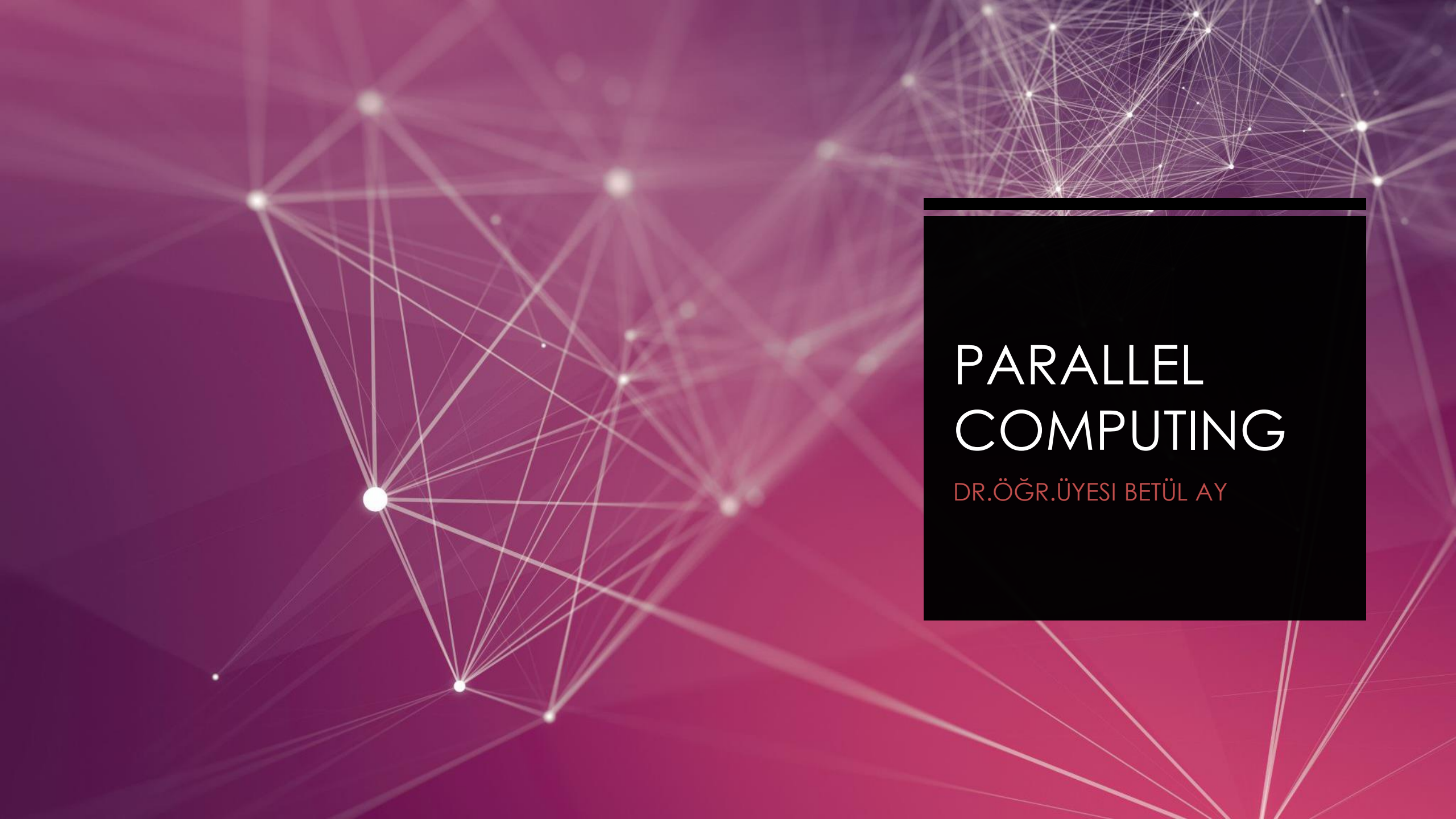


PARALLEL COMPUTING

DR.ÖĞR.ÜYESI BETÜL AY

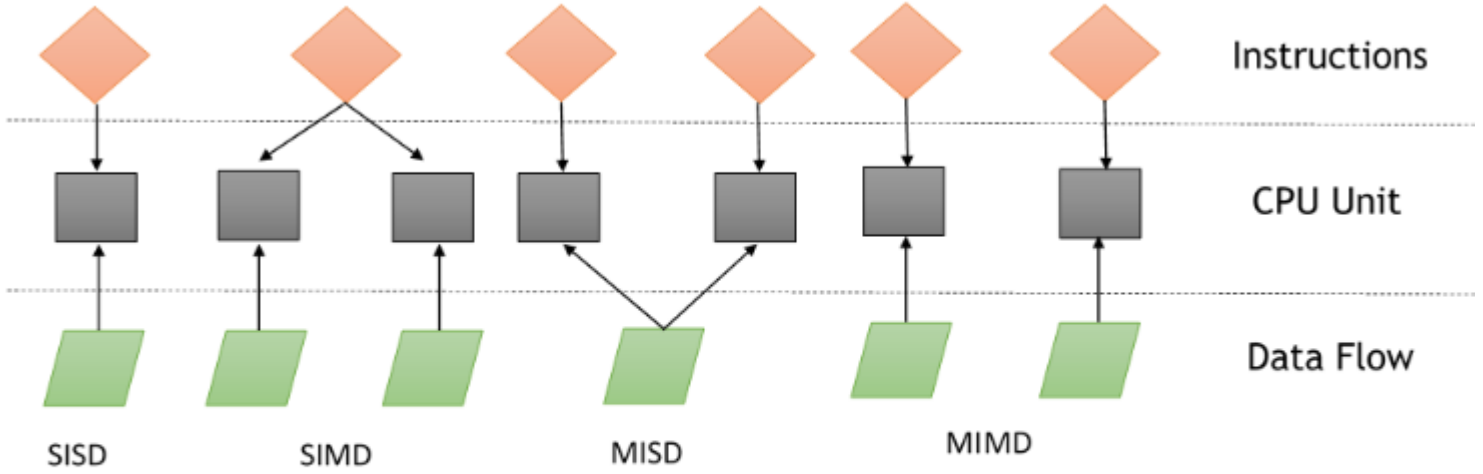
FLYNN'S TAXONOMY

Flynn's taxonomy is a system for classifying computer architectures. It is based on two main concepts:

Instruction flow: A system with n CPU has n program counters and, therefore, n instructions flows. This corresponds to a program counter.

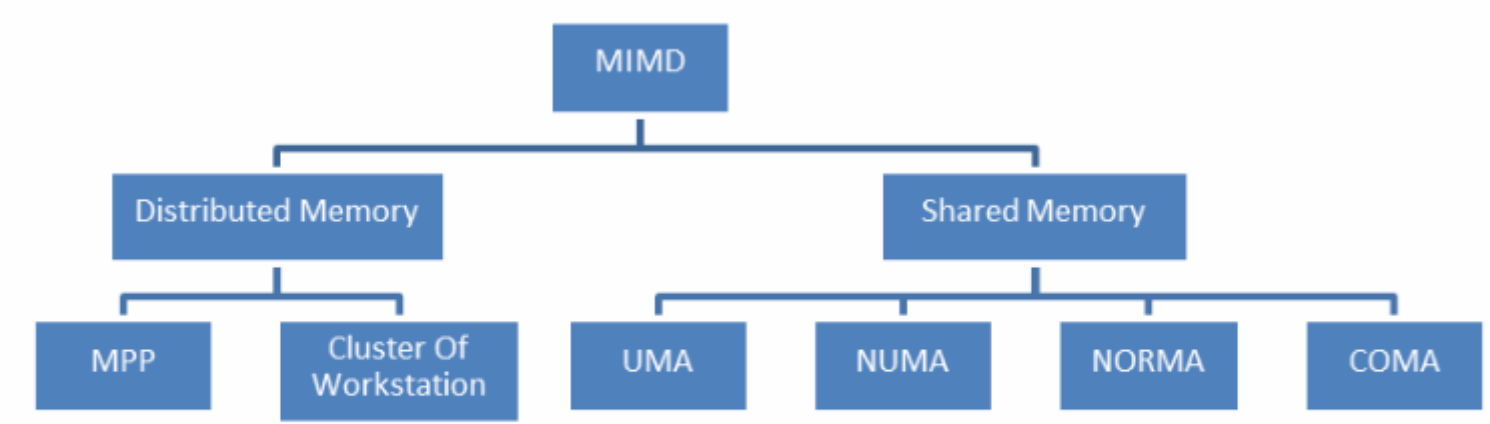
Data flow: A program that calculates a function on a list of data has a data flow. The program that calculates the same function on several different lists of data has more data flows. This is made up of a set of operands.

FLYNN'S TAXONOMY



Flynn's taxonomy

MEMORY ORGANIZATION

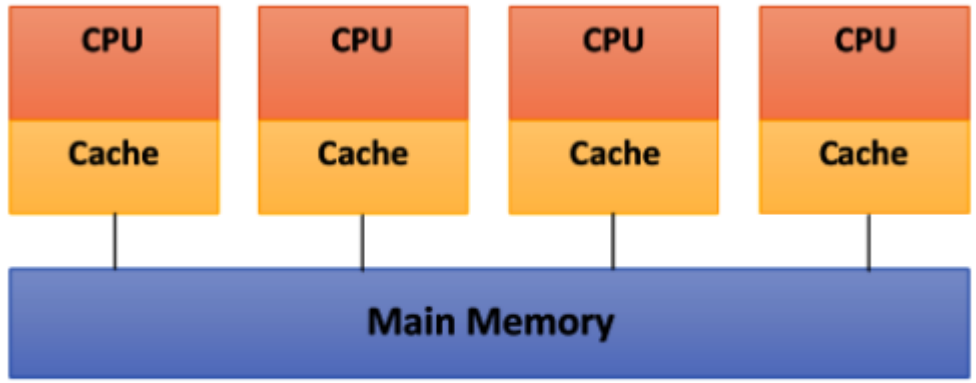


Memory organization in the MIMD architecture

SHARED MEMORY

Main Features:

- The memory is the same for all processors. For example, all the processors associated with the same data structure will work with the same logical memory addresses, thus accessing the same memory locations.
- The synchronization is obtained by reading the tasks of various processors and allowing the shared memory. In fact, the processors can only access one memory at a time.
- A shared memory location must not be changed from a task while another task accesses it.
- Sharing data between tasks is fast. The time required to communicate is the time that one of them takes to read a single location (depending on the speed of memory access).



Shared memory architecture schema

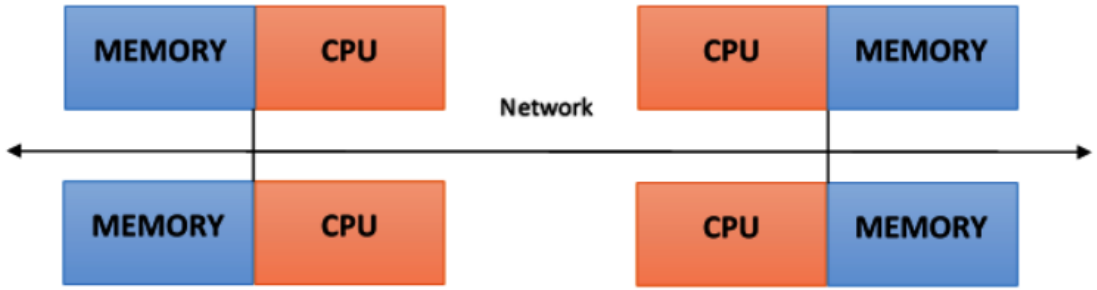
SHARED MEMORY

- **Uniform Memory Access** (**UMA**): The fundamental characteristic of this system is the access time to the memory that is constant for each processor and for any area of memory. For this reason, these systems are also called **Symmetric Multiprocessors** (**SMPs**). They are relatively simple to implement, but not very scalable. The coder is responsible for the management of the synchronization by inserting appropriate controls, semaphores, locks, and more in the program that manages resources.
- **Non-Uniform Memory Access** (**NUMA**): These architectures divide the memory into high-speed access area that is assigned to each processor, and also, a common area for the data exchange, with slower access. These systems are also called **Distributed Shared Memory** (**DSM**) systems. They are very scalable, but complex to develop.
- **No Remote Memory Access** (**NoRMA**): The memory is physically distributed among the processors (local memory). All local memories are private and can only access the local processor. The communication between the processors is through a communication protocol used for exchanging messages, which is known as the **message-passing protocol**.
- **Cache-Only Memory Architecture** (**COMA**): These systems are equipped with only cached memories. While analyzing NUMA architectures, it was noticed that this architecture kept the local copies of the data in the cache and that this data was stored as duplicates in the main memory. This architecture removes duplicates and keeps only the cached memories; the memory is physically distributed among the processors (local memory). All local memories are private and can only access the local processor. The communication between the processors is also through the message-passing protocol.

DISTRIBUTED MEMORY

Main Features:

- Memory is physically distributed between processors; each local memory is directly accessible only by its processor.
- Synchronization is achieved by moving data (even if it's just the message itself) between processors (communication).
- The subdivision of data in the local memories affects the performance of the machine—it is essential to make subdivisions accurate, so as to minimize the communication between the CPUs. In addition to this, the processor that coordinates these operations of decomposition and composition must effectively communicate with the processors that operate on the individual parts of data structures.
- The message-passing protocol is used so that the CPUs can communicate with each other through the exchange of data packets. The messages are discrete units of information, in the sense that they have a well-defined identity, so it is always possible to distinguish them from each other.

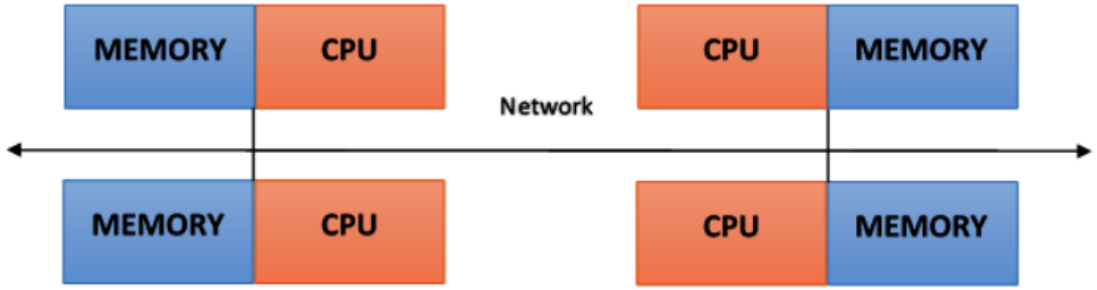


The distributed memory architecture schema

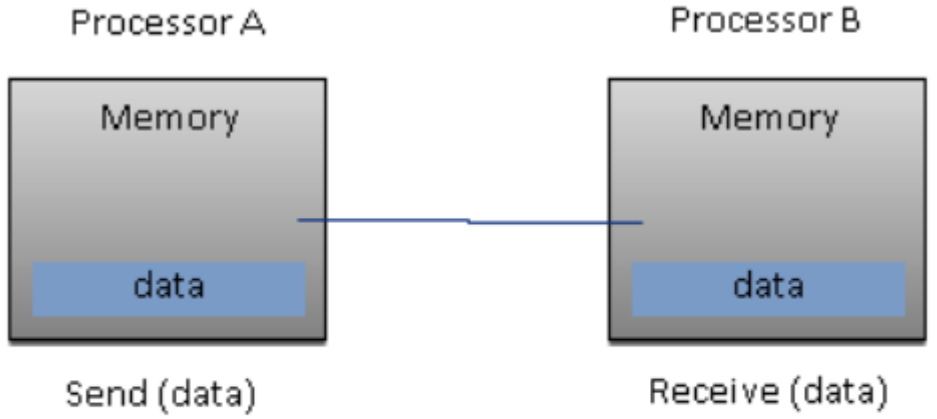
DISTRIBUTED MEMORY

Advantages:

- There are no conflicts at the level of the communication bus or switch. Each processor can use the full bandwidth of their own local memory without any interference from other processors.
- The lack of a common bus means that there is no intrinsic limit to the number of processors. The size of the system is only limited by the network used to connect the processors.
- There are no problems with cache coherency. Each processor is responsible for its own data and does not have to worry about upgrading any copies.



The distributed memory architecture schema



Basic message passing

MASSIVELY PARALLEL PROCESSING (MPP)

- MPP machines are composed of hundreds of processors (which can be as large as hundreds of thousands of processors in some machines) that are connected by a communication network.
- The fastest computers in the world are based on these architectures; some examples of these architecture systems are Earth Simulator, Blue Gene, ASCI White, ASCI Red, and ASCI Purple and Red Storm.

CLUSTERS OF WORKSTATIONS

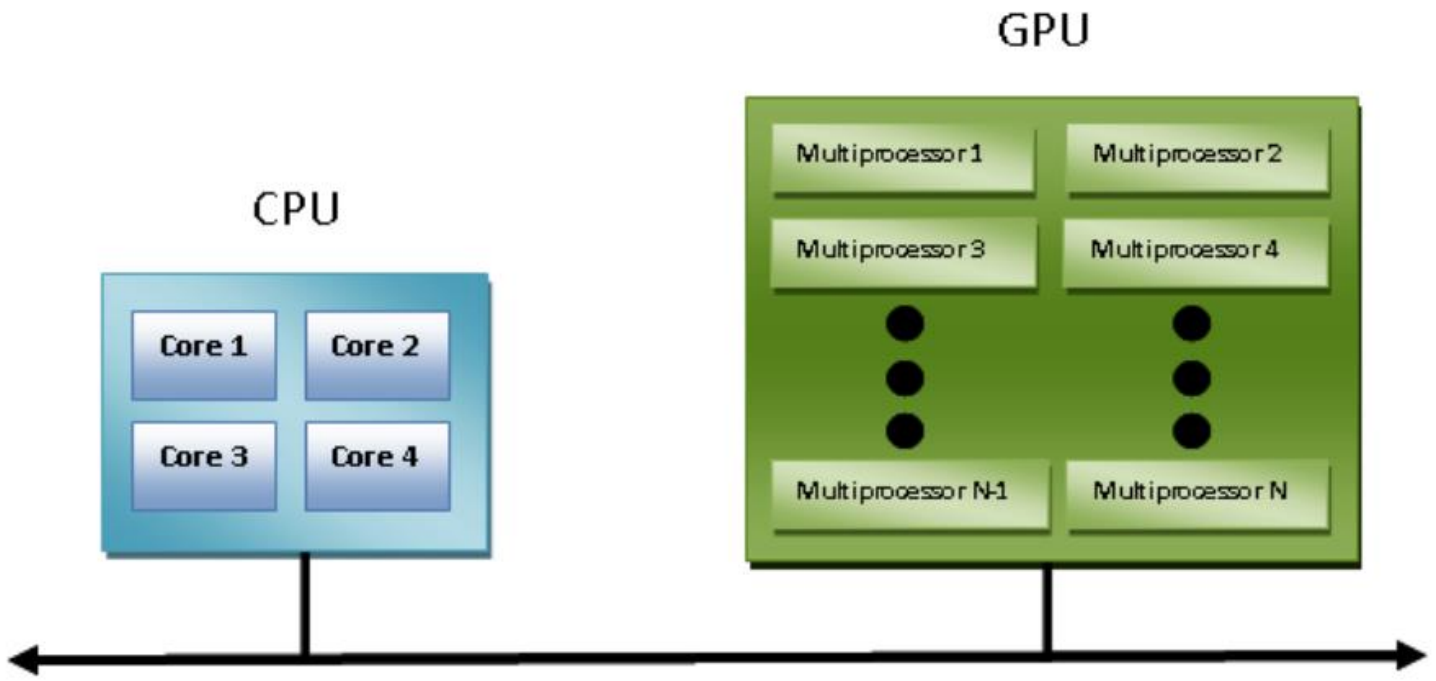
- These processing systems are based on classical computers that are connected by communication networks. Computational clusters fall into this classification.
- In a cluster architecture, we define a node as a single computing unit that takes part in the cluster.
- For the user, the cluster is fully transparent—all the hardware and software complexity is masked and data and applications are made accessible as if they were all from a single node.

CLUSTERS OF WORKSTATIONS

Three types of clusters:

- **Fail-over cluster**: In this, the node's activity is continuously monitored, and when one stops working, another machine takes over the charge of those activities. The aim is to ensure a continuous service due to the redundancy of the architecture.
- **Load balancing cluster**: In this system, a job request is sent to the node that has less activity. This ensures that less time is taken to process the job.
- **High-performance computing cluster**: In this, each node is configured to provide extremely high performance. The process is also divided into multiple jobs on multiple nodes. The jobs are parallelized and will be distributed to different machines.

HETEROGENEOUS ARCHITECTURES



The heterogeneous architecture schema

PARALLEL PROGRAMMING MODELS

The most widely used models for parallel programming are as follows:

- Shared memory model
- Multithread model
- Distributed memory/message passing model
- Data-parallel model

SHARED MEMORY MODEL

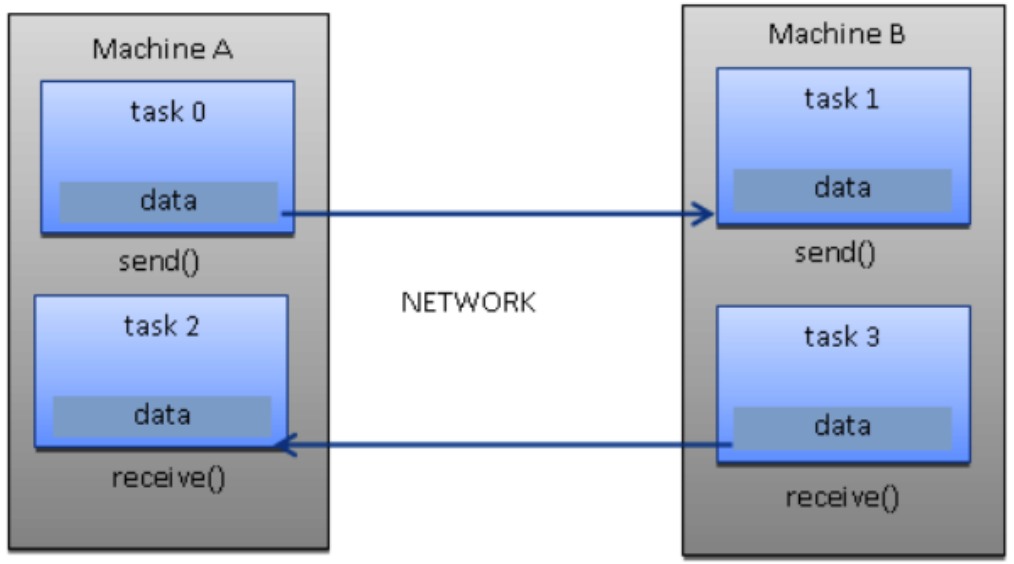
- In this model, tasks share a single memory area in which we can read and write asynchronously.
- There are mechanisms that allow the coder to control the access to the shared memory; for example, locks or semaphores.
- This model offers the advantage that the coder does not have to clarify the communication between tasks.
- An important disadvantage, in terms of performance, is that it becomes more difficult to understand and manage data locality. This refers to keeping data local to the processor that works on conserving memory access, cache refreshes, and bus traffic that occurs when multiple processors use the same data.

MULTITHREAD MODEL

- In this model, a process can have multiple flows of execution.
- For example, a sequential part is created and, subsequently, a series of tasks are created that can be executed in parallel.
- Usually, this type of model is used on shared memory architectures.
- So, it will be very important for us to manage the synchronization between threads, as they operate on shared memory, and the programmer must prevent multiple threads from updating the same locations at the same time.
- The current-generation CPUs are multithreaded in software and hardware.
- **POSIX** (short for **Portable Operating System Interface**) threads are classic examples of the implementation of multithreading on software.
- Intel's Hyper-Threading technology implements multithreading on hardware by switching between two threads when one is stalled or waiting on I/O. Parallelism can be achieved from this model, even if the data alignment is nonlinear.

MESSAGE PASSING MODEL

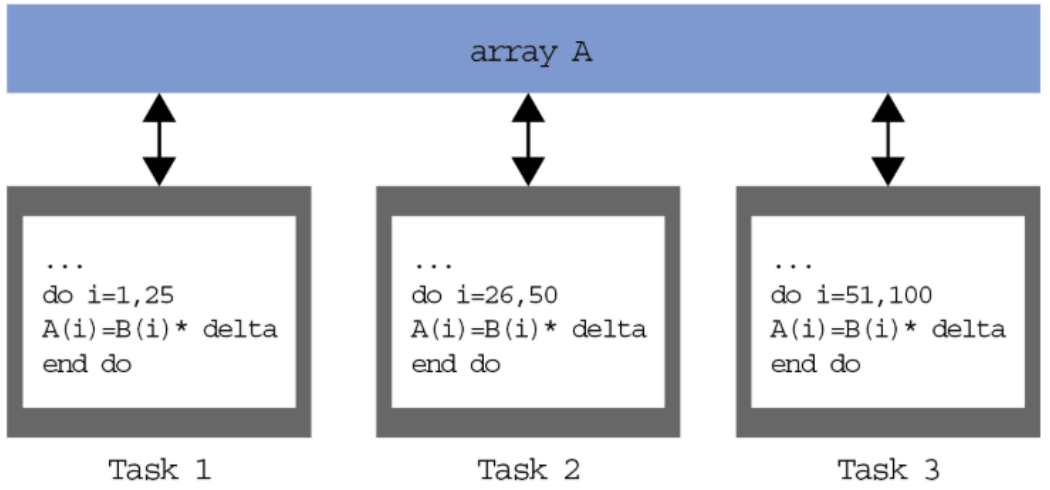
- The message passing model is usually applied in cases where each processor has its own memory (distributed memory system). More tasks can reside on the same physical machine or on an arbitrary number of machines.
- The coder is responsible for determining the parallelism and data exchange that occurs through the messages, and it is necessary to request and call a library of functions within the code.
- Some of the examples have been around since the 1980s, but only in the mid-1990s was a standardized model created, leading to a de facto standard called a **Message Passing Interface** (**MPI**).



Message passing paradigm model

DATA-PARALLEL MODEL

- In this model, we have more tasks that operate on the same data structure, but each task operates on a different portion of data.
- In the shared memory architecture, all tasks have access to data through shared memory and distributed memory architectures, where the data structure is divided and resides in the local memory of each task.
- To implement this model, a coder must develop a program that specifies the distribution and alignment of data; for example, the current-generation GPUs are highly operational only if data (**Task 1**, **Task 2**, **Task 3**) is aligned, as shown in the following diagram:



The data-parallel paradigm model

EVALUATING THE PERFORMANCE OF A PARALLEL PROGRAM

- The development of parallel programming created the need for performance metrics in order to decide whether its use is convenient or not. Indeed, the focus of parallel computing is to solve large problems in a relatively short period of time.
- The factors contributing to this objective are, for example, the type of hardware used, the degree of parallelism of the problem, and the parallel programming model adopted. To facilitate this, the analysis of basic concepts was introduced, which compares the parallel algorithm obtained from the original sequence.
- The performance is achieved by analyzing and quantifying the number of threads and/or the number of processes used. To analyze this, let's introduce a few performance indexes:
  - **Speedup**
  - **Efficiency**
  - **Scaling**

SPEEDUP

- The **speedup** is the measure that displays the benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processing element (**Ts**) to the time required to solve the same problem on **p** identical processing elements (**Tp**).
- We denote speedup as follows:

$$S = \frac{Ts}{Tp}$$

Let's recap these conditions:

- **S = p** is a linear or ideal speedup.
- **S < p** is a real speedup.
- **S > p** is a superlinear speedup.

SCALING

- Scaling is defined as the ability to be efficient on a parallel machine. It identifies the computing power (speed of execution) in proportion to the number of processors.
- By increasing the size of the problem and, at the same time, the number of processors, there will be no loss in terms of performance.
- The scalable system, depending on the increments of the different factors, may maintain the same efficiency or improve it.

EVALUATING THE PERFORMANCE OF A PARALLEL PROGRAM

- The limitations of parallel computation are introduced by **Amdahl**'s law.
- To evaluate the **degree of efficiency** of the parallelization of a sequential algorithm, we have **Gustafson**'s law.

Amdahl's law

Amdahl's law is a widely used law that is used to design processors and parallel algorithms. It states that the maximum speedup that can be achieved is limited by the serial component of the program:

$$S = \frac{1}{1 - P}$$

Tip

1 – P denotes the serial component (not parallelized) of a program.

EVALUATING THE PERFORMANCE OF A PARALLEL PROGRAM

Gustafson's law

Gustafson's law states the following:

$$S(P) = P - \alpha(P - 1)$$

we indicated in the equation the following applies:

- **P** is the **number of processors.**
- **S** is the **speedup** factor.
- **a** is the **non-parallelizable fraction** of any parallel process.

PROCESS AND THREAD

The following table summarizes the main differences between threads and processes:

Threads	Processes
Share memory.	Do not share memory.
Start/change are computationally less expensive.	Start/change are computationally expensive.
Require fewer resources (light processes).	Require more computational resources.
Need synchronization mechanisms to handle data correctly.	No memory synchronization is required.