Parallel and Distributed Computing

Dr. Ali Sayyed
Department of Computer Science
National University of Computer & Emerging Sciences

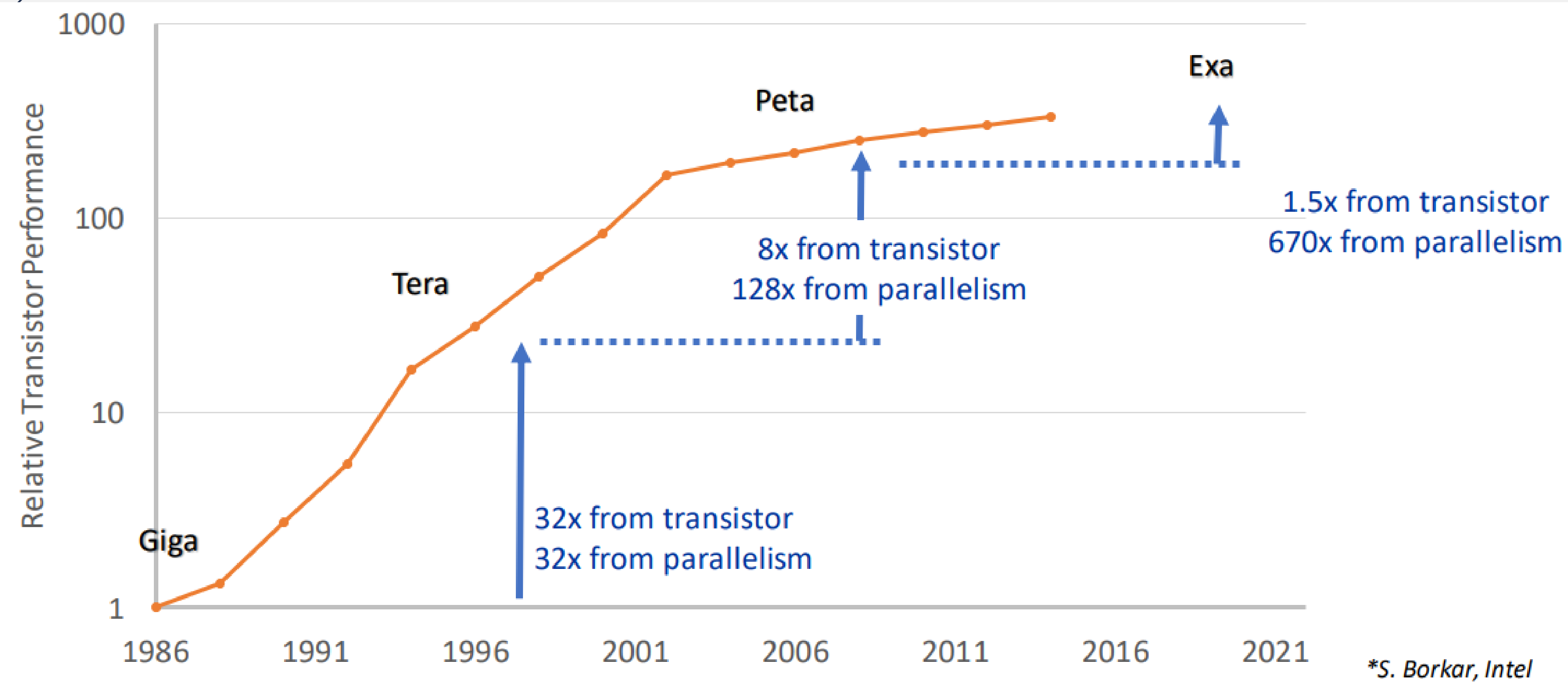
Incorporating Parallelism

What is Parallelism

- Parallelism refers to the simultaneous execution of multiple tasks or instructions.

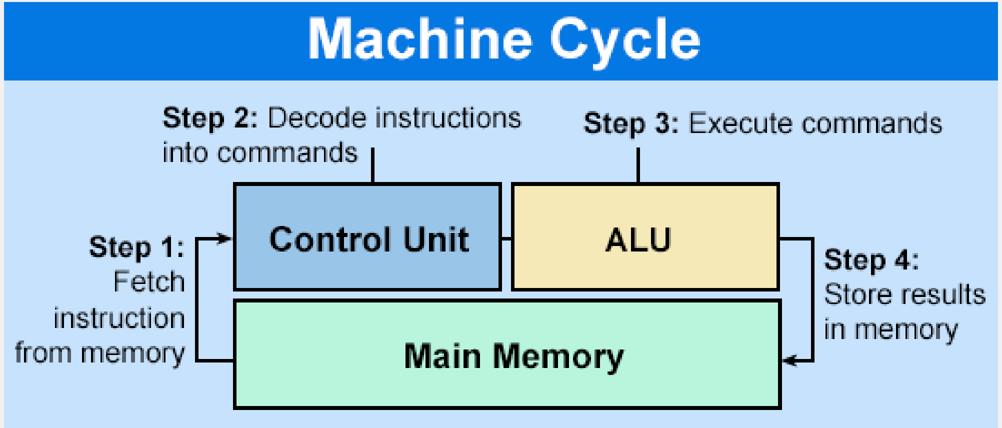
- Parallelism can be achieved in various ways, allowing multiple operations to be performed concurrently.

Performance from Parallelism



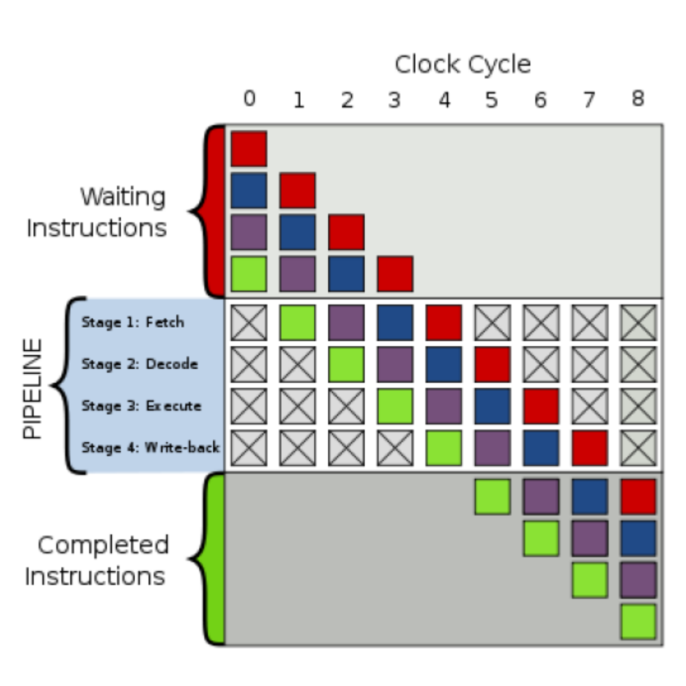
Instruction-Level Parallelism

- Involves executing multiple instructions (called Instruction Pipelining) at the same time.
- Instruction pipelining can reduce idle time in the processor
- Consider a machine cycle (having four-stage) of an Intel processor to execute one instruction.



Instruction-Level Parallelism

- This is a generic pipeline with four stages: **Fetch**, **Decode**, **Execute**, **Write-back**.
- At the top are the list of instructions waiting to be executed
- At the bottom are the list of instructions that have been completed
- In the middle are the instructions ins the pipeline.
- Pipelining improves the performance of a processor by overlapping the execution of multiple instructions.



Without ILP Example

Cycle	Instruction Fetch (IF)	Instruction Decode (ID)	Instruction Execute (EX)	Instruction Write-back (WB)
1	I1 IF			
2		I1 ID		
3			I1 EX	
4				I1 WB
5	I2 IF			
6		I2 ID		
7			I2 EX	
8				I2 WB
9	I3 IF			
10		I3 ID		
11			I3 EX	
12				I3 WB
13	I4 IF			
14		I4 ID		
15			I4 EX	

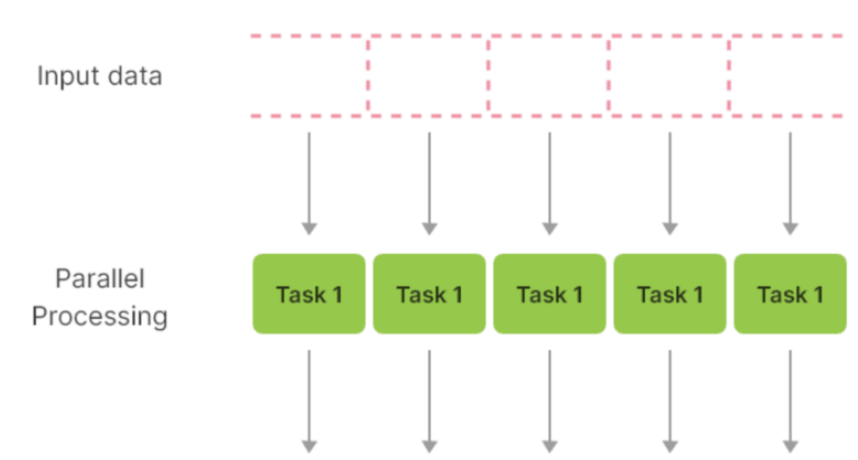
With ILP Example

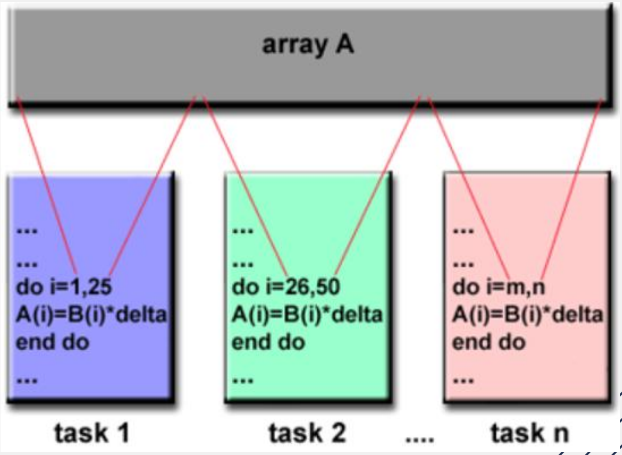
Cycle	Instruction Fetch (IF)	Instruction Decode (ID)	Instruction Execute (EX)	Instruction Write-back (WB)
1	I1 IF			
2	I2 IF	I1 ID		
3	I3 IF	I2 ID	I1 EX	
4	I4 IF	I3 ID	I2 EX	I1 WB
5		I4 ID	I3 EX	I2 WB
6			I4 EX	I3 WB
7				I4 WB

Data Parallelism

Threads involved in doing similar things, but at different locations.

- The data set is organized into a common structure, such as an array.
- A set of tasks **work collectively** on that structure; however, each task **works on a different region**.
- Tasks **perform the same operation on their portion of** data, for example, ”multiply every array element by some value”.





Task-Level Parallelism (TLP)

- Involves dividing a program into smaller tasks that can be executed independently.
- `program:`
- `...`
- `if CPU="a" then do task "A"`
- `else if CPU="b" then do task "B"`
- `end if`
- `...`
- `end program`
- Task Parallelism means concurrent execution of the different task on multiple computing cores or threads using the same or different data.

Threads involved in doing different things.

Task-Level Parallelism (TLP)

- **`Code executed by CPU "a":`**
- `program:`
- `do task "A"`
- `end program`

- **`Code executed by CPU "b":`**
- `program:`
- `do task "B"`
- `end program`
- In the general case, different execution core/threads communicate with one another as they work.

Task-Level Parallelism (TLP)

- This type of parallelism is effective **when the nature of the work doesn't require strict predictability in the decomposition process**.
- **Load balancing becomes crucial** in task parallelism to ensure that tasks are evenly distributed across the available processing resources.
- Examples:
  - Scientific simulations, where different parts of a simulation can be executed concurrently.

Parallel vs Distributed Systems

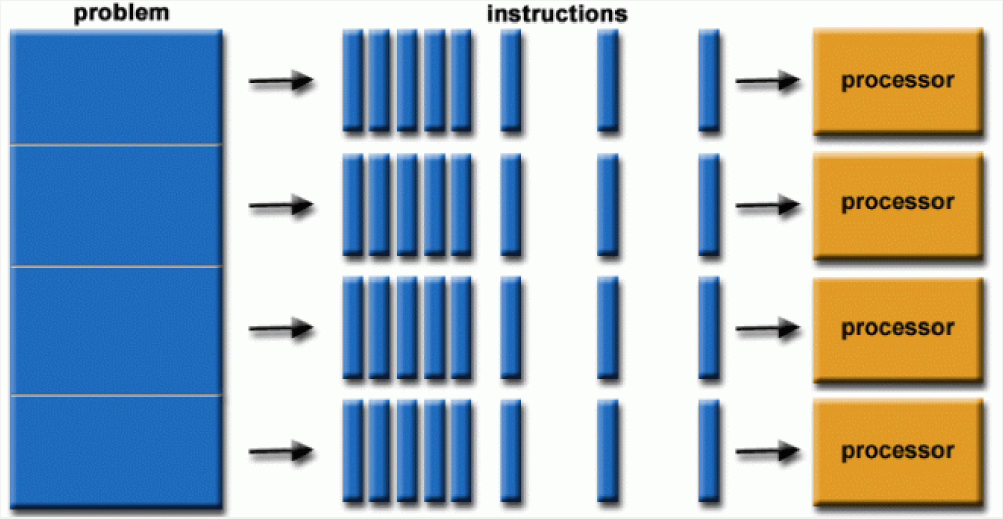
Motivation

- **Parallel** computing and **distributed** computing are two fundamental paradigms used to improve
    - **computational speed,**
    - **efficiency**, and
    - **scalability**.

- **Both approaches involve multiple processors working together**, but they differ in architecture, communication, and execution strategies.

Parallel Computing Systems

- Parallel computing refers to the simultaneous execution of **multiple tasks** using multiple processing elements within a **single system** to solve a problem faster.



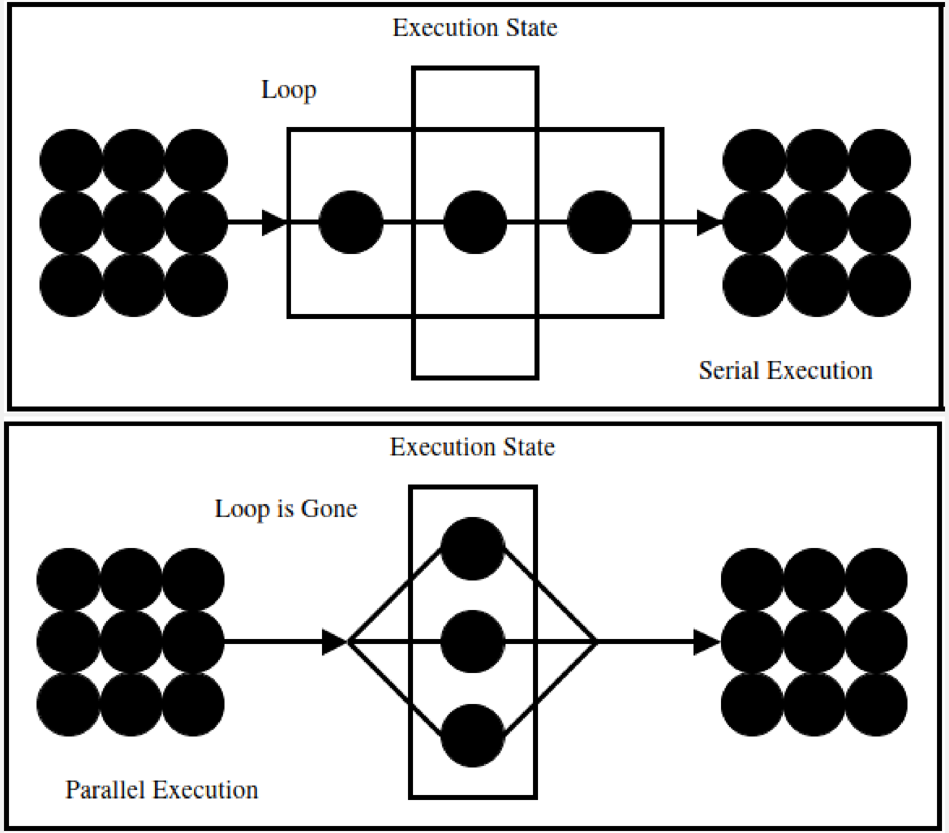
Key Characteristics

- Uses multiple **processors** (or **cores**) within a single system.
- **Shared** memory or **distributed** memory.
- **Tasks are divided** into subtasks, which execute concurrently.
- **Low-latency** communication.
- Requires **tight synchronization** among processors.
- **Example Systems** (e.g., Multicore Systems, high-performance computers, supercomputers etc).

Types of Parallel Computing

- **Shared Memory Architecture**
  - All processors have access to a common memory.
  - Uses synchronization mechanisms (e.g., locks, semaphores).
- **Distributed Memory Architecture**
  - Each processor has its own local memory.
  - Uses message passing for inter-processor communication.
- **Hybrid Architecture**
  - Combination of shared and distributed memory models.

Serial VS parallel Execution



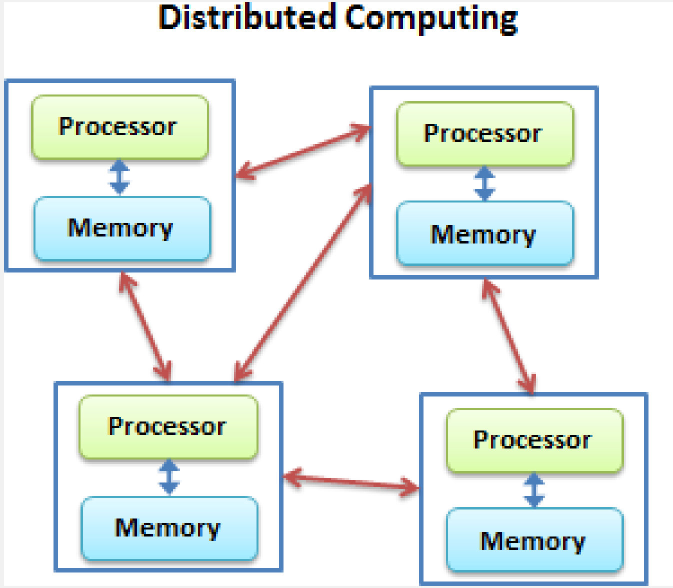
Challenges in Parallel Systems

- **Synchronization Issues**
  - Ensuring correct execution order of parallel tasks.
  - Dealing with race conditions by using locks and barriers.
- **Load Balancing**
  - Efficiently distributing tasks across multiple processors.
  - Dynamic vs. static load balancing strategies.
- **Memory Management**
  - Managing shared memory access efficiently.
  - Avoiding bottlenecks in memory bandwidth.
- **Programming Complexity**
  - Writing efficient parallel algorithms.
  - Adapting sequential programs to parallel execution.

Distributed Computing

- Distributed computing involves multiple independent computers (nodes) working together over a network to achieve a common goal.

- **Goal:** Scalability, fault tolerance, and distributed workload.



Key Characteristics

- Multiple **autonomous computers** (nodes) connected via a network.
- Each node has its **own memory and processing** power.
- Tasks are distributed and **executed independently**.
- Communication occurs via **message passing**.
- Used for **scalability**, **fault tolerance**, and **decentralized control**.

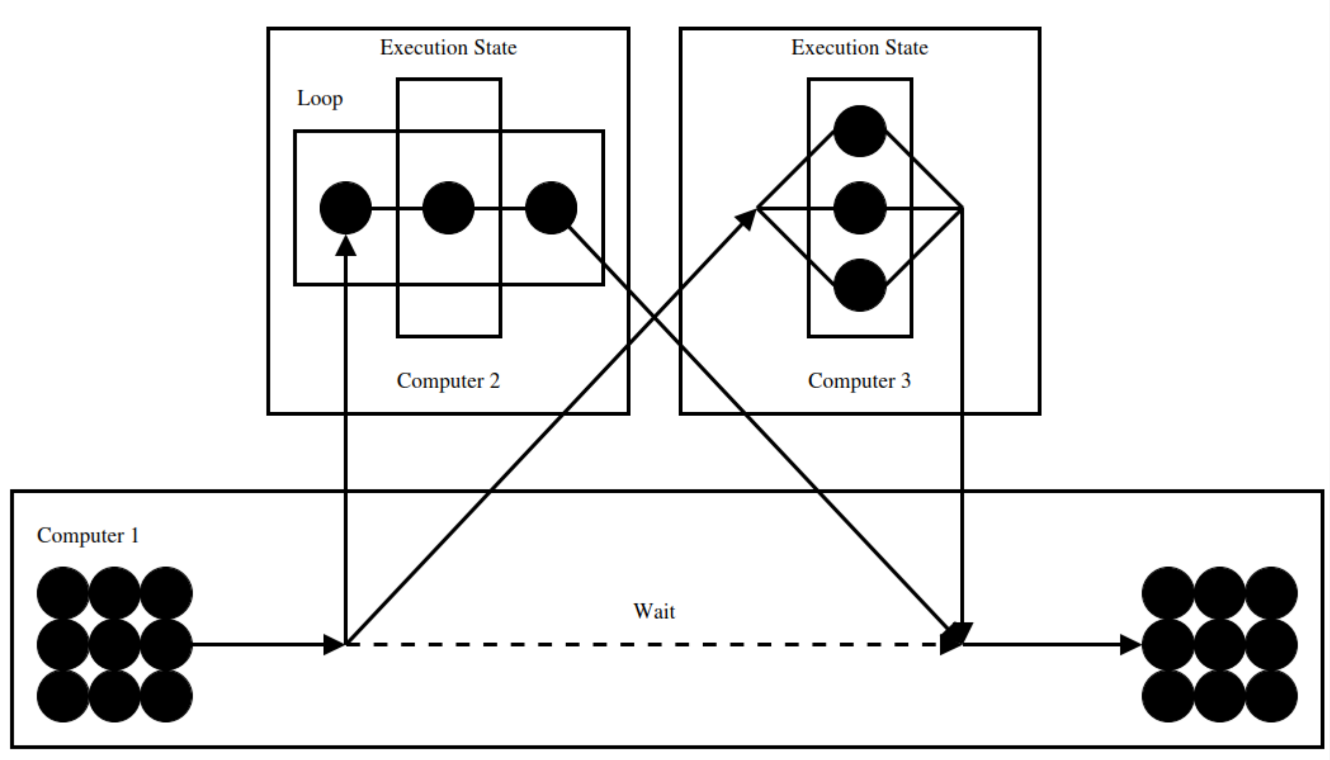
Types of Distributed Computing

- **Cluster Computing**
  - A group of tightly coupled homogeneous computers acting as a single system. e.g. Google Cloud Cluster, AWS Clusters.
- **Grid Computing**
  - Uses geographically distributed computers (same/different) connected via a network. Example: SETI@Home, CERN Grid.
- **Cloud Computing**
  - Provides scalable computing resources on demand via the internet. e.g. Amazon AWS, Microsoft Azure, Google Cloud.

Challenges in Distributed Systems

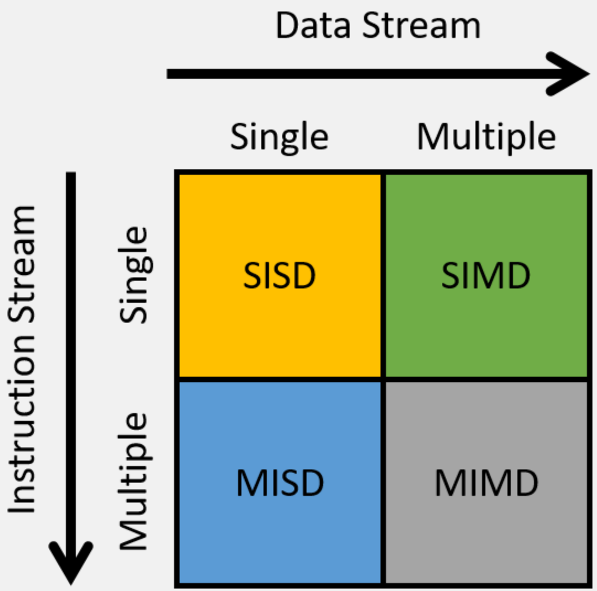
- **Network Latency and Bandwidth**
  - Communication delays due to network overhead.
  - Limited bandwidth causing bottlenecks.
- **Fault Tolerance and Reliability**
  - Handling node failures and system crashes.
- **Data Consistency**
  - Ensuring data consistency across distributed nodes.
- **Security and Privacy Concerns**
  - Protecting data during transmission and storage.
- **Resource Management**
  - Efficient allocation of computing and storage resources.

Execution on a Distributed System



Flynn’s Taxonomy of Architecture

- Flynn’s taxonomy is commonly used to describe the ecosystem of modern computing architecture

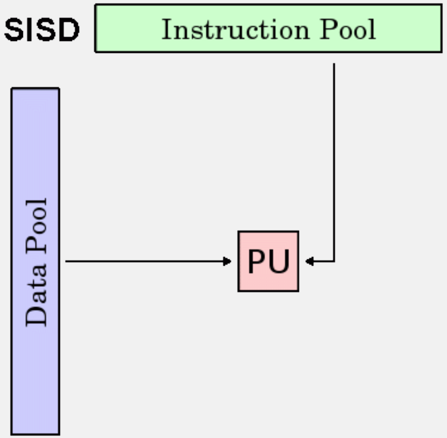


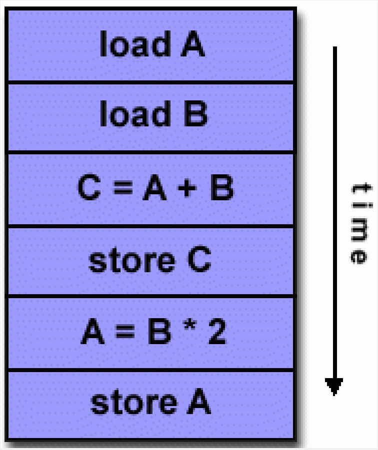
A **stream** in this context is a flow of data or instructions.

A **single stream** issues one element per time unit, similar to a queue. In contrast, **multiple streams** typically issue many elements per time unit (think of multiple queues).

Single Instruction, Single Data (SISD)

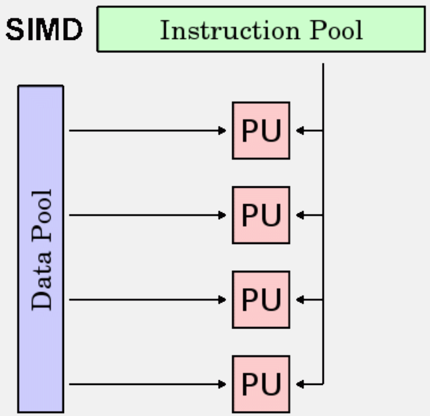
- A serial (non-parallel) computer
- A single compute unit processing a single instruction at a time. Most commercially available processors prior to the mid-2000s were SISD machines.

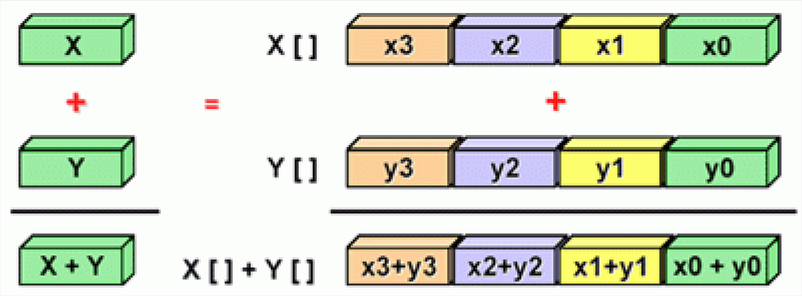




Single Instruction, Multiple Data (SIMD)

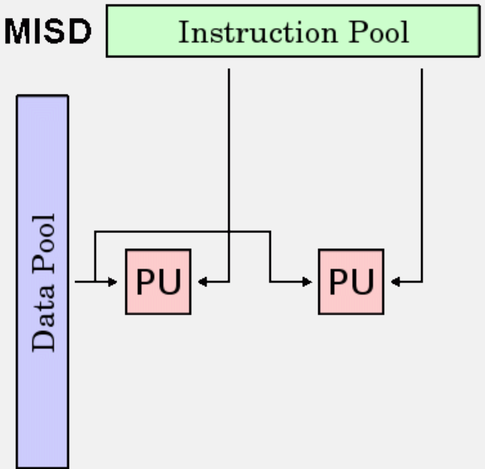
- A type of parallel computer, which execute the same instruction on multiple data simultaneously in lockstep fashion.
- All instructions are placed into a queue, while data is distributed among different compute units. The most well-known example of the SIMD architecture is the graphics processing unit.

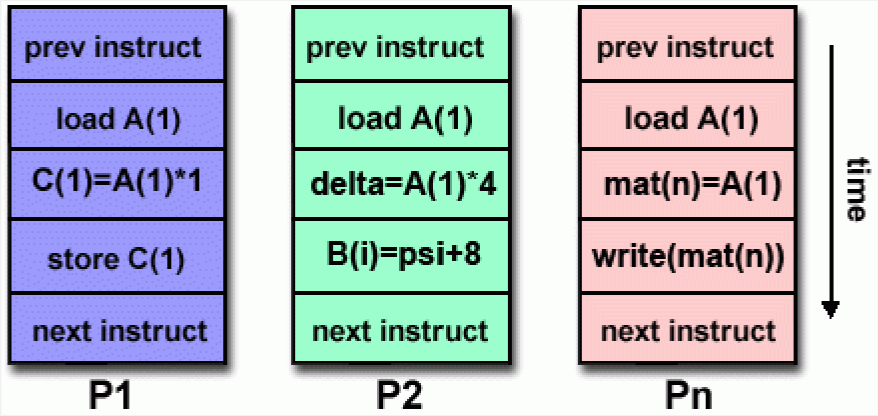




Multiple Instruction, Single Data (MISD)

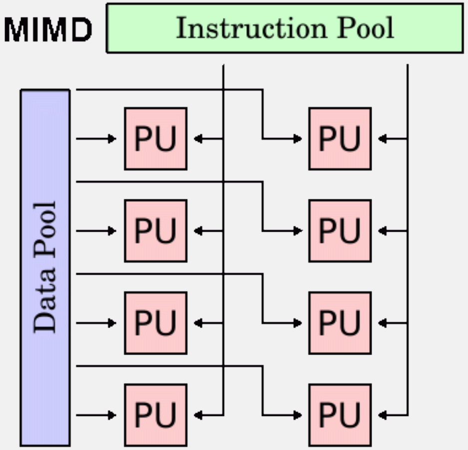
- A type of parallel computer which have multiple instruction units performing on a single data stream. MISD systems were typically designed for incorporating fault tolerance in mission-critical systems, such as the flight control programs for NASA shuttles.

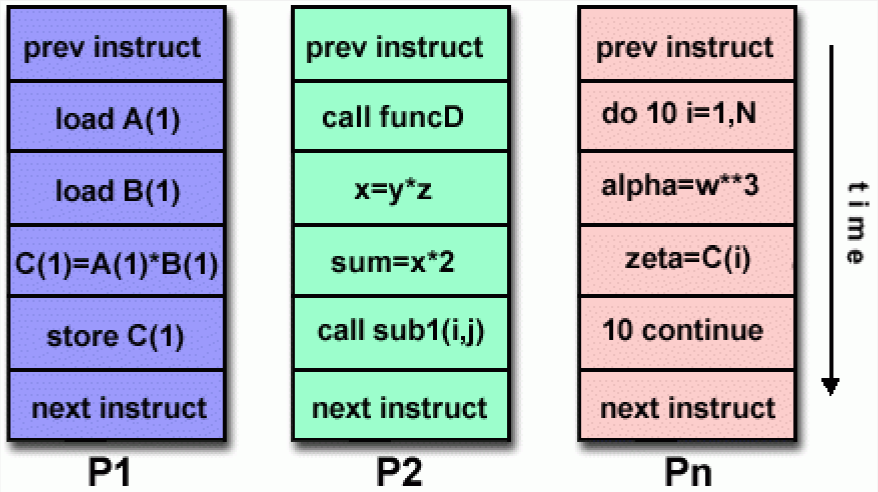




Multiple Instruction, Multiple Data (MIMD)

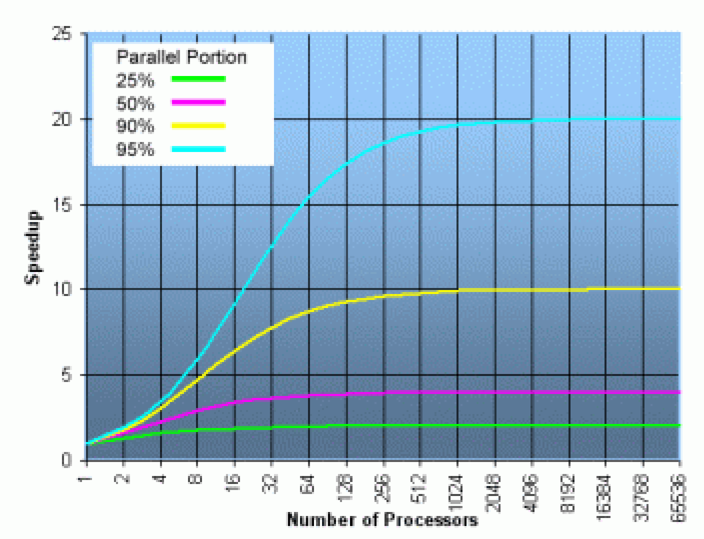
- A type of parallel computer which can work on multiple instructions or multiple data streams. Since nearly all modern computers use multicore CPUs, most are classified as MIMD machines.

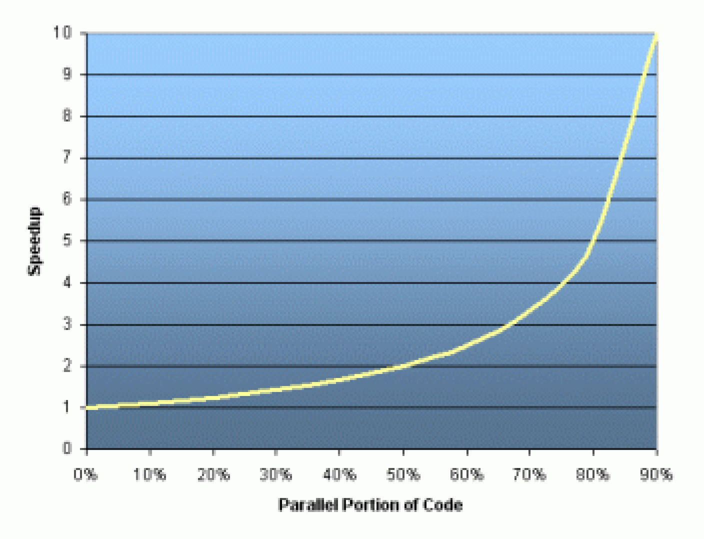




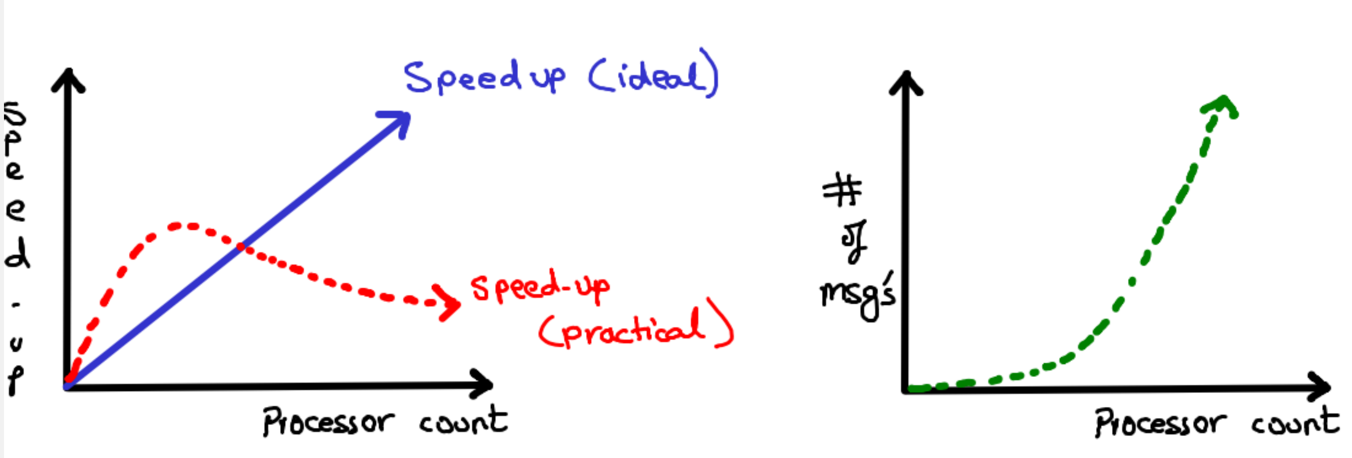
Amdahl's Law

- Amdahl's Law states that potential program speedup is defined by the fraction of code that can be parallelized (P).





Ideal VS Practical Speed Up



- Why ideal speedup is not possible:
- **data transfer** (through message exchanges), **I/O bottlenecks**, **race conditions**, **dependencies**, **load balancing**, **deadlocks**, **synchronization**, **node failures**, **lazy programmers**, . . .

Amdahl's Law

Speedup = 1/(1-p)

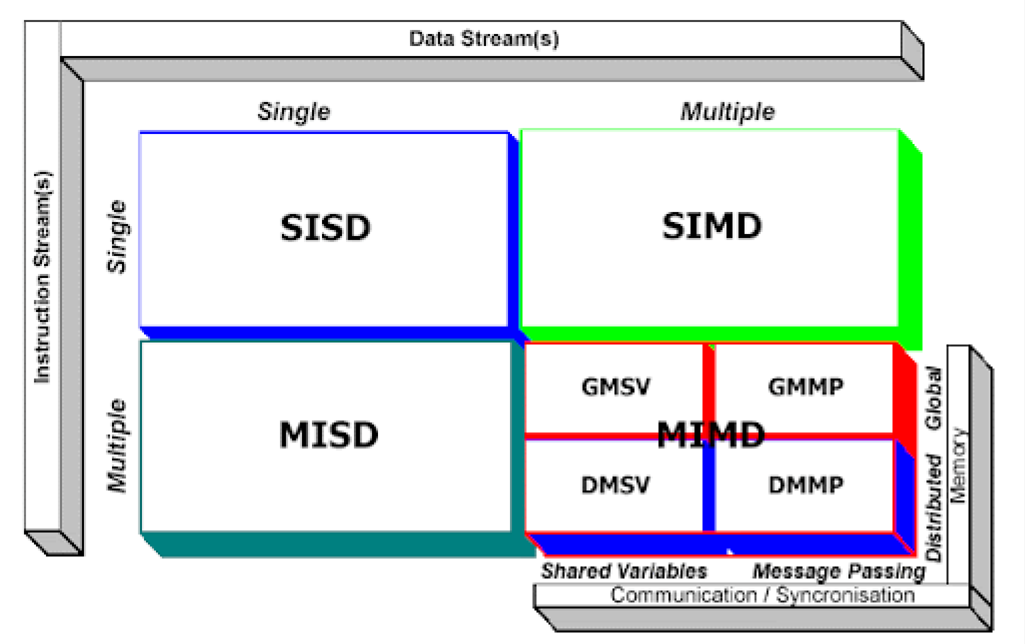
- if none of the code can be parallelized, P = 0 and the speedup = 1 (no speedup).
- If all of the code is parallelized, P = 1 and the speedup is infinite (in theory).
- If 50% of the code can be parallelized, maximum speedup = 2, meaning the code will run twice as fast.
- Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

where P = parallel fraction, N = number of processors and S = serial fraction.

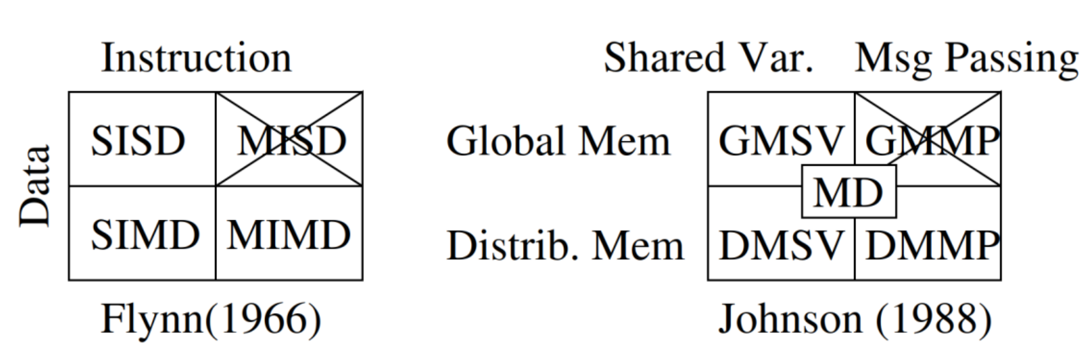
Johnson Taxonomy (Classification)

- Johnson further refined the MIMD model by using the memory system and communications as criteria:



Johnson Taxonomy (Classification)

- **Shared Memory MIMD** (*tightly coupled multiprocessor systems*)
  - GMSV (Global/Shared Memory, Shared Variables)
  - GMMP (Global/Shared Memory, Message Passing)
- **Distributed Memory MIMD** (*loosely coupled multicomputer systems*)
  - DMSV (Distributed Memory, Shared Variable)
  - DMMP (Distributed Memory, Message Passing)



Shared-Variable vs Message-Passing

- Shared-Variable Technique
  - **Common variables are shared among threads/cores**
  - **Synchronization** becomes very important between threads.
  - Threads need to work together but not interfering
- Message-Passing Technique
  - In the case of Distributed Computing System there is **no concept of shared memory**.
  - In this model, the only way to communicate information with another node is **to send data over the network**. We say that nodes are passing messages to each other.
  - Unlike threads, **synchronization happens automatically** as part of the message passing process.

Johnson Taxonomy

- Global Memory-Shared Variables (GMSV) -- **Shared memory machines**
- Distributed Memory-Message Passing (DMMP) -- **Message passing machines**
- Distributed Memory-Shared Variables (DMSV) -- **Hybrid machines - Memory is physically distributed but appears logically shared**

Johnson's Taxonomy

- **GMSV** Symmetric Multi-Processors (OpenMP, PThreads)
- **DMMP** Distributed systems (If simulated: Omnet++, NS2/3, etc.), NUMA based Cluster systems (OpenMPI, RPC), GPU Computing (OpenCL, CUDA), HPC based systems
- **DMSV** Cloud Model: Memory is distributed but access is through same address space (Hadoop, MapReduce)

Applications and Examples

- **General Purpose**
  - Word processor (spell check, auto save, . . .)
  - Spreadsheet (automatic background recalculations after cell edits)
  - Operating system (User threads, Kernel threads, . . .)
  - Server (multi client handling
  - GUI’s (Event listeners)
  - Video games (rendering of animation, taking care of physics, AI)
  - Web browser (tabs, multi-segment download accelerators, . . .)
  - . . .

Applications and Examples

- **Scientific Purposes**
  - Mathematical problems (Ordinary differential equations, partial differential equations, linear algebra, numerical analysis)
  - Climate modelling and weather prediction
  - Fluid dynamics (video games, simulation of dam bursts, tsunamies, etc.)
  - Computational Biology, Finance, Physics, Chemistry
  - Machine learning, statistical methods
  - Image processing (medical images, spectral images, satellite images)
  - Finding Aliens, Finding cure for Alzheimers, Cancers, . . .