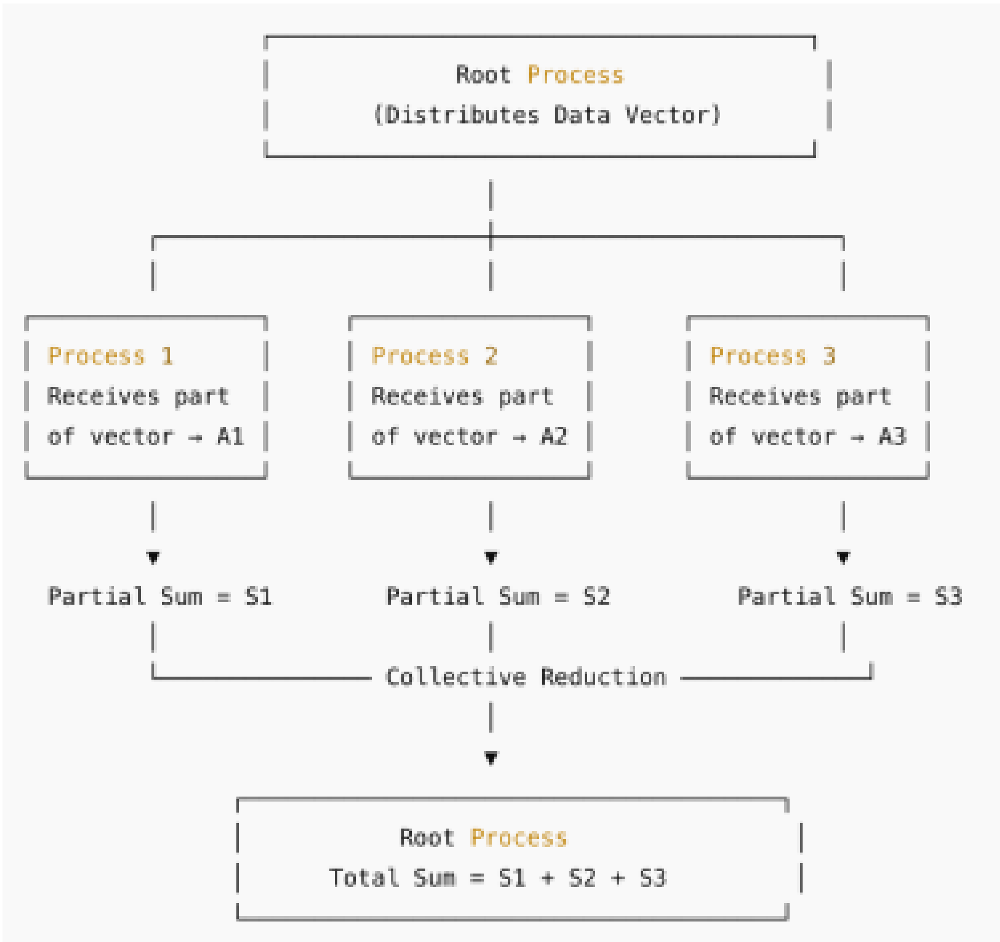
�� 🧮Lab Assignment #5 – Distributed Partial Summation using MPI

�� 🎯Objective

This lab focuses on dividing a large computational task (sum of a big vector) across multiple nodes (processes). Each node will handle a subset of data, compute a **partial sum**, and then combine the results at the **root process** using **collective MPI operations**.

�� 🧠Concepts Covered

- Data decomposition (dividing large datasets)
- Inter-process communication (MPI)
- Reduction and collective operations (`MPI_Reduce`, `MPI_Gather`)
- Understanding root and worker processes



�� 🧩 Step-by-Step Tasks

Task 1: Vector Initialization

• Create a large 1D vector `A` of size **N = 10,000,000** on the **root process**. • Initialize it with values `A[i] = i + 1`.

Task 2: Data Distribution

• Divide `A` into equal sub-vectors and distribute them among all processes using: • `MPI_Scatter()`
• Each process receives its sub-array `local_A`.

Task 3: Local Computation

• Each process computes the **partial sum** of its sub-vector:
• `double local_sum = 0;`

- `for (int i = 0; i < local_size; i++)`
- `local_sum += local_A[i];`

Task 4: Reduction (Collective Operation)

- Use **MPI_Reduce()** to combine all partial sums into one **total sum** at the root node: • `MPI_Reduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);`

Task 5: Display Results

- The **root process** prints the final sum and verifies correctness with the expected result:

  Expected Sum=N(N+1)2\text{Expected Sum} = \frac{N(N+1)}{2}Expected Sum=2N(N+1)

�� Sample Code (C using MPI)

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
 int rank, size;
 long N = 10000000;
 double *A = NULL;
 double local_sum = 0.0, global_sum = 0.0;

 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_size(MPI_COMM_WORLD, &size);

 long local_n = N / size;
 double *local_A = (double*)malloc(local_n * sizeof(double));

 if (rank == 0) {
 A = (double*)malloc(N * sizeof(double));
 for (long i = 0; i < N; i++)
 A[i] = i + 1;
 }

 // Distribute subarrays to each process
 MPI_Scatter(A, local_n, MPI_DOUBLE, local_A, local_n, MPI_DOUBLE, 0,
MPI_COMM_WORLD);

 // Local computation
 for (long i = 0; i < local_n; i++)
 local_sum += local_A[i];
```

```
 // Reduction
 MPI_Reduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);

 if (rank == 0) {
 double expected = (N * (N + 1)) / 2.0;
 printf("Total Sum = %.0f | Expected = %.0f | Difference = %.5f\n",
global_sum, expected, expected - global_sum);  free(A);
 }

 free(local_A);
 MPI_Finalize();
 return 0;
}
```

�� 🔍Discussion Questions

1. What happens if the vector size **N** is not divisible by the number of processes?
2. How can you modify the program to handle **uneven partitions**?
3. How would performance differ between using `MPI_Reduce` vs. `MPI_Gather` + local summation?
4. How could this same approach be extended to **matrix summation or averaging**?

�� 🧠Bonus Challenge

Modify the program to:

- Compute both **sum and average** using a single collective operation (`MPI_Allreduce`).
- Measure execution time using `MPI_Wtime()` and compare with serial CPU computation.

Submission: Code submission will be accepted only on GitHub Public repo