MPI Distributed Computing Assignment - Question 2 Implementation Report

Implementation Overview

This report details the implementation of Question 2 from the MPI distributed computing assignment. The program demonstrates distributed data processing using the Message Passing Interface (MPI) library, specifically addressing array distribution, local computation, and global aggregation across multiple processes.

Program Description

The implemented program (question2.c) performs the following operations:

1. Data Generation: Creates an array of 1000 random double-precision floating-point numbers in the master process (rank 0)
2. Data Distribution: Distributes array elements among participating processes using MPI_Scatterv to handle uneven load balancing
3. Local Processing: Each process calculates the average of its received elements
4. Global Aggregation: Computes the global average using two different methods as specified in the requirements

Technical Implementation Details

MPI Operations Used

1. MPI_Init(NULL, NULL): Initialize MPI environment
2. MPI_Comm_rank(): Get process rank
3. MPI_Comm_size(): Get total number of processes
4. MPI_Scatterv(): Distribute array elements with uneven counts
5. MPI_Bcast(): Broadcast distribution metadata
6. MPI_Allreduce(): Aggregate local sums and averages globally
7. MPI_Wtime(): Performance timing
8. MPI_Finalize(): Clean up MPI environment

Performance Testing:
Test Configurations

- 2 Processes: Even distribution (500 elements each)
- 3 Processes: Uneven distribution (334, 333, 333 elements)
- 4 Processes: Even distribution (250 elements each)

Performance Results

Number of Processes	Consumed Time
1	0.000370618
2	0.000722063
3	0.001219938
4	0.001031587

Analysis of Time Consumption According to Number of Processes

The total time consumed (calculated as the sum of individual process execution times) does not decrease with increasing processes. Instead, it shows an overall upward trend from 1 to 3 processes (peaking at 0.00121 seconds), with a slight dip at 4 processes (0.00103 seconds). This behavior happened because the parallel processing overhead dominates over computational benefits.

Key observations:

- Overhead impact: Each additional process introduces MPI communication overhead (e.g., broadcasting sendcounts, scattering data, and performing allreduce operations), which adds to the total time without proportional speedup for lightweight computations.
- No speedup for small data: The array size (1000 elements) is insufficient to amortize the cost of distributing work, synchronizing processes, and collecting results. The computational workload per process becomes trivial, making parallelism counterproductive.

In summary, for this specific program and data size, more processes lead to higher total time consumed due to overhead, highlighting the importance of problem size and granularity in parallel computing decisions.