

Parallel Processing Computing

→ HPC → high performance computing. (abstract level understanding)
 • performing computational operations collaboratively → multiple 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 deadline.
 • Handle huge amt of data.

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

Appⁿ → • Weather Forecasting • Scientific Simulations • Big data analytics
 • Artificial Intelligence • 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 Architecture

• More complex due to cache management

Performance

• limited by main memory access speed

• improved due to faster cache access

Versatility

• 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

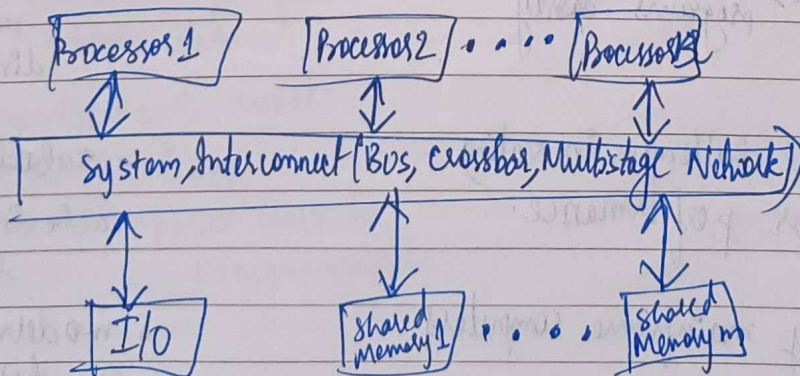
• modern personal computers, laptops.

* Motivating Parallelism -

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

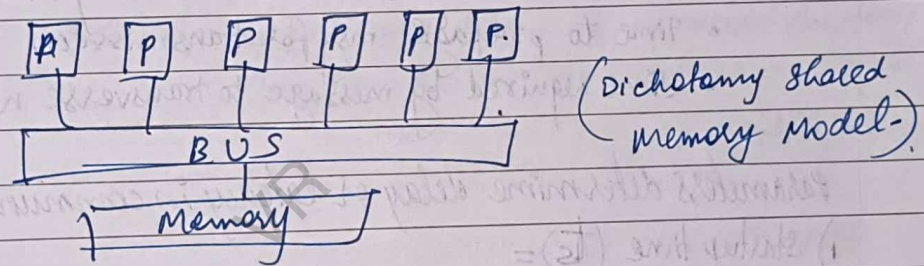
* Implicit Parallelism -

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



* Dichotomy of Parallel Computing Platforms -

- openly parallel program \rightarrow must agree concurrency & communication b/w simultaneous subtasks.
- previous \rightarrow organize structure, latter \rightarrow message model.
- logical & ~~physical~~ physical organization of parallel platform.
 - \rightarrow (programmer's view of platform)
 - \rightarrow (actual hardware organization of platform)



* Physical Organization of Parallel Platforms.

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

PRAM \rightarrow 4 subclass \rightarrow pattern of memory access \rightarrow

- Exclusive-read, exclusive write (EREW) PRAM
- Concurrent-read, exclusive write (CREW) PRAM
- 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

→ Commⁿ time b/w nodes is characterized by sum of -

- Time to prepare msg for transmission
- Time required by message to transverse network to destⁿ node

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.

- $t_w = \frac{1}{\sigma}$ → σ words transferred through channel if bandwidth is σ words/sec.

Parallel Machines (Two routing Techniques)

1) Store & Forward Routing Packet Routing Cut Through Routing.

Concept → • entire packet is received & stored before forwarding

Focus → • Reliability & error correction

Adv → • Simpler Implementation
• Error detection/correction at each hop.

Disadv → • high latency for small msg.
• less efficient

Latency → • High latency
↳ storing entire packet

performance - Reliable but may suffer from latency.

• Forward based on packet destination address

• efficient data movement

• Flexible for various techniques

• requires storage for header info.

• Moderate latency
↳ routing decision.

• balanced performance scalability

• Forwarding begins upon receiving packet header

• low latency for small msg.

• efficient bandwidth usage

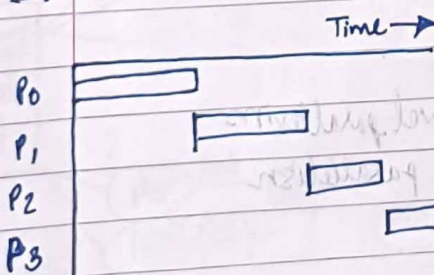
• Increased complexity delivery, potential buffer.
• not suitable for large msg.

• Lowest latency
↳ as begins w/ on header receipt.

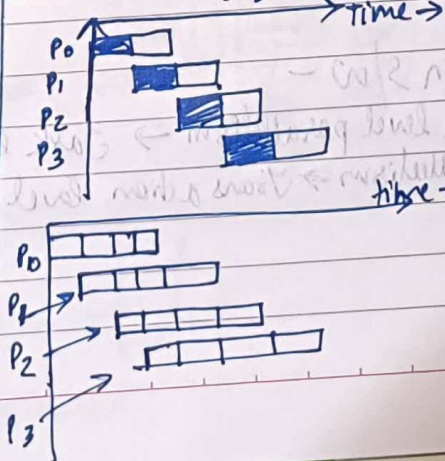
• efficient performance with low latency

$$t_{comm} = t_s + (mtw + t_h)$$

$$t_{comm} = t_s + mltw$$



$$t_{comm} = t_s + t_{w,m} + t_{h1} + t_{w2} (rt_s) + (m-1)t_{w2} (rt_s)$$



$$t_{comm} = t_s + 1t_h + t_{w,m}$$

★ Cost Model for Communicating Messages

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

$$\text{Total cost} \rightarrow t_{\text{comm}} = t_s + t_h + t_{wm}$$

to optimise cost

- communicate in bulk → small msg into single large message
- 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 → $t_{\text{comm}} = t_s + t_{wm}$

★ Levels of parallelism

- Instruction level parallelism
- Thread level parallelism
- Task level parallelism
- Data level parallelism

Parallelism in Hardware (Uniprocessors).

- Pipelining, - Superscalar, VLIW etc.
- SIMD instructions, Vector processors, GPU.

Multiprocessors -

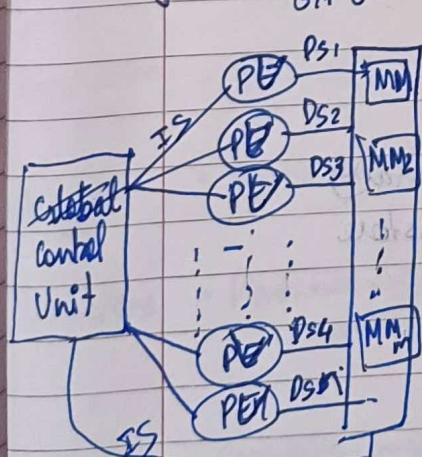
- Symmetric shared memory multiprocessors
- Circulated " "
- Chip multiprocessor aka Multiprocess
- Multi Computer a.k.a clusters

Parallelism in S/W -

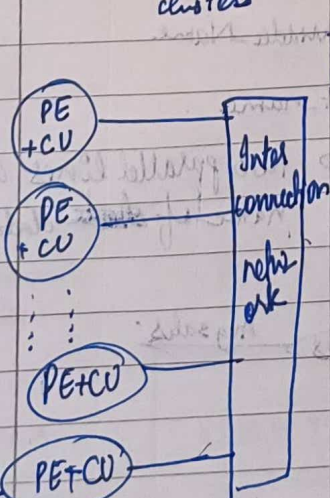
- Instruction level parallelism → Task level parallelism
- Data parallelism → Transaction level parallelism

* Models - (SIMD, MIMD, SIMT, SPMD)

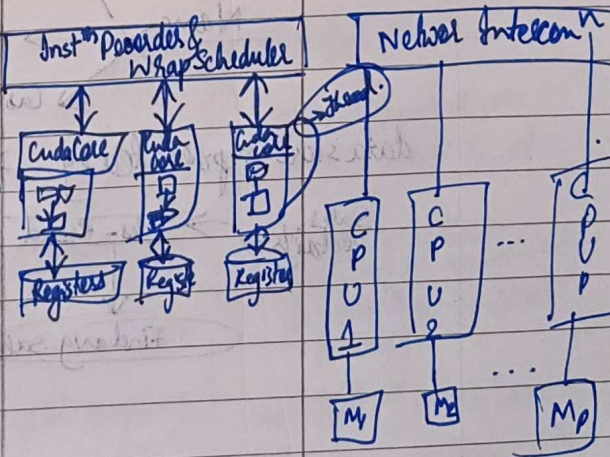
| SIMD | MIMD | SIMT | SPMD |
|---|---|---|---|
| <ul style="list-style-type: none"> Single Instruction Multiple data. single instruction to multiple data element simultaneously all data elements ↳ should have same datatype & operations easier to program for specific tasks. minimal commⁿ b/w processing units/elements High data level parallelism eg → vector processors, GPU. | <ul style="list-style-type: none"> Multiple Instruction Multiple data. Multiple instⁿ to multiple data elements simultaneously data elements ↳ diff data types & operations more complex ↳ data dependencies & race conditions commⁿ may occur for coordination or data sharing variable data level parallelism eg → distributed computing, clusters | <ul style="list-style-type: none"> Single Instruction Multiple Threads single instruction issued to multiple threads all threads → diff data elements similar to SIMD, allows for thread divergence commⁿ within thread groups for synchronization Moderate data level parallelism within thread group eg → GPU architecture, GUDA. | <ul style="list-style-type: none"> Single Program Multiple data. single program executed by multiple processors each processor ↳ operates on own data partition similar to MIMD, focuses on data parallelism with a single program. commⁿ may occur. moderate data level parallelism within same program execution eg → MPI, OpenMP |



PE → processing element unit
MM → memory module
DS → Data Stream



CU → control Unit



* Data Flow Models -

- defines funcⁿ of internal process system with aid of Data Flow Diagrams
- depicts function derivation of data values without indicating how they derived

* Data Flow Diagrams -

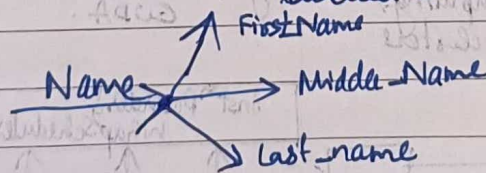
- Functional modelling → hierarchy of DFD
- graphical representation of system → shows input to system, processing upon inputs, the outputs of system as well as internal data structures.
- illustrate series of transformation & computations performed

Parts of DFD →

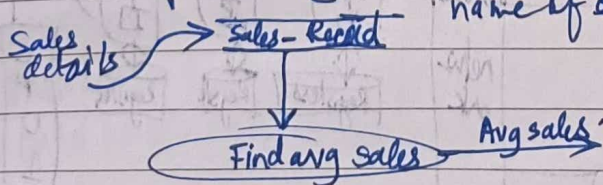
- 1) Processes → transform data values
visualised as high level process system
- 2) Data Flows → flow of data b/w processes or data of process
- 3) Actors → active objects interact with system
- 4) Data Stores → passive object act as repositories of data

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 containing name of data store



*

Demand Driven Computation --

- general framework for deriving demand driven algorithm for interprocedural data flow analysis.
- goal → reduce time &/or space overhead of conventional exhaustive analysis by avoiding collection of info
- Reduction Machines/Lazy Computers.
- approach matches with functional programming language.
- whether minimizing time or space is of primary concern.
- result caching may be incorporated in derived algo.
- if problem's flow function → distributive → derived algo provides as precise info as corresponding exhaustive analysis
- executed when results are required.
- NO USE of shared memory.
- high degree of parallelism.

Applⁿ

Databases servers,
web servers, embedded network,
digital signal processing, multimedia
applications, scientific
application & thread level parallelism

| Feature | N-Wide Superscalar Arch th | Multi-core Architecture | Multi-threaded Architecture |
|-------------------|--|---|---|
| Processing units | • multiple execution units within a core | • Multiple independent cores | • single core with thread contexts switching |
| Parallelism type | • Instruction level | • Task level (program) | • Thread level |
| Benefits | • Improved performance | • significant performance gains for parallelizable workloads. | • Improved performance for thread level. |
| Limitations | • Diminishing returns with more units | • Increase complexity require multiple core aware S/W. | • limited parallelism compared to multi-core, 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 |