Homework 2

Parallel Distributed Num Algorithms

Name: Saurabh Kumar
UIN: 926009924

1. You are required to develop an MPI-based parallel implementation of the Jacobi iterative solver for solving the Laplace equation on a unit square. Compute the solution to the Laplace problem for a mesh of size 1026 x 1026. Use $\sum = 10^{-4}$ as the error tolerance.

Ans: The responses as required in the document are as follows:

1.1 Description of changes made to the code:

Ans: The changes made to the code are primarily for:

- Enabling Sending and Receiving of all the boundary values across 4 neighboring processes in the 2D process grid using MPI_Send/MPI_Recv
- Using MPI_Allreduce to receive the error value from each process and broadcast the maximum error to all processes

1.2 Parallel Computer Used:

Ans: I have used "ada".

1.3 Compilation Steps:

Ans: In order to compile my code, please follow these steps:

- If you are not in local tamu network, please connect to tamu VPN
- SSH to ada.tamu.edu using the command
  - ssh <your-tamu-username>@ada.tamu.edu
- mpicc –o myexe parallel_jacobi_2d_laplace_stripped.c

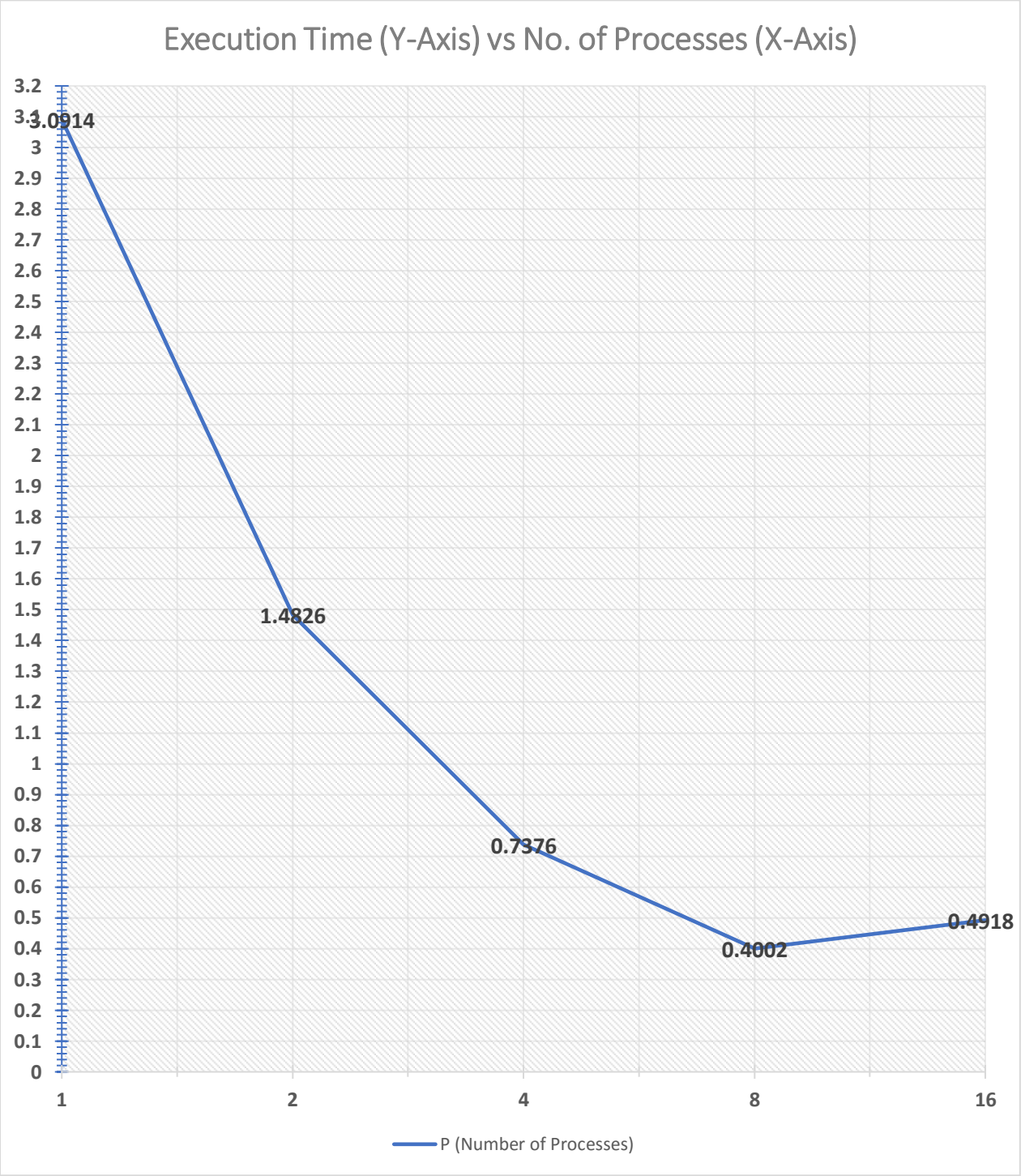
1.4 Execution Steps

Ans: Please follow the following steps to execute my code:

- Uncomment the section in job file parallel_jacobi.job that says "Part 1 run"
- Launch the command:
  - bsub < parallel_jacobi.job

2. Plot the parallel time taken by your code to solve the problem using p processes for p = 1, 2, 4, 8, and 16 (Number of iterations limited to 512)

2.1 Execution Time vs P(Processes) plot for P_x=1,2,4,8,16 & P_y = 1



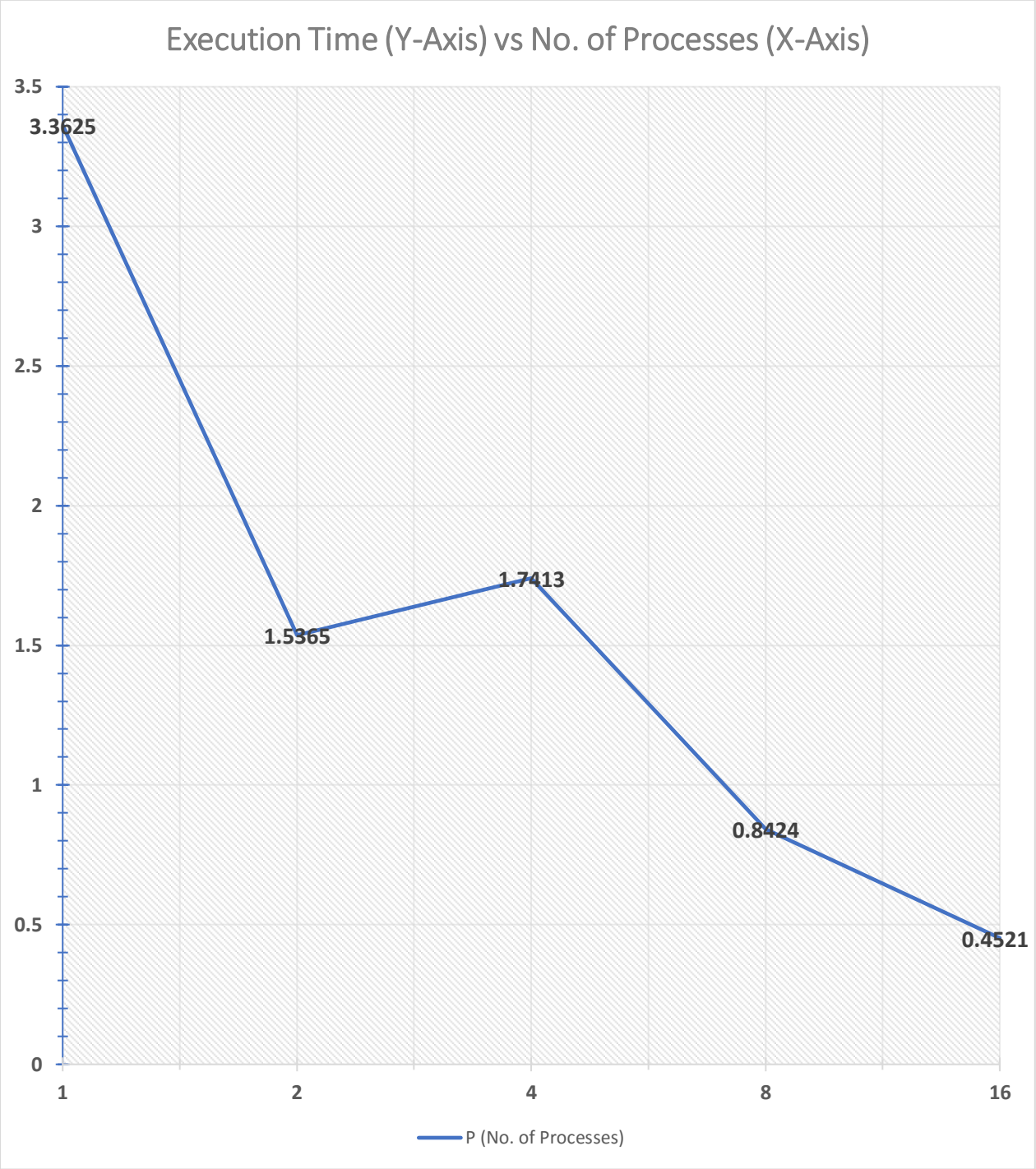
2.2 Speedup and Efficiency calculations for P_x=1,2,4,8,16 & P_y = 1

p	px	py	Execution time	Speedup	Efficiency
1	1	1	3.0914	1	1
2	2	1	1.4826	2.0851	1.0426
4	4	1	0.7376	4.1912	1.0478
8	8	1	0.4002	7.7246	0.9659
16	16	1	0.4918	6.2859	0.3929

Explanation

- From the table above, it is evident that as the number of parallel processes increases, overall execution time decreases until the number of processes becomes 16.
- Although, when the number of concurrent processes becomes 16, it is possible that the communication overhead(setup & transfer) amongst 16 processes dominates the actual computation time which could be the reason for increase in execution time for 16 processes as compared to 8 processes.

2.3 Execution Time vs P(Processes) plot for (p_x, p_y)= (1,1), (1,2), (2,2), (2,4), and (4,4)



2.4 Speedup and Efficiency calculations for (p_x, p_y)= (1,1), (1,2), (2,2), (2,4), and (4,4)

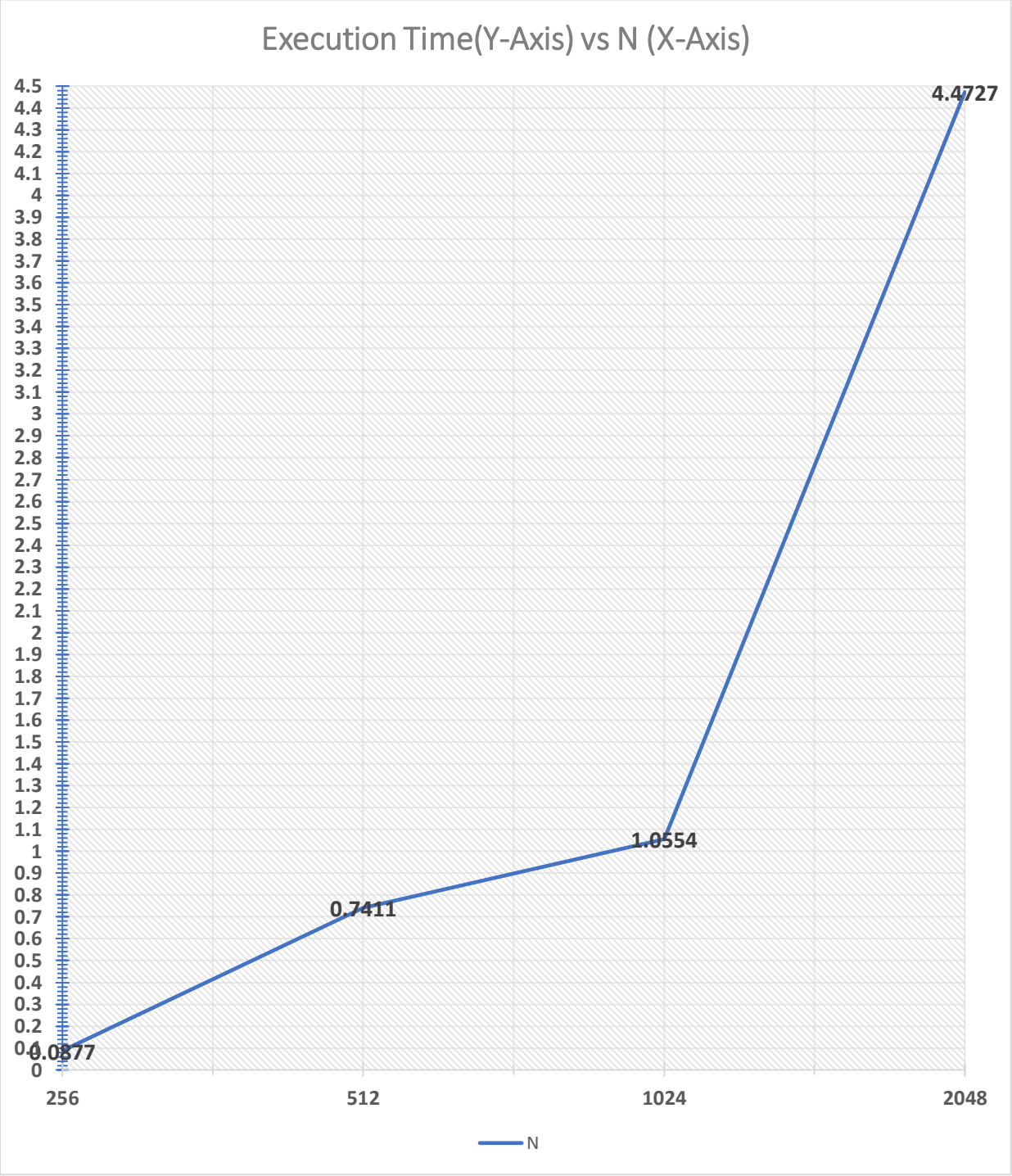
p	px	py	time	Speedup	Efficiency
1	1	1	3.3625	1	1
2	1	2	1.5365	2.1884	1.0942
4	2	2	1.7413	1.9310	0.4828
8	2	4	0.8424	3.9916	0.4989
16	4	4	0.4521	7.4375	0.4648

Explanation:

- As evident from the table above, the execution time is relatively higher for 4 processes in the above case. This could be because of similar reasons as stated above. It is quite possible that the communication overhead in this case also dominates the reduction is computation time using 4 processes. This overhead could be due to the topography of the grid of processes as well.

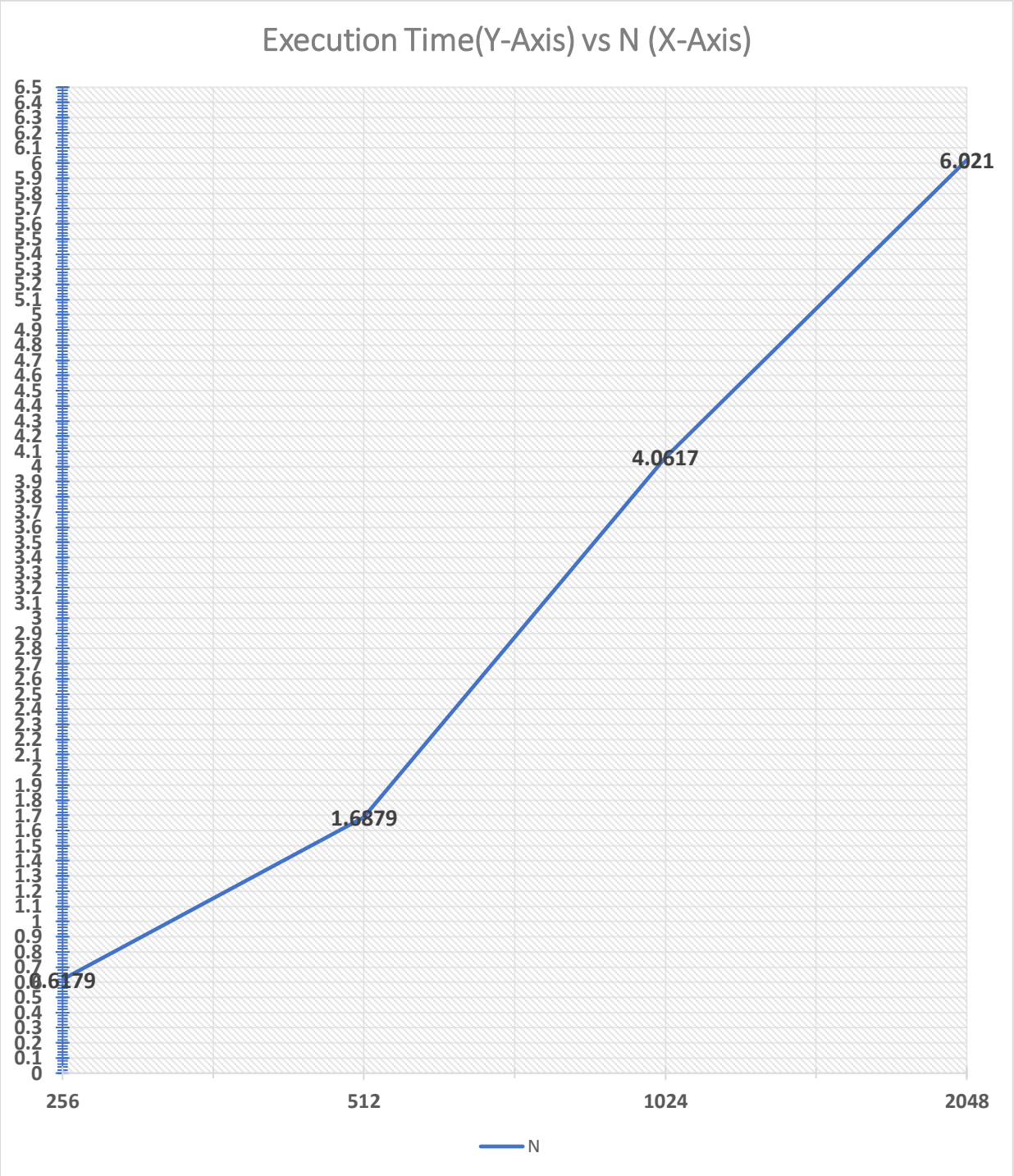
3. Plot the parallel time taken by your code to solve the problem on an (N+2)×(N+2) grid using 16 processes, where N = 2^t, t = 8, 9, 10, 11.

3.1 Case1: When P_y = 1 for all



N	p	px	py	Execution Time
256	16	16	1	.0877
512	16	16	1	.7411
1024	16	16	1	1.0554
2048	16	16	1	4.4727

3.2 Case2: When P_y = SquareRoot(P) for all



N	p	px	py	Execution Time
256	16	4	4	.6179
512	16	4	4	1.6879
1024	16	4	4	4.0617
2048	16	4	4	6.0210

Experimental Observations:

As we can observe from the above data, as the problem size increases, the execution size also increases. However, the execution time increases at a higher rate than the corresponding increase in problem size. This is happening due to a large increase in communication time across different processes which is a direct result of problem size increase.