ADVANCED ARCHITECTURES

PARALLEL ARCHITECTURES: INTRODUCTION AND CLASSIFICATION

Annalisa Massini

Lecture 12

2025-2026

What we will see in this and the next lessons

- Motivation to parallel architectures
- Classification of parallel architectures: Flynn's Taxonomy and other classifications
- SIMD class. Vector processors. Manycore architectures: GPU (and CUDA)
- MIMD class
- Interconnection topologies and interconnection networks
- Performance metrics and measurement

Textbooks

- ***Advanced Computer Architecture and Parallel Processing***

  H. El-Rewini, M. Abd-El-Barr, John Wiley and Sons, 2005

- ***Parallel computing for real-time signal processing and control (Ch. 2 Parallel Architectures)***

  M. O. Tokhi, M. A. Hossain, M. H. Shaheed, Springer, 2003

- ***Multicore and GPU Programming An Integrated Approach***

  G. Barlas - Morgan Kaufmann, 2014

- ***Parallel Computer Architecture: A Hardware/Software Approach***

  D.E. Culler, J. P. Singh, A. Gupta - Morgan Kaufmann, 1998

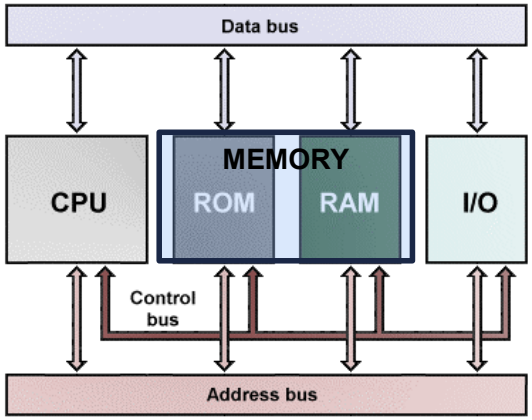
INTRODUCTION

Parallel Computer Architecture: A Hardware/Software Approach

D.E. Culler, J. P. Singh, A. Gupta - Morgan Kaufmann, 1998

Motivations to Parallel Architectures

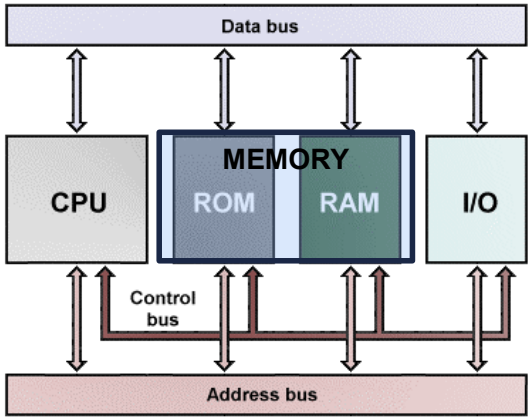
- The leading character of the growth in performance and capability of computer systems is **parallelism**
- Starting from the knowledge of the **conventional computer architecture**, we are interested in
  - Acquiring an **understanding** and **appreciation** of a computer system
  - Learning to harness **parallelism to sustain performance improvements**



Motivations to Parallel Architectures

In fact, the design of *parallel algorithms* and the study *of strategies for problem decomposition* **are sustained by:**

- A deep **knowledge of the computer architecture**
- A careful **use of parallelism**
- The **performance analysis**



Motivations to Parallel Architectures

- **Parallel computer architecture** forms an important thread in the evolution of computer architecture
- It has its roots in the beginnings of computing, and exploits *advancement* over what the *base technology* can provide
- Parallel computer designs have demonstrated a rich ***diversity of structure***, usually motivated by specific higher level parallel programming models
- The **speed** with which computers can process information has been **increasing exponentially** over the time

Motivations to Parallel Architectures

Role of a computer architect:

- To design and engineer the various levels of a computer system to maximize ***performance*** and ***programmability*** within limits of ***technology*** and ***cost***

Parallelism:

- Provides an *interesting perspective* from which to understand computer architecture
- Provides *alternative to faster clock* for performance
- Applies at *all levels of system design*
- Is *increasingly central* in information processing

Motivations to Parallel Architectures

- **A *parallel computer* is a collection of processing elements** that cooperate to solve large problems fast

- This simple definition raises *many questions*

Motivations to Parallel Architectures

- **A *parallel computer* is a collection of processing elements** that cooperate to solve large problems fast

- This simple definition raises *many questions*

- **Resource Allocation**
  - how large is the collection?
  - how powerful are the elements?
  - how big is memory?

Motivations to Parallel Architectures

- **A *parallel computer* is a collection of processing elements** that cooperate to solve large problems fast

- This simple definition raises *many questions*

- **Data access, Communication and Synchronization**
  - how do the elements cooperate and communicate?
  - how are data transmitted between processors?
  - what are the abstractions and primitives for cooperation?

Motivations to Parallel Architectures

- **A *parallel computer* is a collection of processing elements** that cooperate to solve large problems fast

- This simple definition raises *many questions*

- **Performance and Scalability**
  - how does it all translate into performance?
  - how does it scale?

Motivations to Parallel Architectures

- To understand parallel architectures, it is important to examine*:*
  - the principles of computer design at the processor level
  - the design issues present for each of the system components
    - memory systems
    - processors
    - networks
  - the relationships between these components
  - the division of responsibilities between hardware and software

Motivations to Parallel Architectures

- **Parallel machines** have been built at **various scales** since the earliest days of computing, but the approach is more viable today than ever before

- In fact
  - Whatever the performance of a single processor at a given time → **higher performance** can be achieved by utilizing many processors
  - But now the basic **processor** building block is **better suited** to the job

- How much additional performance is gained and at what additional cost ***depends on a number of factors***

CLASSIFICATION

Advanced Computer Architecture and Parallel Processing

H. El-Rewini, M. Abd-El-Barr, John Wiley and Sons, 2005

Parallel Architectures

- ***Parallel processors*** are computer systems consisting of
  - multiple **processing units**
  - connected via some **interconnection network**
  - plus the **software** needed to make the processing units work together

- There are two major factors used to categorize such systems:
  - the **processing units** themselves
  - the **interconnection network** that ties them together

Parallel Architectures

- A **vast number parallel architecture** types have been devised
- Various types of parallel architecture have **overlapping characteristics to different extents**

- It is ***not easy*** to develop a simple **classification** system for parallel architectures

Parallel Architectures

- Parallel architecture can be distinguished under the following broad categories:
  - **Flynn's classification**
  - Classification based on **memory** arrangement
  - Classification based on **interconnections** among PEs and memory modules
  - Classification based on characteristic nature of **PEs**

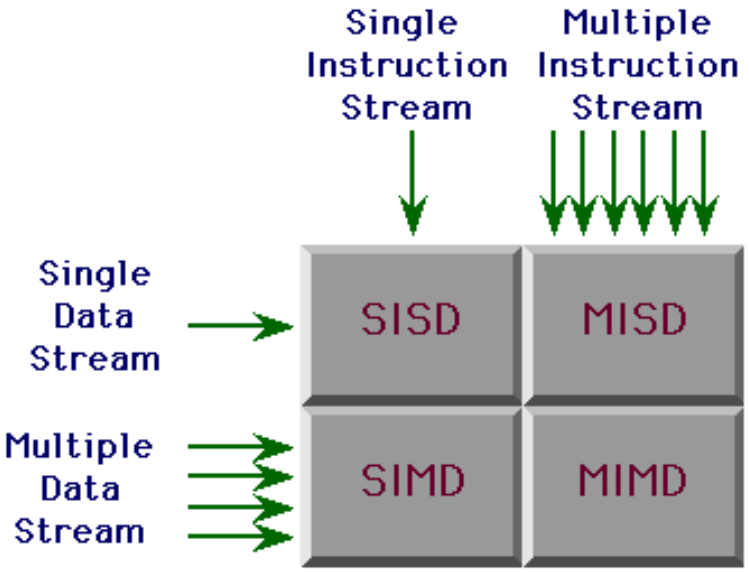
Flynn's classification

- Flynn's classification is based on the notion of a **stream of information**
- There are two types of information flow into a processor:
  - **instruction stream -** defined as the sequence of instructions performed by the processing unit
  - **data stream -** defined as the data traffic exchanged between the memory and the processing unit

- Either of the instruction or data streams can be single or multiple

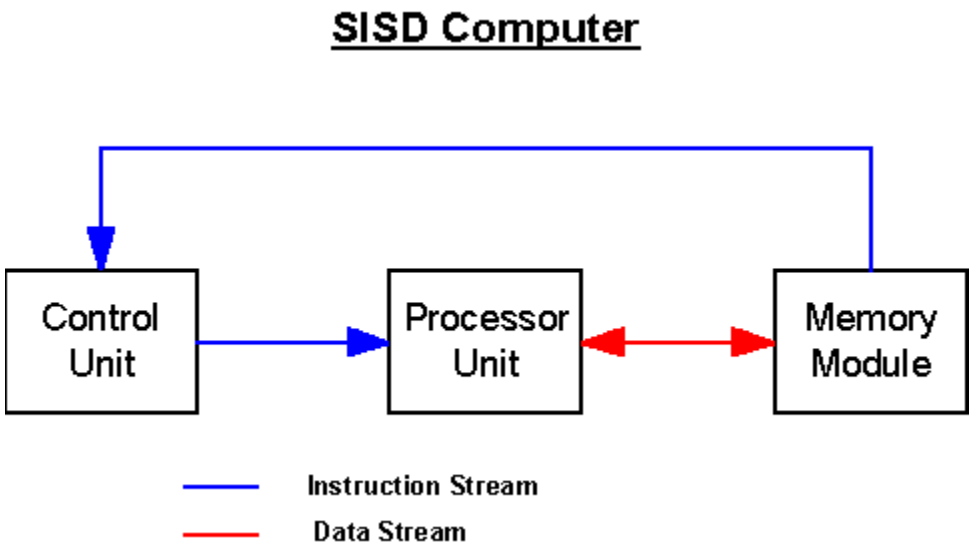
Flynn's classification

- **Four distinct categories**:
  - Single-Instruction Single-Data streams (**SISD**)
  - Single-Instruction Multiple-Data streams (**SIMD**)
  - Multiple-Instruction Single-Data streams (**MISD**)
  - Multiple-Instruction Multiple-Data streams (**MIMD**)



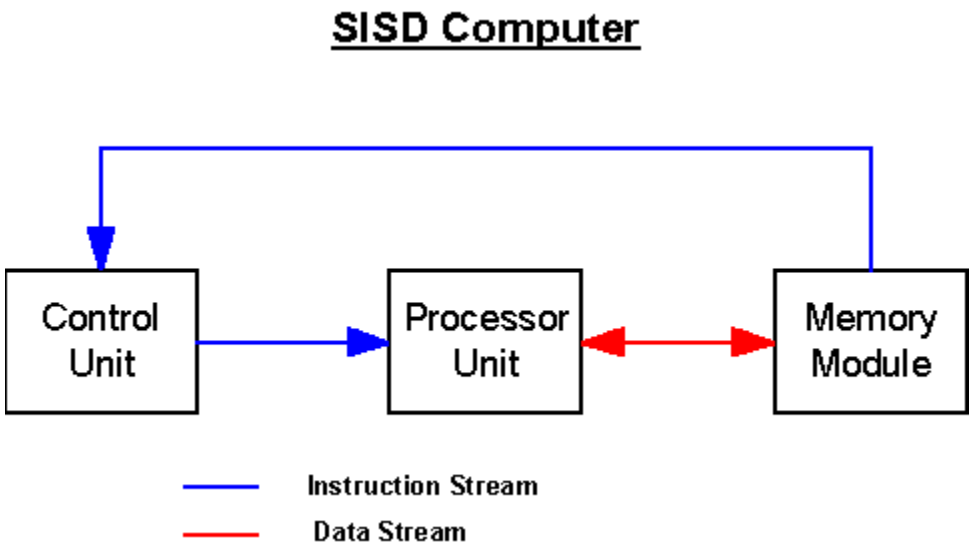
Single Instruction, Single Data Stream - SISD

- **Single** sequential processor *executes a*
  Single instruction stream *to operate on*
  data stored in **single** memory



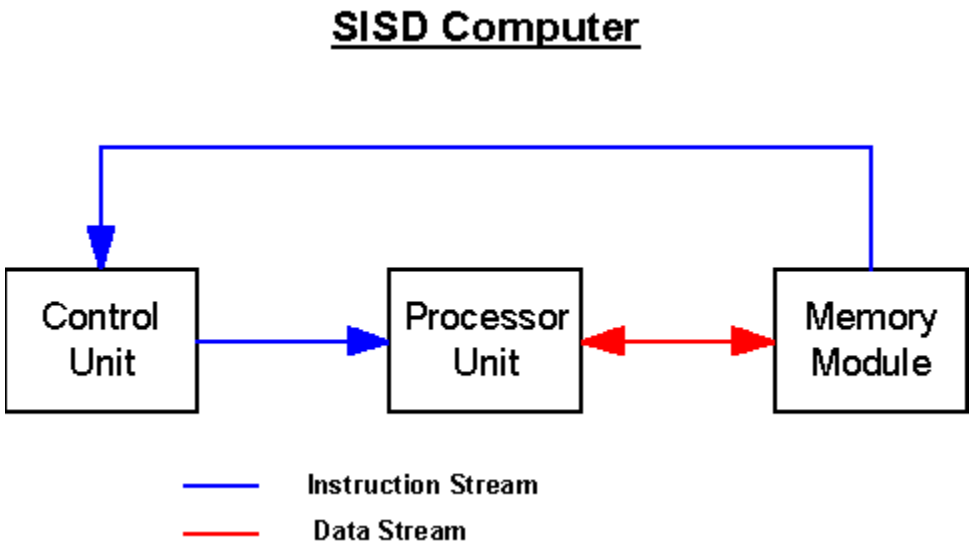
Single Instruction, Single Data Stream - SISD

- During program execution the ***PE***
  - *fetches instructions and data* from the main memory
  - *processes the data* according to the instructions
  - *sends the results to the main memory* after processing has been completed



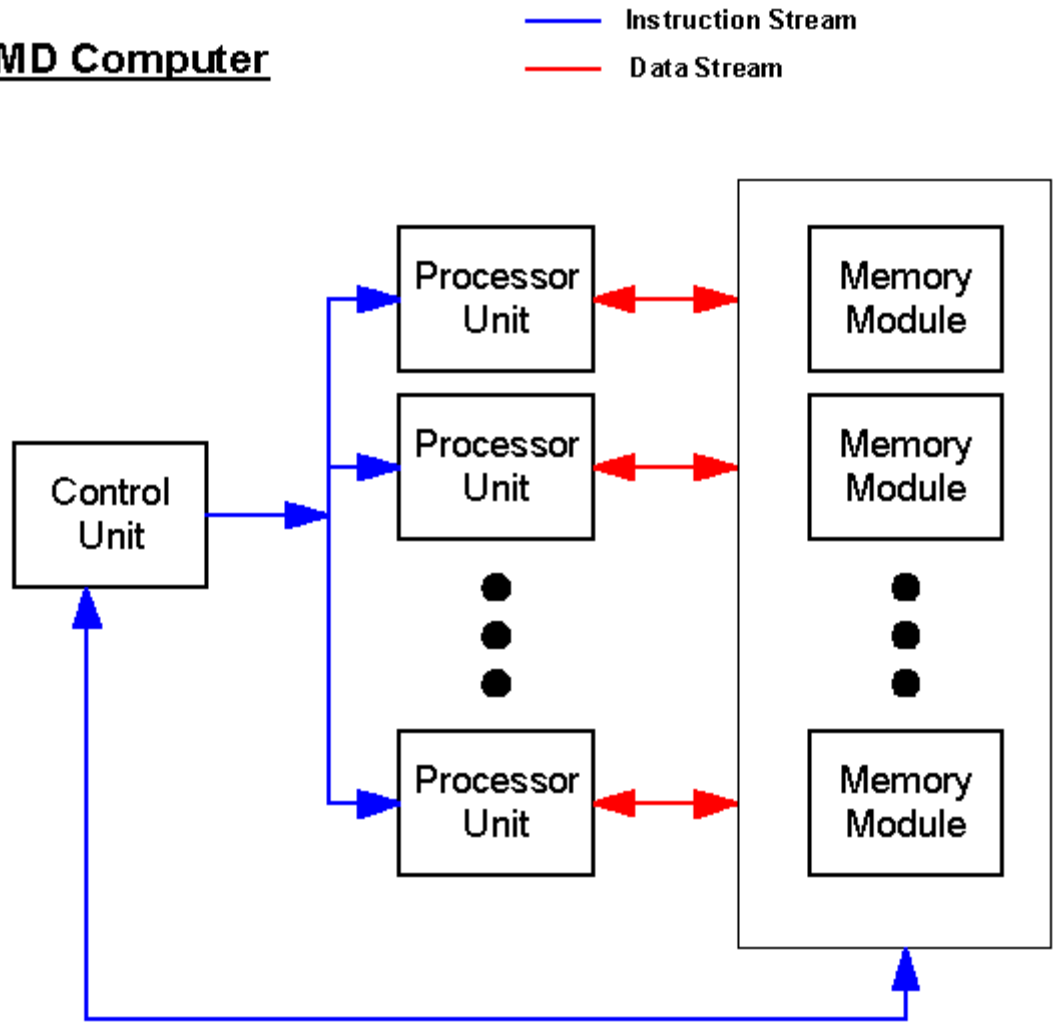
Single Instruction, Single Data Stream - SISD

- The conventional single-processor **Von Neumann computers** falls under this category
- Today's conventional uniprocessors actually have several operations *in flight* at any one time
- In fact, the majority of contemporary CPUs is multicore and not only a **single core CPU** can be considered a SISD machine



Single Instruction, Multiple Data Stream - SIMD

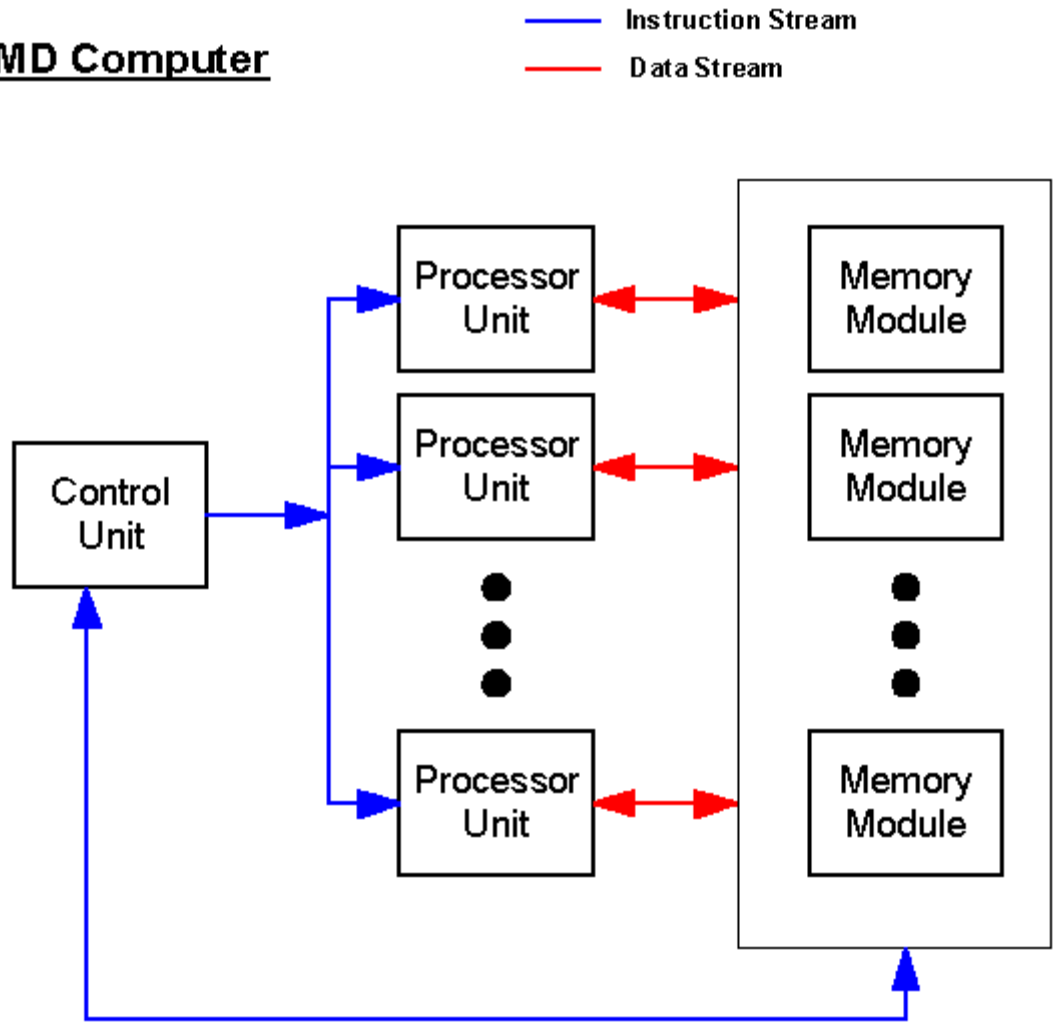
- A single instruction stream **provides parallelism** by operating on multiple data streams concurrently
- A ***single machine instruction*** controls the **simultaneous execution** of a number of processing elements on a lockstep basis



Single Instruction, Multiple Data Stream - SIMD

- Each processing element has an associated data memory

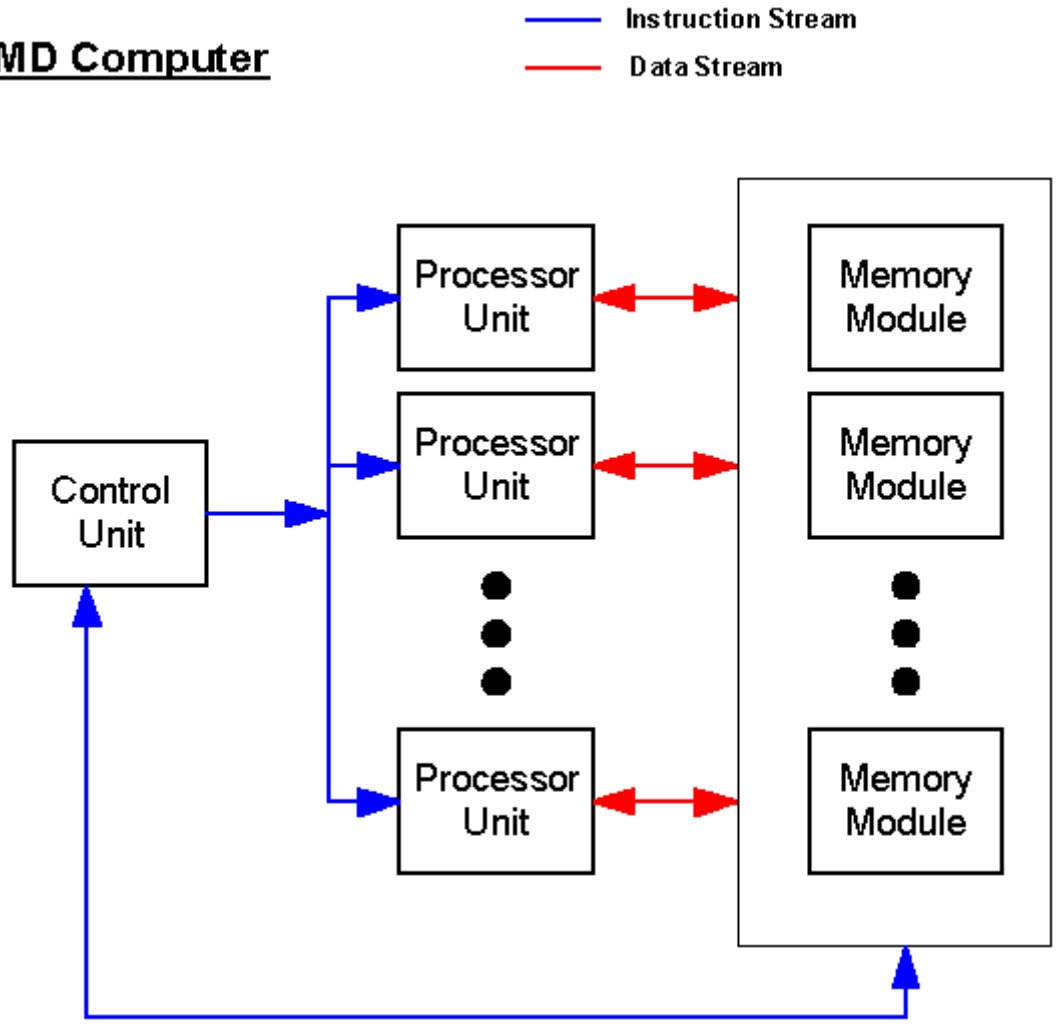
- ***Each instruction*** is executed on a ***different set of data*** by the ***different processors/cores***



Single Instruction, Multiple Data Stream - SIMD

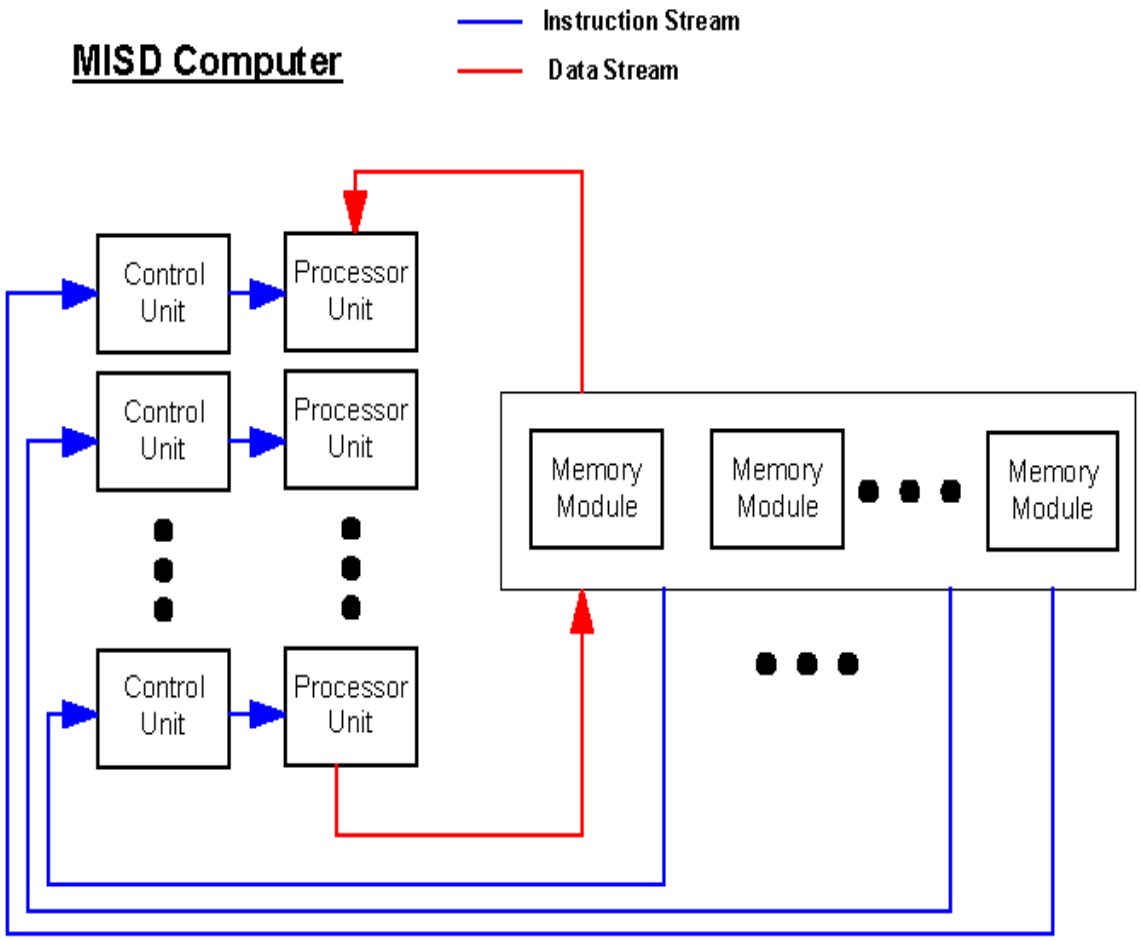
- **Vector processors** were the first SIMD machines
- **GPUs** follow this design at the level of Streaming Multiprocessor (SM)

- Applications:
  - Image processing
  - Matrix manipulations
  - Sorting



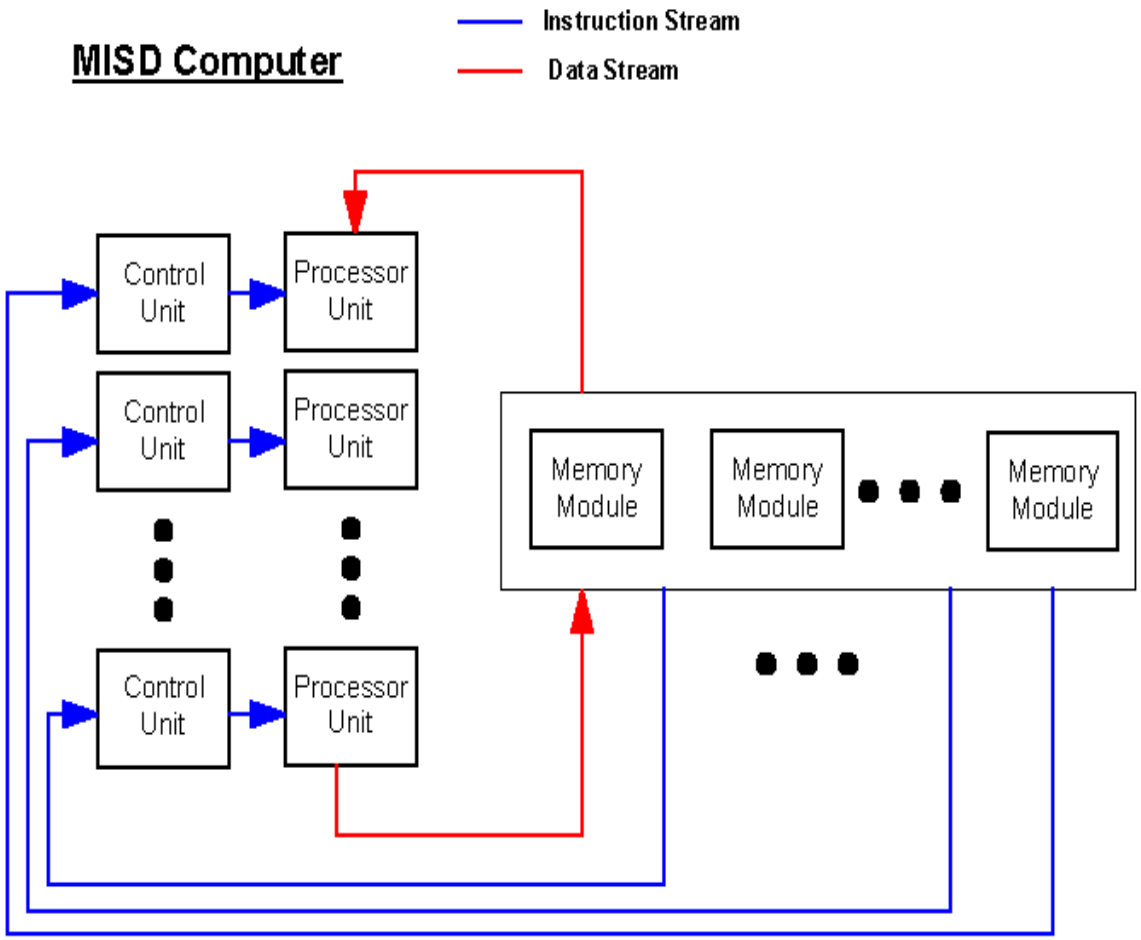
Multiple Instruction, Single Data Stream - MISD

- This is a controversial category
- A ***sequence of data*** is transmitted to a ***set of processors***, each of which *executes a* ***different*** *instruction* sequence
- This structure is not *commercially* implemented



Multiple Instruction, Single Data Stream - MISD

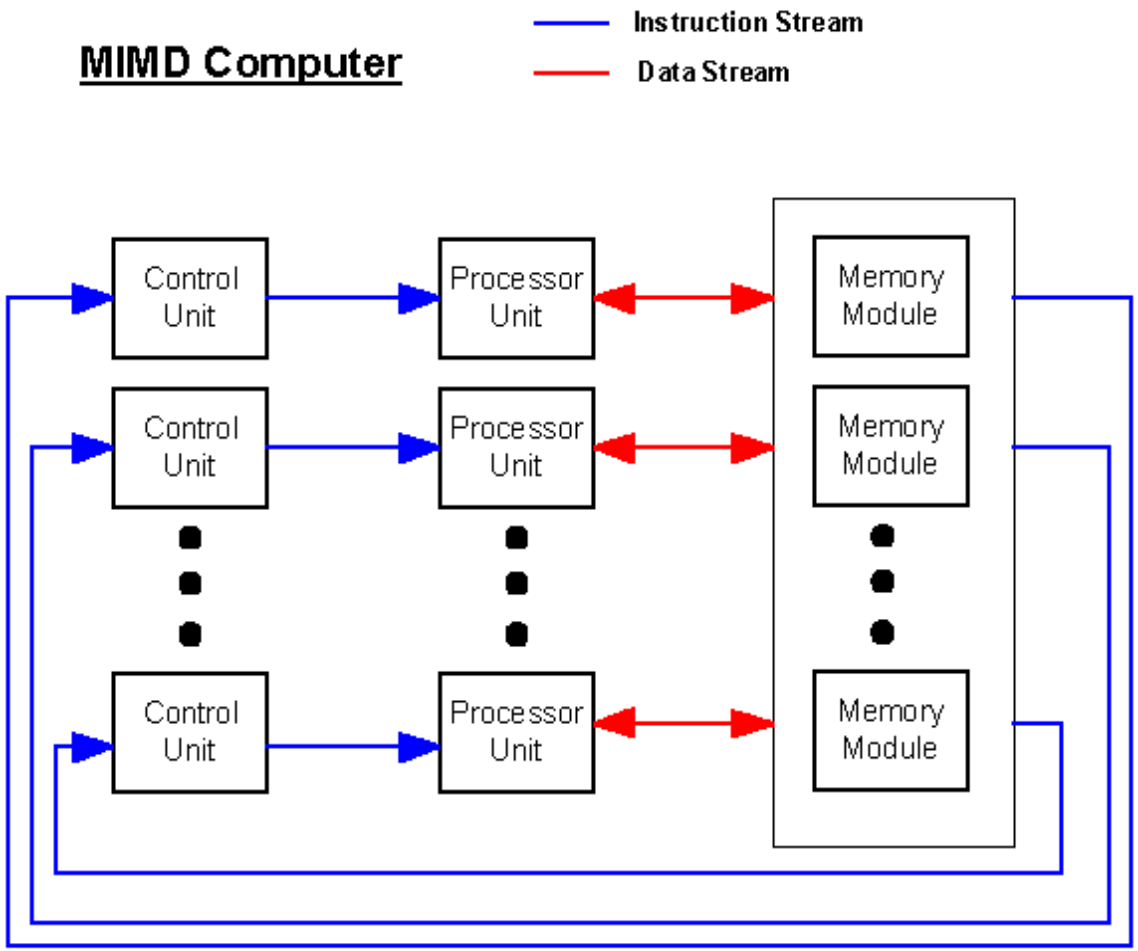
- MISD computers can be useful in **applications** of a **specialized nature**:
  - robot vision
  - when ***fault tolerance*** is required (military or aerospace application) data can be processed by multiple machines and decisions can be made on a majority principle



Multiple Instruction, Multiple Data Stream- MIMD

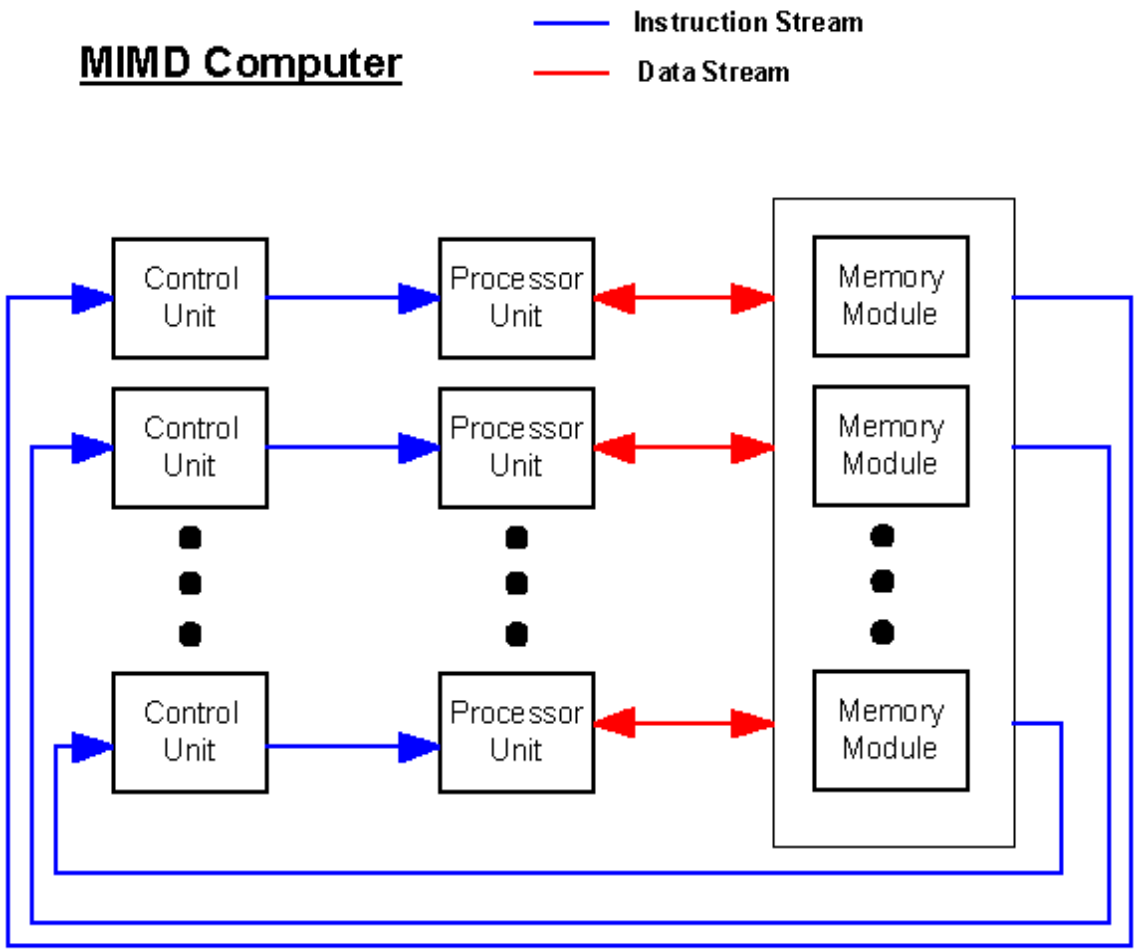
- A **set of processors** simultaneously execute **different instruction sequences** on **different data sets**

- This architecture is the most common and widely used form of parallel architectures



Multiple Instruction, Multiple Data Stream- MIMD

- In the MIMD organization, the processors are general purpose
- Each processor can process all instructions necessary
- **Further classified** by method of processor communication



Flynn's classification

- **Advantages of Flynn**
  - Universally accepted
  - Compact Notation
  - Easy to classify a system

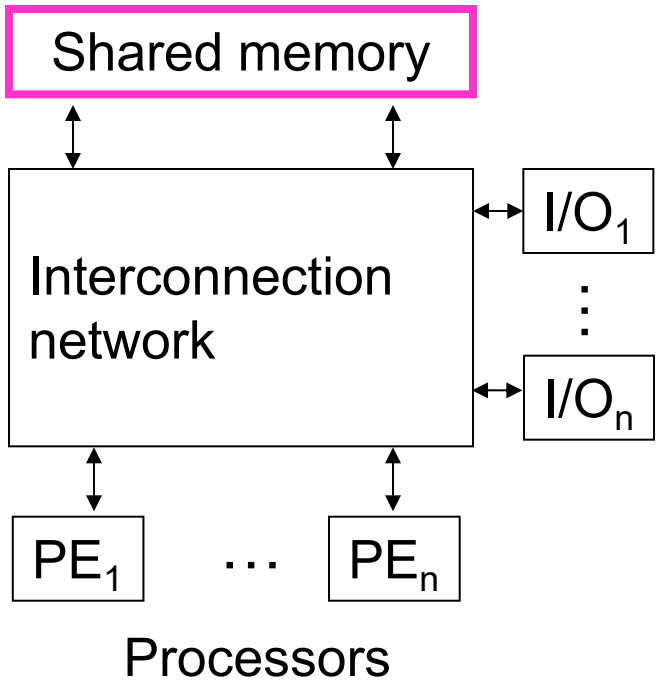
- **Disadvantages of Flynn**
  - Very coarse-grain differentiation among machine systems
  - Comparison of different systems is limited
  - Interconnections, I/O and memory not considered in the scheme

Classification based on memory arrangement

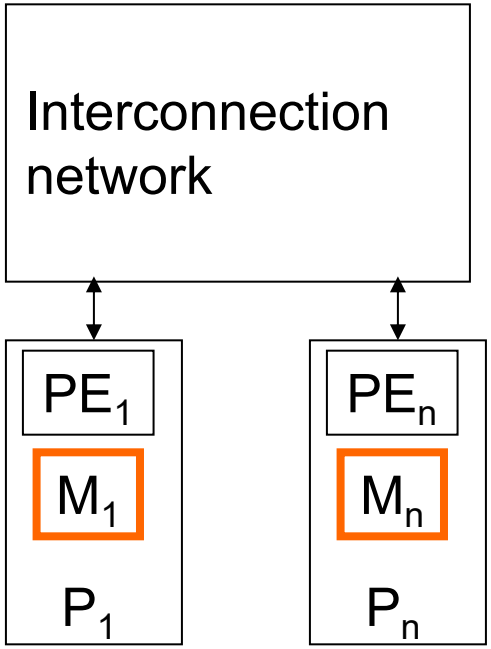
- Parallel architectures can be classified into two major categories in terms of **memory arrangement**:
  - **Shared memory**
  - **Distributed memory** or message passing
- This classification constitutes a subdivision of MIMD parallel architecture and are also known as:
  - *Shared memory* architecture → *tightly coupled* architecture
  - *Distributed memory* architecture → *loosely coupled* architecture

- A third choice is somewhere in between: groups of processors share a memory block while the different groups, also referred to as *nodes*, have distinct memory blocks

Classification based on memory arrangement



Shared memory
multiprocessors



Distributed memory
multicomputers

Shared Memory Multiprocessor

- A **shared-memory multiprocessor** is an architecture consisting of a (modest) number of processors, all of which have direct (hardware) access to all the main memory in the system

- The **memory unit** can comprise **one or more memory modules**
- All the **processors:**
  - have **equal access to the memory modules,**
  - can access data that any of the other processors has created
- The memory modules
  - **store data**
  - are seen as a **single address space** by **all the processors**
  - **allow communication** among the processors via

Shared Memory Multiprocessor

- The key to this form of multiprocessor architecture is the **interconnection network** that directly connects all the processors to the memories
- **Communication** is established through memory access instructions
- There is no direct processor-to-processor communication involved in the programming process

- Processors communicate by *reading* and *writing data in locations into the shared memory*
- The executable programming codes are stored in the memory for each processor to execute
- The data related to each program is also stored in this memory

Shared Memory Multiprocessor

- Each processor may have *registers*, *buffers*, *caches*, and *local memory banks* as additional memory resources
- A number of basic issues in the design of shared memory systems have to be taken into consideration: access control, synchronization, protection, and security
- Access to the memory modules can easily be controlled through appropriate programming mechanisms
- However, this architecture suffers from a **bottleneck** problem when a **number of processors** try to access the global **memory** at the **same time**
- This **limits the scalability** of the system but *nonblocking networks* as crossbars can be used to improve scalability

Shared Memory Multiprocessor

- Depending on the interconnection network, a shared memory system leads to systems that can be classified as:
  - **uniform** memory access (UMA) architecture
  - **non-uniform** memory access (NUMA) architecture
- In the UMA system
  - a shared memory is accessible by all processors through an interconnection network in the same way a single processor accesses its memory
  - the memory access time to the different parts of the memory are almost the same
- **UMA** architectures are also called **symmetric multiprocessors**

Shared Memory Multiprocessor

- An **UMA** architecture comprises two or more **processors with identical characteristics**
- The processors:
  - share the same main memory and I/O facilities
  - are interconnected by networks as some form of bus-based, crossbars, or multiport memories

- Processors perform the same functions under control of an operating system, which provides interaction between processors and their programs at the job, task, file and data element levels

Shared Memory Multiprocessor

- In the case of **NUMA** architectures, memory is *physically distributed* but ***logically shared***
- Each processor has part of the shared memory attached, but the memory has a single address space, therefore, any processor could access any memory location directly
- The **memory access time** of processors **differs** depending on which region of the main memory is accessed

- A subclass of NUMA system is **cache coherent NUMA** (cc-NUMA) where cache coherence is maintained among the caches of various processors
- The main advantage of a cc-NUMA system is that it can deliver effective performance at higher levels of parallelism

Shared Memory Multiprocessor

- Anyway, **cache coherence** mechanisms are required in all cache-based multiprocessor systems, whether they are of the UMA or the ccNUMA kind
- This is because copies of the same cache line could potentially reside in several CPU caches
- Cache coherence ensures that any change in the data of one cache is reflected by some change to all other caches that may have a copy of the same global data location

- **Cache coherence protocols** ensure a consistent view of memory under all circumstances
- The interconnection network that provides cache coherence may employ any one of several techniques

Shared Memory Multiprocessor

An example is the **MESI protocol**, whose name is from the four possible states a cache line can assume:

- **M modified**: The cache line has been modified in this cache, and it resides in no other cache than this one. Only upon eviction will memory reflect the current state.
- **E exclusive**: The cache line has been read from memory but not (yet) modified. However, it resides in no other cache.
- **S shared**: The cache line has been read from memory but not (yet) modified. There may be other copies in other caches of the machine.
- **I invalid**: The cache line does not reflect any sensible data. Under normal circumstances this happens if the cache line was in the shared state and another processor has requested exclusive ownership.

Message Passing Multicomputer

- In a **distributed memory architecture** each unit is a complete computer building block including the processor, memory and I/O system
- Units are referred to as ***nodes***, and there is no sharing of memory between them
- Nodes are typically able to store messages in buffers (temporary memory locations where messages wait until they can be sent or received), and perform *send/receive* operations at the same time as processing
- Hence, communication among the processors is established in the form of I/O operations through message signals and interconnection networks

Message Passing Multicomputer

- **Example**
  - If a processor needs data from another processor
  - It sends a signal to that processor through an interconnection network demanding the required data
  - The remote processor then responds accordingly

- Notice that the further the physical distance to the remote processor, the longer it will take to access the remote data
- The processing units of a message passing system may be connected in a variety of ways ranging from architecture-specific interconnection structures to geographically dispersed networks
- The message passing approach is, in principle, scalable to large proportions

Message Passing Multicomputer

- The **speed performance** of distributed memory architecture largely depends upon **how the processors are connected** to each other
- It is impractical to connect each processor to the remaining processors through independent cables, it can work for a very low number of processors
- Message passing multiprocessors employ a variety of static networks in local communication
  - A common solution is to use **specialized bus** networks to connect all the processors in the system in order that each processor can communicate with any other processor attached to the system
  - **hypercube networks** have received special attention for many years
  - **two-dimensional and three-dimensional mesh** networks have been used as well

Classification based on characteristic of PEs

- Parallel architectures are also classified in terms of the nature of the PEs comprising them
- An architecture may consist of either only **one type of PE** or various types of Pes

- The different types of processors that are commonly used to form parallel architectures are:
  - CISC Processors
  - RISC Processors
  - Vector Processors and DSP (Digital Signal Processor)
  - Homogeneous and Heterogeneous Parallel Architectures

Classification based on characteristic of PEs

- **Homogeneous computing systems** employ a single type of processing component to perform all computation
- The majority of supercomputers are of this type

- Systems comprising two or more types of computer cores or nodes are distinguished from homogeneous computing systems that have only one type, and are designated as **heterogeneous systems**
- Accelerators, for example GPUs, can be attached to a system node via the I/O bus and can be accessed by any of the conventional processor cores of the system within the same node