Parallel and Distributed Computing Lab Report

Saad Tahir 2023-CS-62

February 24, 2026

1 Exercise 1: Hello World using MPI

Figure 1 shows successful compilation and execution of the MPI Hello World program using four processes.

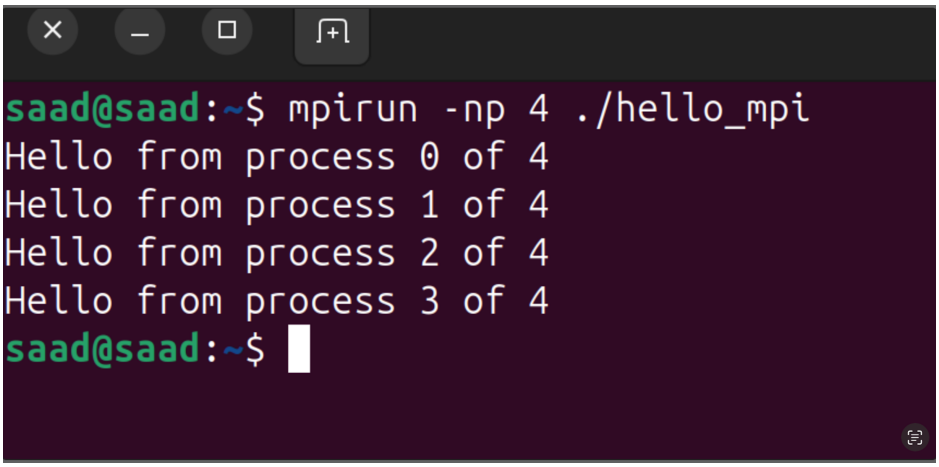

Figure 1: Compilation and execution of Hello MPI with 4 processes

2 Exercise 2: Ping-Pong Communication

The following screenshot demonstrates message exchange between two MPI processes.

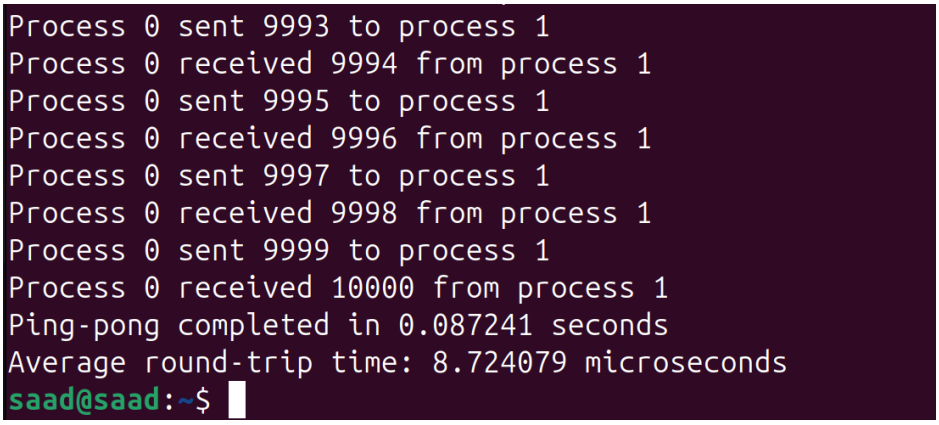

Figure 2: Ping-Pong execution with 2 processes

3 Exercise 3: MPI Counting Program

Figure 3 shows successful compilation and execution of the MPI counting program using four processes.

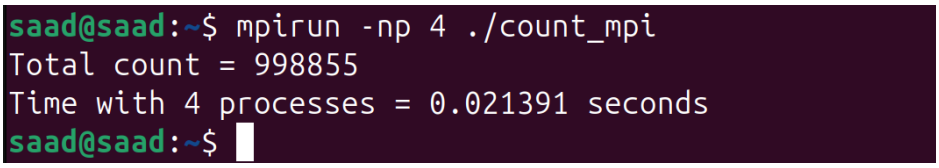

Figure 3: MPI counting program executed with 4 processes

4 Execution Time and Speedup (MPI Counting)

The execution time of the MPI counting program was measured using different numbers of processes. Speedup is calculated using:

$$Speedup(p) = \frac{T_1}{T_p}$$

where T_1 represents execution time with one process.

Number of Processes	Execution Time (seconds)	Speedup
1	0.072654	1.00
2	0.037879	1.92
4	0.021391	3.40

Table 1: Execution time and speedup of MPI counting program

5 Speedup Graph

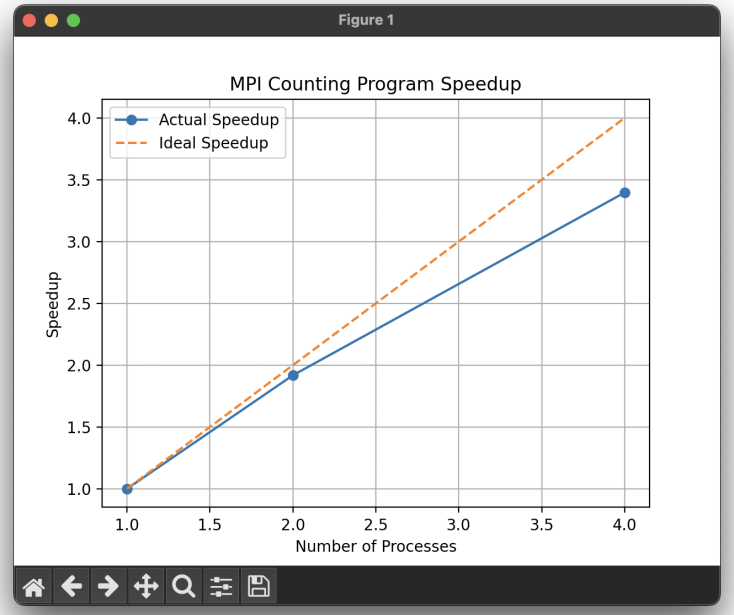

Figure 4: Speedup vs Number of Processes for MPI Counting Program

Table 2: Comparison of Execution Time and Speedup: Pthreads vs MPI

Version	**Threads/Processes**	**Time (s)**	**Speedup**
MPI	1	0.0727	1.00
MPI	2	0.0379	1.92
MPI	4	0.0214	3.40
Pthreads	1	0.040	1.00
Pthreads	2	0.040	1.00
Pthreads	4	0.040	1.00

6 Self-Assessment Answers

1. Shared-memory systems suffer from limited scalability, memory contention, cache coherence overhead, and synchronization conflicts. Distributed memory overcomes these limits by allowing independent memory spaces across multiple nodes.

2. SPMD (Single Program, Multiple Data) means all processes execute the same program but operate on different portions of data. In MPI, each process runs identical code but behaves differently using its rank.

3. Each MPI process generates its own data to avoid communication overhead and memory transfer costs. If process 0 generated all data, distributing it would require expensive communication but would guarantee identical datasets.

4. Without `MPI_Bcast`, processes may use uninitialized or inconsistent values of `n`. The program may produce incorrect results because each process must compute using the same number of intervals.

5. MPI Send/Recv involves network or inter-process communication, buffering, and synchronization, making it much slower than updating a shared variable in Pthreads, which occurs directly in shared memory.

6. Increasing message size increases round-trip time. Small messages are dominated by latency, while larger messages reflect bandwidth limits. Communication cost grows with data size.

7. Amdahl's Law:

$$S_{max} = \frac{1}{(1 - 0.95) + \frac{0.95}{8}} \approx 5.93$$

   Measured speedup is lower due to communication overhead, synchronization delays, and load imbalance.

8. MPI introduces overhead from message passing, data copying, and process management. Pthreads avoids these costs because threads share memory within the same process.

9. Using `MPI_Scatter` reduces redundant computation but increases communication overhead and memory usage at process 0. Local data generation scales better for large systems.

10. MPI is required to utilize all nodes in a cluster since Pthreads works only within shared memory of one node. A hybrid model combining MPI between nodes and Pthreads within each node can efficiently use all cores.