

***LAB Assignment 5***

|  |  |
| --- | --- |
| **Name:** | Noor Fatima |
| **Registration No:** | SP23-BCS-109 |
| **Course:** | Parallel And Distributed Computing |
| **Section:** | C |
| **Teacher:** | Sir Akhzar Nazir |
| **Date:** | 16-10-25 |

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

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

int rank, size;

long N = 10000000; // total elements

double \*A = NULL;

double local\_sum = 0.0, global\_sum = 0.0, global\_avg = 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;

}

double start\_time = MPI\_Wtime();

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];

double results[2];

double local\_results[2] = { local\_sum, (double)local\_n };

MPI\_Allreduce(local\_results, results, 2, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);

global\_sum = results[0];

global\_avg = global\_sum / results[1];

double end\_time = MPI\_Wtime();

double elapsed = end\_time - start\_time;

if (rank == 0) {

double expected = (N \* (N + 1)) / 2.0;

printf("============================================\n");

printf("Total Sum (MPI) = %.0f\n", global\_sum);

printf("Expected Sum (Formula)= %.0f\n", expected);

printf("Average (MPI) = %.5f\n", global\_avg);

printf("Difference = %.5f\n", expected - global\_sum);

printf("Execution Time (MPI) = %.6f seconds\n", elapsed);

double serial\_sum = 0.0;

double serial\_start = MPI\_Wtime();

for (long i = 1; i <= N; i++)

serial\_sum += i;

double serial\_end = MPI\_Wtime();

printf("Execution Time (Serial) = %.6f seconds\n", serial\_end - serial\_start);

printf("============================================\n");

free(A);

}

free(local\_A);

MPI\_Finalize();

return 0;

}

**Discussion Answers**

1. If N not divisible by processes:

* Some processes get one extra element — handled by Scatterv + counts/displs.

1. Uneven partitions fix:

* Distribute remainder elements across first few processes.

1. MPI\_Reduce vs MPI\_Gather:

* Reduce: directly sums values in parallel → faster.
* Gather: collects values, then sums on root → slower, more memory.

1. Extension to matrices:

* Distribute rows instead of elements; each process sums its subset of rows.

**Bonus Challenge:**

|  |
| --- |
| from mpi4py import MPI  import numpy as np  import time  comm = MPI.COMM\_WORLD  rank = comm.Get\_rank()  size = comm.Get\_size()  N = 10\_000\_000  local\_n = N // size  A = None  if rank == 0:  A = np.arange(1, N + 1, dtype=np.float64)  else:  A = None  local\_A = np.zeros(local\_n, dtype=np.float64)  comm.Scatter([A, MPI.DOUBLE], [local\_A, MPI.DOUBLE], root=0)  local\_sum = np.sum(local\_A)  comm.Barrier()  start = MPI.Wtime()  global\_sum = comm.allreduce(local\_sum, op=MPI.SUM)  average = global\_sum / N  comm.Barrier()  end = MPI.Wtime()  if rank == 0:  expected = N \* (N + 1) / 2  print(f"\nMPI\_Allreduce Results:")  print(f"Total Sum = {global\_sum:.0f}")  print(f"Average = {average:.4f}")  print(f"Expected = {expected:.0f}")  print(f"Time Taken = {end - start:.5f} sec") |

|  |
| --- |
| import numpy as np  import time  N = 10\_000\_000  A = np.arange(1, N + 1, dtype=np.float64)  start = time.time()  total\_sum = np.sum(A)  end = time.time()  expected = N \* (N +1)/2  print(f"Serial sum={total\_sum:.0f}")  print(f"Expected={expected:.0f}")  print(f"Time taken={end - start:.5f}sec") |