**Matrix Multiplication with MPI: Distributed Computation, Performance & Benchmarking**

1. **Objective**

To implement and benchmark a matrix multiplication algorithm using:

* A standard serial version
* A distributed version using **MPI (blocking and non-blocking)**
* Performance metrics and scalability analysis
* Visual benchmarking against the serial baseline

1. **Serial Matrix Multiplication**

Implemented in serial.py, the function run\_serial(N) performs:

* Matrix initialization (random floats)
* Standard NumPy np.dot(A, B) operation
* Timing using time.perf\_counter()

**Used as the baseline** for all MPI benchmarks.

1. **MPI-Based Distributed Matrix Multiplication**

Implemented two variants of MPI matrix multiplication:

* **Blocking MPI Version**

File: mpi\_blocking.py

* Uses MPI.Scatterv and MPI.Gatherv for distributing and collecting data
* Master process splits matrix A row-wise
* Broadcasts matrix B using MPI.Bcast
* Each process computes partial matrix multiplication
* Final result is gathered on rank 0

**Pros:** Simple to implement  
**Limitation:** Blocking communication can cause idle time while waiting

* **Non-Blocking MPI Version**

File: mpi\_nonblocking.py

* Uses MPI.Isend, MPI.Irecv, and MPI.Ibcast for non-blocking comm
* Allows overlapping communication with computation
* Each rank receives its chunk asynchronously
* Computation starts only after required data is received

**Pros:** Better efficiency in real-world HPC clusters  
**Limitation:** Slightly more complex to synchronize correctly

1. **Performance Metrics System**

File: run\_tests.py

* Runs:
  + Serial version once
  + MPI (blocking) with n processes
  + MPI (non-blocking) with n processes
* Measures:
  + Wall-clock execution time
  + Process count
  + Mode (serial/blocking/non-blocking)
* Outputs a CSV: log\_performance.csv

1. **Sample log:**

serial,3000,1,0.39

mpi,3000,4,0.36

mpi,3000,8,0.37

mpi,3000,10,0.50

mpi,3000,12,0.42

mpi\_nb,3000,4,0.38...

1. **Scalability Testing**

* Performed runs with varying process counts: [4, 8, 10, 12]
* Matrix size N = 3000
* Serial baseline: T\_serial = 0.39 sec

1. **Benchmarking & Analysis**

File: plot\_results.py

* Reads log\_performance.csv
* Computes:
  + **Speedup** = T\_serial / T\_parallel
  + **Efficiency** = Speedup / Num\_Processes
* Plots:
  + Speedup vs. Process Count
  + Efficiency vs. Process Count
* Saves as: performance\_plot.png

1. **Deliverables Summary**

| **Deliverable** | **Description** |
| --- | --- |
| * serial.py | Serial matrix multiplication with timing |
| * mpi\_blocking.py | MPI blocking implementation |
| * mpi\_nonblocking.py | MPI non-blocking implementation |
| * run\_tests.py | Automates benchmark runs, logs times |
| * log\_performance.csv | Output of all performance runs |
| * plot\_results.py | Visualization of speedup and efficiency |
| * performance\_plot.png | Final benchmarking visualization |
| * README or this doc | Full documentation (this file) |