Assignment 2: Intermediate Concepts in MPI Programming

Task 1: Process Role Identification and Dynamic Tasking

Write an MPI program that assigns different roles to each process based on its rank:

- Process 0: Master (coordinator) - reads an array of 16 integers and distributes segments to all processes.

- All Other Processes: Workers - receive their portion, compute the square of each value, and send results back.

1. Implement this using MPI_Send and MPI_Recv.

2. Master process should collect results and display the final array.

3. Add support for arbitrary number of processes <= 16.

Questions:

a. How is workload distribution affected by the number of processes?

b. Can this design scale for larger arrays? Why or why not?

Task 2: Safe Non-Blocking Communication

Modify Task 1 to use non-blocking versions: MPI_Isend, MPI_Irecv, and MPI_Waitall.

1. Create an array `requests[]` to manage multiple asynchronous communications.

2. Ensure correct synchronization using `MPI_Waitall`.

Questions:

a. Explain why `MPI_Waitall` is needed.

b. What happens if you omit waiting for non-blocking messages? Simulate it and report.

Task 3: Custom Communication Protocol

Write an MPI program with at least 4 processes, using the following logic:

- Process 0 sends two arrays to Process 1 and 2.

- Process 1 and 2 process the arrays and send results to Process 3.

- Process 3 performs final aggregation and displays the result.

1. Use different tags for each message.

2. Use MPI_Status in receiving functions to determine source and tag dynamically.

Questions:

a. How do message tags help in handling multiple simultaneous messages?

b. What can go wrong if two messages arrive with the same tag from different sources?

Task 4: Implement Circular Ping-Pong

Create a ring of N processes (N >= 4), where each process passes a counter to the next process in the ring. The counter starts at 0 and is incremented on each pass.

1. Process 0 starts the counter.

2. After M complete cycles, the process 0 terminates the loop and ends execution on all processes.

Requirements:

- Implement safe termination using message flags.

- Avoid deadlocks.

Discussion:

a. What are common pitfalls in ring-based communication?

b. What would be different if communication was bi-directional? Implement and test.

Task 5: Performance Timing and Barriers

1. Use MPI_Wtime to time the execution of Tasks 1 and 2.

2. Introduce MPI_Barrier to synchronize processes before timing starts and ends.

Report:

- Time taken for blocking vs non-blocking communication.

- Explain the overhead introduced by synchronization.

Note:

- Use only MPI standard functions (no external libraries).

- Avoid use of collective communication (e.g., MPI_Bcast, Scatter) for this assignment.