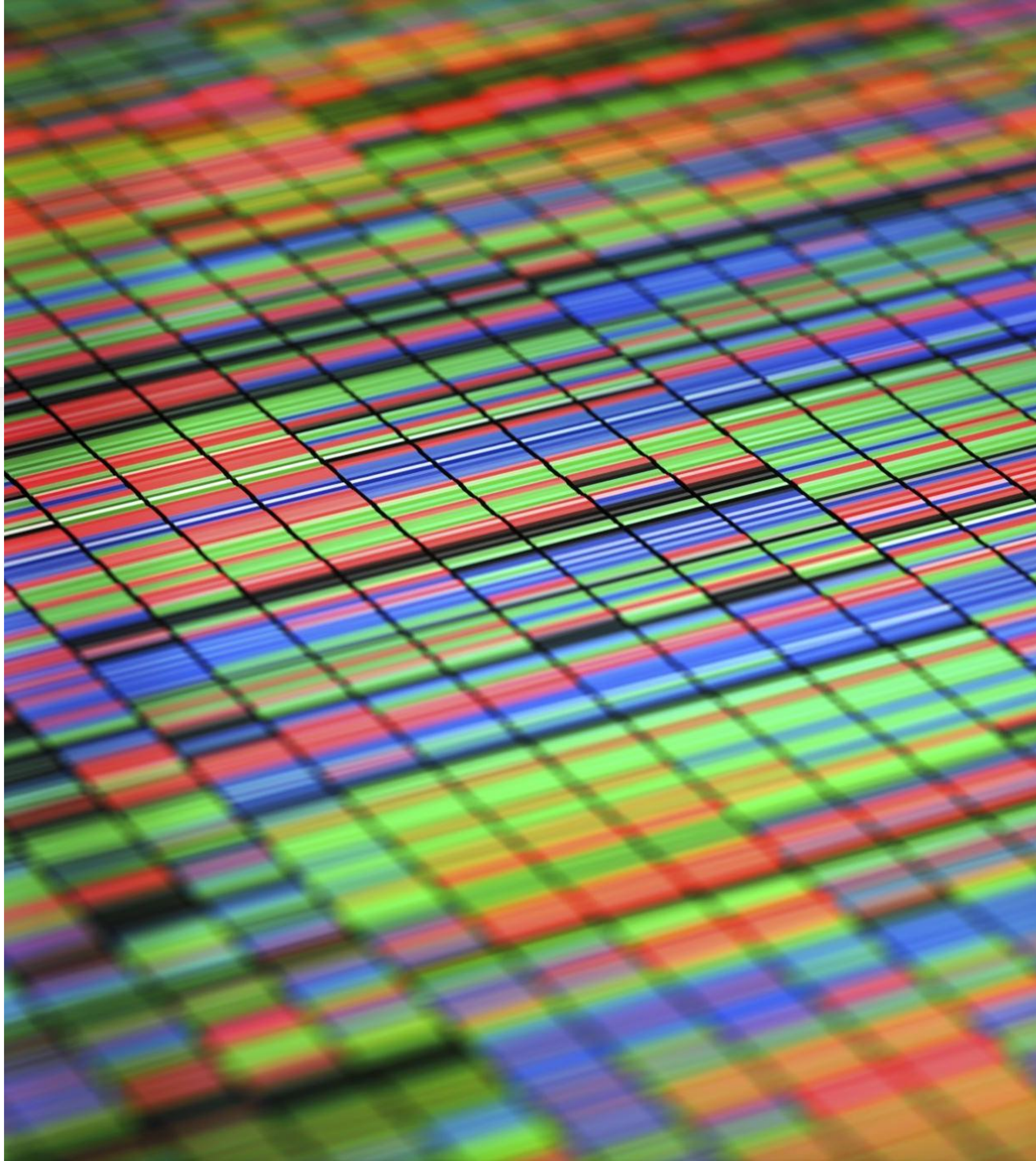
# Parallel & Distributed Computing

# Lecture # 8

By,

Dr. Ali Akbar Siddique

# Parallel Algorithms



**Topics Covered Today:**

- Parallel Search
- Parallel Sorting
- Reduction Algorithms
- Hands-on: Parallel Merge Sort in Python

# What Are Parallel Algorithms?

Parallel algorithms are algorithms designed to **divide a problem into multiple independent tasks** that can be executed **simultaneously** on multiple processing units (CPU cores, GPU threads, or distributed nodes).

They aim to reduce the total execution time and improve scalability when dealing with large or complex datasets.

# Why Use Parallel Algorithms?

- **Performance Improvement:** More work done in less time by using multiple processors concurrently.
- **Scalability:** As data size grows, parallelism ensures performance does not degrade significantly.
- **Resource Utilization:** Modern hardware (multi-core CPUs, GPUs, clusters) is designed for parallel workloads.
- **Solve Larger Problems:** Some tasks are computationally expensive and only feasible using parallel techniques.

## Key Metrics in Parallel Computing

### 1. Speedup (S)

How much faster a parallel algorithm is compared to a sequential one.

$$S = \frac{T_{sequential}}{T_{parallel}}$$

### 2. Efficiency (E)

How effectively processors are being used.

$$E = \frac{S}{P}$$

where **P** is the number of processors.

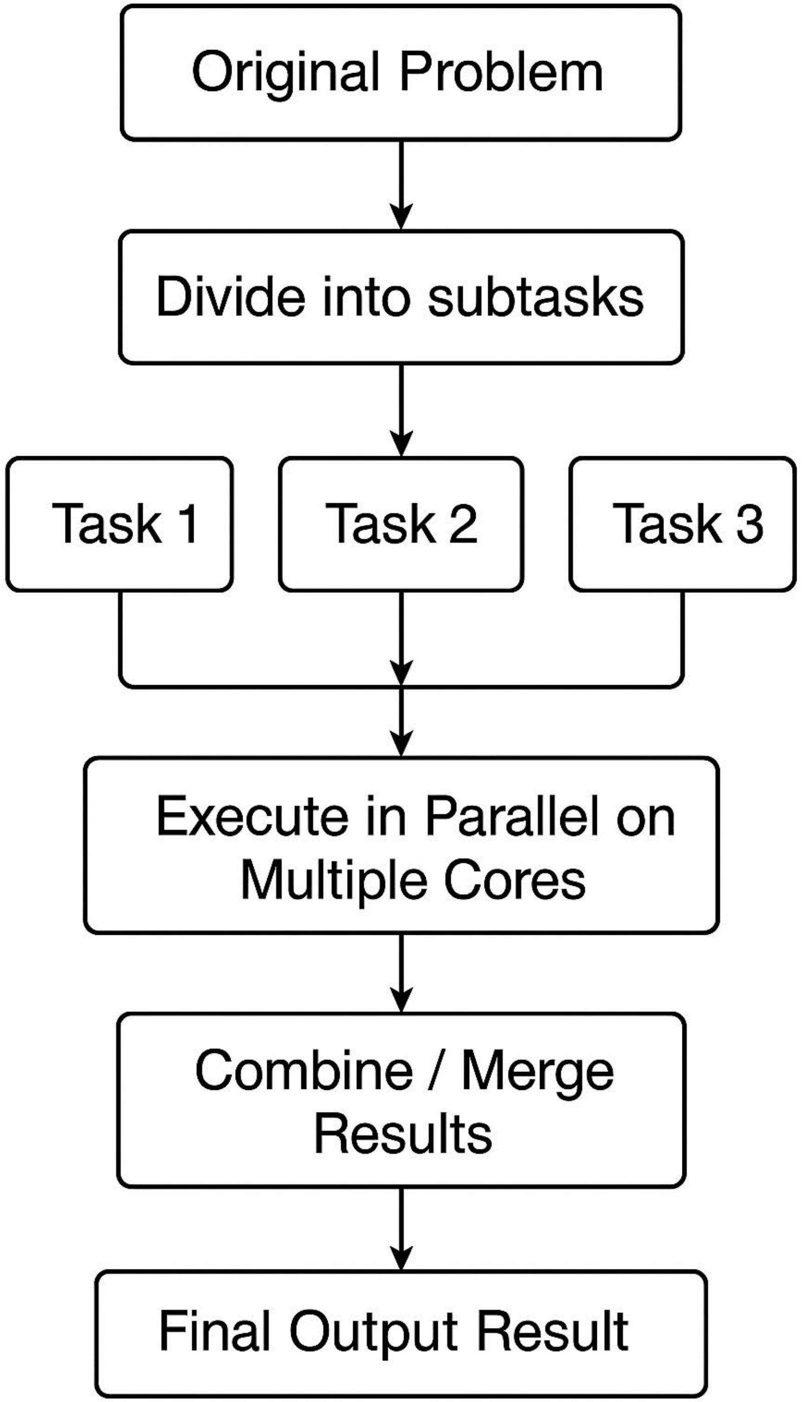
### 3. Scalability

An algorithm is scalable if increasing processors leads to proportionate performance gains.

# Types of Parallelism

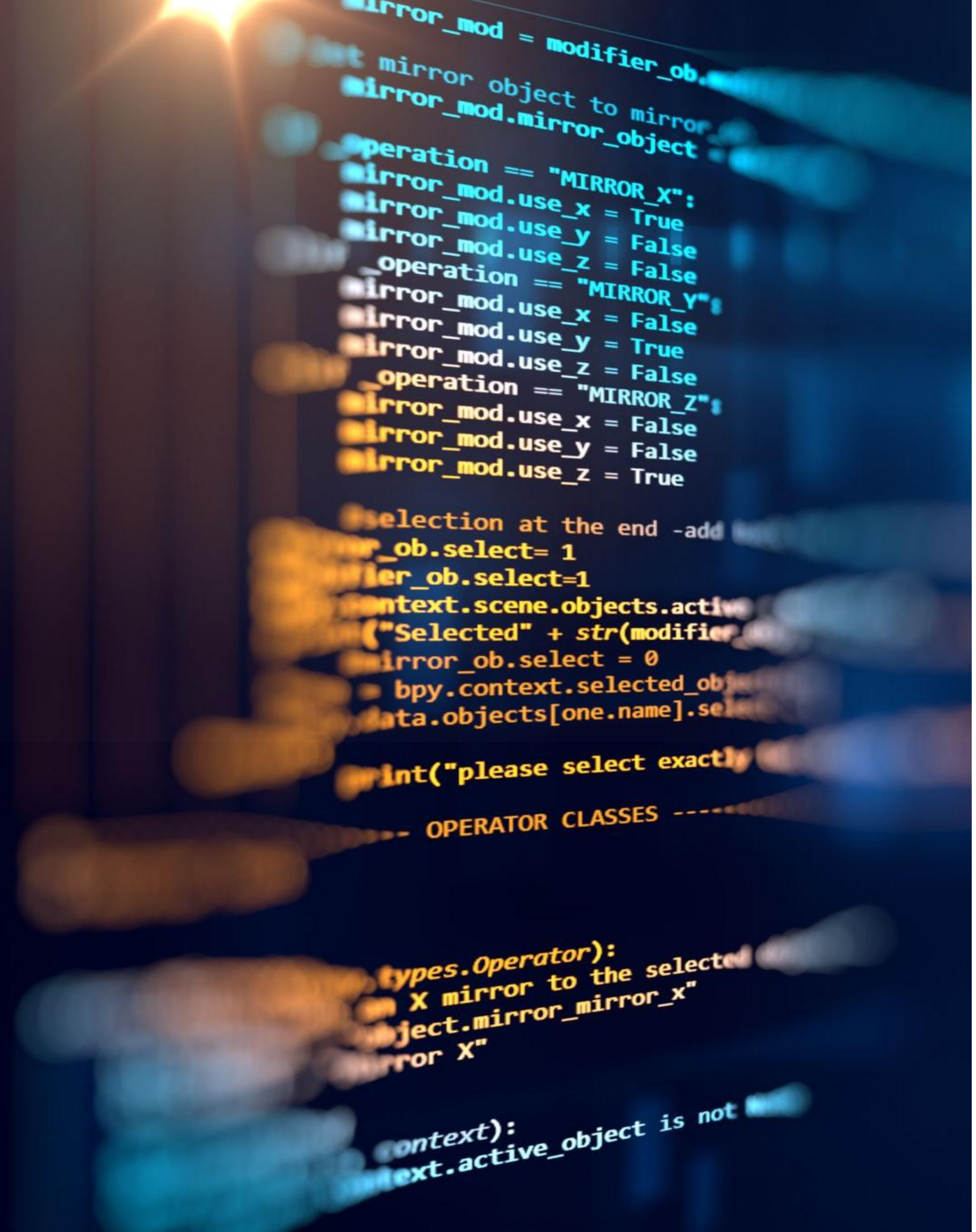
- **Data Parallelism:**
  Same operation applied on different chunks of data.
  *Example:* splitting an array for parallel searching.
- **Task Parallelism:**
  Different tasks run simultaneously (independent functions or steps).
  *Example:* web servers handling multiple requests.
- **Hybrid Parallelism:**
  Combination of both, common in HPC and deep learning.

# How Parallel Algorithms Work



# What is Parallel Search?

- Parallel search refers to techniques that allow searching through large datasets by **dividing the data into multiple segments** and searching those segments **simultaneously** using multiple processors or threads.
- The goal is to reduce the search time by leveraging hardware parallelism.




# Why Parallelize Search?

Searching is one of the most common operations in computing. For extremely large datasets (millions or billions of elements), sequential search becomes slow.

Parallel search helps because:

- **Data is divisible:**
  Arrays, lists, and databases can be partitioned easily.
- **Multiple processors can work independently:**
  Each processor searches a chunk of the data without interfering with others.
- **Reduced latency:**
  Overall search time depends on *how quickly any one processor finds the element*.
- **Ideal for large-scale data applications:**
  Used in databases, search engines, bioinformatics, and scientific computing.

# Parallel Search Approaches

Word Document (Search Techneques.docx)

# Parallel Linear Search

- Parallel Linear Search divides the array among multiple processors/threads.
- Each processor searches its own sub-array **simultaneously**.
- If any processor finds the target, it signals the others to stop.

# Example

- Search for the value **45** in the following array using **4 processors**:

  Array A = [12, 7, 45, 23, 56, 89, 14, 32]

- **Step 1 — Partition the array equally**

  8 elements, 4 processors → each gets **2 elements**.

| Processor | Elements Assigned |
|---|---|
| P0 | 12, 7 |
| P1 | 45, 23 |
| P2 | 56, 89 |
| P3 | 14, 32 |

# Example: (Cont.)

- **Step 2 — Processors search in parallel**

**P0** → checks 12, 7 → *not found*

**P1** → checks 45 → 🎉 **found at global index 2**

**P2** → would check 56, 89 but gets cancel signal

**P3** → stops early because result found

- **Step 3 — Result returned**

Target **45 found at index 2**
Speedup achieved by parallelism since not all processors needed to finish.

# Numerical Problem

- File named (Parallel Linear Search.docx)
- File named (Parallel Linear Search (Code with reasoning).docx)

# What is Parallel Sorting?

- Parallel sorting refers to a family of algorithms that sort data by **dividing the input array into smaller subarrays**, sorting these parts **simultaneously** on multiple processors, and then **merging or combining** the results.
- Sorting is one of the most commonly parallelized tasks because it is a fundamental operation in computing and is frequently applied to large datasets.

# Why Parallelize Sorting?

- File named (Why Parallelize Sorting.docx)
- File named (sorting Numerical.docx)