CS633A

Parallel Computing

Assignment 1

Halo Exchange Simulator

Aditya Jain
20111004

Mohit Kumar
20111034

Instructor:
Dr. Preeti Malakar

February 23, 2021

1 Steps to run the code

- **NOTE:** For automated plot generation, please install python and following python libraries - matplotlib and numpy.
- Execute run script from terminal as :- $ ./run.sh
- Above command will make the program, generate the host file on fly, execute the program with all required configurations, and will generate the output files which are required to generate the boxplots.

2 Implementation Details

2.1 Explanation of Code

2.1.1 General Background

- In 2-D grid decomposition of processes, every process do not has the same number of neighbours. For example :- Process 0 has only two neighbouring process with whom it has to communicate where as Process 1 has 3 neighbouring process with whom it has to communicate in $n * n$ grid ($n > 1$).
- We categorized processes on the basis of number of neighbours they have. There exists 3 such categories:
    1. Corner Processes: Process having only 2 neighbour. There are 4 such processes. These represents the 4 corners of the 2-D grid. Suppose if the grid is 4 * 4. The processes having only 2 neighbour are process 0, process 3, process 11, and process 15.
    2. Border Processes: Process having only 3 neighbours. These processes lie in the first row, first column, last row, last column (excluding the type 1 (mentioned above)) of the 2-D grid. Suppose if the grid is 4 * 4. The processes having only 3 neighbour are process 1, process 2, process 4 etc.
    3. Center Processes: Process having 4 neighbours. These processes do not lie in the first row or first column or last row or last column of the 2-D grid.Suppose if the grid is 4 * 4. The processes having only 4 neighbour are process 5, process 6, process 9, process 10.

- In our program, we coded separately for the categories mentioned above.

2.1.2 Method 1. Multiple Send(), Recv()

- To send each double element explicitly, we have used MPI Send() and MPI Recv() in for loop. In this method, N messages are sent per neighbour.

2.1.3 Method 2. MPI Pack(), MPI Unpack()

- To send elements of a halo exchange region. We first pack the data elements together that lies in one halo exchange region using MPI_Pack().
- After packing the data elements, we communicate among the neighbouring processes by using MPI Send(). Only message is sent per neighbour.
- The receiver receives the packed data element by using MPI Recv().
- For unpacking the data elements, MPI Unpack() is executed N times inside a for loop.

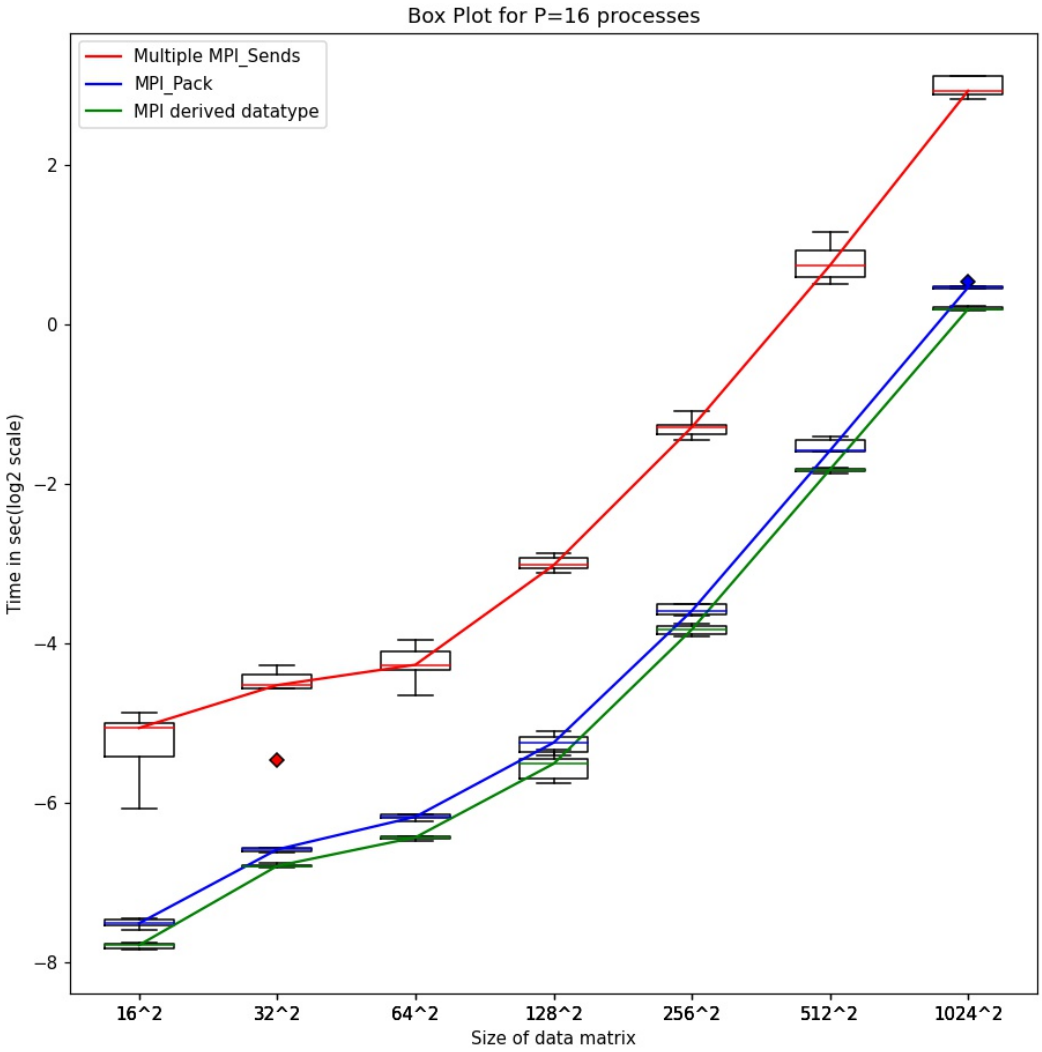
2.1.4 Method 3. MPI derived datatypes

- To send and receive the data elements using MPI derived data type, we use two techniques:-

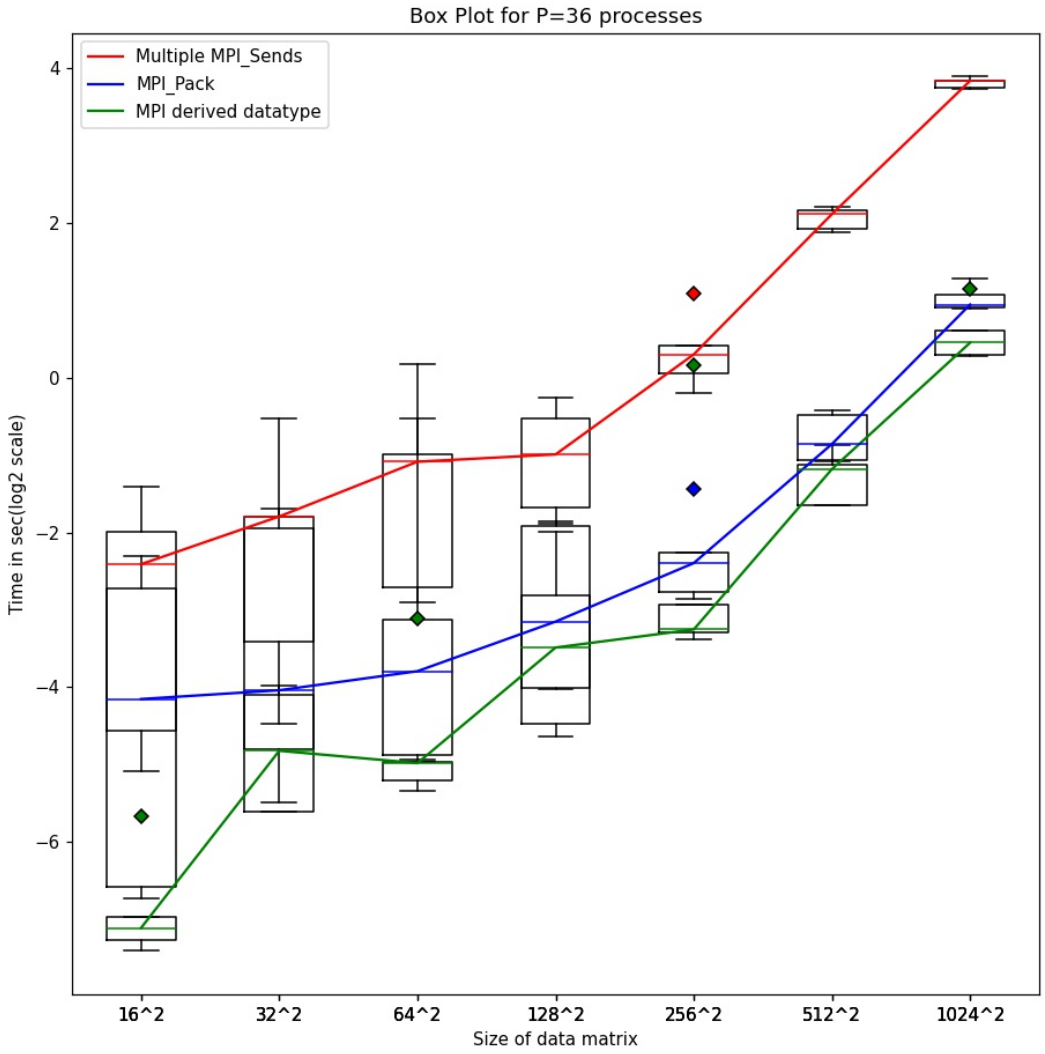
    1. For sharing top and bottom row elements: We created MPI_contiguous data type (MPI derived data type) which clubs the elements of a row of the data matrix into one data variable. We then sent and receive this MPI_contiguous data type using MPI Send() and MPI Recv() respectively.
    2. For sharing first and last column elements: We created a MPI_vector data type (MPI derived data type) which clubs the elements of a column of the data matrix into one variable. We then sent and receive this MPI_vector data type using MPI Send() and MPI Recv() respectively.

3 Simulation Results

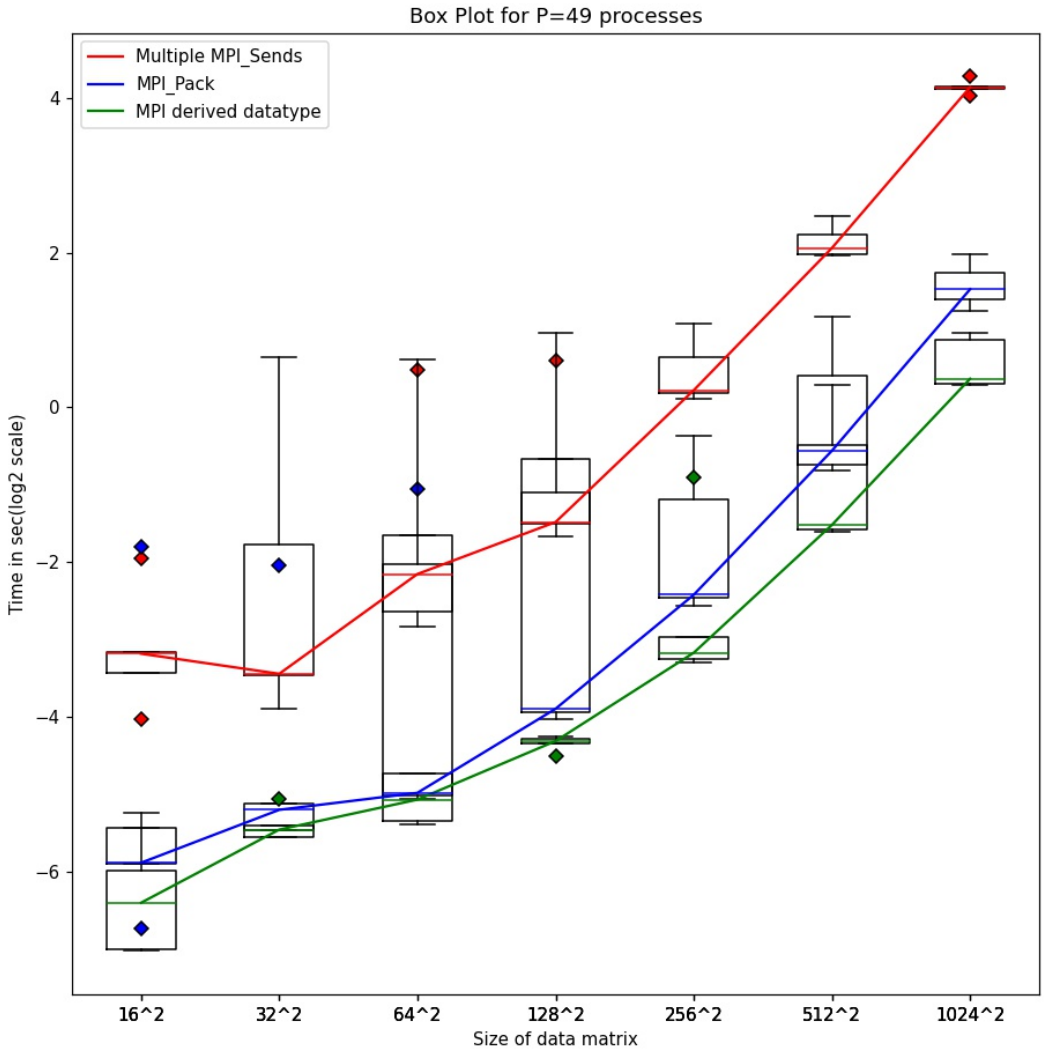
3.1 Box Plot for P=16 processes



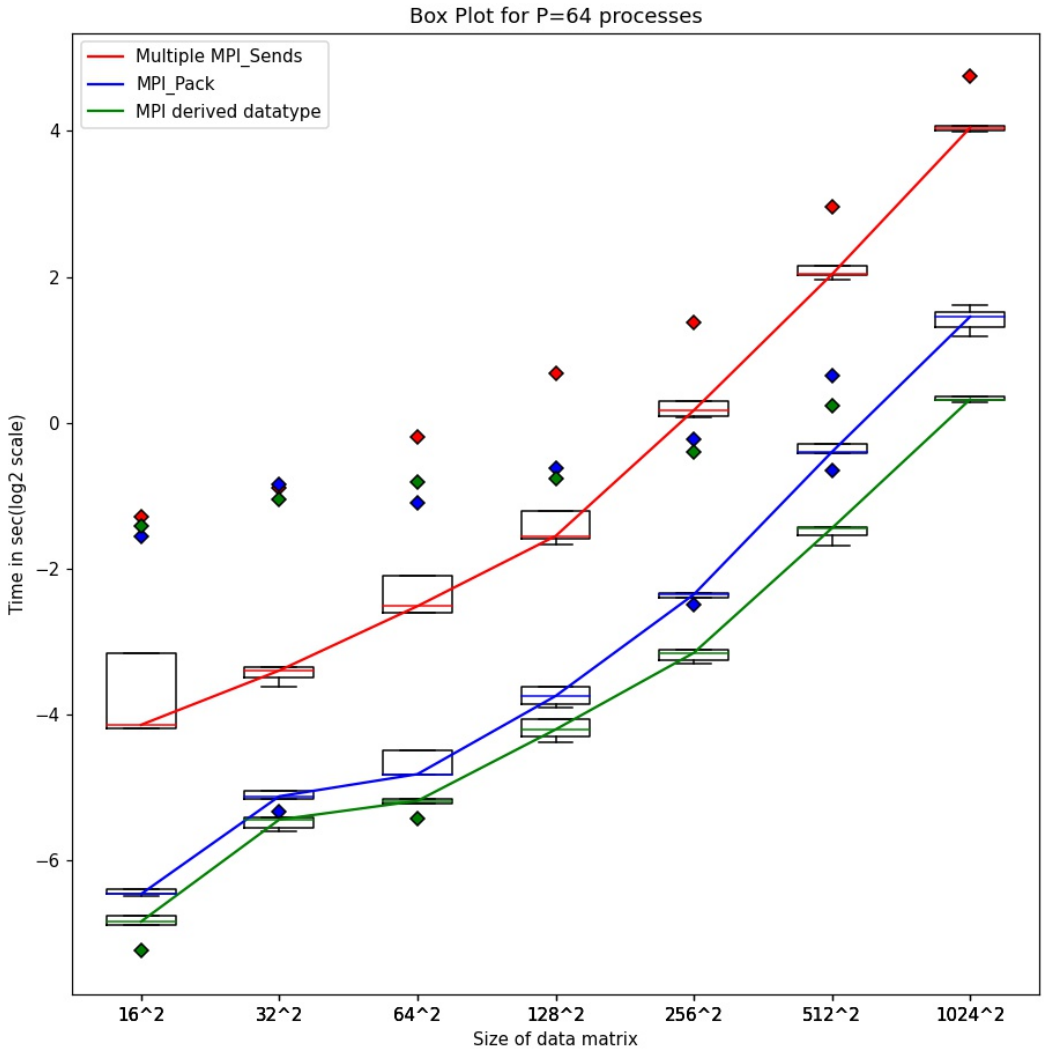
3.2 Box Plot for P=36 processes



3.3 Box Plot for P=49 processes



3.4 Box Plot for P=64 processes



4 Simulation Analysis

4.1 Time increases if the data size increases

- For a fixed number of processes, with increase in size of data matrix from 16^2 to 1024^2, the time taken also increases.
- This happens as more data need to be transfered from one node to another and more elements are needed to be computed.

4.2 Time increases in exponential order in case of Multiple MPI Sends

- As in case of Multiple MPI Sends, we are sending each data element explicitly. That is N elements are sent per neighbour.
- So on increasing the size of the data matrix, i.e. the value of N, number of MPI_Send() function calls increases. Since more messages are travelling across the communication channel, time for the program increases.
- From graph, we can observed that it follows linear order. Since the time is reprsented in logarithmic scale in graph, program follows exponential order of time increase when value of N is increased.

4.3 Trend of time taken in MPI Pack is almost same as trend of time taken in MPI derived datatype

- The trend of time taken with increase in size of data matrix for fixed number of processes is almost similar in MPI Pack and MPI derived data type.
- MPI Pack and MPI Unpack requires a loop to pack/unpack data elements respectively.
- This causes MPI Pack method to take more time as compare to MPI derived data type.
- So,overall MPI derived data type gives best performance in terms of time taken.