

***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 |

from mpi4py import MPI

import numpy as np

import time

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

local\_n = N // size

A = None

if rank == 0:

A = np.arange(1, N + 1, dtype=np.float64)

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)

global\_sum = comm.reduce(local\_sum, op=MPI.SUM, root=0)

if rank == 0:

expected = N \* (N + 1) / 2

print(f"Total Sum = {global\_sum:.0f}")

print(f"Expected = {expected:.0f}")

print(f"Difference = {abs(global\_sum - expected):.5f}")

**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") |