# Parallel Programming (SE 295)
## Assignment 01

**Satish Kumar**
M.Tech. SERC
Roll No. -11052

# Problem 1

**Programming Strategies:**

1. I used **MPI-Cart** topology to create and handle 2D process topology and to find out rank of neighboring processes, in the topology. This helped in **accessing neighboring process** quite easily.

2. I used **4x4** dims for 16 processes and **4x8** dims for 32 processes. To ensure balanced assignment of tasks, I used same no. of rows and columns of data to each processes.

3. 4 ghost row has been taken for each process, to communicate array of data with left, right, up and down immediate process. I have used MPI_Sendrecv function keeping in mind of deadlock situations.

4. To communicate a column between left and right immediate processes, I first stored elements of that column into temporary 1d array, and then used 1 MPI_Sendrecv per column.

5. To put **boundary conditions** for our 2D domain, I deployed Null Processes whose rank happens to be MPI_PROC_NULL while using MPI_CART_SHIFT function for processes onto boundaries (*Note: non-periodic topology chosen*). Hence, I added boundary conditions (Dirichlet type) by putting values corresponding the Null Processes.

6. To avoid **data discrepancies** between different iterations, I used MPI_BARRIER. I have chosen Maximum no. of iterations as **stopping criteria** for the problem.

**Results:**
Data for Execution time in seconds, obtained for 1000 iterations, after taking average of 5 run for each of 8 experiments, are as follows:

| np | 2000 | 4000 | 6000 | 8000 |
|---|---|---|---|---|
| 1 | 101.88 | 434.36 | 947.64 | 2127.94 |
| 16 | 7.58 | 30.67 | 65.07 | 119.72 |
| 32 | 4.75 | 20.10 | 40.23 | 78.27 |

Table 1. Execution time in seconds np = 1, 16 and 32 processes for different problem size

Speedup and Efficiency computed with respect to sequential execution time are as follows:

| np | 2000 | 4000 | 6000 | 8000 |
|---|---|---|---|---|
| 16 | 13.4406 | 14.1624 | 14.5634 | 17.7743 |
| 32 | 21.4484 | 21.6100 | 23.5556 | 27.1872 |

| np | 2000 | 4000 | 6000 | 8000 |
|---|---|---|---|---|
| 16 | 0.8400 | 0.8851 | 0.9102 | 1.1109 |
| 32 | 0.6703 | 0.6753 | 0.7361 | 0.8496 |

Table 2. Speedup and Efficiency for different N and np

Problem 1 continued on next page...

Plots Obtained are shown below:

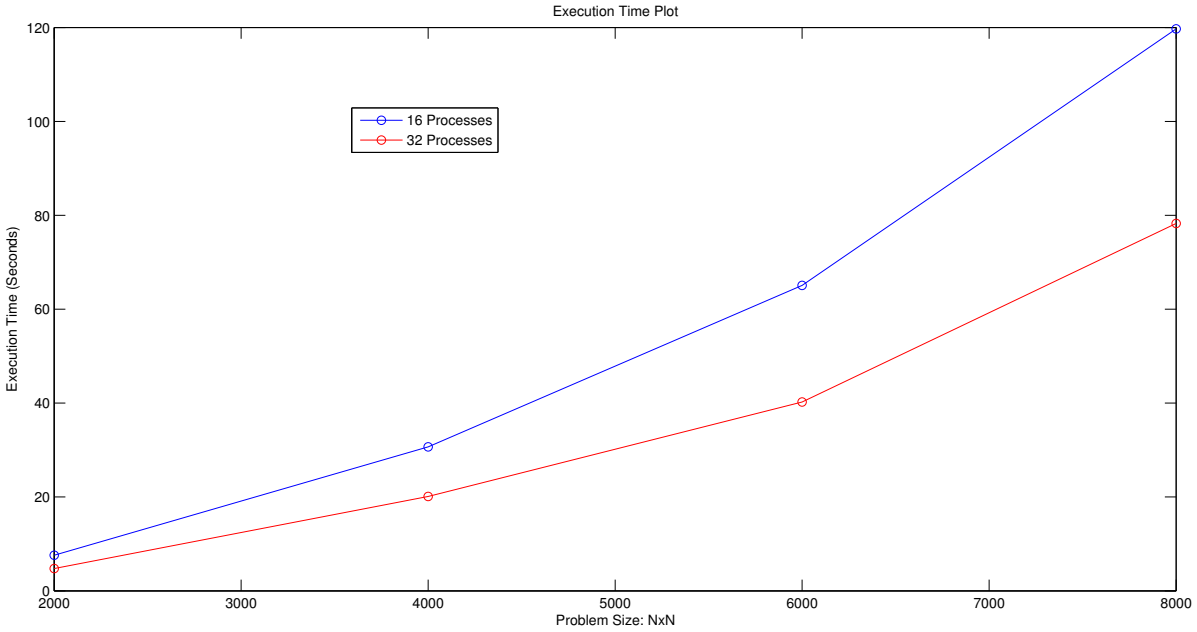


Fig. 1. Execution Time Plot for 16 and 32 processes

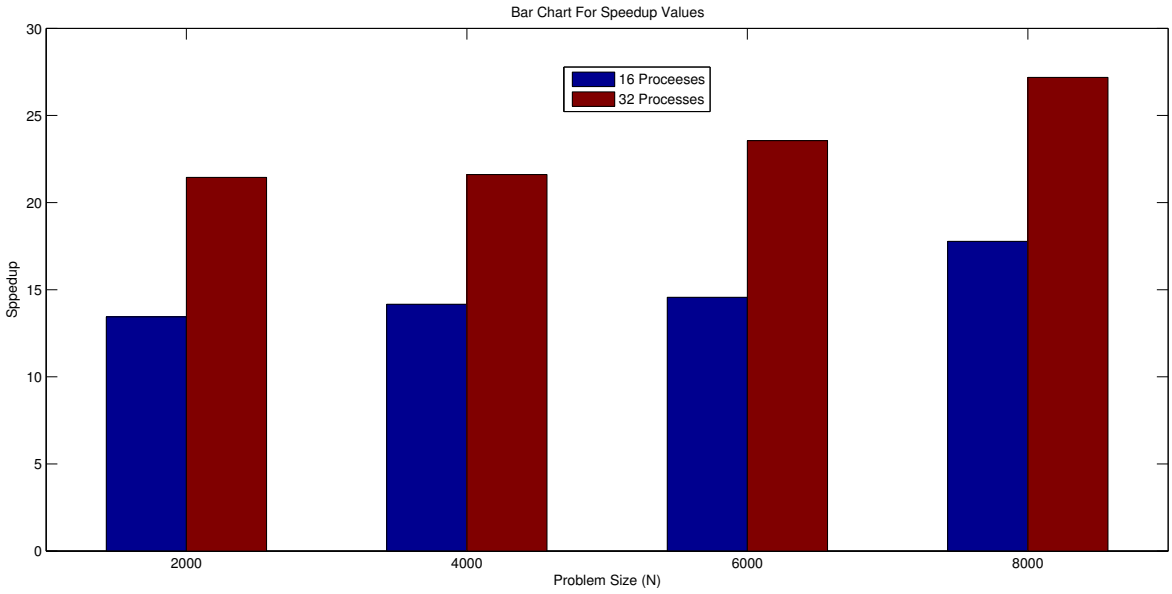


Fig. 2. Speedup Bar-Chart for 16 and 32 processes

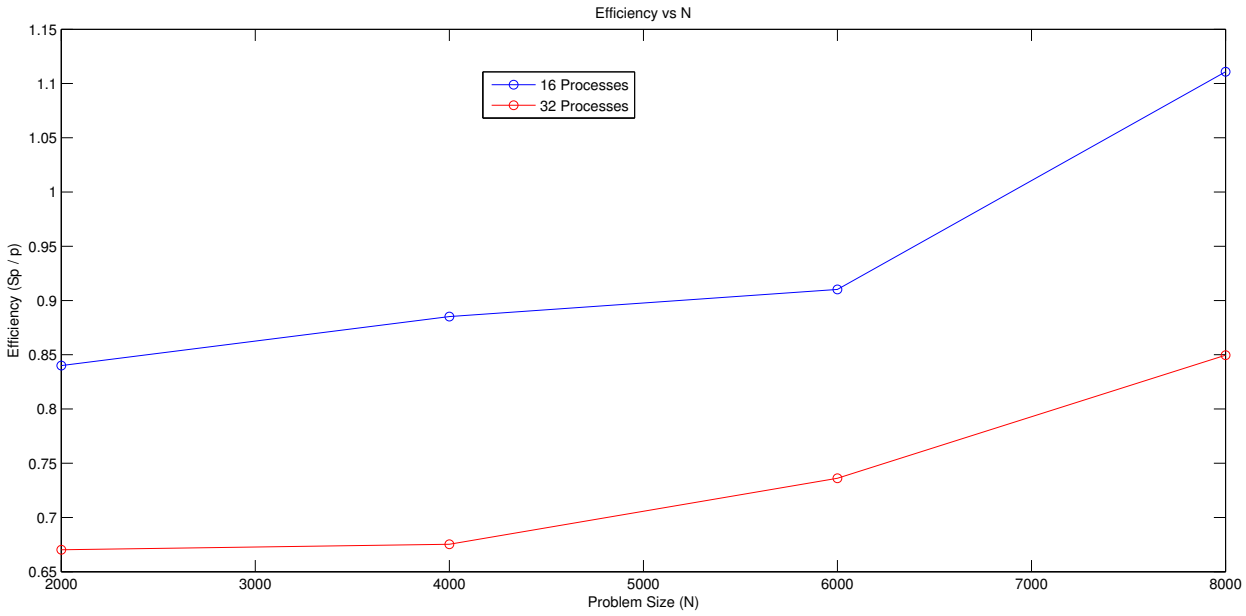


Fig. 3. Efficiency plot for 16 and 32 processes

**Observations:**

- With increase in problem size(N), Execution time and hence speedup increases. It can be noted that, **Efficiency** also increases with increase in N.
- When increase in no. of processes (16 to 32), Execution time and hence speedup increases. But, **Efficiency** decreases, as expected.
- MPI Cartesian topology helps in no. of ways, like, easier mapping of higher dimensional data onto processes, accessing neighbors, replacing blocks.
- With the knowledge of actual processor topology, if Cartesian, It can be used to reduce communication time by mapping processes similar to processor.

**Profiling Outputs:**

With the screen shots shown on the next page, we can see the distribution of various function calls over the time. For, example, For 2000 iterations, time spent on MPI_Barrier and MPI_Sendrecv function for 16 and 32 processes can be see by yellow and green lines respectively.

Problem 1 continued on next page... 

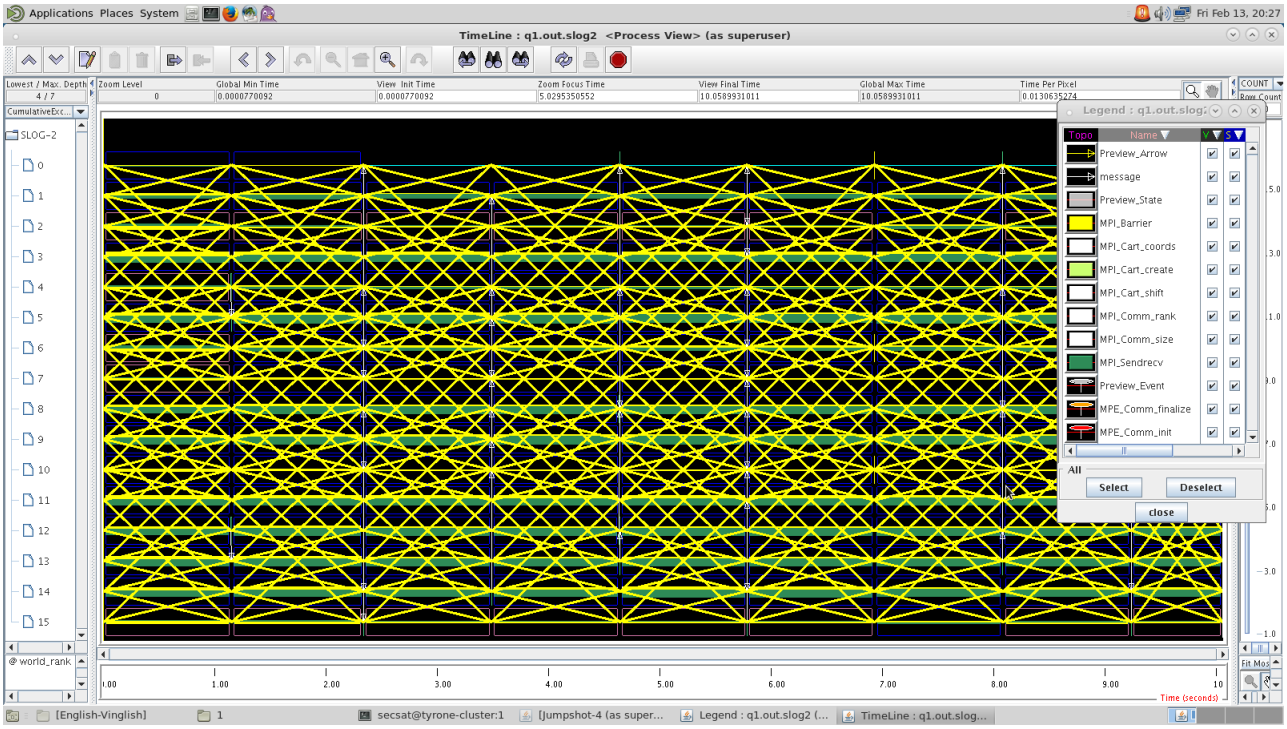


Fig. 4. Screen-shot of process view for 16 processes with N=2000

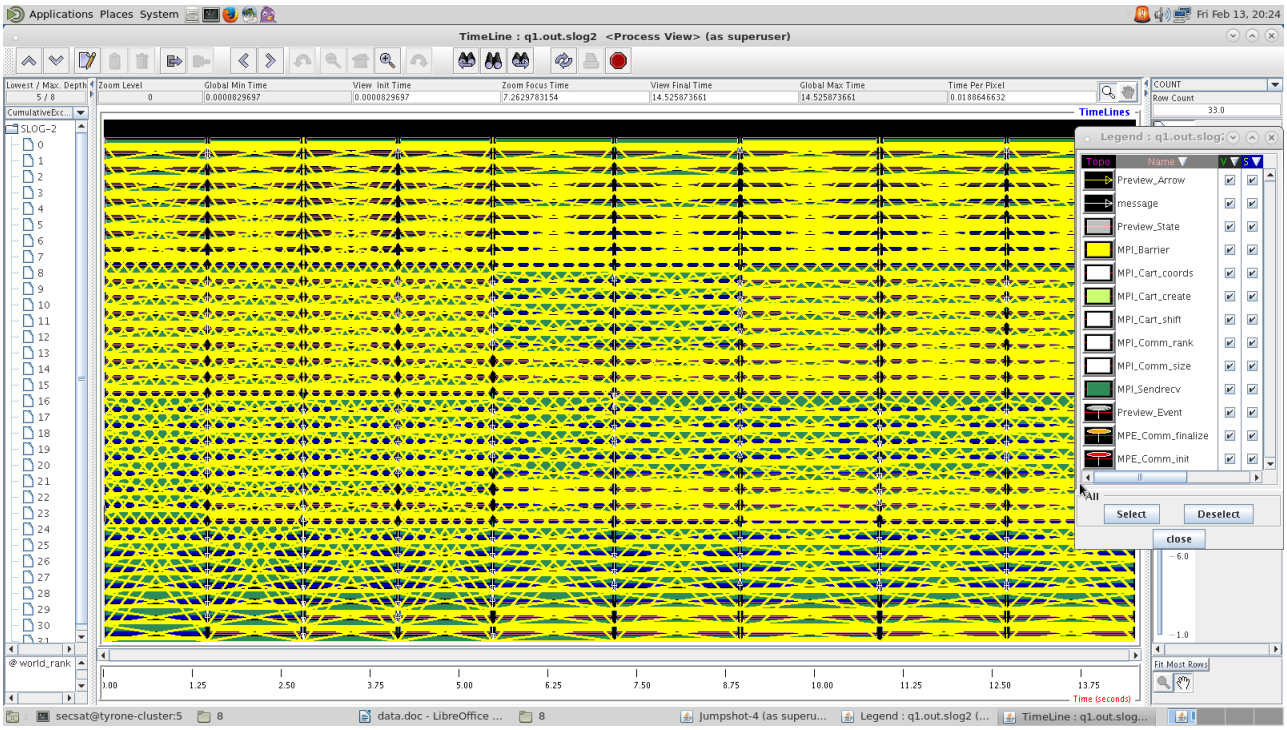


Fig. 5. Screen-shot of process view for 32 processes with N=2000

# Problem 2

**Programming Strategies:**

1. To store **Harwell-Boeing matrices** I have used CSR format. To obtained data, I used *https://math.nist.gov/MatrixMarke Boeing/*. Data obtained can be put separately into three files corresponding to three vectors of CSR format (Val, Column Index and Row-pointer).

2. To maximize the parallelism, my strategy is to perform 1 row computation each threads.

3. Due to limitation on no. of threads per block, I used maximum no. of threads per block as 1000 and no. of blocks as (No. of rows / 1000).

4. To obtained maximum possible data size to fit into GPU, Error-Checking function was used.

**Results**

1. For upto 8,80,000(double) no. of nonzero entries, GPU-paralel program was working. Here, I used square matrix of size 80,000.

2. Execution time obtained is 3.319 seconds in parallel implementation, while sequential program was taking 1.218 seconds. In parallel GPU program, time taken in transferring data took 3.125 seconds while computations took 0.019 seconds.

**Observations:**

1. Since CSR Format of storing sparse matrix is quite space efficient, We were able to perform operation for 8,80,000 nonzero entries.

2. Despite of maximizing no. of threads, due to transfer of data between CPU and GPU, parallel program took more time.

**Note:** Due to some unknown reasons, Even after trying many-many times, I was unable to generate csv file to perform profiling.

END.