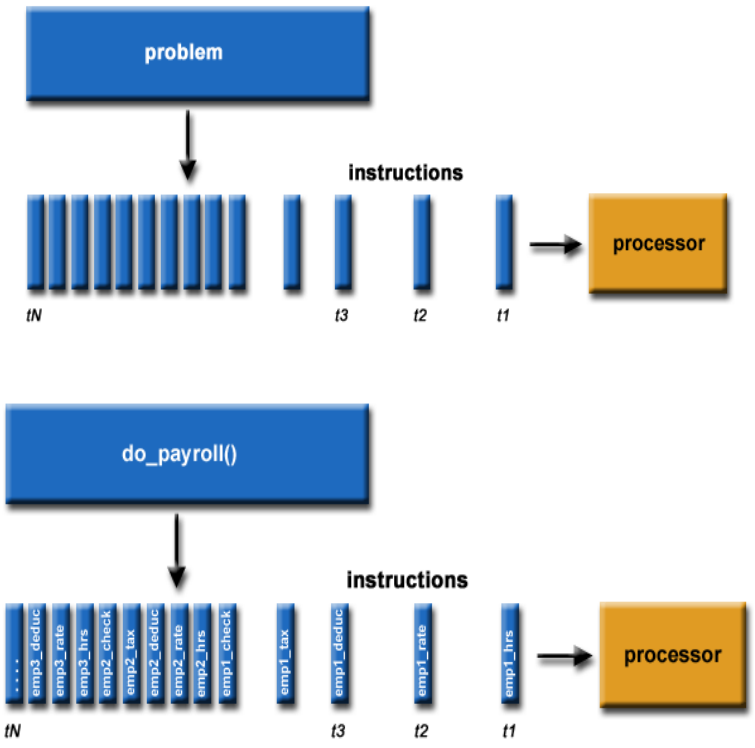
Module 1: Introduction to Parallel Computing

Serial Computing:

Traditionally, software has been written for ***serial*** computation.

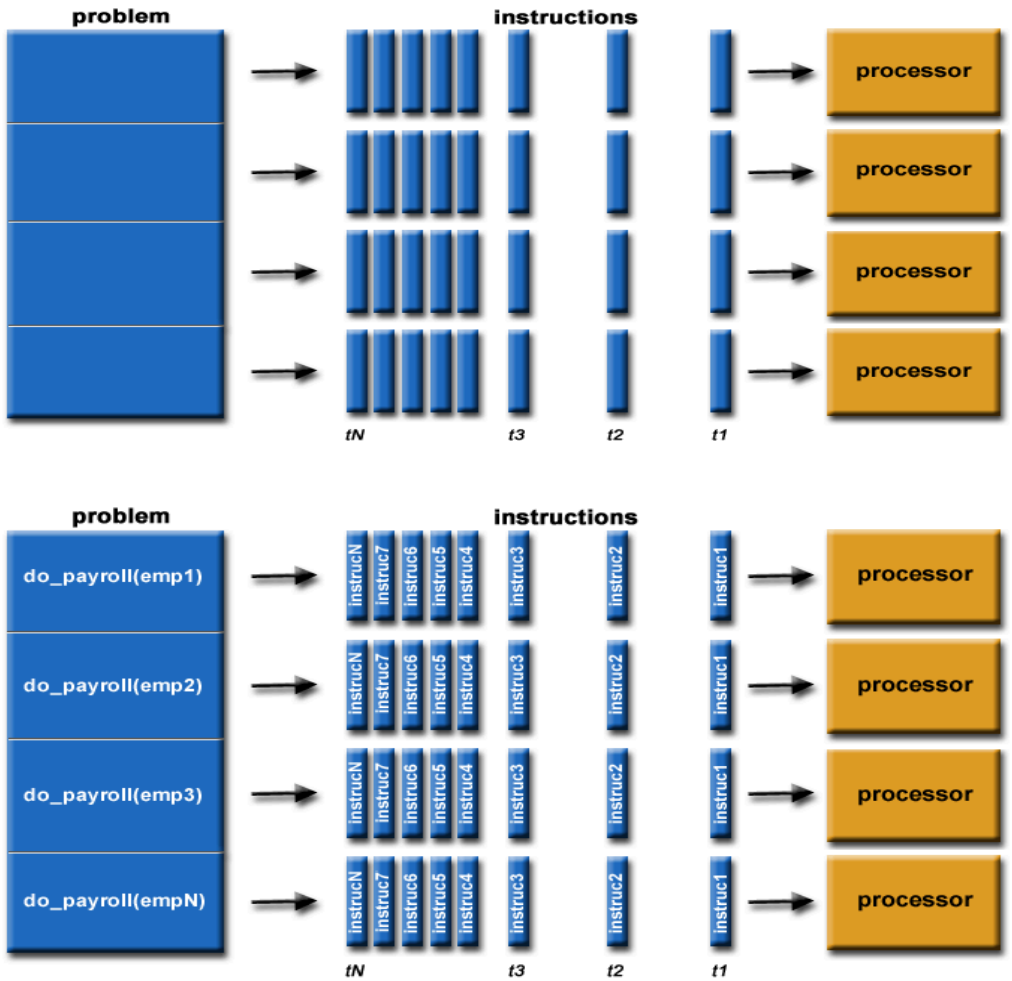
- A problem is broken into a discrete series of instructions
- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time



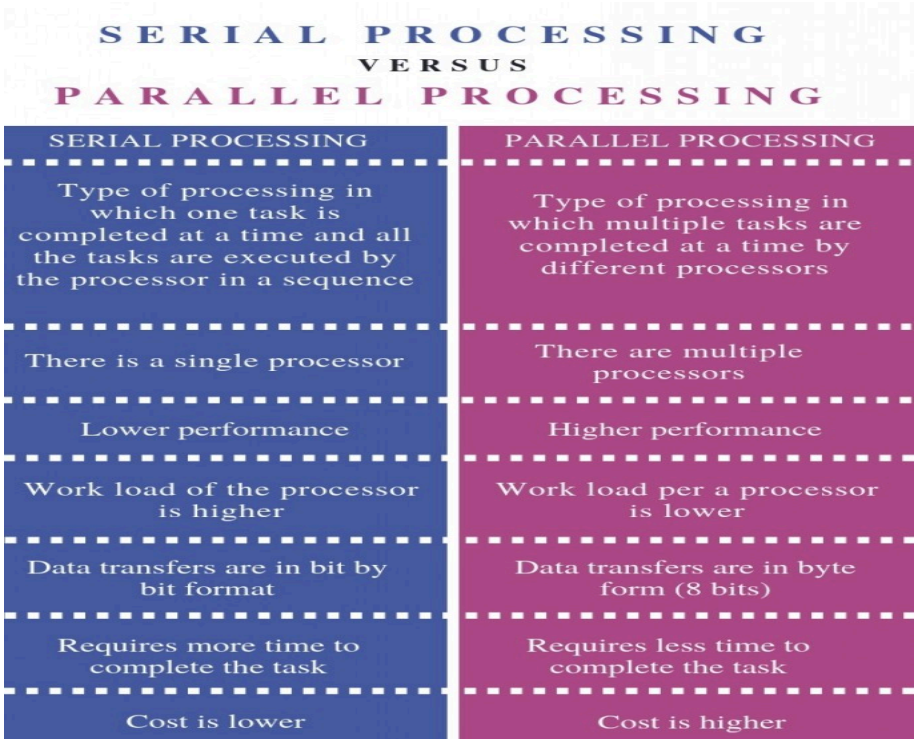
Parallel Computing:

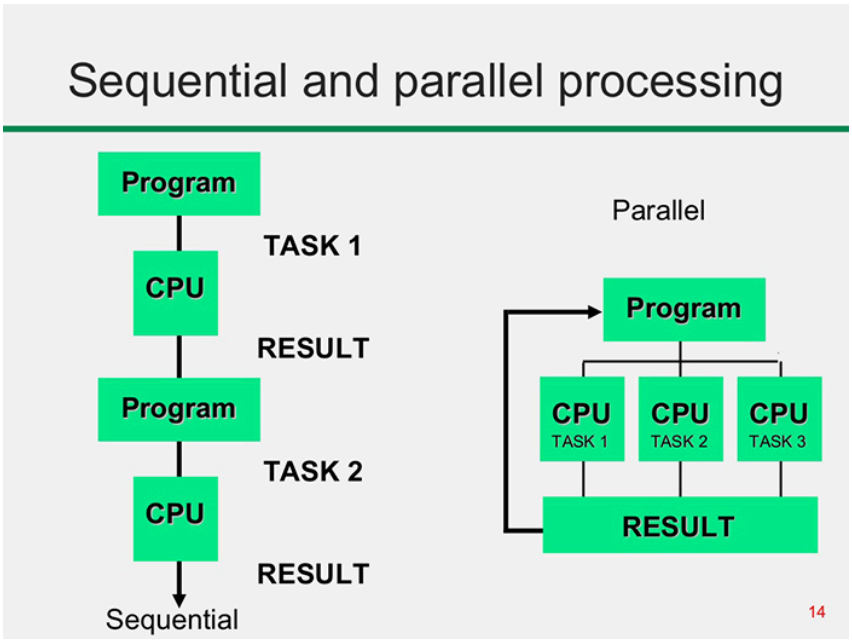
Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.

- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed



Difference between Serial and Parallel Processing:





Fundamental limits on Serial Computing: Three "Walls"

1. Power Wall (The Computational Power Argument – from Transistors to FLOPS):

Increasingly, microprocessor performance is limited by achievable power dissipation rather than by the number of available integrated-circuit resources (transistors and wires).Thus, the only way to significantly increase the performance of microprocessors is to improve power efficiency at about the same rate as the performance increase.

2. Frequency Wall(The Data Communication Argument):

Conventional processors require increasingly deeper instruction pipelines to achieve higher operating frequencies.

This technique has reached a point of diminishing returns, and even negative returns if power is taken into account.

3. Memory Wall(The Memory/Disk Speed Argument):

On multi-gigahertz symmetric processors --- even those with integrated memory controllers --- latency to DRAM memory is currently approaching 1,000 cycles.

As a result, program performance is dominated by the activity of moving data between main storage (the effective-address space that includes main memory) and the processor.

Why is Parallel Computing Important?

1. THE REAL WORLD IS MASSIVELY PARALLEL

- In the natural world, many complex, interrelated events are happening at the same time, yet within a temporal sequence.
- Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real world phenomena.

Example: Weather Prediction(one of the 1989 Grand Challenges to Computational Science Categories)

Atmosphere is divided into 3D cells; Data includes temperature, pressure, humidity, wind speed and direction, etc

–Recorded at regular time intervals in each cell ;There are about 5×10^3 cells of 1 mile cubes.

Calculations would take a modern computer over 100 days to perform calculations needed for a 10 day forecast (Then)

2. SAVE TIME AND/OR MONEY:

- In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings.
- Parallel computers can be built from cheap, commodity components.

3. SOLVE LARGER / MORE COMPLEX PROBLEMS:

- Many problems are so large and/or complex that it is impractical or impossible to solve them using a serial program, especially given limited computer memory.
- Example: "Grand Challenge Problems" requiring petaflops and petabytes of computing resources. Web search engines/databases processing millions of transactions every second.

4. PROVIDE CONCURRENCY:

- A single compute resource can only do one thing at a time. Multiple compute resources can do many things simultaneously.
- Example: Collaborative Networks provide a global venue where people from around the world can meet and conduct work "virtually".

5. TAKE ADVANTAGE OF NON-LOCAL RESOURCES:

- Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient.
- Example: SETI@home (setiathome.berkeley.edu) has over 1.7 million users in nearly every country in the world. (May, 2018).

6. MAKE BETTER USE OF UNDERLYING PARALLEL HARDWARE:

- Modern computers, even laptops, are parallel in architecture with multiple processors/cores.
- Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc.
- In most cases, serial programs run on modern computers "waste" potential computing power.

Who is Using Parallel Computing? (Scope of Parallel Computing)

1. Applications in Engineering and Design:

- Design of airfoils (optimizing lift, drag, stability)
- Internal combustion engines (optimizing charge distribution, burn),
- High-speed circuits (layouts for delays and capacitive and inductive effects)
- Structures (optimizing structural integrity, design parameters, cost, etc.)
- Design of microelectromechanical and nanoelectromechanical systems (MEMS and NEMS)

2. Scientific Applications:

- Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
- Bioscience, Biotechnology, Genetics
- Chemistry, Molecular Sciences
- Geology, Seismology
- Applications in astrophysics have explored the evolution of galaxies, thermonuclear processes,and the analysis of extremely large datasets from telescopes.
- Weather modeling, mineral prospecting, flood prediction, etc.

2. Commercial Applications:

- "Big Data", databases, data mining
- Web search engines, web based business services
- Financial and economic modeling
- Management of national and multinational corporations
- Advanced graphics and virtual reality, particularly in the entertainment industry
- Networked video and multimedia technologies
- Collaborative work environments

4. Applications in Computer Systems:

- Network intrusion detection
- Cryptography
- Embedded systems

- Artificial Intelligence

Why is Parallel Computing Hard?(Limitations)

1. Algorithm development is harder

- The algorithms must be managed in such a way that they can be handled in the parallel mechanism.
- The algorithms or program must have low coupling and high cohesion. But it's difficult to create such programs.
- complexity of specifying and coordinating concurrent activities

2. Software development is much harder

- Lack of standardized & effective development tools, programming models, and environments
- It addresses things such as communication and synchronization between multiple sub-tasks and processes which are difficult to achieve.

3.Rapid pace of change in computer system architecture

- Today's hot parallel algorithm may not be suitable for tomorrow's parallel computer!
- More technically skilled and expert programmers can code a parallelism based program well.

Levels of Parallelism

Bit-level Parallelism:

- This form of parallelism is **based on doubling the processor word size**. Increased bit-level parallelism means faster execution of arithmetic operations for large numbers.
- For example, an 8-bit processor needs two cycles to execute a 16-bit addition while a 16-bit processor just one cycle.
- This level of parallelism seems to have come to an end with the introduction of 64-bit processors.

Instruction-level parallelism (ILP):

- This type of parallelism is trying to exploit the potential overlap between instructions in a computer program.
- Multiple instructions from the **same instruction stream** can be executed concurrently.
- Generated and managed by **hardware** (ex:superscalar) or by **compiler** (ex:VLIW).
- Limited in practice by data and control dependencies.

Instruction-level parallelism (ILP):

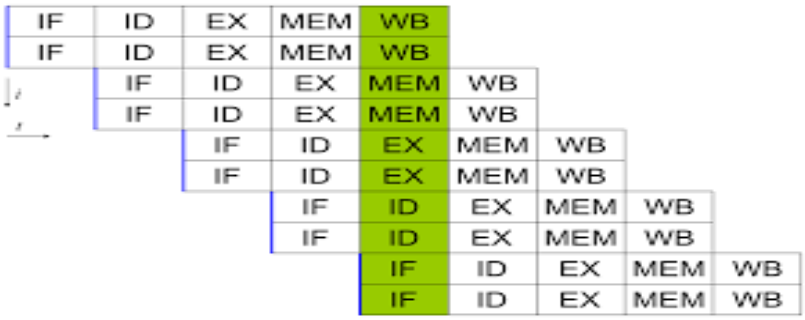
Implemented on hardware level:

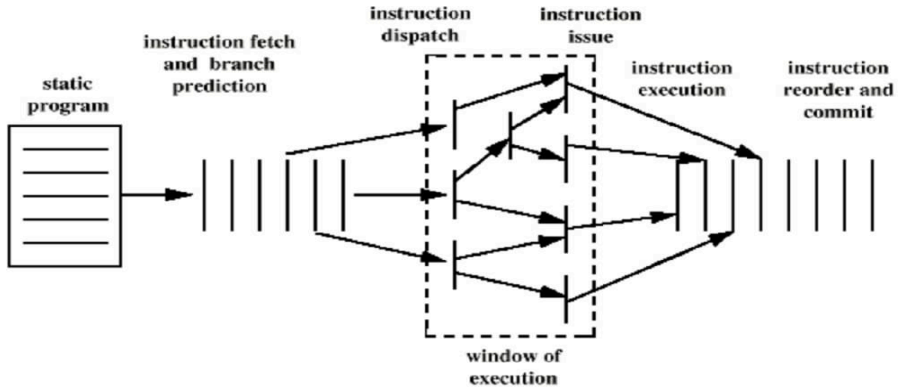
Instruction Pipelining —Execute different stages of different independent Instructions in the same cycle thus making full use of idle resources.

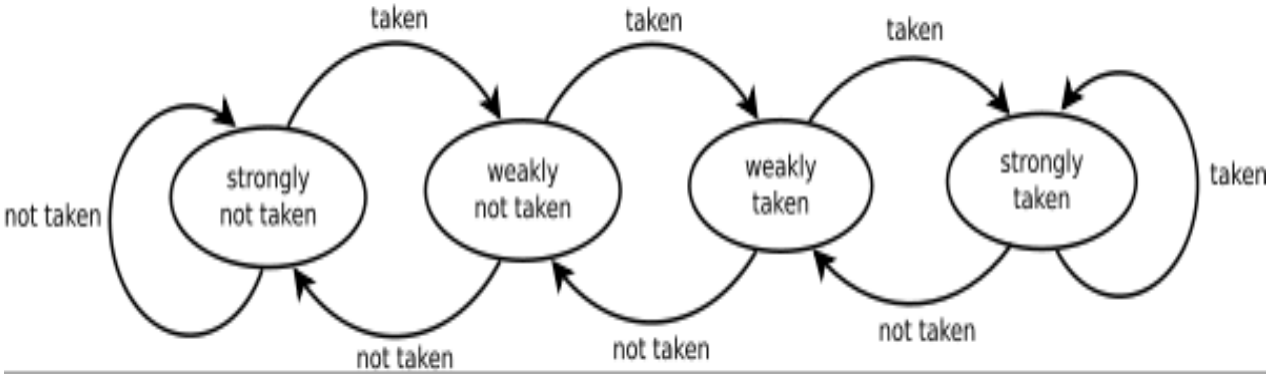
Superscalar Processors — Utilize multiple execution units in a single processor die, thus following instructions can be executed without waiting on complex preceding instructions to finish executing.

Out-of-order execution — Instructions not violating any data dependencies are executed when a unit is available even when preceding instructions are still executed.

Speculative execution / Branch prediction — The processor tries to avoid stalling while control instructions (branches — i.e. if or case commands) are still resolved by executing the most probable flow of the program.

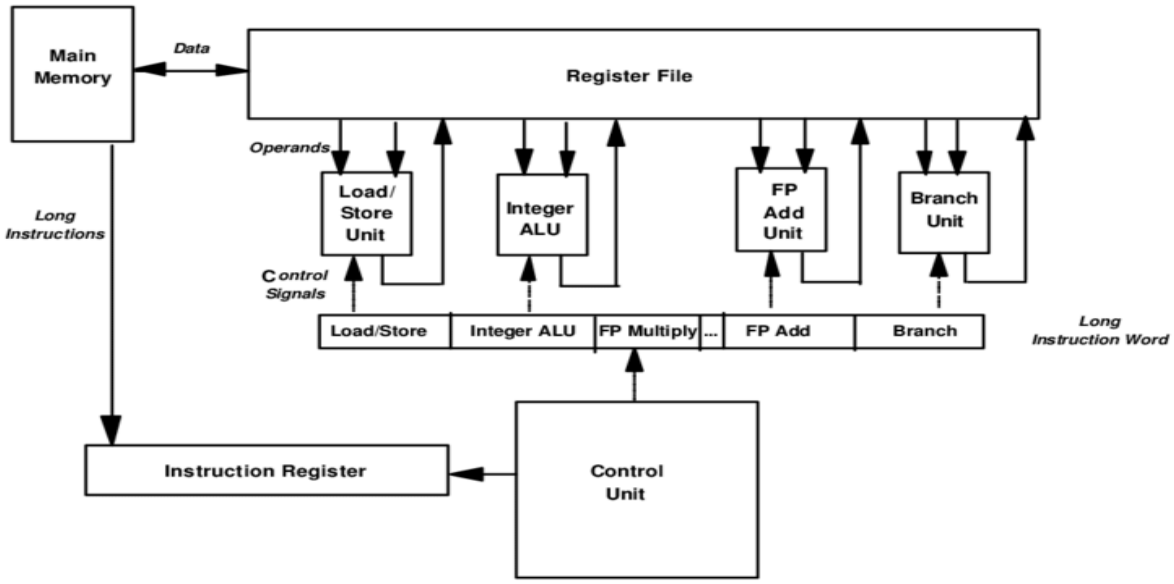






Realized on software level:

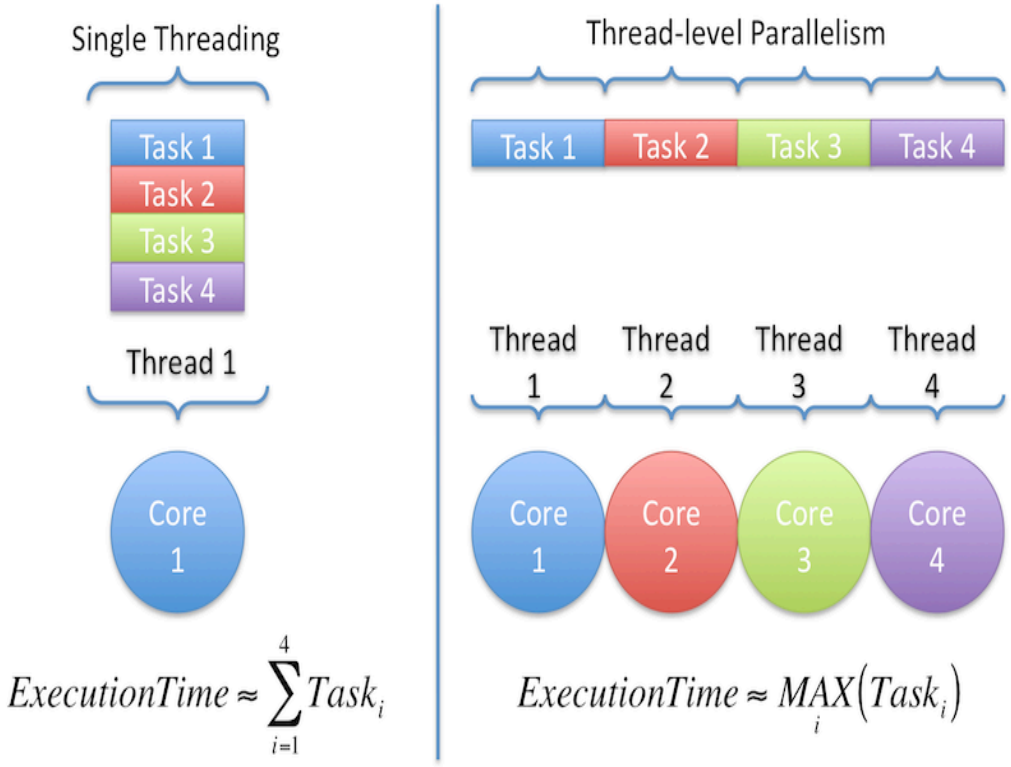
Using specialized compilers for *Very long instruction word* (VLIW) processors. VLIW processors have multiple execution units organized in multiple pipelines and require the compilers to prepare the parallel instruction streams to fully utilize their resources.



Note: Most processors use a combination of the above ILP techniques. For example, Intel Pentium series used all of the above techniques to achieve higher performance with each product generation.

Function/ Task / Thread-level parallelism:

- This high-level form of parallel computing is focusing on partitioning the application to be **executed in distinct *tasks* or *threads,*** that can be then executed simultaneously on different computation units.
- Threads can work on independent data fragments or even share data between them.
- Multiple threads or instruction sequences from the same application can be executed concurrently
- Generated by compiler and managed by compiler and hardware
- Limited in practice by communication/ synchronization overheads and by algorithm characteristics



$$ExecutionTime \approx \sum_{i=1}^{4} Task_i$$

$$ExecutionTime \approx \underset{i}{MAX}(Task_i)$$

Data parallelism:

- Instructions from a single stream operate concurrently on **several data**
- Limited by non-regular data manipulation patterns and by memory bandwidth
- Tries to partition the data to different available computation units instead.
- The cores execute the *same* task code over the data assigned to each.
- Data parallelism is the only available option for high-level parallelism in computer graphics because the graphic processor units (GPUs) are designed to execute each graphic processing task as fast as possible by partitioning each frame in regions.
- The task on command is then executed independently on each data region by their hundreds of processing units.

Data Parallelisms	Task Parallelisms
1. Same task are performed on different subsets of same data.	1. Different task are performed on the same or different data.
2. Synchronous computation is performed.	2. Asynchronous computation is performed.
3. As there is only one execution thread operating on all sets of data, so the speedup is more.	3. As each processor will execute a different thread or process on the same or different set of data, so speedup is less.
4. Amount of parallelization is proportional to the input size.	4. Amount of parallelization is proportional to the number of independent tasks is performed.
5. It is designed for optimum load balance on multiprocessor system.	5. Here, load balancing depends upon on the e availability of the hardware and scheduling algorithms like static and dynamic scheduling.

Transaction-level parallelism: Multiple threads/processes from different transactions can be executed concurrently–Limited by concurrency overheads

Memory-level parallelism (MLP): is a term in computer architecture referring to the ability to have pending multiple memory operations, in particular cache misses or translation lookaside buffer(TLB) misses, at the same time.

1.2 Classification Models

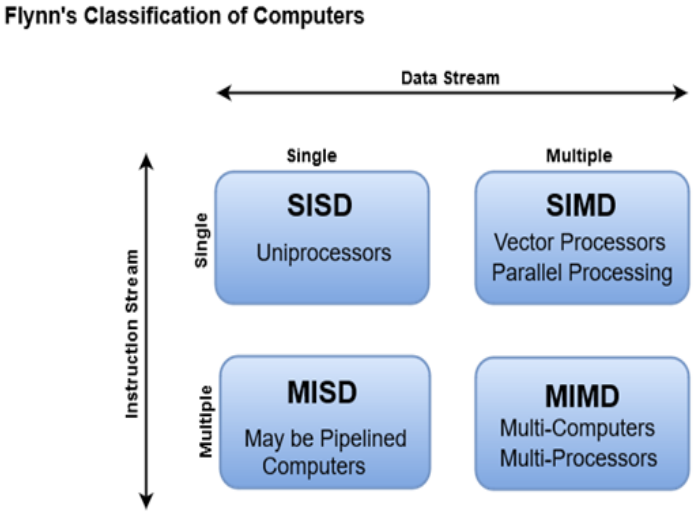
Classification Models based on Architectural Schemes

Architectural Classification schemes

1) Flynn's classification: based on terms of streams of data and instructions
2) Feng's Classification: based on the degree of parallelism
3) Shore's Classification : based on the terms of organization of constituent elements in the computer
4) Handler's Classification: based on the degree of parallelism and pipelining

Flynn's Classification:

- Michael J Flynn classified computers **on the basis of multiplicity of instruction stream and data streams** in a computer system.
- It gives how sequence of instructions or data will be executed upon a processor
- **Instruction stream:** is the sequence of instructions as executed by the machine.
- **Data Stream** is a sequence of data including input, or partial or temporary result, called by the instruction Stream
- Instructions are decoded by the control unit and then ctrl unit send the instructions to the processing units for execution.
- Data Stream flows between the processors and memory bi-directionally.



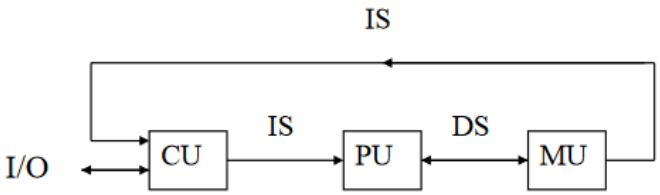
Flynn's Classification scheme

Single-instruction, single-data (SISD) systems –

- An SISD computing system is a uniprocessor machine which is **capable of executing a single instruction, operating on a single data stream.**
- In SISD, machine instructions are processed in a sequential manner and computers adopting this model are popularly called **sequential computers.**
- Most conventional computers have SISD architecture. All the instructions and data to be

processed have to be stored in primary memory.

- The speed of the processing element in the SISD model is limited(dependent) by the rate at which the **computer can transfer information internally.**
- Dominant representative SISD systems are IBM PC, workstations.



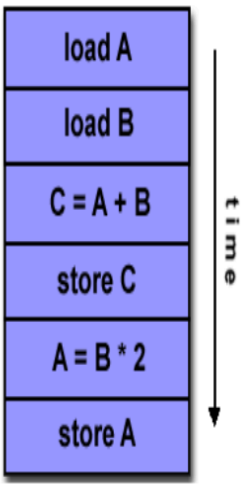
(a) SISD Uniprocessor Architecture

Captions:

CU - Control Unit ; PU – Processing Unit

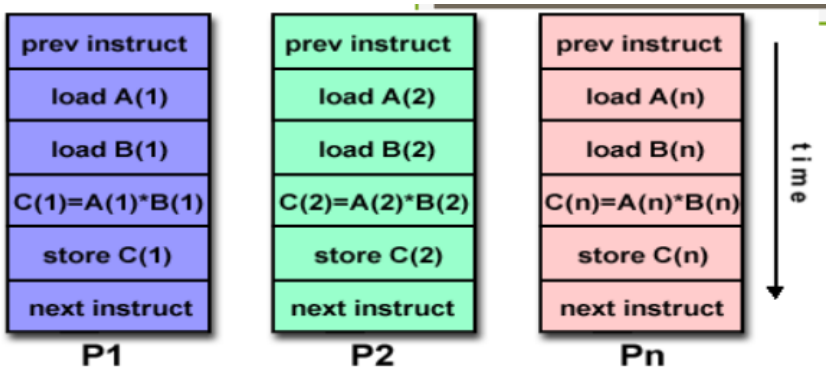
MU – Memory Unit ; IS – Instruction Stream

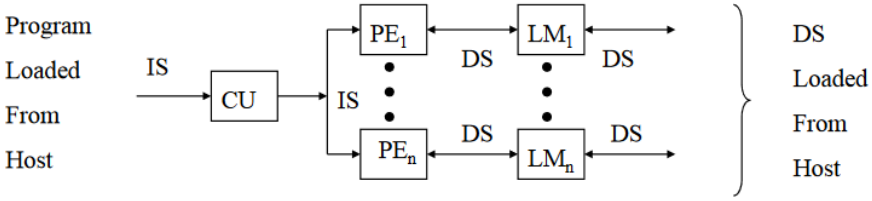
DS – Date Stream



Single-instruction, multiple-data (SIMD) systems –

- An SIMD system is a multiprocessor machine capable of executing the **same instruction on all the CPUs but operating on different data streams.**
- Machines based on an SIMD model are well suited to scientific computing since they **involve lots of vector and matrix operations.**
- So that the information can be passed to all the processing elements (PEs) organized data elements of vectors can be divided into multiple sets(N-sets for N PE systems) and each PE can process one data set.
- Dominant representative SIMD systems is Cray's vector processing machine.





(b) SIMD Architecture (with Distributed Memory)

Captions:

CU - Control Unit ; PU - Processing Unit

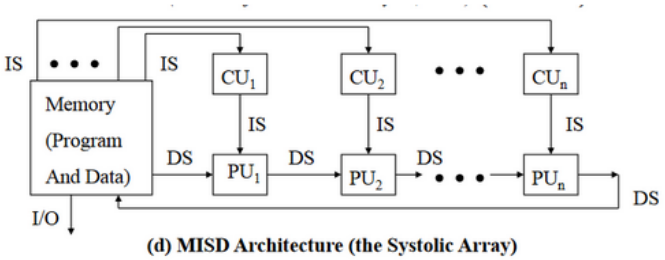
MU - Memory Unit ; IS - Instruction Stream

DS - Date Stream ; PE – Processing Element

LM – Local Memory

Multiple-instruction, single-data (MISD) systems –

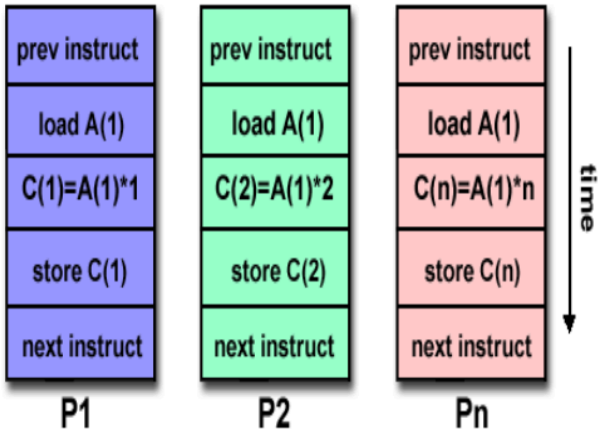
- An MISD computing system is a multiprocessor machine **capable of executing different instructions on different PEs but all of them operating on the same dataset .**
- Example Z = sin(x)+cos(x)+tan(x)
- The system performs different operations on the same data set. Machines built using the MISD model are not useful in most of the application, a few machines are built, but none of them are available commercially.



(d) MISD Architecture (the Systolic Array)

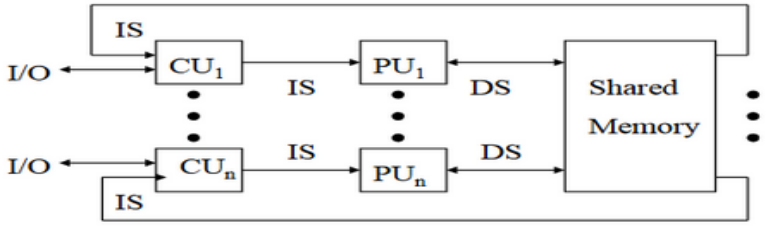
Captions:

CU - Control Unit	;	PU - Processing Unit
MU - Memory Unit	;	IS - Instruction Stream
DS - Date Stream	;	PE – Processing Element
LM – Local Memory		



Multiple-instruction, multiple-data (MIMD) systems –

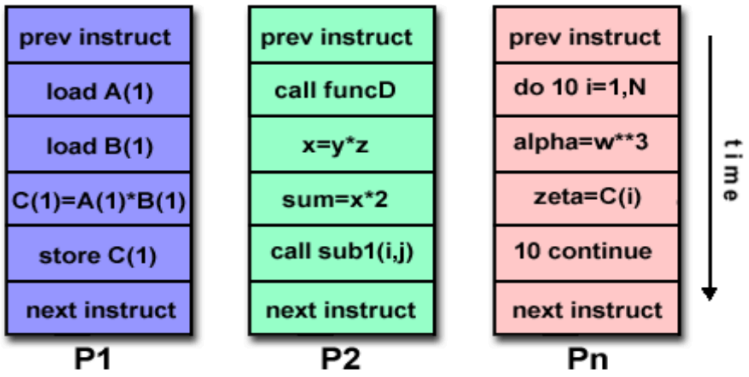
- An MIMD system is a multiprocessor machine which is **capable of executing multiple instructions on multiple data sets.**
- Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application.
- Unlike SIMD and MISD machines, PEs in MIMD machines **work asynchronously.**
- MIMD machines are broadly categorized into **shared-memory MIMD** and **distributed-memory MIMD** based on the way PEs are coupled to the main memory.



(c) MIMD Architecture (with Shared Memory)

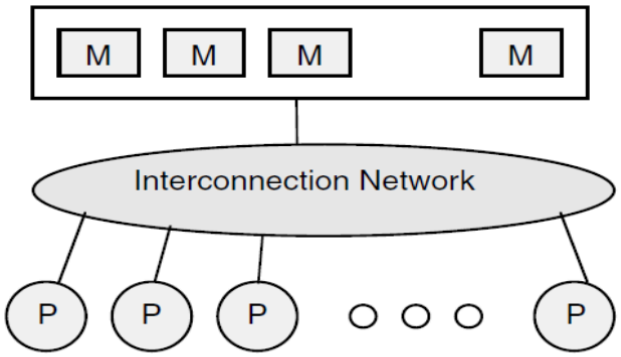
Captions:

CU - Control Unit ; PU - Processing Unit
MU - Memory Unit ; IS - Instruction Stream
DS - Date Stream ; PE – Processing Element
LM – Local Memory



Shared Memory MIMD:

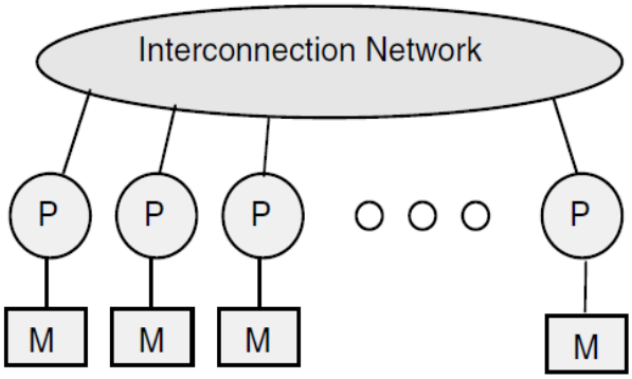
- In the shared memory MIMD model (tightly coupled multiprocessor systems), **all the PEs are connected to a single global memory** and they all have access to it.
- The communication between PEs in this model takes place through the shared memory, modification of the data stored in the global memory by one PE is visible to all other PEs.
- Dominant representative shared memory MIMD systems are Silicon Graphics machines and Sun/IBM's SMP (Symmetric Multi-Processing).



Shared Memory MIMD Architecture

Distributed Memory MIMD:

- In Distributed memory MIMD machines (loosely coupled multiprocessor systems) **all PEs have a local memory.**
- The communication between PEs in this model takes place through the interconnection network (the inter process communication channel, or IPC). It is also known as **message passing system.**
- The network connecting PEs can be configured to tree, mesh or in accordance with the requirement.



Message Passing MIMD Architecture

Difference between Shared Memory MIMD and Distributed Memory MIMD:

- The shared-memory MIMD architecture is **easier to program** but is **less tolerant to failures** and harder to extend with respect to the distributed memory MIMD model.
- Failures in a shared-memory MIMD **affect the entire system**, whereas this is not the case of the distributed model, in which each of the PEs can be easily isolated.
- Moreover, shared memory MIMD architectures are **less likely to scale** because the addition of more PEs leads **to memory contention.** This is a situation that does not happen in the case of distributed memory, in which each PE has its own memory.
- As a result of practical outcomes and user's requirement , distributed memory MIMD architecture is superior to the other existing models.

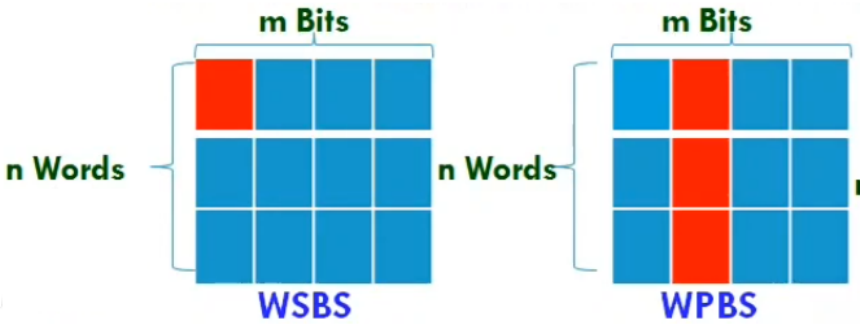
- Tse-yun Feng suggested classification is mainly **based on degree of parallelism to classify parallel computer architecture**.
- The maximum number of binary digits that can be processed per unit time is called maximum parallelism degree P.
- A bit slice is a string of bits one from each of the words at the same vertical position

Feng's Classification scheme

There are four types of processing methods :
word length= n; bit-slice length=m

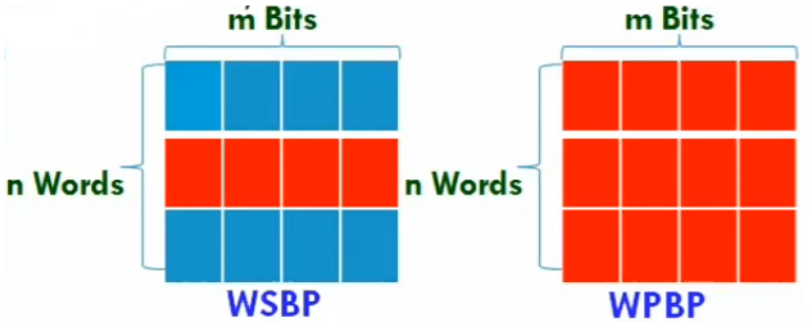
Word–serial and bit-serial(WSBS):has been called as bit –serial processing because one bit (n=m=1) is processed at a time, which was a slow process. This was done in only first generation computers. Example: The "MINIMA"

Word–parallel and bit-serial(WPBS):(n=1,m>1) has been called bis(bit slice) processing because an m-bit slice is processed at a time. Example:STARAN



Word–serial and bit-parallel(WSBP):(n>1,m=1) has been called word-slice processing because one word of n bits is processed at a time. These are found in most existing computers. Example:Burrough 7700

Word –Parallel and bit-parallel(WPBP):(n>1,m>1) is known as fully parallel processing, in which an array of n.m bits is processed at a time. This is the fastest mode of the four .Example:Illiac IV



Mode	Computer model (manufacturer)	Degree of parallelism (n, m)
WSPS	The "MINIMA"	(1, 1)
$n = 1$	(unknown)	
$m = 1$		
WPBS	STARAN	(1, 256)
$n = 1$	(Goodyear Aerospace)	
$m > 1$	MPP	(1, 16384)
(bit-slice processing)	(Goodyear Aerospace)	
	DAP	(1, 4096)
	(ICL, England)	
WSBP	IBM 370/168 UP	(64, 1)
$n > 1$	CDC6600	(60, 1)
$m = 1$	Burrough 7700	(48, 1)
(word-slice processing)	VAX 11/780	(16/32, 1)
	(DEC)	
WPBP	Illiac IV	(64, 64)
$n > 1$	(Burroughs)	
$m > 1$	TI-ASC	(64, 32)
(fully parallel processing)	C.mmp	(16, 16)
	(CMU)	
	S-1	(36, 16)
	(LLNL)	

Shore's Classification scheme

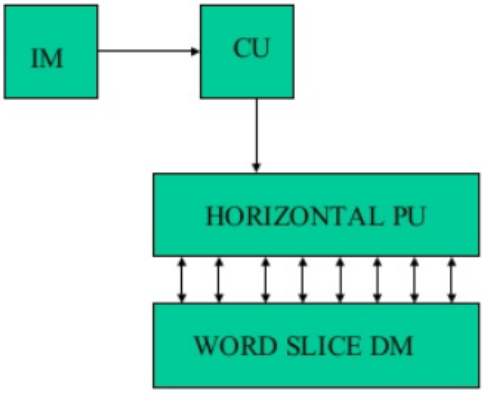
- In this classification computers are classified on the basis of organization of the constituent elements in the computer.
- He proposed 6 machines which are recognized and distinguished by numerical designators.

Machine 1:

- This machine is based on von Neumann architecture with following units:
  1) Control unit 2) Processing unit 3) Instruction memory and data memory
- A single data memory reads the word for parallel processing and generates all bits for that word.
- PU contains two types of functions which may be pipelined or may not.
- As a result of that this machine contains both type of computer namely scalar computer

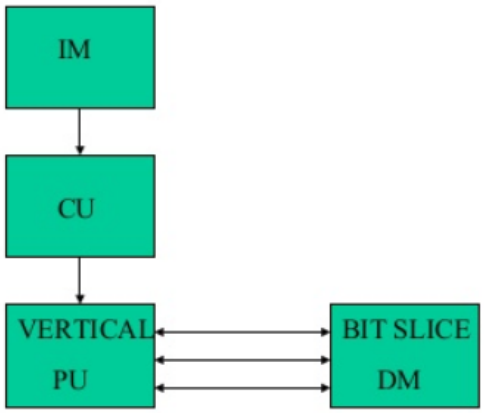
and pipeline vector computer.

- Example: IBM360/91, Cray1



Machine 2:

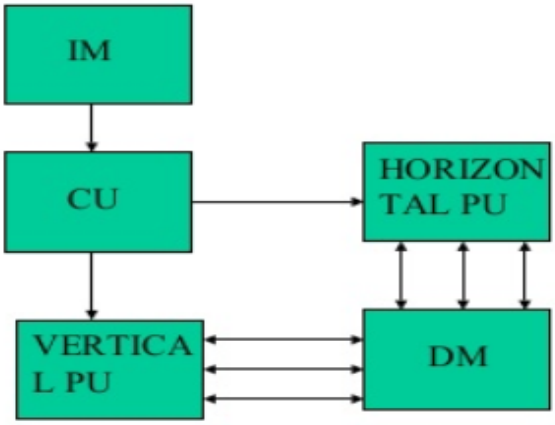
- This machine is same as machine 1 but here DM fetches bit slice of the word from memory. PU performs parallel operation on the word.
- If the memory contains two dimension array of bit with one word stored per row in this case machine 2 reads them vertically and processes the same element.
- Example: ICL DAP , MPP



Machine 3:

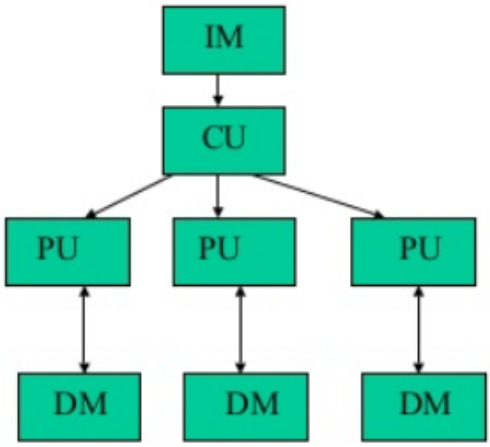
- This machine is a combination of machine 1 and machine 2.
- In this machine both vertically and horizontally reading and processing are possible.
- Hence it contain both horizontal and vertical processing unit.

- Example: OMENN 60



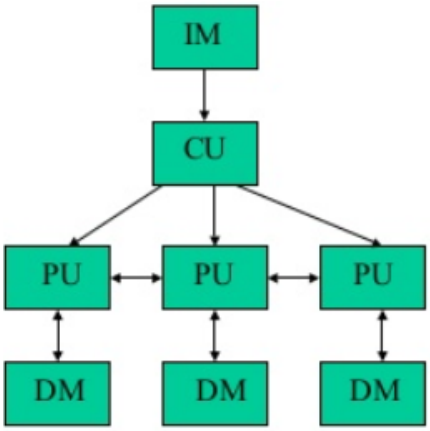
Machine 4:

- This machine is obtained by duplicating the PU and DM of machine 1.
- Combining PU and DM called as Processing Elements (PE's).
- Instructions are given to PE's for processing through the single control unit.
- Here there is no communication between PE's.
- This machine limits the applicability of the machine due to absence of communication between PE's.
- Example: PEPE



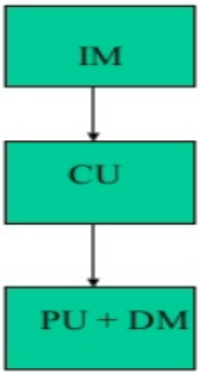
Machine 5:

- This machine is similar to the machine 4. Here there is
- communication between PE's.
- Example: ILLIAC IV



Machine 6:

- Machine 1 to 5 maintain separation between DM and PU with some data bus or connection unit providing the communication between them.
- In this machine PU and DM are combined and called as associative process(as logic is included in the memory itself).
- Machines based on this architecture span a range from simple associative memories to complex associative processors



Handler's Classification scheme

Handler's proposed an elaborate notation for expressing the pipelining and parallelism of computers. He divided the computer at three levels.

- Processor Control Unit(PCU)
- Arithmetic Logic Unit(ALU)
- Bit Level Circuit(BLC)

PCU corresponds to CPU, ALU corresponds to a functional unit or PE's in an array processor. BLC
corresponds to the logic needed for performing operation in ALU. He uses three pairs of integers to describe computer:
Computer = (k*k', d*d' , w*w')

Where, k= no. of PCUs and k'=no. of PCUs which are pipelined
d=no. of ALUs control by each PCU and d'=no. of ALUs that can be pipelined
w=no. of bits or processing elements in ALU and w'=no. of pipeline segments or stages

Consider the following model and observe how handlers differentiate them on the basis of degree of
parallelism and pipelining.
Computer = (k*k', d*d' , w*w')
Where, k= no. of PCUs and k'=no. of PCUs which are pipelined
d=no. of ALUs control by each PCU and d'=no. of ALUs that can be pipelined
w=no. of bits or processing elements in ALU and w'=no. of pipeline segments or stages

1. **Texas Instrument's Advanced Scientific Computer (TI ASC)** have one controller controlling four arithmetic pipelines. Each has 64-bit word lengths and eight stages. Therefore, we have:

   TI ASC = <1*1, 4*1, 64 * 8> = <1, 4, 64 * 8>

2. **CDC 6600** has only a single CPU with an ALU that has 10 specialized hardware functions, each of a word length of 60-bits. Up to 10 of these functions can be linked into a longer pipeline. It has 10 peripheral I/O processors which can operate in parallel with the CPU and with each other also. Each I/O processor has one ALU with a word length of12 bits. Thus we specify CDC 6600 into 2 parts:

   CDC 6600= <Central processor> *<I/O processors)>
   = <1*1, 1*10, 60*1> * <10*1, 1 * 1, 12*1>
   = <1, 10, 60> * <10, 1, 12>

Single Program Multiple Data (SPMD) Model

Single Program Multiple Data (SPMD) is a special case of the Multiple Instruction Multiple Data model (MIMD) of Flynn's classification. In the SPMD model, a single program is executed simultaneously on multiple data elements. Here, each processing element (PEs) runs the same program but with different data elements, allowing for parallel processing and increased efficiency.

In the SPMD model, the process id is used for branching. Each instance of the program works on its own data and may follow different conditional branches or execute loops differently.

Execution Process

In the SPMD model, the program is written once but replicated many times, with each PE executing the same program but with different data elements. The program is divided into tasks, which are assigned to different PEs for processing. The PEs operate independently of each other and can execute conditional branches or loops differently depending on their assigned data elements.

Synchronization Methods

Synchronization is a crucial aspect of the SPMD model to ensure that all PEs complete their assigned tasks before moving to the next phase of the program. One common method of synchronization in SPMD is Barrier Primitives, which allows multiple threads or processes to wait for each other to reach a specific point in the program before continuing to execute. Barrier

Primitives make each PE wait at its primitive until all other PEs have completed their tasks.

Use Cases

The SPMD model is used extensively in high-performance computing (HPC) applications that require massive amounts of data processing. Some of the most common use cases of the SPMD model include weather forecasting, scientific simulations, and financial modeling. The SPMD model can significantly improve processing speed and efficiency, making it an ideal choice for large-scale computing tasks.

Data Flow Model and Demand Driven Computation

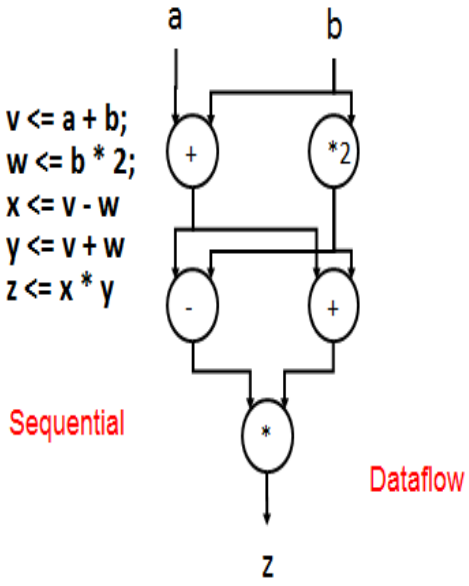
Von Neumann model: An instruction is fetched and executed in **control flow order**

–As specified by the **instruction pointer**

–Sequential unless explicit control flow instruction

Dataflow model: An instruction is fetched and executed in **data flow order** i.e., when its operands are ready and there is **no instruction pointer.**

- A number of data flow operators, each capable of doing an operation are used. Operators form the nodes of the graph and arc shows data movement between the nodes.
- Token-output node when it computes the function with data on its inputs arcs. **Called "Firing" of a node.**
- As soon as the node fires, it removes the input tokens to signify the data have been consumed.



- **Nothing like an instruction Pointer:** an instruction is enabled if and only if all the required input values have been computed.
- An instruction does not introduce sequencing constraints other than the ones imposed by **data dependencies in the algorithm.**
- Any two instructions in the program memory may be executed concurrently.
- If sufficient resources are provided, the processor can exploit all concurrency present in the program.

Advantages:

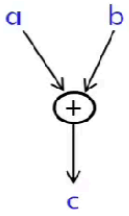
- Very good at exploiting **irregular parallelism.**
- Only real dependencies(data dependencies) constrain processing.

Disadvantages:

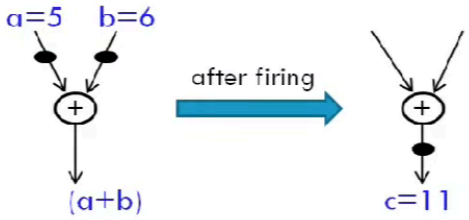
- Debugging difficult
- Implementing dynamic data structures difficult in pure data flow models
- Parallelism control needed
- High bookkeeping overhead (tag matching, data storage)

- **Data flow Computers/ Data-driven Computers/ Eager Machines**
- In **data flow computers**, the *execution of an instruction is driven by data (operand) availability* instead of being guided by a program counter
- These machines are ***eager*** for data to be present. If the data is present then the instruction is executed these computers are also known as **data driven computers** or **Eager machines**.
- In a ***data driven program***, the instructions are not ordered. There is **no shared memory**. Thus, *data is stored inside instructions*.
- The **result** of data tokens are *passed directly between the instructions*. Actually a copy of data is passed to instructions. Once data is consumed by instruction it will no longer available for reuse by other instructions.

- In a data flow machine, a program consists of ***data flow nodes*** i.e. a ***program*** *is represented using* ***directed acyclic graph*** ***(nodes and edges)***.
  - ***Instructions*** is represented by a ***node*** and the ***data dependency relationship*** is represented by the ***edge*** between the connected node.



- A *data flow node fires* (fetched and executed) when all its inputs are ready i.e. when all inputs have tokens
  - **Eg:** Find c=a+b
    if a=5 & b=6



- **Data Flow Features:**
  - No need for shared memory
  - No program counter
  - No control sequencer
- **Special mechanisms are required to**
  - detect data availability
  - match data tokens with instructions needing them
  - enable chain reaction of asynchronous instruction execution
- **Advantage:**
  - High potential for parallelism and throughput
  - Freedom from side effects
- ***Disadvantage:***
  - High control overhead,
  - Lose time waiting for unneeded arguments,
  - Difficulty in manipulating data structures.

- **Demand-driven Computers/ Reduction Machines/ Lazy Computers**
- In **demand driven machines**, if there is a *demand for the result then only the computation triggers.*
- **Data-driven machines** takes a **bottom-up approach** as it select instructions for execution based on the availability of their operands.
- **Demand-driven machines** takes a **top-down approach**, attempting to execute the instruction (a **demander**) that yields the final result. This triggers the execution of instructions that yield its operands, and so forth.
- The demand-driven approach matches naturally with functional programming languages (e.g. LISP and SCHEME).

- Example Consider the following example of a arithmetic expression :

  a=[(b+1) * c – (d÷e)]

- The ***data-driven computation*** chooses a **bottom-up approach** , starting from **b+1** and **d ÷ e** then proceeding to the '*' operation finally to the outermost operation '-'.
- A ***demand-driven computation*** chooses a **top- down approach** by *first demanding the value of* **'a'** ,which triggers the demand for evaluating the next level expression **(b+1)*c** and **d ÷ e** ,which in turns triggers the demand for evaluating **b+1** at the inner most level. The results are returned to the nested ***demander*** in the reverse order before **'a'** is evaluated.

- A ***demand-driven machines*** corresponds to **lazy machines** because operation are executed only when their result are required by another instruction. While ***data-driven machines*** are called **eager machines** because operation are carried out immediately after all their operands becomes available.
- **Reduction machines (*demand-driven machines*)**
- ***Advantages:***
  - **High parallelism potential,**
  - Easy manipulation of data structures, and
  - Only execute required instructions.
- **Disadvantage:**
  - They do not share objects with changing local state, and
  - Do require time to propagate demand tokens.