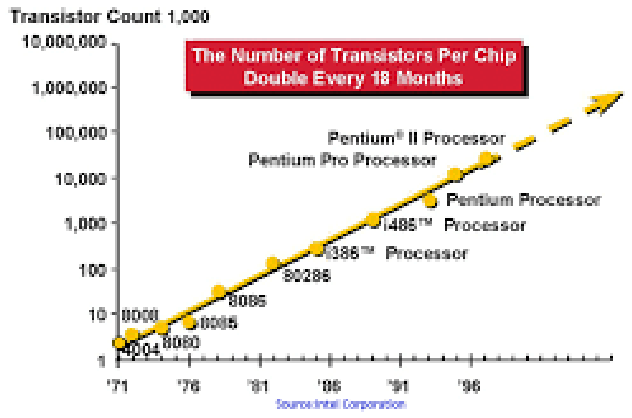
Moore's Law and Parallel Computer Classification

1. Moore's Law

Moore's Law was observed by Gordon Moore in 1965, stating that the number of transistors on a microchip doubles approximately every 18 to 24 months, leading to exponential growth in computing power and performance.

Formula:
Number of Transistors ∝ 2^(Time / 18 months)

Implications:
- Faster processors
- More memory capacity
- Cheaper and smaller devices
- Increased energy efficiency over time



Applications of Moore's Law include consumer electronics, AI advancements, cloud computing, IoT devices, healthcare diagnostics, and autonomous vehicles.

2. Classification of Parallel Computers

A. Based on Memory Architecture

1. Shared Memory Systems:
- Processors share a global memory space.
- Examples: Multi-core CPUs, SMP systems.

2. Distributed Memory Systems:
- Each processor has private memory.
- Communicate via message passing (e.g., MPI).
- Examples: Clusters, MPP.

3. Hybrid Systems:
- Combine shared and distributed memory.
- Examples: IBM Summit, Fugaku.

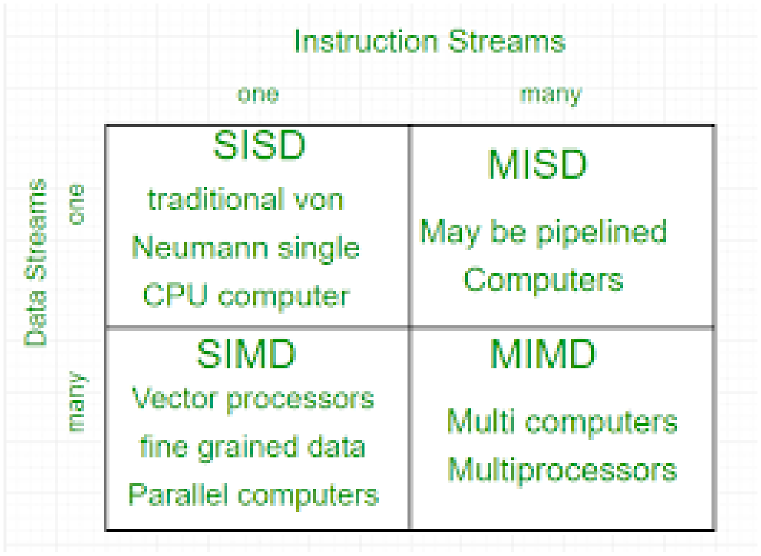
B. Based on Processing Element Organization (flynn's classification)

1. SISD (Single Instruction Single Data):
- Classic Von Neumann architecture.
- Example: Traditional single-core CPU.

2. SIMD (Single Instruction Multiple Data):
- One instruction operates on multiple data streams.
- Example: GPUs used for image processing.

3. MISD (Multiple Instruction Single Data):
- Multiple instructions on the same data stream.
- Rarely used in practice.

4. MIMD (Multiple Instruction Multiple Data):
- Multiple processors execute different instructions on different data.
- Examples: Multi-core CPUs, Supercomputers.


Instruction Streams
one
many
Data Streams
one
many
SISD
traditional von Neumann single CPU computer
MISD
May be pipelined Computers
SIMD
Vector processors fine grained data Parallel computers
MIMD
Multi computers Multiprocessors

C. Based on Interconnection Network

1. Bus-based Interconnection:
- Simple but limited bandwidth as processors increase.

2. Crossbar Switch:
- Provides dedicated paths between processors but is expensive.

3. Network Topologies:
- Mesh (2D/3D grid)
- Torus
- Hypercube
- Fat-tree

Summary Table

Classification	Examples
Shared Memory	Multi-core CPU, SMP
Distributed Memory	Cluster with MPI
SIMD	GPUs
MIMD	Supercomputers, Multi-core CPUs
Hybrid	IBM Summit, Fugaku
Interconnects	Mesh, Torus, Hypercube

Performance-Based Computing

Performance-Based Computing refers to designing and optimizing computer systems and architectures to achieve the highest possible performance for specific applications or workloads. The goal is to maximize speed, throughput, efficiency, and responsiveness.

- **Throughput:** Amount of work completed per unit time (e.g., transactions per second).
- **Latency:** Time taken to complete a single operation or task.
- **Scalability:** The ability to maintain or increase performance as resources are added (e.g., more processors).
- **Efficiency:** How well a system uses its resources (CPU, memory, bandwidth) to perform tasks.

Performance Metrics:

- **FLOPS (Floating Point Operations Per Second):** Measures raw compute capability.
- **IPC (Instructions Per Cycle):** Average number of instructions executed per clock cycle.
- **Benchmarking:** Running standard tests to compare performance of different architectures (e.g., SPEC CPU, LINPACK).

Strategies to Improve Performance:

1. **Pipelining:** Overlapping the execution of multiple instructions.
2. **Superscalar Execution:** Multiple instructions processed in parallel in a single processor cycle.
3. **Parallelism:** Using multiple processors or cores to divide work.
4. **Caching:** Reducing memory access latency by storing frequently used data closer to the CPU.
5. **Specialized Hardware:** Using GPUs, TPUs, or FPGAs for tasks like AI, graphics, or signal processing.

Example:

- A server running AI inference uses **TPUs (Tensor Processing Units)** optimized for matrix multiplications, achieving much higher performance per watt compared to a general-purpose CPU.

The Myopic View of Computer Architecture:

The Myopic View of Computer Architecture refers to the narrow focus on improving processor speed (clock rate, transistor count) at the expense of considering broader system-level performance factors like memory bandwidth, power consumption, I/O limitations, and parallelism.

- Historically, computer architects focused primarily on making processors faster by increasing clock speed and transistor density (following Moore's Law).
- However, this led to problems such as:
  - **Memory Wall:** CPU speeds increasing faster than memory speeds, creating bottlenecks.
  - **Power Wall:** Increased power consumption and heat dissipation with higher clock speeds.
  - **Diminishing Returns:** Adding more complex features (like speculative execution) increases hardware complexity with marginal performance gain.

Why Is It Called Myopic?

- **Myopic** means short-sighted. Architects focusing solely on improving the CPU ignored important aspects like:
  - Parallel computing
  - Energy efficiency
  - Memory hierarchy design
  - I/O system optimization

Modern Shift:

- The focus shifted towards balanced system design:
  - Multi-core architectures
  - Heterogeneous computing (CPUs + GPUs + accelerators)
  - Energy-efficient design
  - Optimizing data movement rather than just compute speed