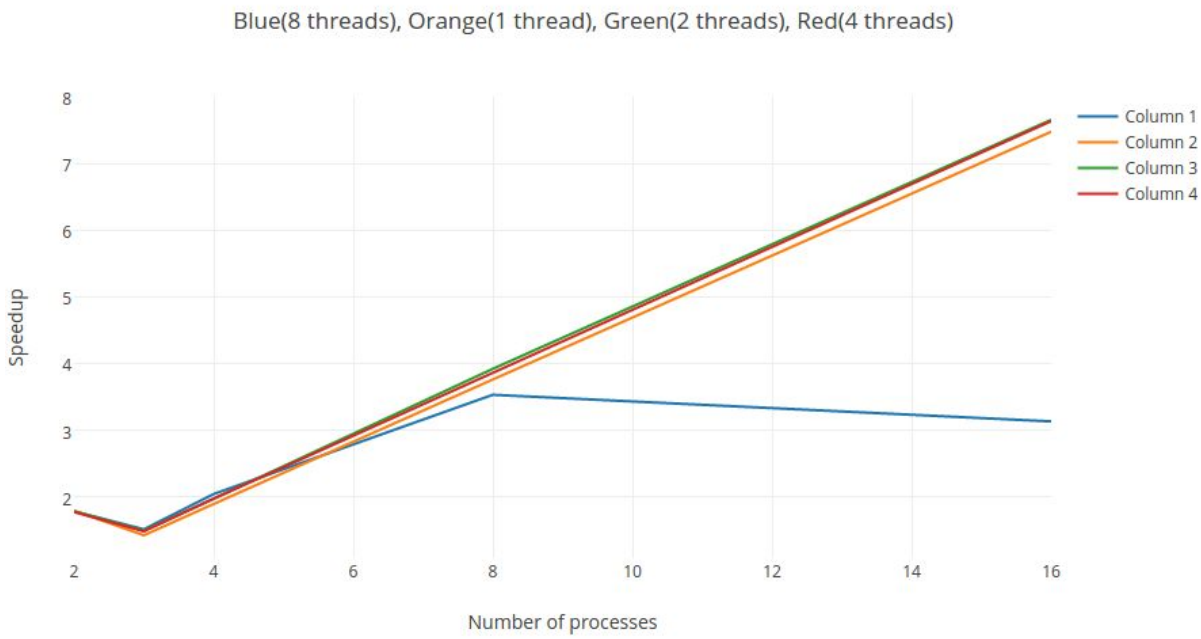
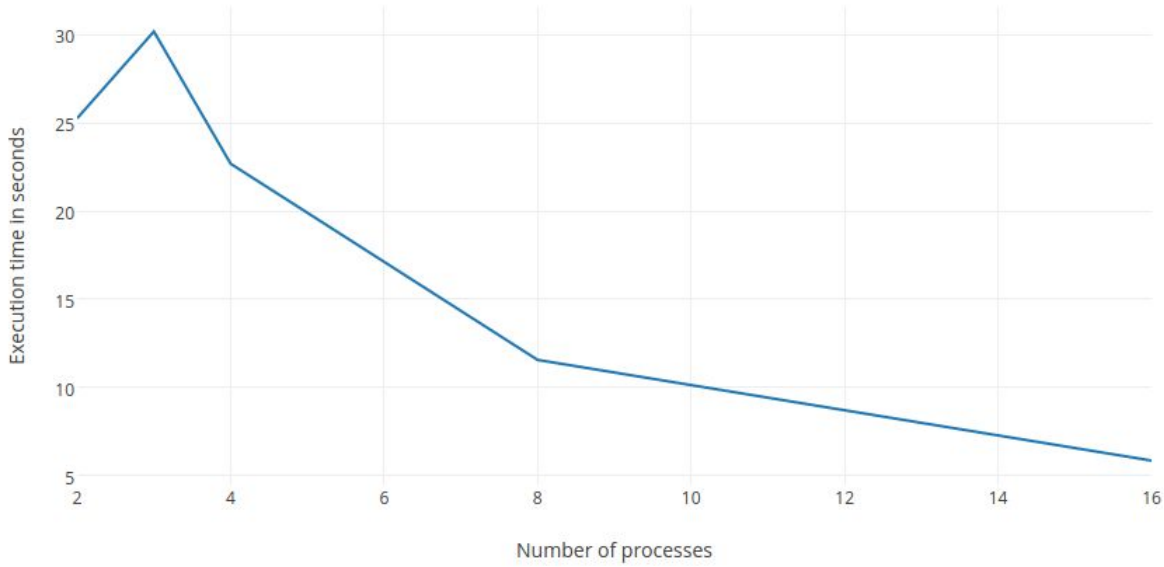
1. The implementation I used for solving this problem is Hybrid OpenMPI + OpenMP. Each slave process would work on a column strip of the matrix and when they are done, they would send back the results to the master process who would assemble all the strips.

The slave processes need to exchange ghost regions before each iteration. To copy columns from current matrix into buffer and vis viz., I used OpenMP. I prefered this approach over CUDA approach because I felt that the if conditions while copying ghost regions would not be able to run so optimally on the GPU and the performance would take a hit.

2.

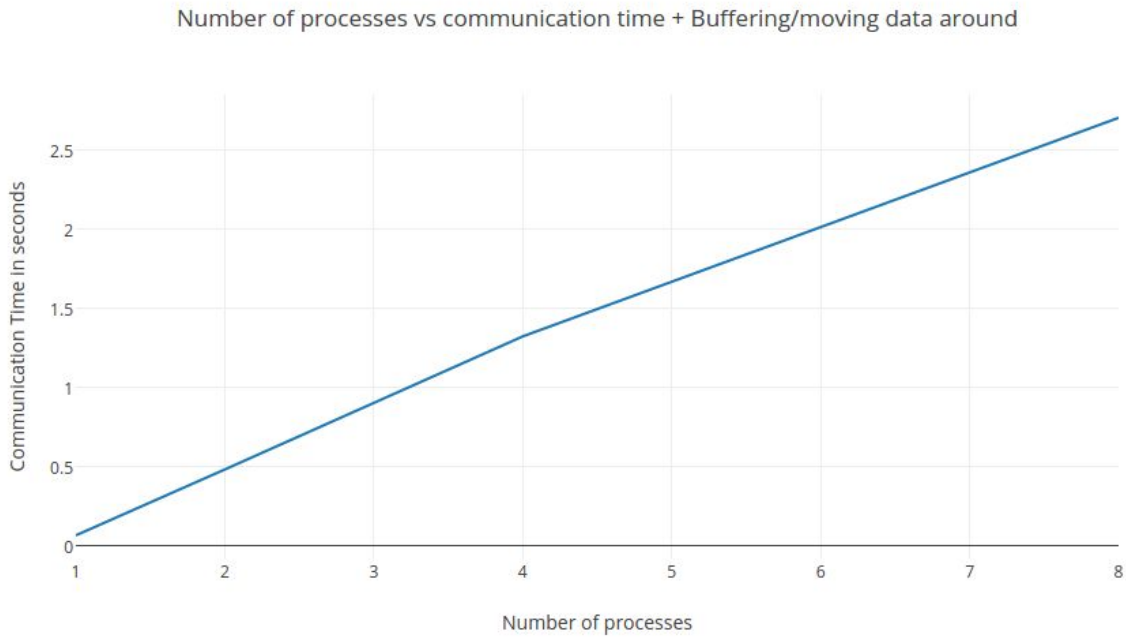


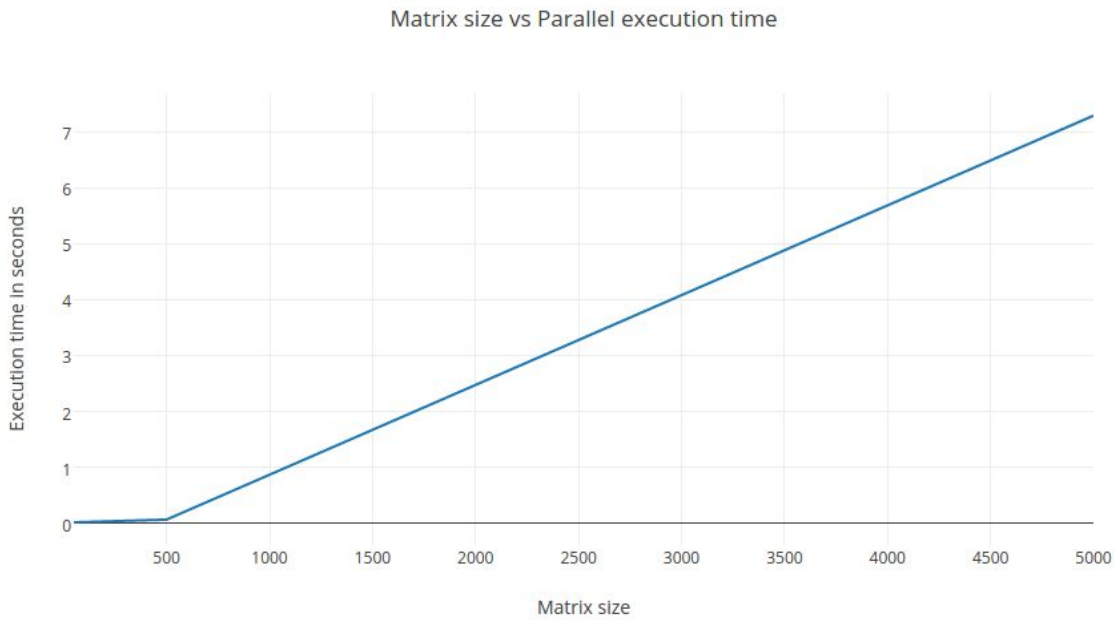


- **Work Imbalances:** I was dividing my matrix into vertical strips and giving each process a strip. Since the size of the matrix is rarely uniformly divided by the number

of process I have, one of the processes on the boundaries(right and left boundaries) ends up doing less work than the rest.

- **Parellization overheads:** To make this heat distribution problem parallel I had to use exchange ghost regions and copy them into their right places. I tried to mitigate this extra work by using OpenMP but still the overhead remained. Below is the plot of the overhead vs number of processes I used. As you can see the overhead steadily increases with the number of processes used.

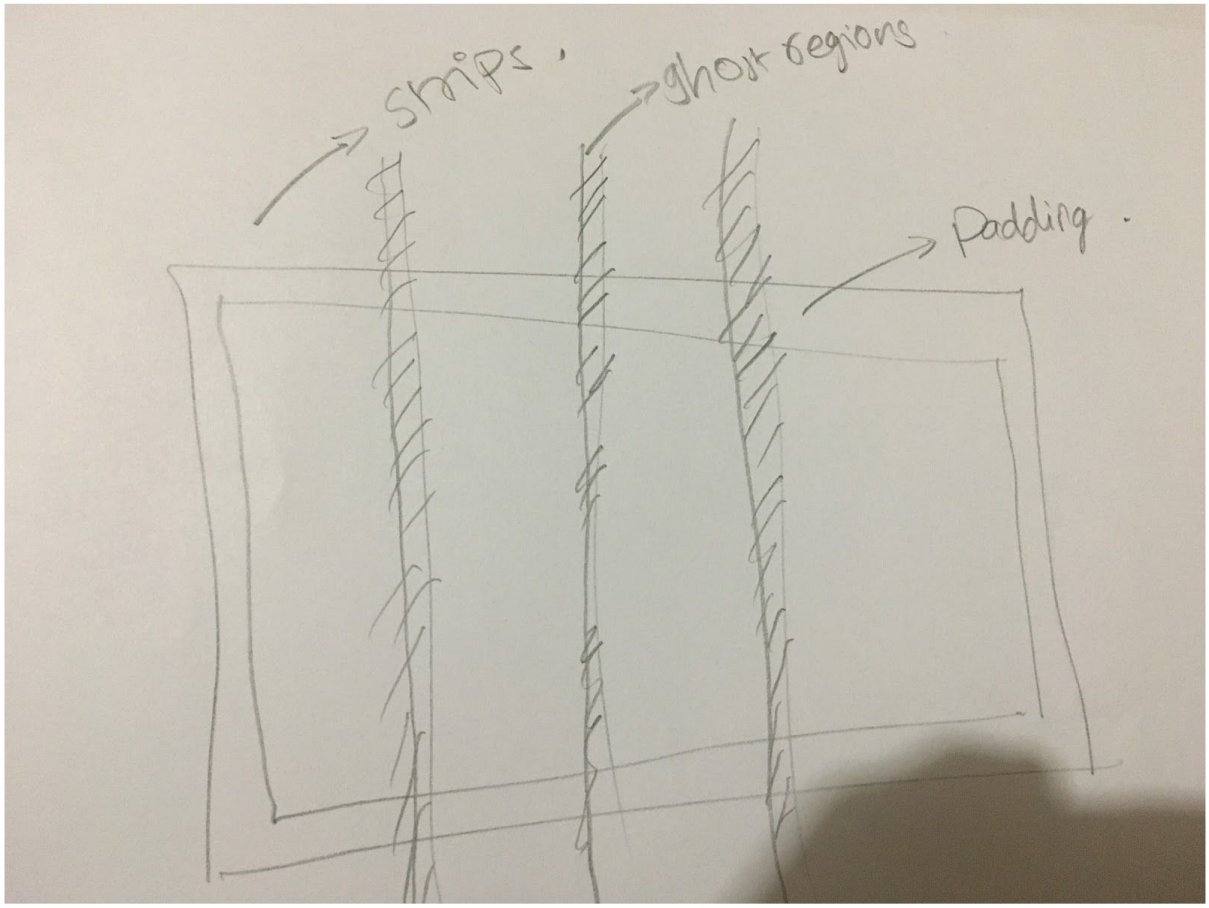




Data partion strategy: The two code division strategies that I have tried were dividing the matrix into strips of rows or columns. I got a better performance by dividing the matrix into

columns because I was allocating memory for my matrix in a row major way so when I need to swap out ghost regions, I could just change the column pointers and I didn't need to copy data. I couldn't have done something like that if I would have divided my matrix into grids. Alternatively I would have had more boundary regions in grid data partition strategy. Which would have required me to copy more data from buffers into its exact places. More parallel overheads..

(3)



Design decisions

I divided my matrix into vertical strips.

I choose vertical strips because it made sharing ghost regions easy. Instead of copying data to and from buffer, I just had to swap pointers for copying ghost regions. It also was less work to do then dividing the matrix into grids.

Unit of work: I let the number of processes decide the number of strips I need to make.
Tradeoffs: More threads = better handling of parallelization overhead but more threads = less time for processes.
Analytical model:

In a single threaded model each point needs to be individually processed which takes constant time. Therefore for single threaded model, the run time complexity is O(N * N). Where N is the size of the matrix.

For Multithreaded model. Each process needed to deal with copying ghost regions which were of length N, So the other overhead for doing that was O(N * P) But the overall work was done in O((N * N)/P). So total time complexity is O((N*N)/P + N*P)