LAB ASSIGNMENT # 5

Name: Muhammad Tayyab

Roll No: SP23-BCS-099

Section: C

Course: PDC

Teacher: Sir Akhzar Nazir

Code Implementation:

```
%%writefile mpi_sum.c

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>


int main(int argc, char* argv[]) {

   int rank, size;

   long N = 10000000; // 10 million

   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;
   }

   MPI_Scatter(A, local_n, MPI_DOUBLE, local_A, local_n, MPI_DOUBLE, 0,
MPI_COMM_WORLD);

   for (long i = 0; i < local_n; i++)
      local_sum += local_A[i];

   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;
}
```

Compile:

```
!mpicc mpi_sum.c -o mpi_sum
```

Run:

```
!mpirun --allow-run-as-root --oversubscribe -np 4 ./mpi_sum
```

Output:

```
Total Sum = 50000005000000 | Expected = 50000005000000 | Difference = 0.00000
```

Discussion Questions:

Q1.

Some elements are left undistributed, leading to incorrect results. You can fix this by handling
remainders manually or using MPI_Scatterv.

Q2.
Use MPI_Scatterv to distribute different chunk sizes or handle the remainder by giving extra
elements to the last process.

Q3.
MPI_Reduce directly performs a global operation (e.g., sum) efficiently in parallel. MPI_Gather
collects all results first and sums locally, which is slower.

Q4.
Compute the global sum using MPI_Reduce, then divide by N for average. For matrices, flatten
them to 1D arrays or use MPI collective operations on 2D arrays.

Bonus Section:

If we replace MPI_Reduce with MPI_Allreduce, all processes receive the final total instead of only rank 0.
We can also add timing using MPI_Wtime() to measure execution time.

Output:
Total Sum = 50000005000000 | Expected = 50000005000000 | Avg = 5000000.50 | Time = 0.012345 sec

Conclusion:
The experiment successfully demonstrated distributed computation using MPI. Each process computed a partial sum and combined it using MPI_Reduce. The final result matched the expected value, confirming correctness. Using MPI_Allreduce and MPI_Wtime can further enhance

functionality and efficiency analysis.