HPC(V-1)

Parallel Processing Computing

→ • HPC → high performance computing. (abstract level understanding)
• peforming computational operations collaboratively → mutiple computers.

→ Need
↳ perform a high no. of operations per seconds (FLOPS)
↳ complete a time - consuming operation in less time.
↳ complete an operation under a tight dealine.
↳ Handle huge amt of data.

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

Appl^n →: • Weather Forecasting • Scientific simulations • Big data analytics • Artifical Intellgnee • Computational Biology • Financial Modeling.

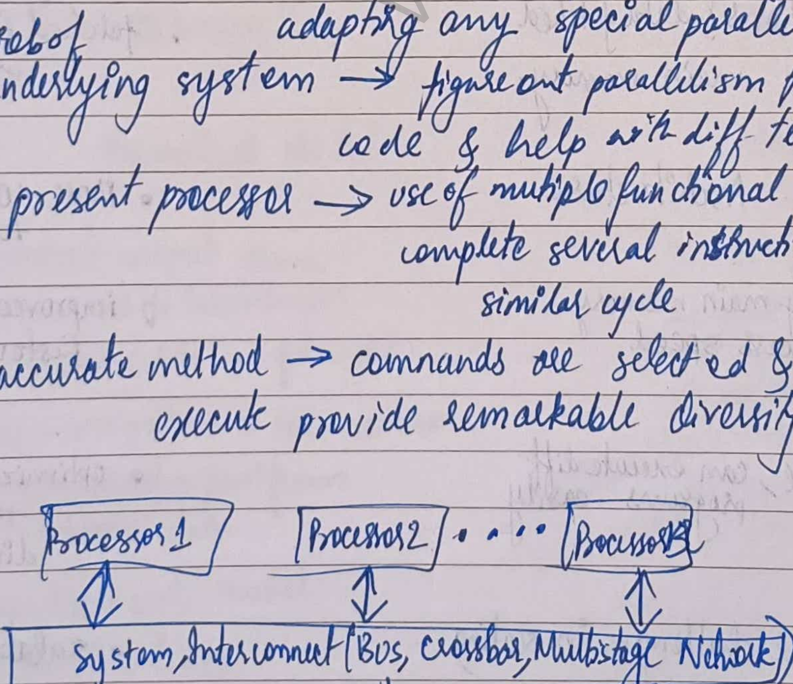
Feature	✯ Stored Program Architecture	✯ General Purpose Cache Base Microprocessor Architect.
Memory Access.	• all instructions & data fetched from main memory.	• fetched from cache or main memory
complexity	• Simpler Architechure	• More complex due to cache management
Performance	• Limited by main memory access speed	• improved due to faster cache access
versatality	• versatile, can execute diff programs easily	• optimized for specific tasks (not support diverse tasks)
scalability	• May face challenges in scaling for performance.	• scalable by increasing cache sizes or adding levels
examples →	• early mainframe computers	• modern personal computers, laptops.

✱ Motivating Parallelism -

- accelerating computing speeds
- multiplicity of datapaths
- increased access to storage elements.
- scalable performane & low cost

✱ Implicit Parallelism -

- allows programmers to write their programming without any concern about exploitation of parallelism.
- compiler, runtime system & underlying hardware → imp role in exploiting parallelism implicity.
- parallelism → transparent to programmer
  ↳ write standard sequential program without adapting any special parallel constructs
- underlying system → role of figure out parallelism from seqetial code & help with diff techniques.
- present processor → use of multiple functional units & complete several instruction in similar cycle
- accurate method → commands are selected & execute provide remarkable diversity in architecture

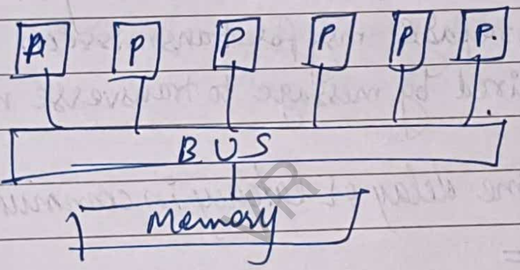


* Dichotomy of Parallel Computing Platforms -

- openly parallel program → must agree concullency & communication b/w simutaneous subtasks-

- previous → organize structre • latter → message model-

- logical & ~~phystat~~ physical organization of parallel platform :
  - → (programmer's view of platform)
  - → (actual hardwalwarde) organization of platform



(Dichotomy shared memory Model-)

* Physical Organization of Parallel Platforms.

- Ideal Parallel Computer → PRAM (Parallel Random Access Machine)
  - Random Access Machine (RAM) → simple model of computation-
  - , memory → unbounded sequence of registers.
- , each register → integer value-
  - - control unit of RAM → program
  - , programer counter → determines statement to be executed next-
  - , processes in PRAM → share same address space:

PRAM → 4 subclass → pattern of memory access →
- Exclusive-red, exclusive write (EREW) PRAM
- concurrent-read, exclusive write (CREW) RRAM
- Exclusive-read, concurrent-write (ERCW) PRAM
- concurrent-read, concurrent-write (CRCW) PRAM

* Communication Costs of Parallel Machines

processing elements leads to major overhead in parallel computing.

→

cost of communication → depends on features
- Programming modes semantic
- Network Topology
- Data handling & routing
- Software protocols associate to program.

Message Passing Costs within Parallel Computers

→ Comm^n time b/w nodes is characterized by sum of -
- Time to prepage msg for transmission
- Time required by message to transverse network to destination

Parameters determine delay or latency in communication.

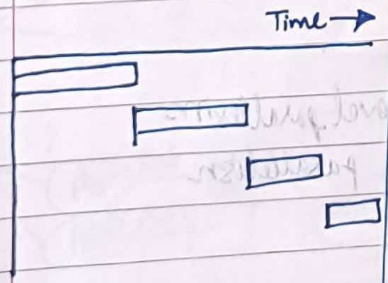
1) Startup time (t_s) =
- time consumed at sending & receiving nodes.

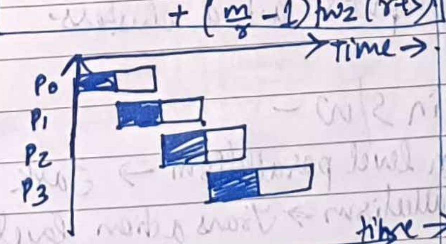
2) Per-hop time (t_h) =
- time taken by header of a message to travel b/w two directly connected nodes in network.
- also called as Node Latency
- t_h depends on delay in routing switch

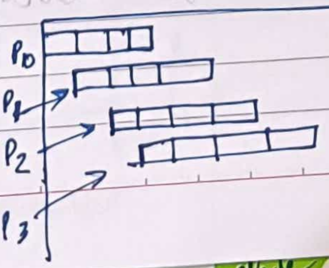
3) Per word transfer time (t_w) -
- time required by each word to transverse link.
- $\boxed{t_w = \frac{1}{r}}$ → r words transferred through channel if bandwidth is r words/sec

Parallel Machines (Two routing Techniques)

	1) Store & Forward Routing	Packet Routing	Cut through Routing.
Concept →	• entire packet is received & stored before forwarding	• Forward based on packet destination address	• Forwarding begins upon receiving packet header
Focus →	• Reliability & error correction	• efficient data movement	• Low latency for small msgs.
Adv →	• Simpler Implementation • Error detection/correction at each hop.	• Flexible for various techniques	• efficient bandwidth usage
Disadv →	• high latency for small msg. • less efficient	• requires storage for header info.	• Increased complexity delivery, potential buffer- • not suitable for large msg.
Latency →	• High latency ↳ storing entire packet	• Moderate latency ↳ routing decision.	• Lowest latency ↳ as begins upon header receipt.
performance	– Reliable but may suffer from latency.	• Balanced performance & scalability	• efficient performance with lower latency
	• $t_{comm} = t_s + (mt_w + t_h)l$ $t_{comm} = t_s + mlt_w$	• $t_{comm} = t_s + t_{w1}m + t_h1 + t_{w2}(r+s) + (\frac{m}{s} - 1)t_{w2}(r+s)$	• $t_{comm} = t_s + 1t_h + t_wm$







Cost Model for Communicating Messages

→ communicate message b/w two nodes 1 hops away using cut-through routing.

Total cost → $t_{comm} = t_s + 1t_h + t_w m$.

To optimise cost
- → communicate in bulk → small msg into single large messag
- → minimize vol. of data ⇒ cost of per word transfer time can be reduced.
- → Minimize distance of data transfer → min. no. of hops.

cost b/w two nodes. → $\boxed{t_{comm} = t_s + t_w m}$

Levels of parallelism

- → Instruction level parallesim
- → Threadlevel/ task level paralasim
- → Data level parallelism.

Parallism in Hardware (Uniprocessor).
- Pipelining, - Superscalar, VLIW etc.
- SIMD instructions, Vector processors, GPU.

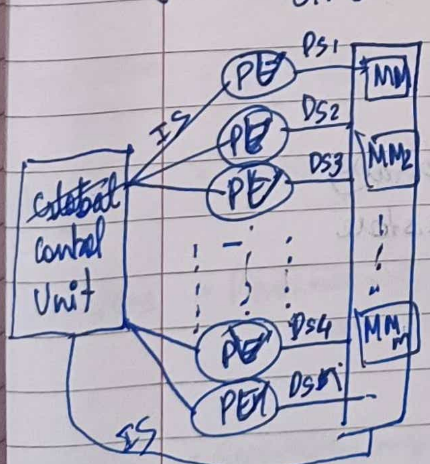
Mutiprocessor —
- symmetric shared memory multiprocessors
- Circulated " "
- chip multiprocessor aka Multicores
- Multi computer a.k.a clusters.

Parallelism in S/W —
- Instruction level parallelism → Task level parallesim
- Data parallelism → Trans action level parallelism

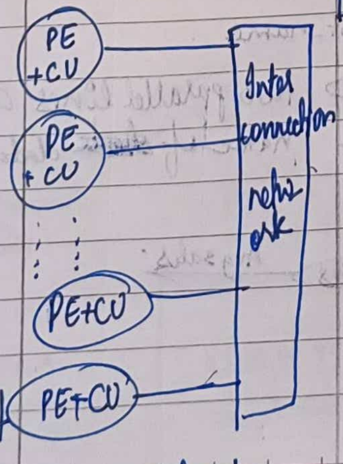
(1-9, 1-22)

Models - (SIMD, SIMT, ~~SM~~MIMD, SPMD)

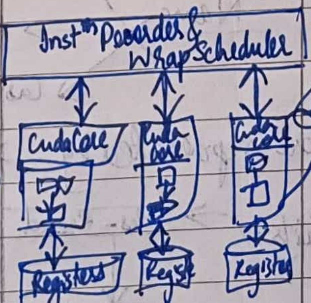
SIMD	MIMD	SIMT	SPMD
Single Instruction Multiple data.	Multiple Instruction Multiple data.	Single Instruction Multiple Threads	Single Program Multiple data.
single Instruction to multiple data element simultaneously	Multiple instⁿ to multiple data elements simultaneously	single instruction issued to multiple threads	single program executed by multiple processors
all data elements ↳ should have same datatype & operations	data elements ↳ diff data types & operations	all threads → diff data elements	each processor ↳ operates on own data partition
easier to program for specific tasks.	more complex ↳ data dependencies & race conditions	similar to SIMD allows for thread divergence	similar to MIMD, focusses on data parallelism with a single program.
minimal commⁿ b/w processing units/elements	commⁿ may occur for coordination or data sharing	commⁿ within thread groups for synchronization	commⁿ may occur.
High data level parallelism	variable data level parallelism	Moderate data level parallelism within thread group	Moderate data level parallelism within same program execution
eg → vector processors, GPU.	eg → distributed computing, clusters	eg → GPU architecture CUDA.	eg → MPI, OpenMP

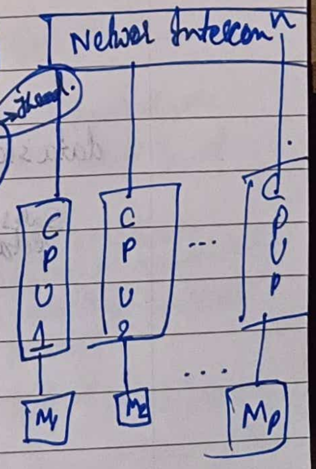


PE → Processing ~~element~~ Unit
MM → memory module
DS → Data Stream



CU → control Unit.





Data Flow Models –

- defines fun^n of internal process system with aid of Data Flow Diagram
- depicts function derivation of data values without indicating how they derived

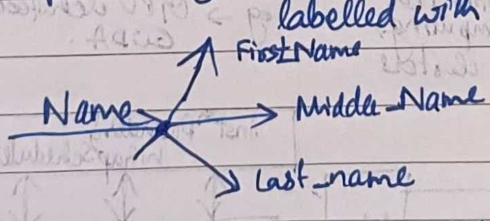
Data Flow Diagrams –

- Functional modelling → hierarchy of DFD.
- graphical representation of system → shows input to system, processing upon inputs, the outputs of system as well as internal data structures.
- illustrate series of transformation & computations performed.

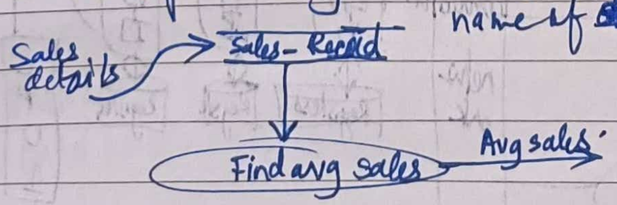
4 parts of DFD → 1) Processes → • transform data values • visualised as high level process → system
2) Data Flows → flow of data b/w processes → b/w actor & process or data store & process
3) Actors → active objects interact with system
4) Data stores → passive object act as repositories of ~~data~~

other parts → constraints & ~~data~~ control flows

data flow represented by → directed arc or an arrow, labelled with name of data item.



data store represented by → two parallel lines containg name of ~~data~~ data store



* Demand Driven Computation –

- general framework for deriving demand driven algorithm for interprocedural data flow analysis.

- goal → reduce time &/or space overhead of covertional exhaustive analysis by avoidig collection of info

- Reduction Machines/Lazy Computers.
- approach matches with fuctional programming lnguage.
- whetes minimizing time or spce is of primary concern. result caching may be incorpoted in derived algo.
- If problem's flow function → distributive → derived algo provides as preciseinfo as corresponding exhaustive analysis
- executed when results are required.
- NO USE of shared memory.
- high degree of parallelism.

Appl^n

- Databaseservers, webservers, embeddednetwork, digital signal processing, multimedia applications, scientific application of thread level parallelism

Feature	N-Wide Superscalar Architth	Multicore Architecture	Multithreaded Architecture
Processing units	• mutiple execution units within a core	• Mutiple independent cores	• single core with thread contex switching
parallelism Type	• Instruction level	• Task level (program)	• Thread level
Benefits	• Improved performaci	• significant performance gains for parallelizable workloads.	• Improved perfomace for thread level.
Limitations	• Diminishing returns with more units	• Increase complexity require multicore aware S/w.	• limited parallelism compared to multicore, overhead of context switching
Typical Use cases	• Scientific Computing., Image/Video Processing	• General purpose computing, multitasking, server workloads	Multitasking, web browsing I/O bound tasks

* Pipelining -

- Breakes down instruction execution into stages.
- Allows overlapping of instruction execution.
- Increases throughput by processing multiple instructions simultaneoust
- Reduces overall execution time.
- Instructions progress through stages sequentially.

* Superscalar Execution -

- utilizes multiple execution units with single processor core.
- executes mutiple instructions simutaneonsly.
- can exploit instruction level parallelism with a program.
- Dispatches instruction - level paralelism to mutiple execution units.
- acheives greater performance gain compared to pipeline alone

* VLIW (Very long Instruction Word) processor.

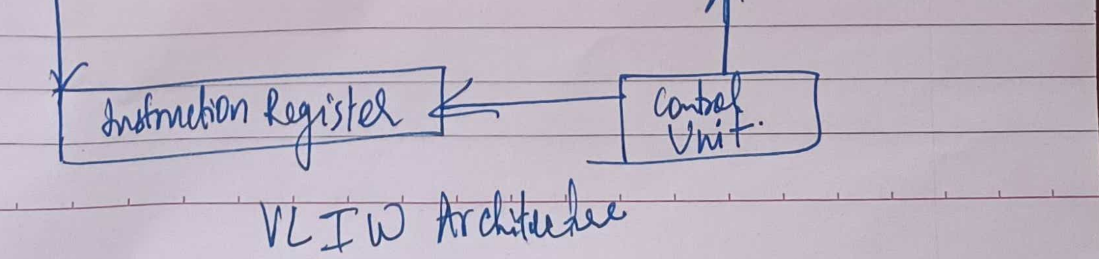
- executes mutiple instructions in parallel using single VLIW.

Process it follows → 1) Instruction Bundling 2) Static Scheduling 3) Parallel Execution 4) No dependency checking 5) efficiency & performance 6) compiler dependency.

Adv → • reduced hardware complexity • high performance for parallel appln

Disadv → • relies on complex compiler for parallel instruction • Less flexible code

Appl^n → • Digital Signal Processing • Scientific Computing • Embedded systems



VLIW Architecture

NUMA Multiprocessor	UMA multiprocessor.
• Non uniform (varies by location)	• Unifor (equal accesstime)
• more complex, mutiple memery ctrls	• simpler, simple mem ctrls
• scalability is more	• limited scalablility
• performance higher (for workload with locality)	• lower performance (potential bottleneck).
• require msg passing b/w processors	• Interprocess communication simpler due to shared memory
• eg → HPC, large DBs	• eg → General purpose, file sharing