Halo Exchange

Shobhit Sinha (20111059)

Feb 23, 2021

Submitted Files and execution command

The script to run the configurations is in run.sh which can be executed using **bash run.sh**. The script generates four plot files P16boxplot.png. P36boxplot.png. P49boxplot.png. P64boxplot.png which are the plots for the cases when P=16,36,49 and 64 respectively. The plot files will be generated in the same location where run.sh is present. Also, we have submitted four plots in subdirectory SubmittedPlots.

Implementation Details

We have used **MPI_Sendrecv()** to communicate with neighbours. We initially, planned to use non-blocking MPI_Isend() and MPI_Irecv() with a MPI_Waitall() but observed that MPI_Sendrecv() makes the code more clean as Halo exchange communication leads to bidirectional exchanges with the neighbours.
There was no need to use MPI_Barrier() due to the implicit blocking nature of MPI_Sendrecv() and our communication pattern ensures that if a process has received the data from neighbours and communicated data to the neighbours, it is free to proceed towards computation.
Our implementation is **deadlock free** as there is no cycle in the communication dependency graph. The communication takes places in four phases - Left to Right (Process receives from Right and communicates to Left), Right to Left, Top to Bottom and Bottom to Top. We used **MPI_Cart_create()** to generate a Cartesian Topology and used **MPI_Cart_shift()** to generate source and destination for each of the fours phases. As MPI_Cart_shift() produces -1 if a source/destination does not exist and -1 is nothing but MPI_PROC_NULL, it automatically takes care of boundary communication cases.
We have timed the communication and computation phases for 50 time steps and taken the max using **MPI_Reduce()**. We have also ensured that 2D matrix allocation using malloc() is contiguous as we have used **MPI_Type_vector** as our derived data type.

About Plots

The boxes in the boxplots were overlapping heavily with each other. So, **we split the boxplots for each method and have plotted them side-by-side**. We have joined the median values through a line for each of the three methods.

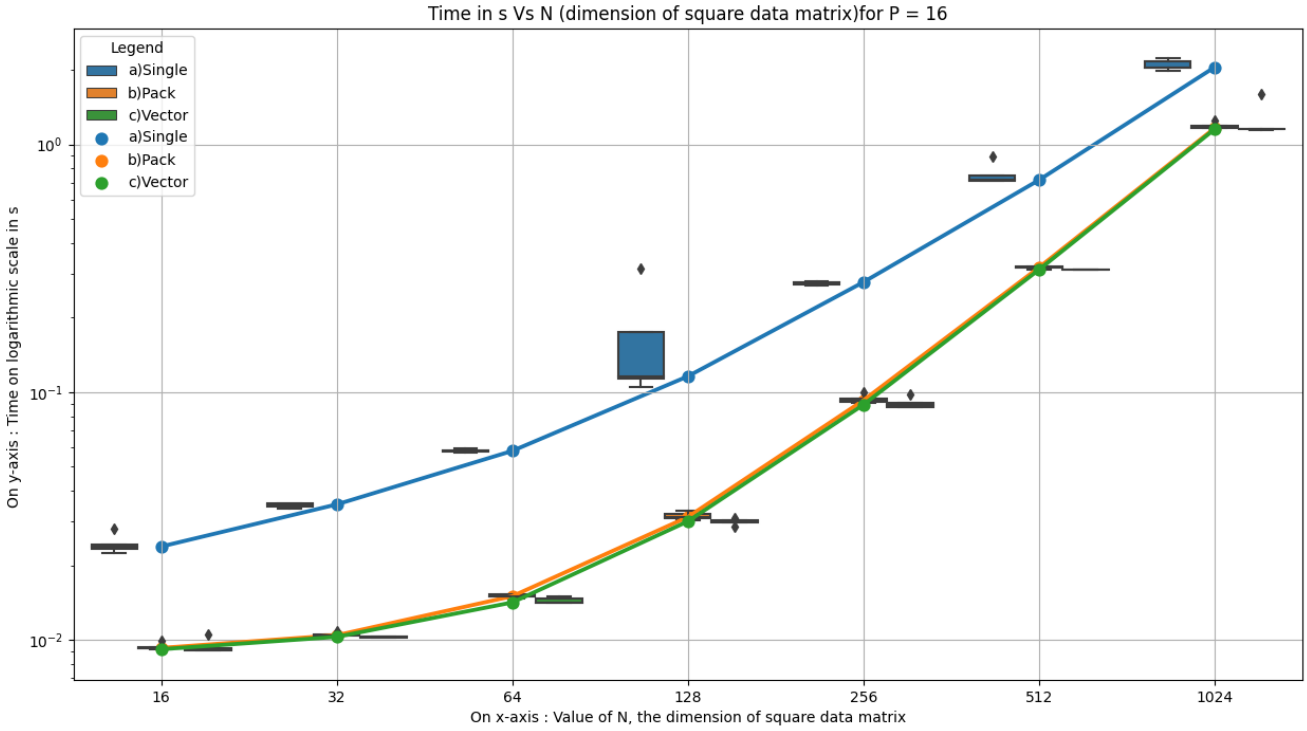

Figure 1: Time in s Vs N (dimension of square data matrix)for P =16

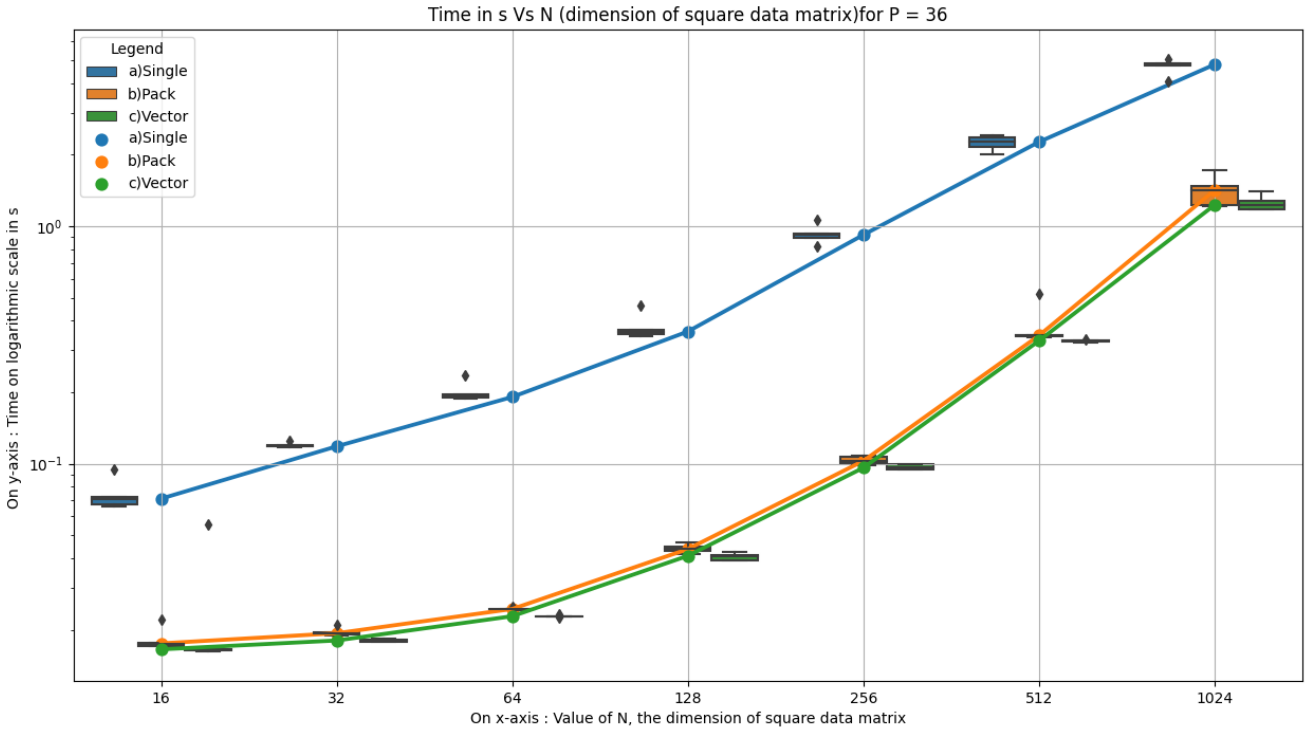

Figure 2: Time in s Vs N (dimension of square data matrix)for P = 36

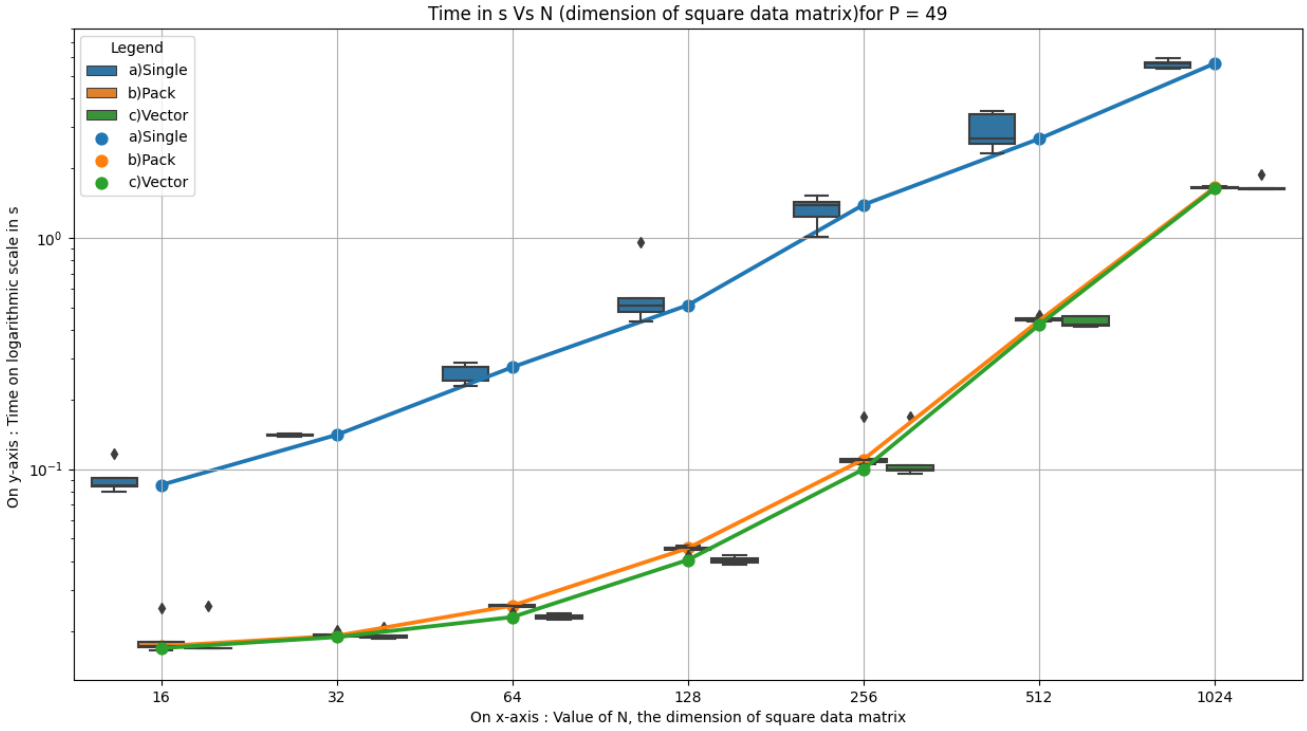

Figure 3: Time in s Vs N (dimension of square data matrix)for P =49

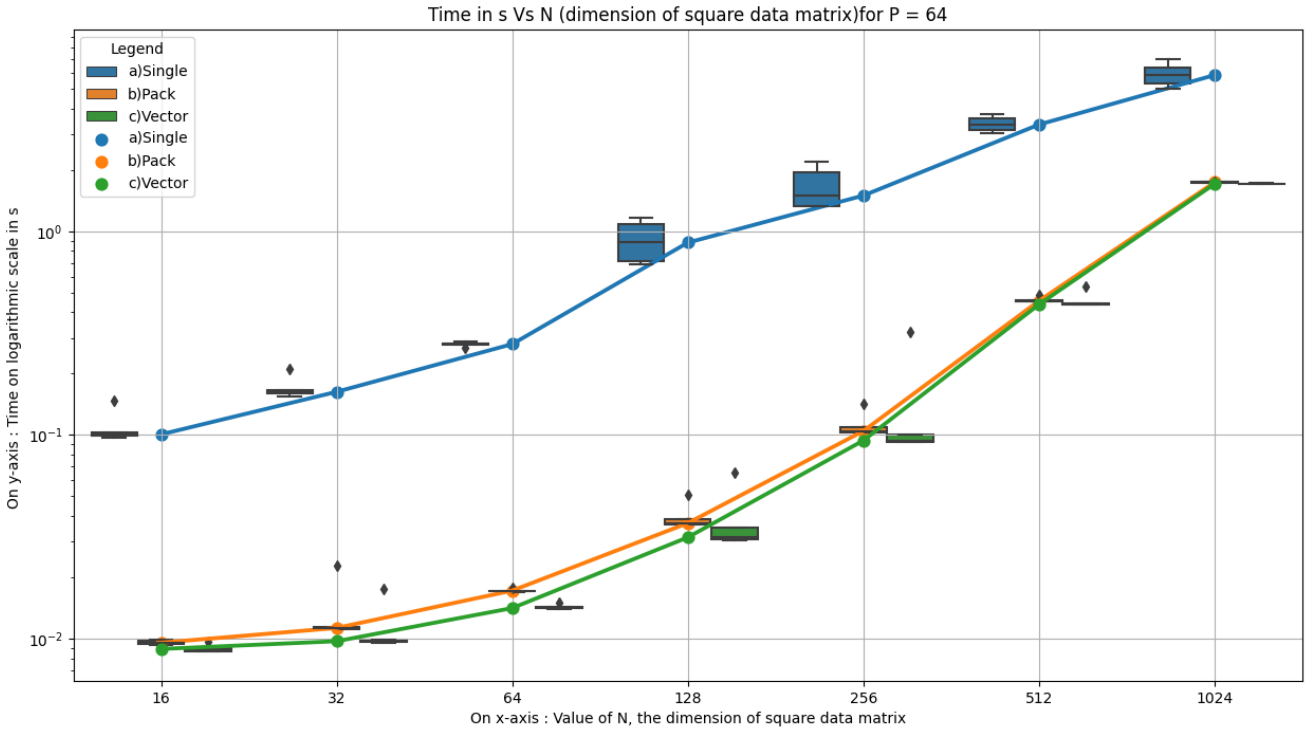

Figure 4: Time in s Vs N (dimension of square data matrix)for P = 64

Observations

As value of N increases exponentially, Time taken also increases in an exponential manner. Though in graph, the plot looks linear as Scale on y-axis is logarithmic.
Sending matrix values, one value at a time (Single) takes significantly more time as more number of small messages have to be transmitted and this is observed in Figures 1,2,3 and 4. Also, the boxplots for this method have more height as more number of messages are in transit and hence, range of observed times has wider spread.
Also, times for both Pack and Vector is almost equal and plot lines are overlapping as message lengths are same. However, as value of P and N increase, there appears increasing gap between the Pack and Vector plot lines with Pack starting to take more time due to additional overhead of Packing and Unpacking. Packing requires extra storage for a copy of the data (which increases as N value increases). On the other hand the Derived Data Type requires extra storage for only the description of the data layout (which is less than the overhead for packing).