README

Ankita Dey, 20111013 P J Leo Evenss, 20111038

This assignment helps us understand the comparision of 3 types of communications used in MPI.

Code Explanation:

1. All the variables needed are initialized appropriately.
2. The number of rows and columns in Processor matrix (logical topology) is obtained by sqrt from the MPI_size.
3. The N^2 datapoints (subdomain) obtained from the command line is built into a N X N matrix using dynamic allocation.
4. The NXN matrix is initialised with random doubles using rand() function.
5. The boundary processors are detected based on their ranks using :

   myrank<P - filters out all processes in the top most row,

   myrank%P!=0 - filters out all processes in the left most column,

   (myrank+1)%P!=0 - filters out all processes in the right most column,

   (myrank+P)<size - filters out all processes in the botoom most row,

6. The data is sent for halo exchange as given in question.

   To send top halo region to the processor just above, destination = myrank-P

   To send left halo region to the processor just left, destination = myrank-1

   To send right halo region to the processor just right, destination = myrank+1

   To send bottom halo region to the processor just below, destination = myrank+P

7. Multiple MPI_Sends:

   Using MpI_Send() each element is sent individually with a count of 1.

   Row is fixed to 0 and N-1 for top and bottom rows respectively, and columns iterate from 0 to N-1 using a for loop to indicate buffer start position each time a datapoint is sent.

   Column is fixed to 0 and N-1 for left and right columns respectively, and rows iterate from 0 to N-1 using a for loop to indicate buffer start position each time a datapoint is sent.

   Corresponding multiple MPI_Recv is used at the receiver process to receive the individual data points.

8. MPI_Pack() - is used to pack a row/column of datapoint doubles into a buffer of size N*sizeof(double).

   (separate buffers for up,down,left,right are used for both send and recv)\

   MPI_Send() - is used to send the packed data with datatype MPI_PACKED.

   MPI_Recv() - corresponding to send is used to recv the packed data.

   MPI_Unpack() - is used to unpack the data into an array.

9. MPI_Send/Recv using MPI derived datatypes:

   MPI_Vector() is the derived datatype used to pass a regular blocks of contiguous or non contiguous datapoints together.

   For top and bottom rows, 1 block of N contiguous elements is sent (count = 1, blocklength = N, stride = N) to the processor myrank-P and myrank+P respectively.

   For left and right columns, N blocks of 1 element is sent (using count = N, blocklength = 1, stride = N) to the processor myrank-1 and myrank+1 respectively.

10. All the data received at the receiver side is updated into the datapoint matrix accordingly, right after MPI_Recv, for every timestep using the formula,

    [value (P, t+1) = [value (P_left, t) + value (P_right, t) + value (P_top, t) + value (P_bottom, t)] / 4]

    and time is recorded for 50 timesteps. Thus, processes alternate between communication and computation.

11. The time is measured using MPI_Wtime() from start to end of communication and computation and MPI_Reduce is used to measure the maximum time taken by any process.

    The time is displayed as

    <time for MPI_Send/Recv>

    <time for MPI_Pack/Unpack>

    <time for MPI_Vector>

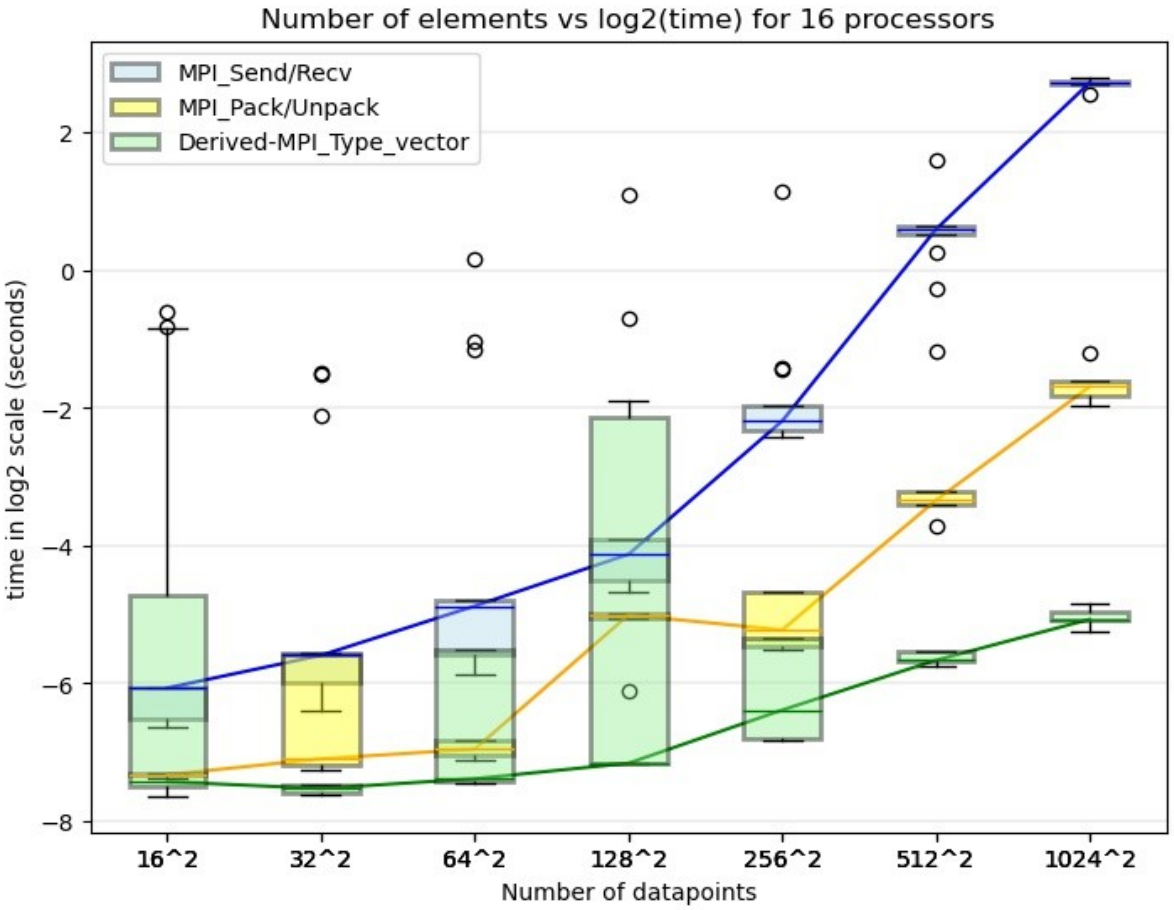
    for each N for each P for 5 iterations as given in assignment.

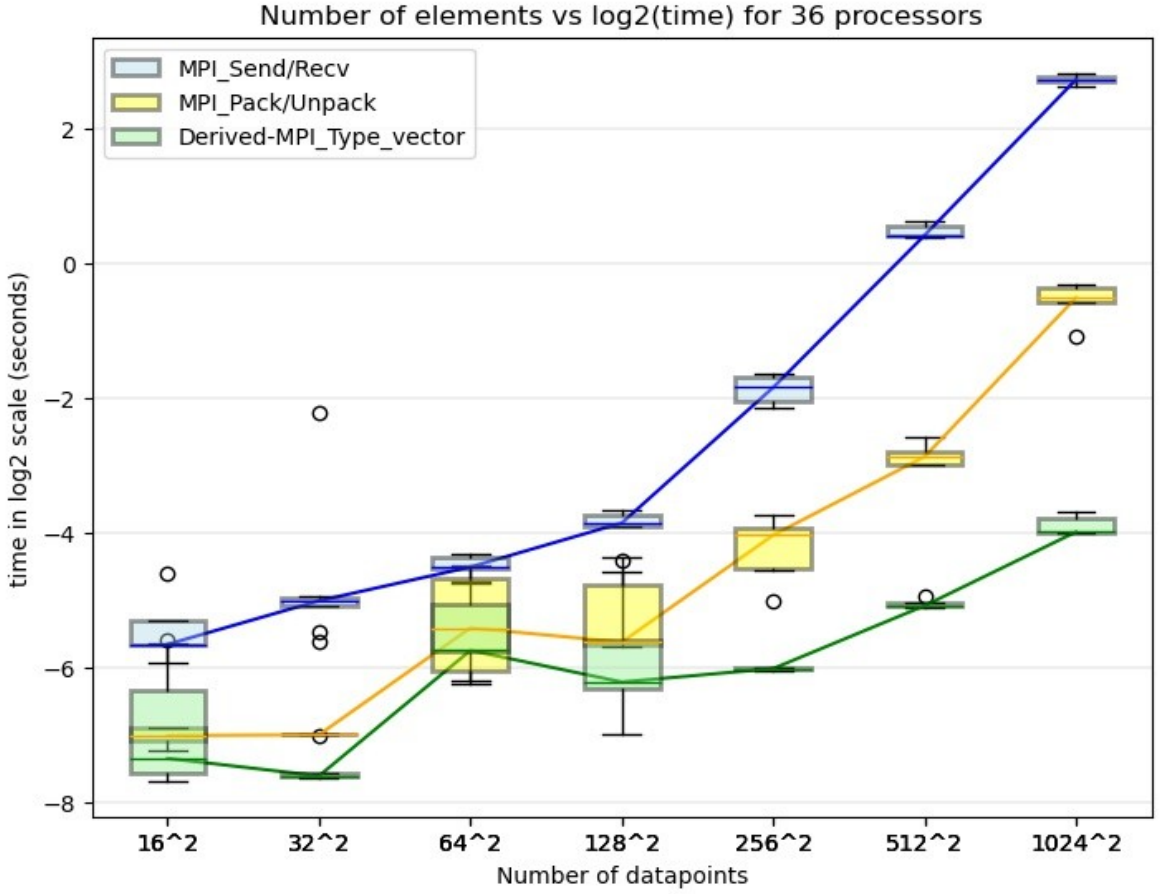
12. The time is stored in a text file, data.txt. It is used to generate the plots using python script, plot.py after src.c. is run.

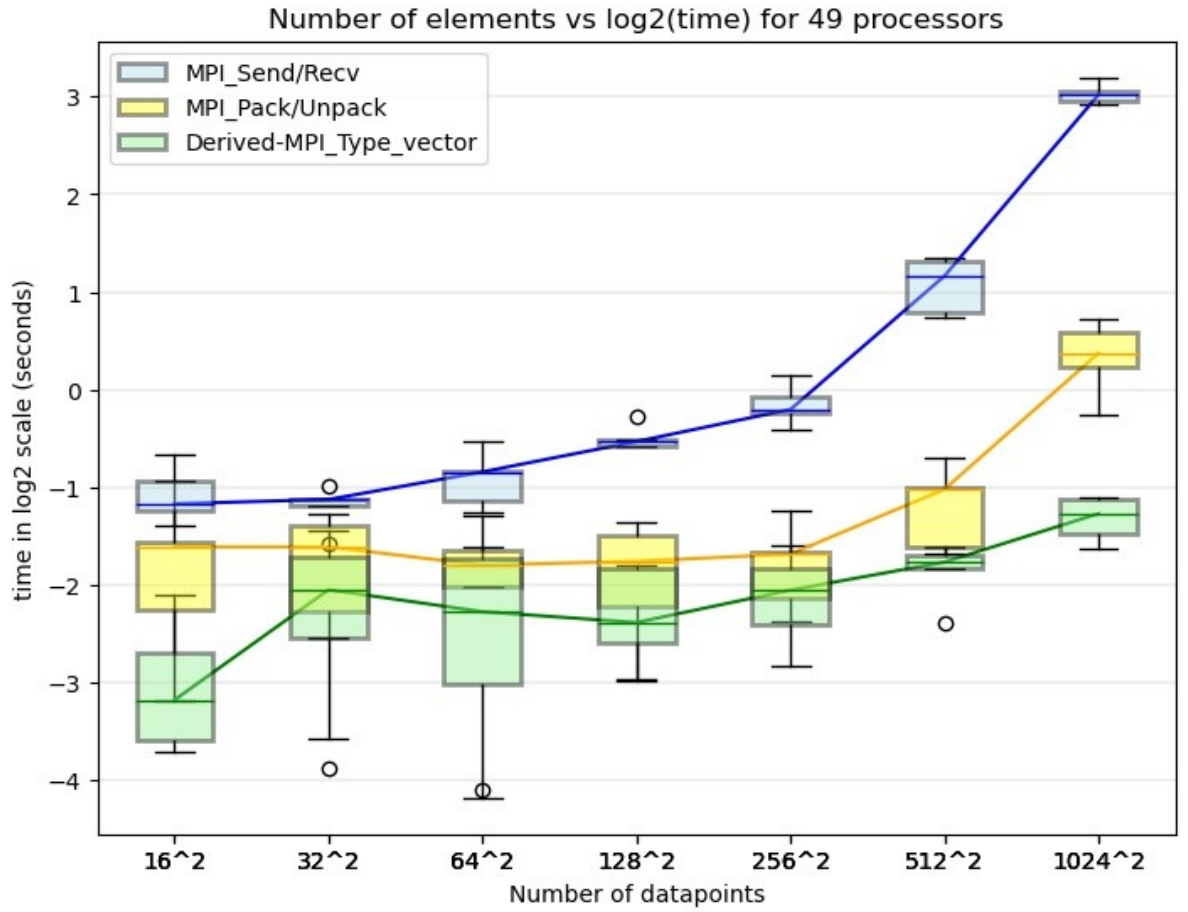
    The script, plot.py plots the time (in log2 scale) for each data size per method per process count as asked in the assignment.

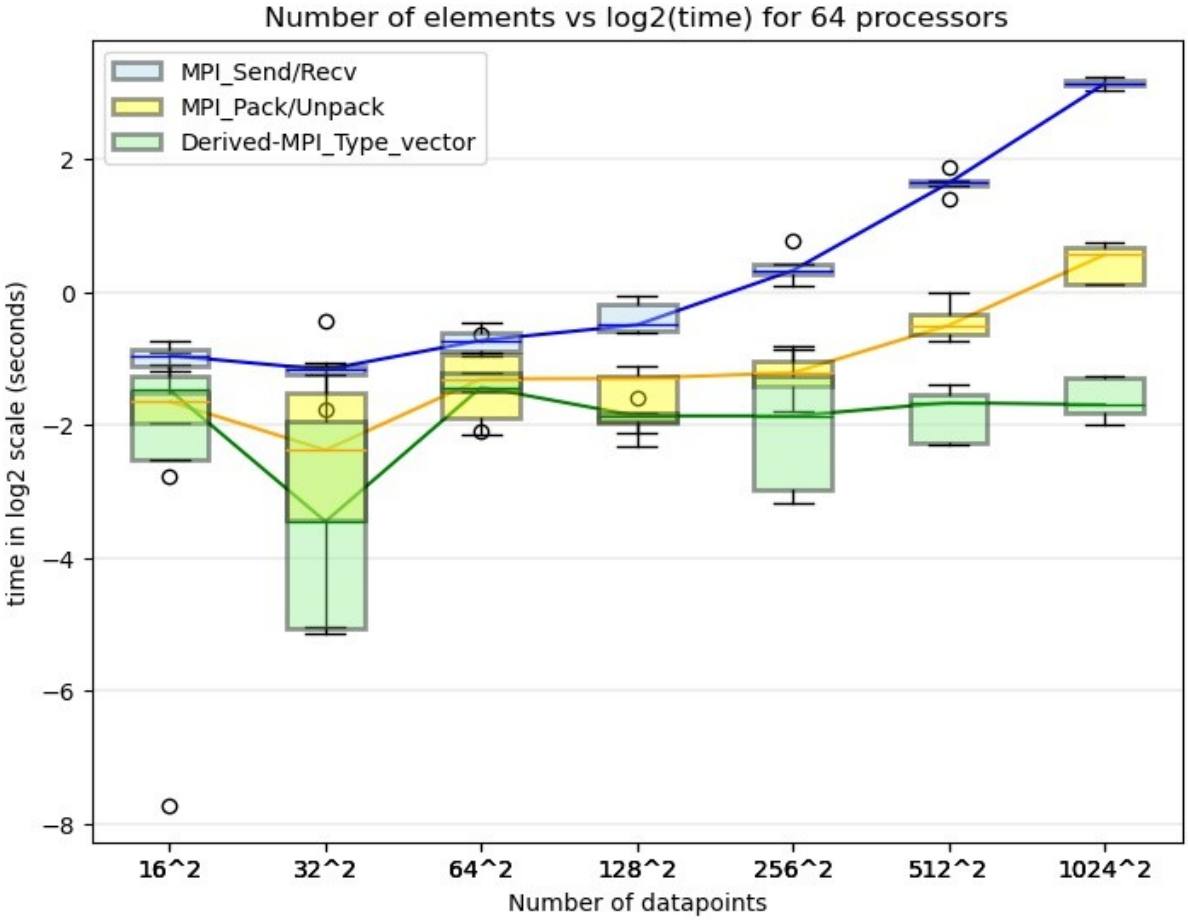
Plots:

1. Boxplots (from the 5 executions) is used to denote every data point in the plots.
2. Line graph connects the median of 5 execution of each data transfer method.
3. Number of data points, N is in x axis and time in seconds in log2 scale is in y axis.
4. Light blue, yellow and light green is used to fill box plots for MPI_Send/Recv, MPI_Pack/Unpack and MPI_Type_vector respectively.
5. Blue, orange and green is used to mark median of box plots and line graphs connecting them for MPI_Send/Recv, MPI_Pack/Unpack and MPI_Type_vector respectively.
6. The 4 plots for 16, 36, 49 and 64 processors are shown in the next 2 pages.
7. Note: The graphs can change according to the load on the nodes


Number of elements vs log2(time) for 16 processors
MPI_Send/Recv
MPI_Pack/Unpack
Derived-MPI_Type_vector
time in log2 scale (seconds)
2
0
−2
−4
−6
−8
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
Number of datapoints


Number of elements vs log2(time) for 36 processors
MPI_Send/Recv
MPI_Pack/Unpack
Derived-MPI_Type_vector
time in log2 scale (seconds)
2
0
−2
−4
−6
−8
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
Number of datapoints


Number of elements vs log2(time) for 49 processors
MPI_Send/Recv
MPI_Pack/Unpack
Derived-MPI_Type_vector
time in log2 scale (seconds)
3
2
1
0
−1
−2
−3
−4
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
Number of datapoints


Number of elements vs log2(time) for 64 processors
MPI_Send/Recv
MPI_Pack/Unpack
Derived-MPI_Type_vector
time in log2 scale (seconds)
2
0
−2
−4
−6
−8
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
Number of datapoints

Observations:

1. For any number of processors, it can be seen that the time taken for multiple MPI_Send/Recv is much more than MPI_Pack/Unpack and MPI_Type_vector. (Time has been plotted in log2 scale, so actual difference in time is more than that shown in the graph).

   Reason: Since each datapoint is sent using a separate MPI_Send, there is extra network overhead for each of the datapoints resulting in more communication time.

2. Time taken for MPI_Type_vector is more than MPI_Pack/Unpack.

   Reason: Although both MPI_Pack and MPI_Type_vector require a single MPI_Send from each source to destination for each execution, MPI_Pack involves copying the datapoints to be sent in a buffer which is then sent using MPI_Send as opposed to derived datatypes (MPI_Type_vector in our case). This overhead due to copying results in extra time taken. The difference gets more stark for more number of data points due to even more copying involved.

3. The more the number of datapoints, that is, the more parallel work required to be done, more is the benefit obtained by optimizing the copying or the network overhead.

Experimental setup:

1. Installed Node Allocator by following the readme and created host files through it.

2. Created a job script run.sh to include all the execution commands for compiling and executing src.c and then running the plot scripts.

3. Included the ~/UGP/allocator/src/allocator.out 64 8 to allocate 64 processes with 8 process per node on the fly.

4. Created a Makefile used for compiling src.c and is included inside the job script.

5. Boxplots are created using plot.py with the help of matplotlib and numpy python libraries and 4 plots were created.

IssuesFaced:

1. Due to overlap between the box plots of the various methods of data exchange, they were getting hidden behind one another. Finally we increased the trasparency of the body colour of box plots and used colors that easily distinguishable from one another.

2. As already warned by the instructors, we did face slow lab clusters specially at night in the last few days. Thankfully most of our work was done till then, so it didn’t pose a big problem.

Code Documentation:

1. Run chmod u+x run.sh

2. Now run ./run.sh (it starts running whole configuration, time to complete depends on load on nodes, from 15 mins to 45 mins).

3. The code starts compiling using make command and output of src.x is obtained.

4. The hostfiles is generated on the fly using allocator.out.

5. The src.x is executed using mpiexec command with for loop execution sequence as specified in assignment.

6. The data.txt file is generated which captures the timing of all the configurations as per the execution sequence.

7. The plots script is executed using python3 plot.py and generates four plots with names plotP.jpg where P is the number of processes.