CS633 Assignment Group Number 19

Akshay Narayan O
Roll No. 210097

Chetan
Roll No. 210281

Kinjarapu Gnan
Roll No. 210520

Pasupuleti Jaya Siva Shankar
Roll No. 210707

14 April 2025

1 Code Description

We first read the complete data into an array on process rank 0. Then, we created a copy of this array and rearranged its contents such that, after scattering the modified array, each process receives its corresponding matrix in XYZ order.

After each process receives its corresponding matrix in XYZ order as a one-dimensional array, we convert this array into a three-dimensional matrix. Now, each element in this 3D matrix is itself an array of size `NC` which is the number of time steps. Now, for calculating the count of local minima and local maxima, each process needs to exchange boundary data with its neighboring processes.

To achieve this, we first compute the 3D coordinates (x, y, z) of the current process based on its rank in the process grid. Then, we allocate memory dynamically for the send and receive buffers, but only for the faces that require communication with a neighboring process. We use `MPI_Isend` and `MPI_Irecv` to exchange the necessary boundary data so that the performance is improved as it is non blocking.

To calculate the count of local minima and local maxima, we iterate through each element in the 3D matrix. For each element, we check whether it is a local minima or maxima by comparing it with its neighboring points. Here, we have the definition that an element is a local minimum if it is *strictly less* than all of its neighbors, and a local maximum if it is *strictly greater* than all of its neighbors. This procedure is repeated for each time step.

Within each process, we also keep track of the smallest local minima and the largest local maxima encountered. These represent the local extrema for that process.Then, we use `MPI_Reduce` to compute the sum of local minima and local maxima across all processes. We also used `MPI_Reduce` to determine the global minimum and global maximum values among all the processes.

2 Code Compilation and Execution Instructions

To compile the code, use the following command:

```
mpicc -o ./executable_file_name file_name.c
```

Next, create a job script named `job.sh` with the following content:

```
#!/bin/bash
#SBATCH -N 2
#SBATCH --ntasks-per-node=8
#SBATCH --error=job.%J.err
#SBATCH --output=job.%J.out
#SBATCH --time=00:20:00          # wall-clock time limit
#SBATCH --partition=standard     # can be "standard" or "cpu"
```

```
echo `date`
mpirun -np P ./executable_file_name input_data_file_name PX PY PZ NX NY NZ NC output_file_name
echo `date`
```

In the above script:

- `PX` – Number of processes in the X-dimension
- `PY` – Number of processes in the Y-dimension
- `PZ` – Number of processes in the Z-dimension
- `NX` – Number of grid points in the X-dimension
- `NY` – Number of grid points in the Y-dimension
- `NZ` – Number of grid points in the Z-dimension
- `NC` – Number of columns (i.e., number of time steps)
- `P` – `PX` * `PY` * `PZ`

Note: Ensure that both `executable_file_name` and `input_data_file_name` are located in the same directory as `job.sh`.

To schedule the job, use the following command:

```
sbatch job.sh
```

After the job completes, the output file `output_file_name` will be created in the current working directory. This file contains the final computed results.

3 Code Optimizations

Bottleneck 1

We observed that `MPI_File_open` and `MPI_Comm_split` were consuming significantly more time compared to the rest of our code. Initially, we used `PZ` number of processes to read the file using `MPI_File_open`, and then performed `MPI_Comm_split` to group processes with the same **Z-index** in the process grid.

However, since this approach used so much time, we optimized it by handing the file reading task to only **process rank 0**, using `MPI_COMM_SELF` instead of `MPI_COMM_WORLD` in `MPI_File_open`. This change allowed us to eliminate many `MPI_File_open` and `MPI_Comm_split` calls. The time difference we noticed was around 0.5 seconds decrease on average when we used the optimised approach.

Bottleneck 2

Initially, we used `MPI_Send` to distribute the data read by process rank 0 to all other processes. However, we observed that this one-to-many communication pattern introduced significant overhead and slowed down the overall execution.

To optimize this, we replaced it with `MPI_Scatter`, which internally uses a recursive halving strategy to distribute data more efficiently. This change resulted in a noticeable reduction in communication time.

Bottleneck 3

During the process of finding local minima and local maxima, we needed to exchange data between neighboring processes. Initially, we used a nearest-neighbor exchange to send and receive the data; however, this method was slower compared to directly using `MPI_Isend` and `MPI_Ireceive` for communication. We found a significant difference of 0.5 seconds decrease when we use `MPI_Isend` and `MPI_Ireceive` instead of nearest-neighbour exchange.

4 Results

4.1 Test Case 1

For the input data_64_64_64_3.bin
We varied two key parameters varied in the experiment:

1. **Total number of MPI processes**
2. **Number of tasks per node**

For each run, three time metrics were recorded (using the maximum value among all processes):

- **Communication Time**
- **Computation Time**
- **Total Time (Communication + Computation)**

The following sections present box plots and tables summarizing the data for various process counts (8, 16, 32, and 64 processes).

4.1.1 8-Process Configuration

For 8 processes the available number of tasks per node settings are 1, 2, 4, and 8.

- **Number of tasks per node = 1:** lower whisker = 0.1522, lower quartile = 0.1967, median = 0.2589, upper quartile = 0.2656, upper whisker = 0.2948.
- **Number of tasks per node = 2:** lower whisker = 0.0843, lower quartile = 0.1015, median = 0.1255, upper quartile = 0.1327, upper whisker = 0.1684.
- **Number of tasks per node = 4:** lower whisker = 0.0584, lower quartile = 0.0725, median = 0.0873, upper quartile = 0.0920, upper whisker = 0.1300.
- **Number of tasks per node = 8:** lower whisker = 0.0221, lower quartile = 0.1312, median = 0.1430, upper quartile = 0.1763, upper whisker = 0.1763.

4.1.2 Box Plot for 8 Processes

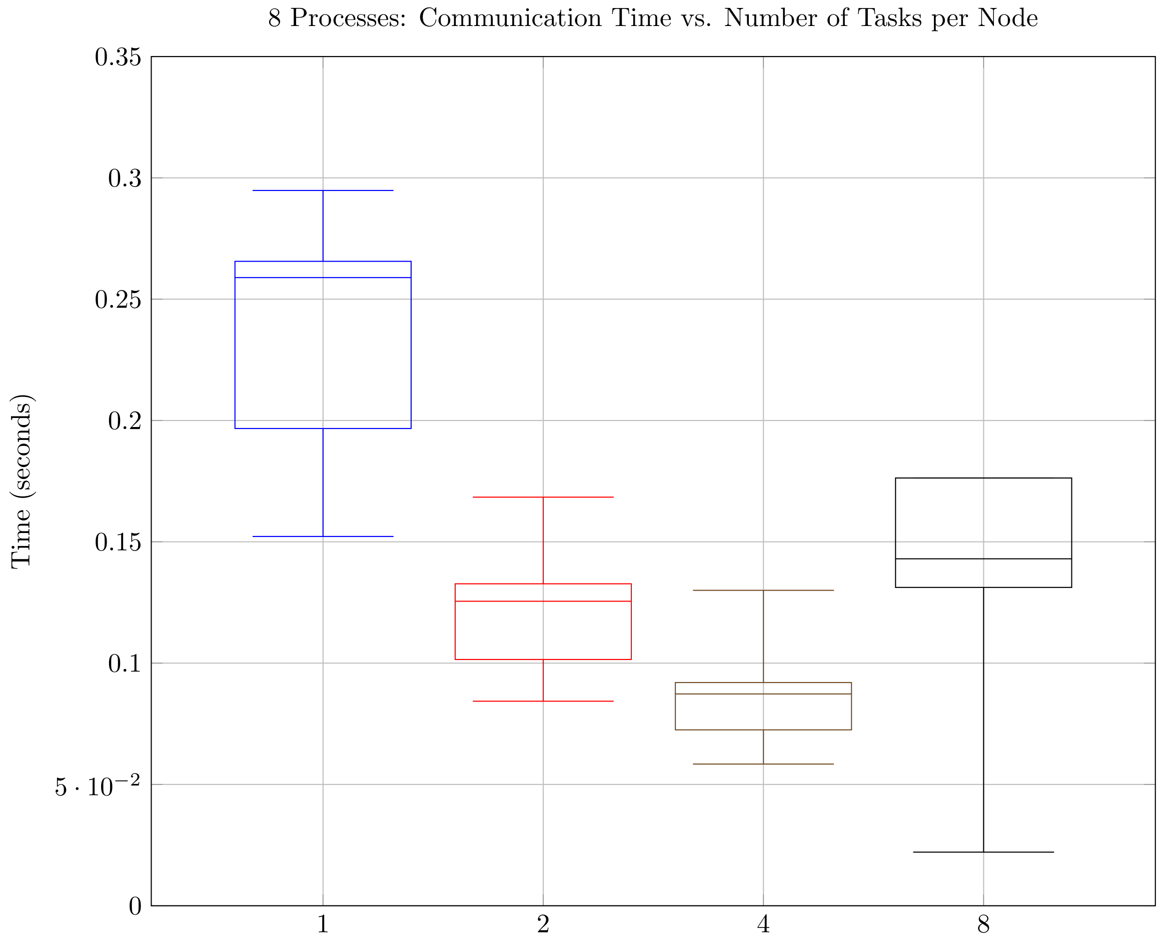

Figure 1: Communication Times for 8 Processes

4.1.3 Table for 8 Processes

Number of Tasks per Node	**Communication (sec)**	**Computation (sec)**	**Total (sec)**
1	0.1522–0.2948	0.5943–0.6094	0.6702–0.8137
2	0.0843–0.1684	0.2635–0.2894	0.3473–0.3966
4	0.0584–0.0865	0.1444–0.1498	0.2281–0.2296
8	0.0221–0.1763	0.1341–0.1793	0.1552–0.2071

4.1.4 16-Process Configuration

For 16 processes the available number of tasks per node settings are 1, 2, 4, 8, and 16.

- **Number of tasks per node = 1:** lower whisker = 0.4935, lower quartile = 0.5000, median = 0.5270, upper quartile = 0.5400, upper whisker = 0.5604.
- **Number of tasks per node = 2:** lower whisker = 0.2438, lower quartile = 0.2500, median = 0.2839, upper quartile = 0.3100, upper whisker = 0.3239.
- **Number of tasks per node = 4:** lower whisker = 0.1228, lower quartile = 0.1300, median = 0.1386, upper quartile = 0.1500, upper whisker = 0.1544.
- **Number of tasks per node = 8:** lower whisker = 0.0905, lower quartile = 0.1200, median = 0.1855, upper quartile = 0.2500, upper whisker = 0.2805.
- **Number of tasks per node = 16:** lower whisker = 0.1686, lower quartile = 0.2000, median = 0.2583, upper quartile = 0.3200, upper whisker = 0.3479.

4.1.5 Box Plot for 16 Processes

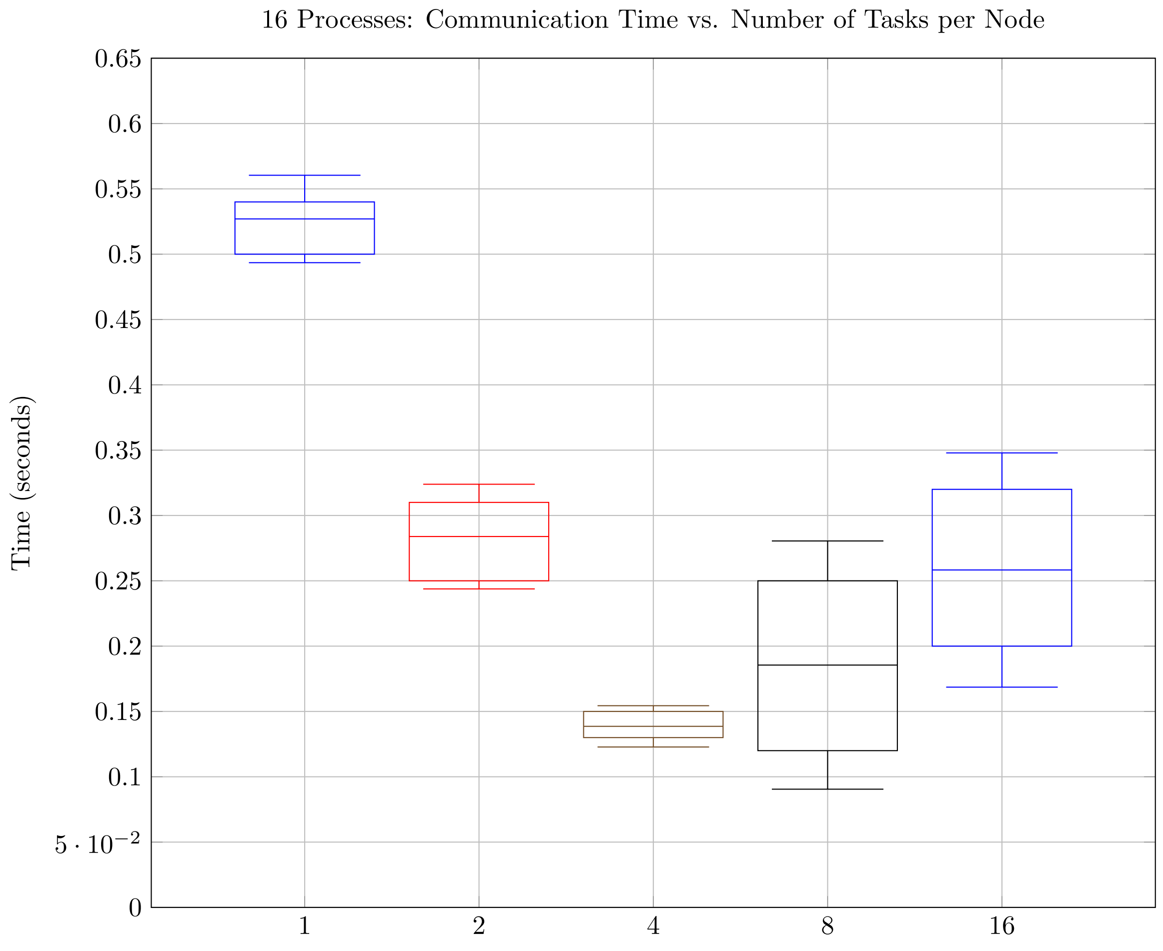

Figure 2: Communication Times for 16 Processes

4.1.6 Table for 16 Processes

Number of Tasks per Node	Communication (sec)	Computation (sec)	Total (sec)
1	0.4935–0.5604	1.0916–1.4275	1.4444–1.7065
2	0.2438–0.3239	0.5447–0.6756	0.7370–0.8067
4	0.1228–0.1544	0.2791–0.3084	0.3643–0.3804
8	0.0905–0.2805	0.1293–0.3043	0.3247–0.4085
16	0.1686–0.3479	0.1596–0.3395	0.1833–0.3643

4.1.7 32-Process Configuration

For 32 processes the available number of tasks per node settings are 1, 2, 4, 8, 16, and 32.

- **Number of tasks per node = 1:** lower whisker = 0.9879, lower quartile = 1.1000, median = 1.2818, upper quartile = 1.4500, upper whisker = 1.5757.
- **Number of tasks per node = 2:** lower whisker = 0.5242, lower quartile = 0.5500, median = 0.6169, upper quartile = 0.6800, upper whisker = 0.7095.
- **Number of tasks per node = 4:** lower whisker = 0.2076, lower quartile = 0.2400, median = 0.2717, upper quartile = 0.3000, upper whisker = 0.3358.
- **Number of tasks per node = 8:** lower whisker = 0.1608, lower quartile = 0.1650, median = 0.1729, upper quartile = 0.1800, upper whisker = 0.1849.
- **Number of tasks per node = 16:** lower whisker = 0.2214, lower quartile = 0.2500, median = 0.3181, upper quartile = 0.3800, upper whisker = 0.4148.
- **Number of tasks per node = 32:** lower whisker = 0.0234, lower quartile = 0.1000, median = 0.3056, upper quartile = 0.5000, upper whisker = 0.5878.

4.1.8 Box Plot for 32 Processes

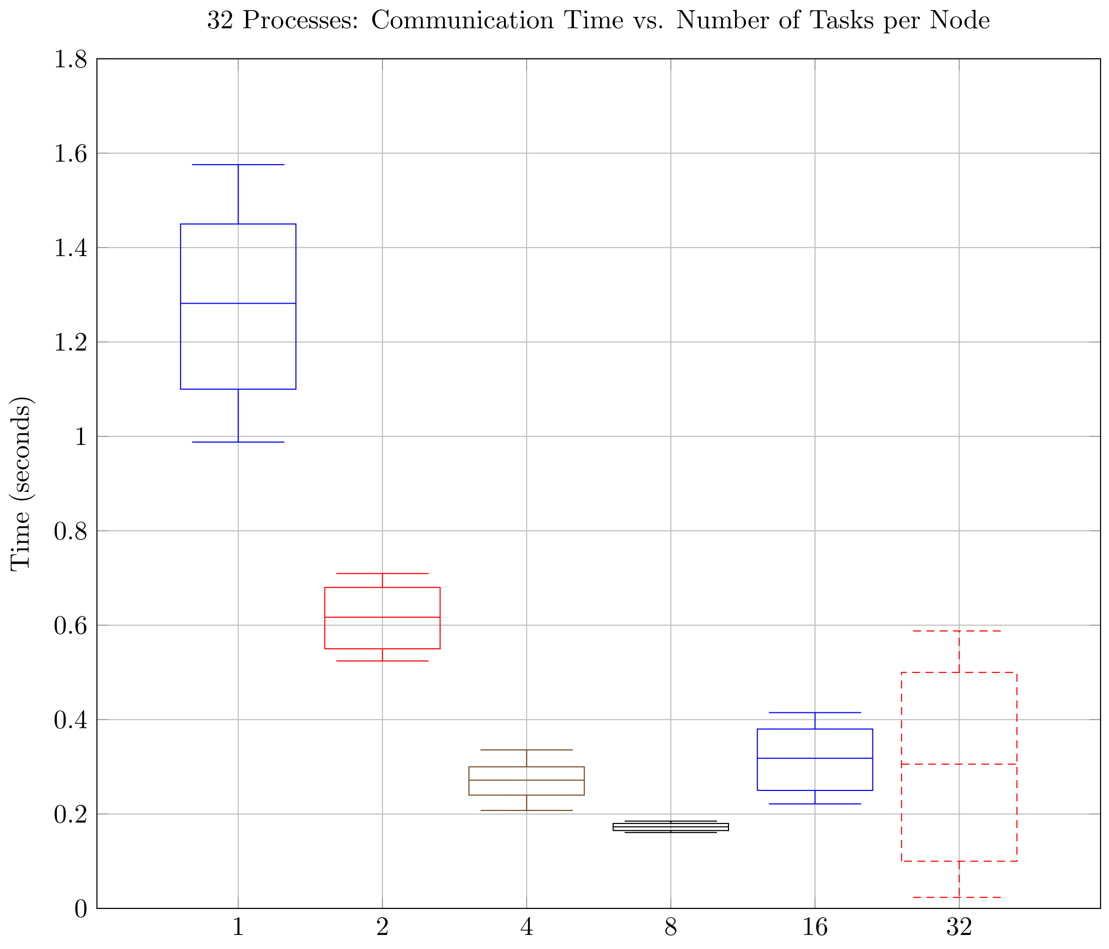

Figure 3: Communication Times for 32 Processes

4.1.9 Table for 32 Processes

Number of Tasks per Node	Communication (sec)	Computation (sec)	Total (sec)
1	0.9879–1.5757	2.2448–2.6016	2.9878–3.3306
2	0.5242–0.7095	1.0679–1.3107	1.4424–1.5919
4	0.2076–0.3358	0.4780–0.5907	0.6555–0.7728
8	0.1608–0.1849	0.2586–0.3187	0.3348–0.3807
16	0.2214–0.4148	0.1335–0.4193	0.3247–0.5415
32	0.0234–0.5878	0.0093–0.5767	0.0316–0.5962

4.1.10 64-Process Configuration (Single Node)

For the 64-process configuration on a single node the available number of tasks per node settings are 1, 2, 4, 8, 16, and 32.

- **Number of tasks per node = 1:** lower whisker = 1.4472, lower quartile = 1.6000, median = 1.8908, upper quartile = 2.1000, upper whisker = 2.3344.
- **Number of tasks per node = 2:** lower whisker = 0.5817, lower quartile = 0.6000, median = 0.6523, upper quartile = 0.6900, upper whisker = 0.7229.
- **Number of tasks per node = 4:** lower whisker = 0.3607, lower quartile = 0.3700, median = 0.3981, upper quartile = 0.4200, upper whisker = 0.4354.
- **Number of tasks per node = 8:** lower whisker = 0.1821, lower quartile = 0.1800, median = 0.1937, upper quartile = 0.2050, upper whisker = 0.2052.
- **Number of tasks per node = 16:** lower whisker = 0.5223, lower quartile = 0.5500, median = 0.6393, upper quartile = 0.7000, upper whisker = 0.7562.
- **Number of tasks per node = 32:** lower whisker = 0.7978, lower quartile = 0.8500, median = 0.9299, upper quartile = 1.0000, upper whisker = 1.0619.

4.1.11 Box Plot for 64 Processes (Single Node)

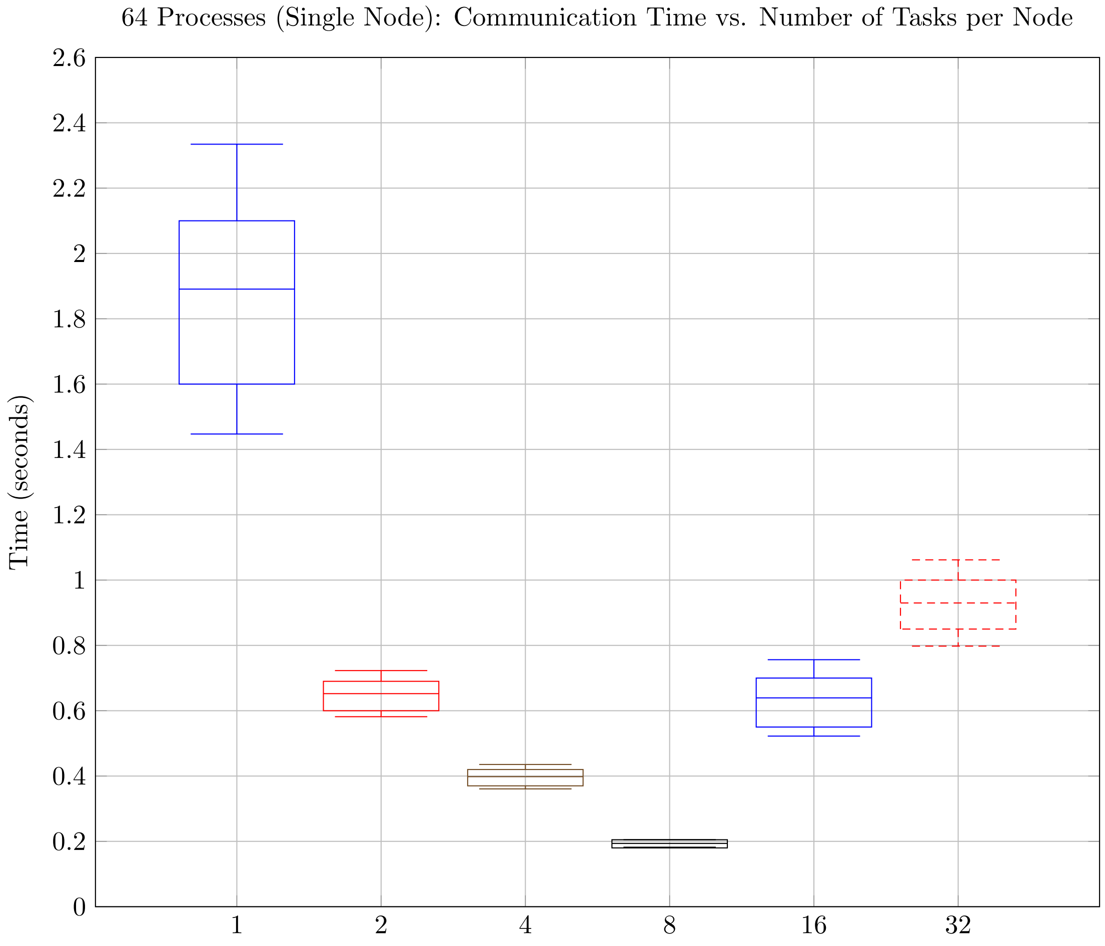

Figure 4: Communication Times for 64 Processes on a Single Node

4.1.12 Table for 64 Processes (Single Node)

Number of Tasks per Node	Communication (sec)	Computation (sec)	Total (sec)
1	1.4472–2.3344	0.4374–1.6760	1.7123–2.8577
2	0.5817–0.7229	0.1571–0.4810	0.8319–1.0523
4	0.3607–0.4354	0.1254–0.2330	0.4718–0.5894
8	0.1821–0.2052	0.0479–0.1209	0.2032–0.2537
16	0.5223–0.7562	0.0881–0.6049	0.5595–0.7808
32	0.7978–1.0619	0.6092–0.6985	0.8022–1.0694

4.1.13 64-Process Configuration (2 Nodes)

For the 64-process run distributed over 2 nodes the available number of tasks per node settings are 1, 2, 4, 8, 16, and 32.

- **Number of tasks per node = 1:** lower whisker = 1.3193, lower quartile = 1.6700, median = 3.8816, upper quartile = 4.0600, upper whisker = 4.3330.
- **Number of tasks per node = 2:** lower whisker = 0.7389, lower quartile = 0.6900, median = 1.8047, upper quartile = 1.8900, upper whisker = 1.9526.
- **Number of tasks per node = 4:** lower whisker = 0.4148, lower quartile = 0.9300, median = 0.9902, upper quartile = 1.0400, upper whisker = 1.2486.
- **Number of tasks per node = 8:** lower whisker = 0.7963, lower quartile = 1.0000, median = 1.0379, upper quartile = 1.1150, upper whisker = 1.1980.
- **Number of tasks per node = 16:** lower whisker = 1.1523, lower quartile = 1.5050, median = 1.5156, upper quartile = 1.5200, upper whisker = 1.5282.
- **Number of tasks per node = 32:** lower whisker = 1.2166, lower quartile = 1.2000, median = 1.4278, upper quartile = 1.4240, upper whisker = 1.4403.

4.1.14 Box Plot for 64 Processes (2 Nodes)

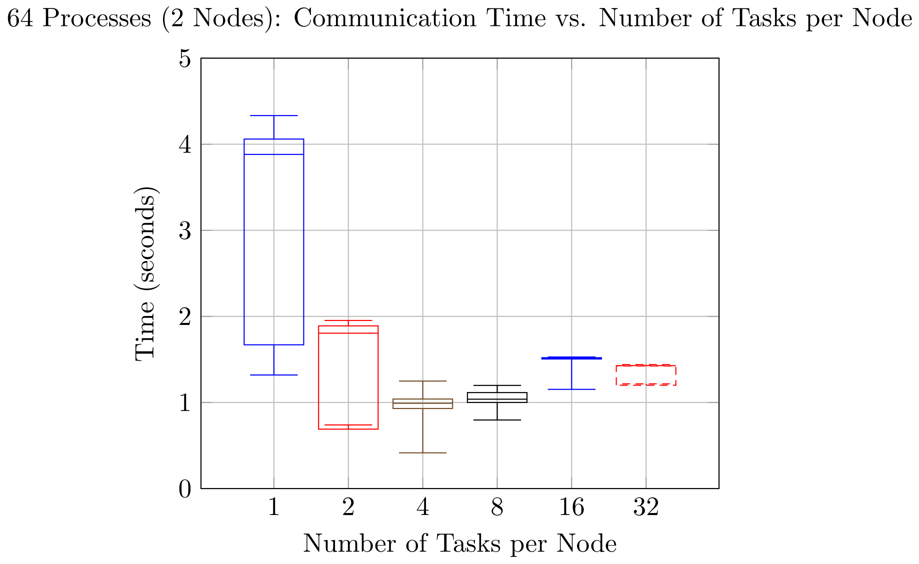

Figure 5: Communication Times for 64 Processes on 2 Nodes

4.1.15 Table for 64 Processes (2 Nodes)

Number of Tasks per Node	Communication (sec)	Computation (sec)	Total (sec)
1	1.3193–4.3330	2.8468–3.2732	3.5722–4.2596
2	0.7389–1.9526	1.3206–1.8917	1.7236–2.2268
4	0.4148–1.2486	0.6641–0.9100	0.9902–1.1566
8	0.7963–1.1980	0.8809–0.9804	1.1155–2.0873
16	1.1523–1.5282	1.1763–1.2710	1.4992–1.5672
32	1.2166–1.4403	1.1798–1.2430	1.3866–2.2904

4.2 Test Case 2

For the input data_64_64_96_7.bin
We varied two key parameters in the experiment:

1. **Total number of MPI processes**
2. **Number of tasks per node**

For each run, three time metrics were recorded (using the maximum value among all processes):

- **Communication Time**
- **Computation Time**
- **Total Time (Communication + Computation)**

The following sections present box plots and tables summarizing the data for various process counts (8, 16, 32, and 64 processes).

4.2.1 8-Process Configuration

For 8 processes the available tasks per node settings are 1, 2, 4, and 8. The data (in seconds) are:

- **task/node = 1:** lower whisker = 0.1522, lower quartile = 0.1967, median = 0.2589, upper quartile = 0.2656, upper whisker = 0.2948.
- **task/node = 2:** lower whisker = 0.0843, lower quartile = 0.1015, median = 0.1255, upper quartile = 0.1327, upper whisker = 0.1684.
- **task/node = 4:** lower whisker = 0.0584, lower quartile = 0.0725, median = 0.0873, upper quartile = 0.0920, upper whisker = 0.1300.
- **task/node = 8:** lower whisker = 0.0221, lower quartile = 0.1312, median = 0.1430, upper quartile = 0.1763, upper whisker = 0.1763.

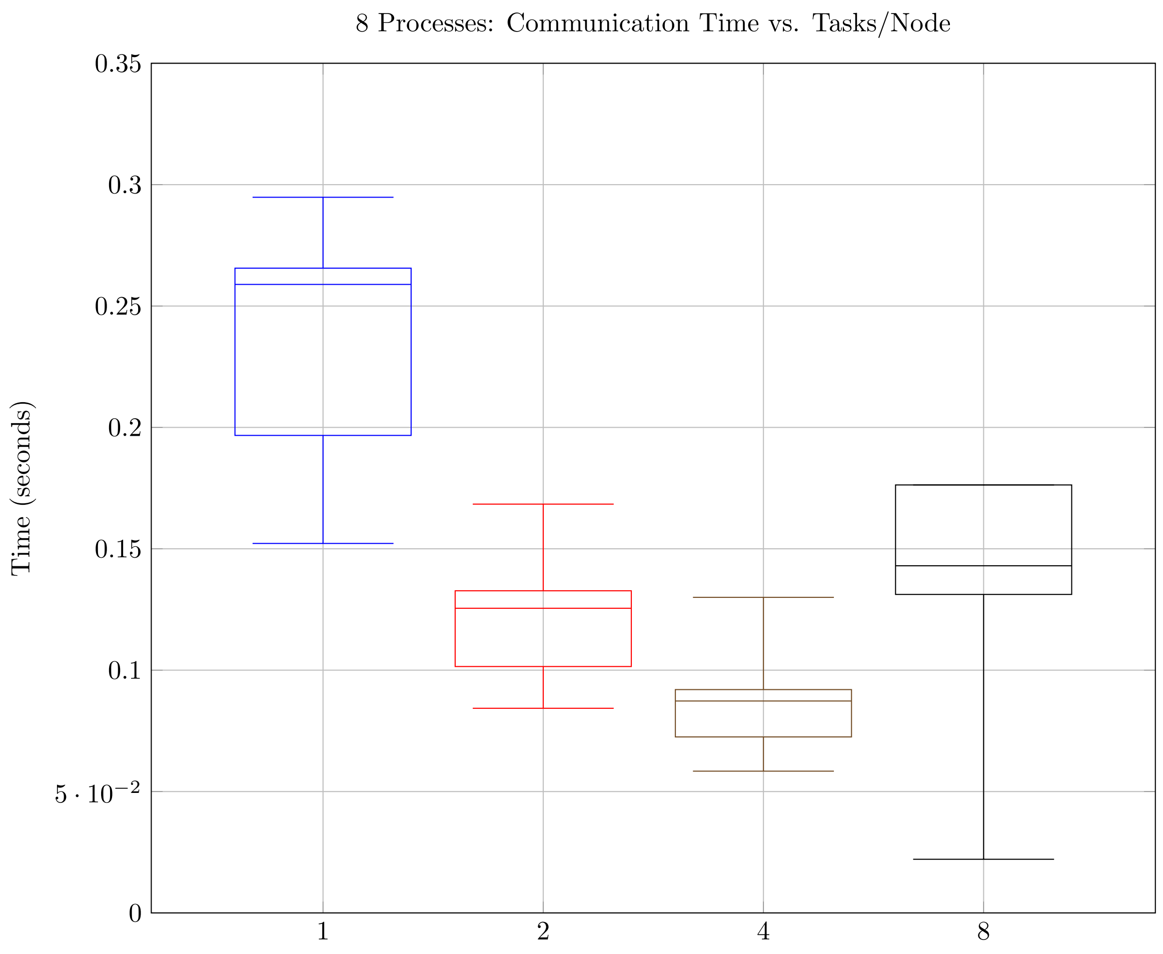

Figure 6: Communication Times for 8 processes

4.2.2 8-Process Configuration

Task/Node	**Communication (sec)**	**Computation (sec)**	**Total (sec)**
1	0.1522–0.2948	0.5943–0.6094	0.6702–0.8137
2	0.0843–0.1684	0.2635–0.2894	0.3473–0.3966
4	0.0584–0.0865	0.1444–0.1498	0.2281–0.2296
8	0.0221–0.1763	0.1341–0.1793	0.1552–0.2071

4.2.3 16-Process Configuration

For 16 processes the available tasks per node settings are 1, 2, 4, 8, and 16.

- **task/node = 1:** lower whisker = 0.4935, lower quartile = 0.5000, median = 0.5270, upper quartile = 0.5400, upper whisker = 0.5604.
- **task/node = 2:** lower whisker = 0.2438, lower quartile = 0.2500, median = 0.2839, upper quartile = 0.3100, upper whisker = 0.3239.
- **task/node = 4:** lower whisker = 0.1228, lower quartile = 0.1300, median = 0.1386, upper quartile = 0.1500, upper whisker = 0.1544.
- **task/node = 8:** lower whisker = 0.0905, lower quartile = 0.1200, median = 0.1855, upper quartile = 0.2500, upper whisker = 0.2805.
- **task/node = 16:** lower whisker = 0.1686, lower quartile = 0.2000, median = 0.2583, upper quartile = 0.3200, upper whisker = 0.3479.

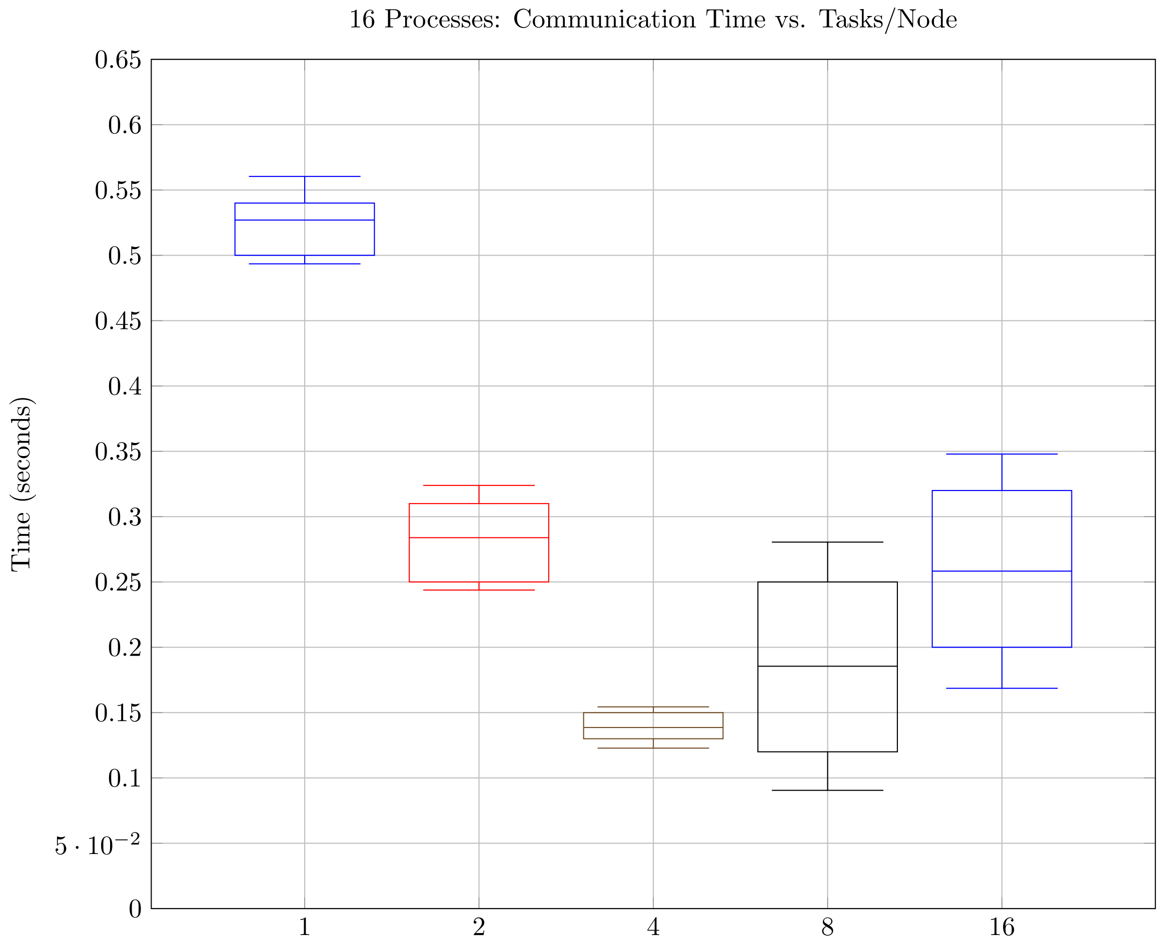

Figure 7: Communication Times for 16 processes

4.2.4 16-Process Configuration

Task/Node	Communication (sec)	Computation (sec)	Total (sec)
1	0.4935–0.5604	1.0916–1.4275	1.4444–1.7065
2	0.2438–0.3239	0.5447–0.6756	0.7370–0.8067
4	0.1228–0.1544	0.2791–0.3084	0.3643–0.3804
8	0.0905–0.2805	0.1293–0.3043	0.3247–0.4085
16	0.1686–0.3479	0.1596–0.3395	0.1833–0.3643

4.2.5 32-Process Configuration

For 32 processes the available tasks per node settings are 1, 2, 4, 8, 16, and 32.

- **task/node = 1:** lower whisker = 0.9879, lower quartile = 1.1000, median = 1.2818, upper quartile = 1.4500, upper whisker = 1.5757.
- **task/node = 2:** lower whisker = 0.5242, lower quartile = 0.5500, median = 0.6169, upper quartile = 0.6800, upper whisker = 0.7095.
- **task/node = 4:** lower whisker = 0.2076, lower quartile = 0.2400, median = 0.2717, upper quartile = 0.3000, upper whisker = 0.3358.
- **task/node = 8:** lower whisker = 0.1608, lower quartile = 0.1650, median = 0.1729, upper quartile = 0.1800, upper whisker = 0.1849.
- **task/node = 16:** lower whisker = 0.2214, lower quartile = 0.2500, median = 0.3181, upper quartile = 0.3800, upper whisker = 0.4148.
- **task/node = 32:** lower whisker = 0.0234, lower quartile = 0.1000, median = 0.3056, upper quartile = 0.5000, upper whisker = 0.5878.

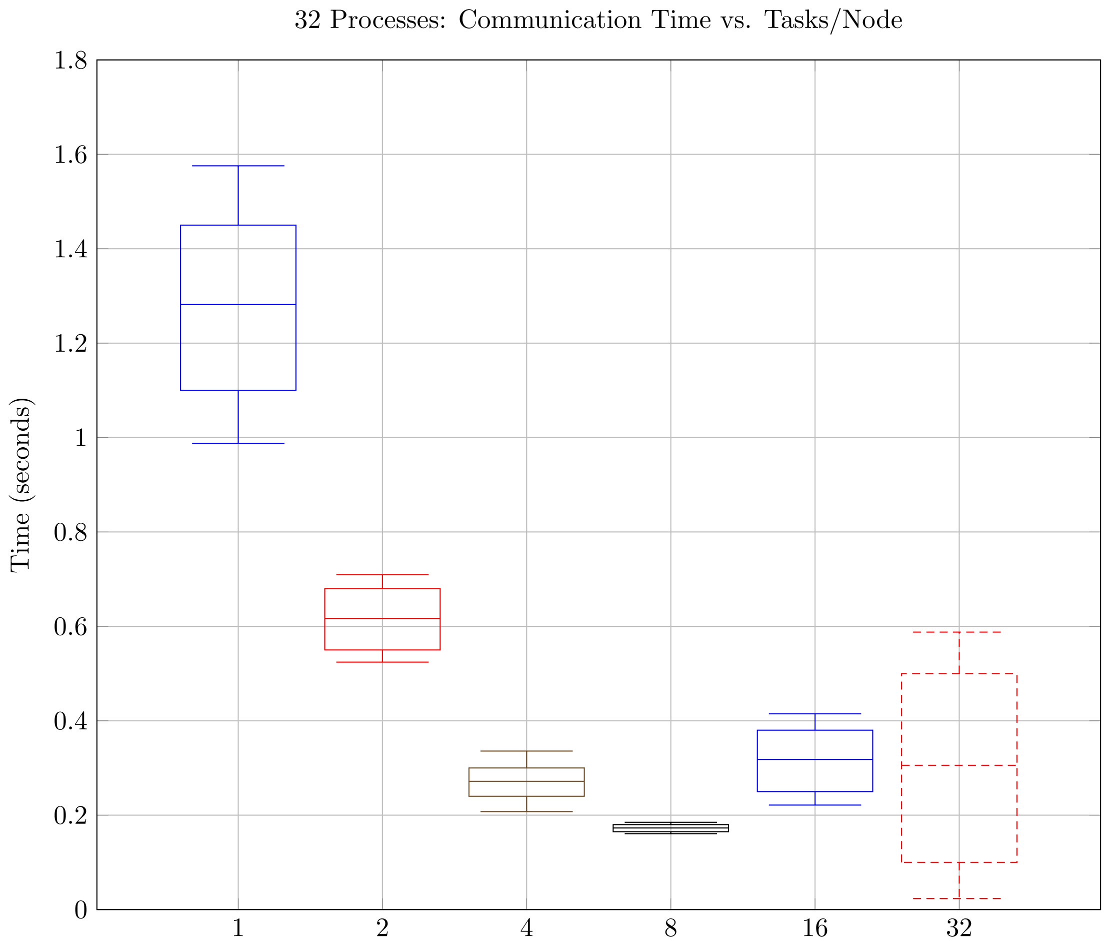

Figure 8: Communication Times for 32 processes

4.2.6 32-Process Configuration

Task/Node	Communication (sec)	Computation (sec)	Total (sec)
1	0.9879–1.5757	2.2448–2.6016	2.9878–3.3306
2	0.5242–0.7095	1.0679–1.3107	1.4424–1.5919
4	0.2076–0.3358	0.4780–0.5907	0.6555–0.7728
8	0.1608–0.1849	0.2586–0.3187	0.3348–0.3807
16	0.2214–0.4148	0.1335–0.4193	0.3247–0.5415
32	0.0234–0.5878	0.0093–0.5767	0.0316–0.5962

4.2.7 64-Process Configuration (Single Node)

For the 64-process configuration (single node) the available tasks per node settings are 1, 2, 4, 8, 16, and 32.

- **task/node = 1:** lower whisker = 1.4472, lower quartile = 1.6000, median = 1.8908, upper quartile = 2.1000, upper whisker = 2.3344.
- **task/node = 2:** lower whisker = 0.5817, lower quartile = 0.6000, median = 0.6523, upper quartile = 0.6900, upper whisker = 0.7229.
- **task/node = 4:** lower whisker = 0.3607, lower quartile = 0.3700, median = 0.3981, upper quartile = 0.4200, upper whisker = 0.4354.
- **task/node = 8:** lower whisker = 0.1821, lower quartile = 0.1800, median = 0.1937, upper quartile = 0.2050, upper whisker = 0.2052.
- **task/node = 16:** lower whisker = 0.5223, lower quartile = 0.5500, median = 0.6393, upper quartile = 0.7000, upper whisker = 0.7562.
- **task/node = 32:** lower whisker = 0.7978, lower quartile = 0.8500, median = 0.9299, upper quartile = 1.0000, upper whisker = 1.0619.

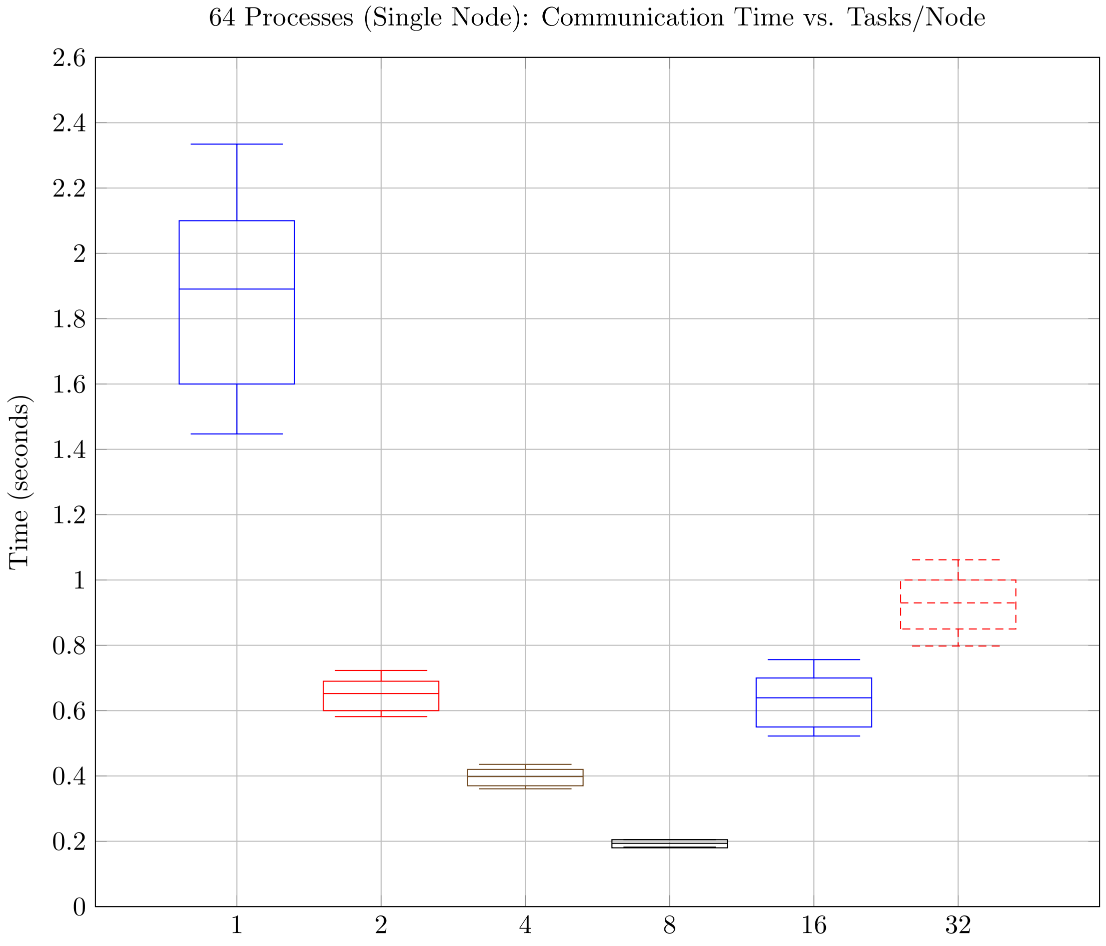

Figure 9: Communication Times for 64 processes on a Single Node

4.2.8 64 Processes – Single Node

Task/Node	Communication (sec)	Computation (sec)	Total (sec)
1	1.4472–2.3344	0.4374–1.6760	1.7123–2.8577
2	0.5817–0.7229	0.1571–0.4810	0.8319–1.0523
4	0.3607–0.4354	0.1254–0.2330	0.4718–0.5894
8	0.1821–0.2052	0.0479–0.1209	0.2032–0.2537
16	0.5223–0.7562	0.0881–0.6049	0.5595–0.7808
32	0.7978–1.0619	0.6092–0.6985	0.8022–1.0694

4.2.9 64 Processes – 2 Nodes

Task/Node	Communication (sec)	Computation (sec)	Total (sec)
1	1.3193–2.1486	2.8468–3.2732	3.5722–4.2596
2	0.7389–1.3578	1.3206–1.8917	1.7236–2.2268
4	0.4148–0.7014	0.6641–0.9100	0.9902–1.1566
8	0.7963–1.2509	0.8809–0.9804	1.1155–2.0873
16	1.1523–1.2686	1.1763–1.2710	1.4992–1.5672
32	1.2166–1.2816	1.1798–1.2430	1.3866–2.2904

4.3 Observations and Inferences

1. There is a balance between computation and communication with other processes. More tasks can reduce the work per process, but too many can cause extra communication overhead.

2. When all processes run on one node, performance is much better than when they are split over two nodes. That's because inside one node, the system can share memory quickly, but communicating between nodes is much slower.

3. The best performance is achieved by choosing a moderate number of tasks per node: about 2–4 tasks for 16 processes and around 8 for 32 processes.

4. As seen from the below two graphs, as the data size increases, the total time also increases.

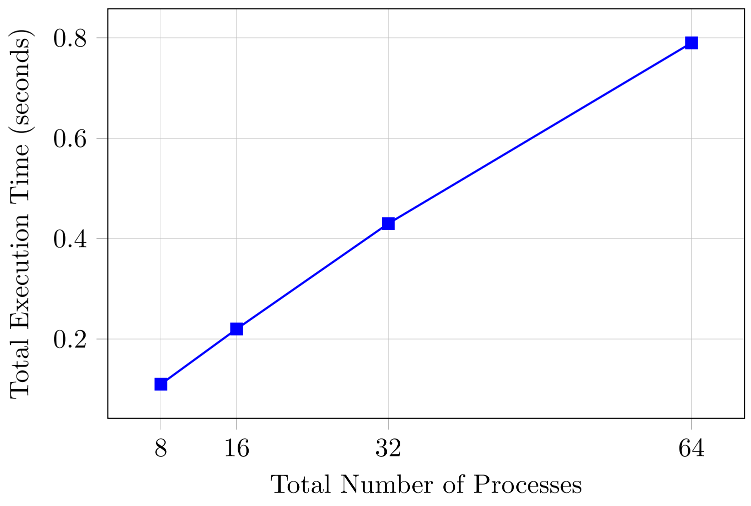

Figure 10: Total Execution Time vs Total Number of Processes(data_64_64_64_3.bin)

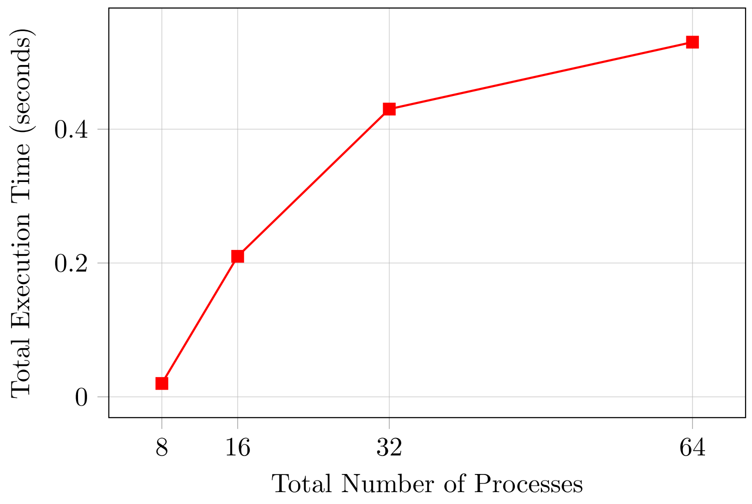

Figure 11: Total Execution Time vs Total Number of Processes(data_64_64_96_7.bin)

5 Conclusions

All team members collaborated to discuss and evaluate various approaches for writing the code. After selecting the most suitable method, we developed the initial version of the program. Subsequently, we focused on optimizing the code to achieve the best possible performance. The entire code was written on a single laptop. As for the report, we worked on it together as well, with each member contributing to different sections, though the responsibilities were shared and not strictly divided.