## Assignment No 3

Title: Develop a Distributed System to Find Sum of N Elements in an Array by Distributing N/n Elements to n Processors using MPI or OpenMP. Display Intermediate Sums at Each Processor.

### **Objectives**

- To understand and implement parallel programming using MPI (Message Passing Interface) or OpenMP (Open Multi-Processing).
- To divide a computational task (array sum) among multiple processors or threads for performance optimization.
- To study concepts of data distribution, parallel execution, synchronization, and result aggregation.
- To visualize how different processors work independently and collaboratively to solve a problem.

#### **Problem Statement**

You are required to develop a distributed application that calculates the **sum of N elements in an array** by dividing the task among **n processors** or threads. Each processor should compute the **sum of its assigned N/n elements**. After all partial sums are computed, the final sum should be obtained by aggregating these values. The application must display the **intermediate sums calculated at each processor** and finally the **total sum**. The system must use **MPI** for distributed memory systems or **OpenMP** for shared memory systems.

#### **Expected Outcomes**

- Hands-on experience in parallel/distributed programming using MPI or OpenMP.
- An application that:
  - o Divides the array among processors.
  - o Computes partial sums concurrently.
  - o Aggregates results efficiently.
  - o Prints all intermediate and final results.
- Improved understanding of:
  - o Task decomposition.
  - o Process/thread communication.
  - o Synchronization and data sharing.
- Performance benefits from parallel execution (visible with large arrays and multiple processors).

## **Software Requirements**

**Component Description** 

OS: Linux (preferred), Windows (with WSL or MPI tools), macOS

**Language:** C / C++ / Fortran

Compilers: gcc, g++, mpicc, mpirun, etc.

**MPI Tools:** MPICH, OpenMPI

OpenMP Support: GCC (with -fopenmp flag), Clang

Editor/IDE: VS Code, Eclipse, Code::Blocks, Terminal

## **Hardware Requirements**

# **Component** Specification

CPU: Multi-core (for OpenMP) or access to a cluster/multi-node setup (for MPI)

RAM: Minimum 4 GB Storage: 100 MB or more

Network: Required for MPI if using multiple machines

# Theory:

#### **Parallel Computing Overview**

Parallel computing divides large problems into smaller sub-problems that are solved simultaneously using multiple processors or cores. It is categorized into:

- Shared memory (OpenMP): Multiple threads share the same memory space.
- **Distributed memory** (MPI): Each process has its own local memory, and data is passed via messages.

#### 1. MPI (Message Passing Interface)

## What is MPI?

MPI is a standardized and portable **message-passing system** designed to allow processes to communicate with one another in a distributed environment.

#### **How it Works in This Context**

- The array of N elements is divided among n processes.
- Each process calculates the sum of its portion (N/n elements).
- The root process gathers all partial sums using MPI\_Gather() or MPI Reduce().
- The final result is printed by the root process.

#### **Common MPI Functions Used**

- MPI Init() Initializes the MPI environment.
- MPI Comm rank() Gets the rank(ID) of the calling process.
- MPI Comm size() Gets the total number of processes.
- MPI Scatter() Distributes chunks of data from the root to all processes.
- MPI Gather() or MPI Reduce() Collects results from all processes.

## 2. OpenMP (Open Multi-Processing)

## What is OpenMP?

OpenMP is a programming interface that supports **multi-threaded programming** on shared memory architectures. It uses **compiler directives (pragmas)** to manage thread creation and synchronization.

#### **How it Works in This Context**

- The array is shared among threads.
- A parallel for loop is used where each thread processes a chunk of the array.
- A reduction clause is used to compute the global sum in a thread-safe manner.
- Each thread can optionally print its partial sum (with threadprivate or manual sum logic).

# **Common OpenMP Directives**

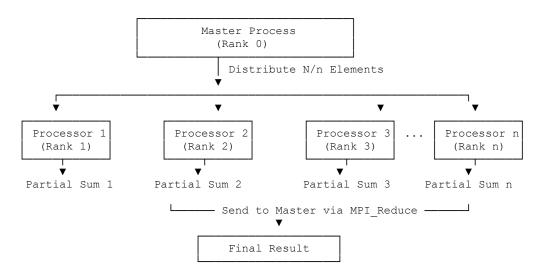
- #pragma omp parallel Creates a parallel region.
- #pragma omp for Distributes loop iterations among threads.
- reduction (+: sum) Combines values computed by threads into a single result.

## Comparison: MPI vs OpenMP

Feature	MPI	OpenMP
Memory Model	Distributed	Shared
Communication	Explicit (messages)	Implicit (shared variables)
Scalability	Very high (across machines)	Limited to single machine
Use Case	Clusters, supercomputers	Multi-core desktops/servers

# **Architecture Diagram**

# **Distributed Sum Using MPI**



# **Conclusion:**