in ability to overcome throughput losses

→ arises due to pipeline start-up costs

→ A processor w/ coarse-grained multithreading issues instructions from a single thread, when a stall occurs, the pipeline must be emptied or frozen.

→ The new thread that begins executing after the stall must fill the pipeline before instructions can complete - causes start up overhead

∴ coarse grained multithreading is much more useful for reducing the penalty of high-cost stalls, where pipeline refill is negligible compared to the stall time.

* Simultaneous Multithreading?

→ a variation of hardware multithreading that uses the resources of a multiple issue, dynamically scheduled pipelined processor to exploit thread-level parallelism.

→ multiple-issue processors have more functional unit parallelism than most single threads

→ SMT relies on existing dynamic mechanisms, does not switch resources every cycle.

* Multicore and other Shared Memory Multiprocessors

Multicore : more than one processor available within a single chip

Multiprocessor : a computer system with at least 2 processors.

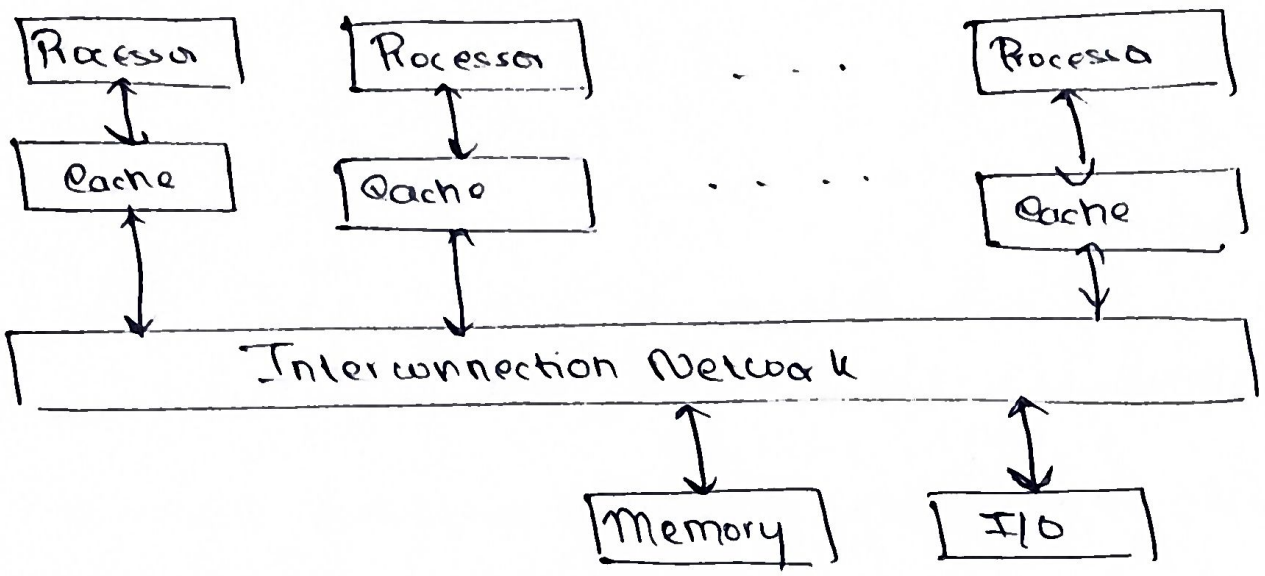
A. Shared Memory Multiprocessor (SMP)

→ offers the programmer a single physical address space across all processors.

→ processors communicate through shared variables in memory

→ Systems can still run independent jobs in their own virtual address space, even if they all share a physical address space

→ Shared data usage is coordinated via synchronization primitives (locks) that allow access to data only one processor at a time.



Memory Access

* ### Uniform Memory Access

→ main memory is uniformly shared by all processors, and each processor has equal access time to the shared memory.

→ used for time-sharing applications in a multi-user environment

→ used w/ tightly coupled systems, with a high degree of resource sharing.

(symmetric & asymmetric SMP)

* ### Non-Uniform Memory Access

→ some memory accesses are much faster than other.

→ systems differ in how memory & peripheral resources are shared

→ time taken depends on which processor accesses which word

→ harder programming challenges for NUMA, but can scale to larger sizes.

synchronization & lock definition

* Challenges from multicore computing

→ relies on effective exploitation of multiple thread parallelism

→ memory latency

→ fragmentation of L3 cache.

→ require mechanisms for efficient IP comm

(i) synchronization

(ii) mutual exclusion

(iii) context switching

Advantages

→ lesser space on PCB

→ higher throughput

→ consume less power

Applications

→ db servers

→ web servers

→ compilers

→ multimedia applications