Assignment 1, CS633

Saketh Maddamsetty 170612,
Ayush Tharwani, 170201

Directory - Assignment 1
Files

- src.c - source code
- Makefile
- run.py - helper code for execution and plotting
- Readme.pdf - this file

Non standard Libraries : *matplotlib* from python3

Steps to run the code:

1. *python3 run.py*
   make clean, followed by make to compile source, then generates host file using Node Allocator, runs the executable for 5 times for P=16,36,49,64 processes for N^2 (N=16,32,64,128,512,1024) size of arrays, outputs the times for each method in "output.data" and plots the boxplots corresponding to each P number of process in 'plotP.png'.

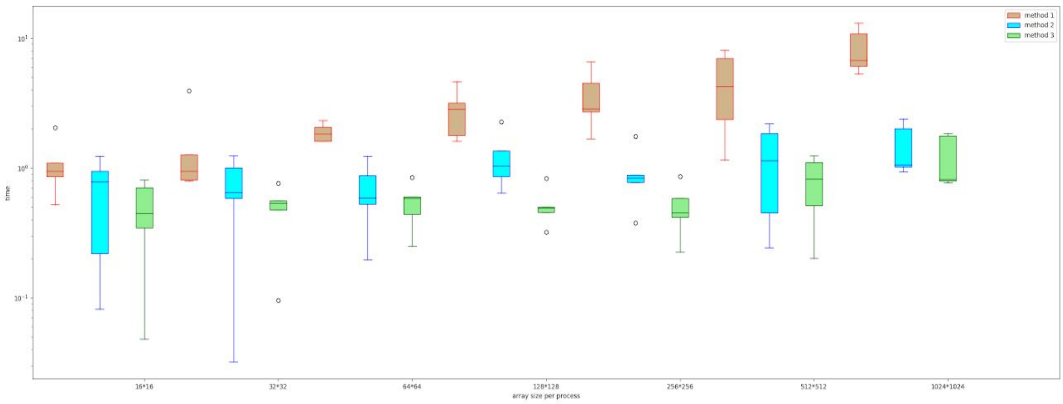
Code Explanation

src.c

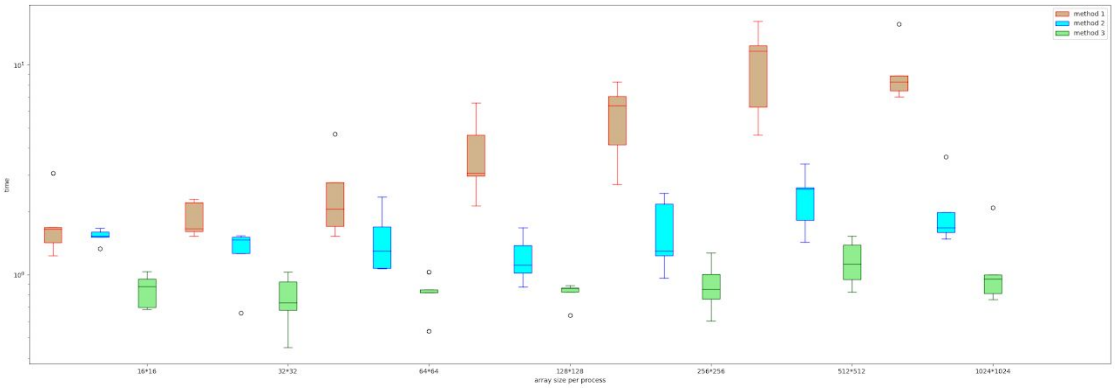
- **run_stencil_computation** function is defined which given the local data points array and all the received data points in form of 4 receive arrays performs stencil computation for that time step.
- In the main, we first compute the four adjacent processes and store their rank in variables left,right,top,bot respectively. (-1 is stored is none exist)
- Data initialisation is performed before each method starts.
- In method 1 we send and receive each individual element separately using MPI_Issend and MPI_Irecv. We use the MPI_Waitall to apply barriers in code and use the **run_stencil_computation** function to perform stencil computation.
- For method 2 we send packed data and receive the entire array of data at once. Rest remains the same as method 1.
- For method 3 we send data using derived data type which is constructed using MPI_Type_vector. Rest remains the same as method 1.
- The output is sent in format of "**timeformethod1**,**timeformethod2**,**timeformethod3**" which is further processed by run.py to output in the required format.

Plots

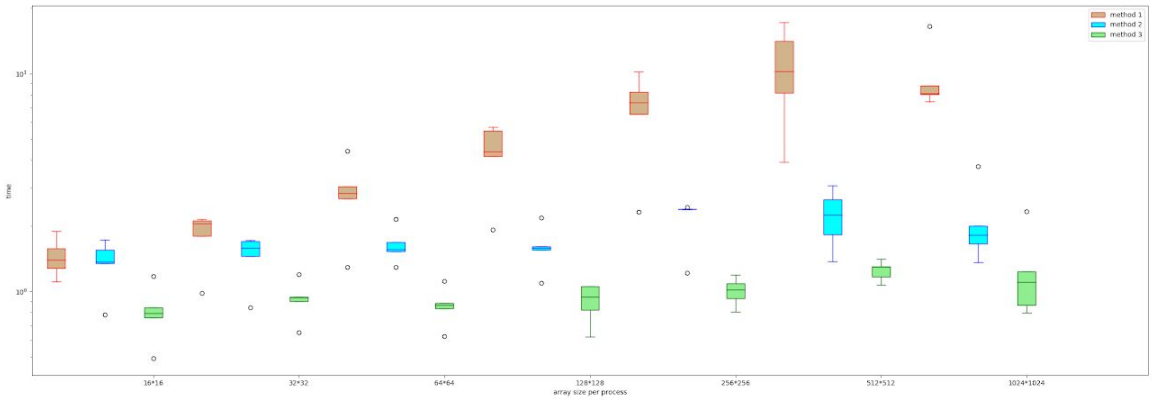
Number of processes = 16



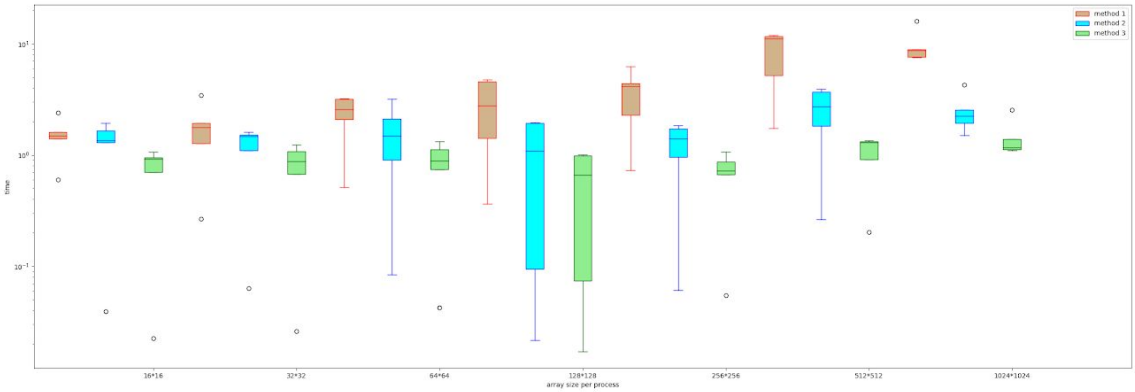
Number of processes = 36



Number of processes = 49



Number of processes = 64



Trends observed in the plots

- **Orange** is **method 1**, **Blue** is **method 2** and **Green** is **method 3**.
- On average, method 3 is taking the least time followed by method 2 followed by method 1. This is justified since method 1 sends each array element individually adding to extra overheads. Method 2 is performing worse than method 3 since we are packing data into a buffer before sending in method 2 whereas we are directly sending data from the array used for computation in method 3 using derived data types. This extra copying is avoided.
- For a given number of processes, Time increases for method 1 as the data points per array(array size) increases since the number of sends and receives are increasing proportional to array size.
- For a given number of processes, Time for method 2 and 3 almost remain constant since the number of sends would be constant and the network latency might be a major contributing factor for total latency.

- When we increase the total number of processes, the time remains constant on an average (i.e. when we compare method2 for p = 16 and p = 32 for a particular array size). This is because the load on an individual server is constant (~ 8 processes per node) and the amount of communication done per MPI process is also constant (at most 4 to its left, right, top, bottom).

Issues faced

- make command leading to clock skew, sometimes interrupt with mpiexec call
- Code takes approx 20-30 mins from compilation to plotting
- Generating host files for every mpiexec call is faster than generating hosts for only different process counts in terms of complete execution of code.
- [**Important**] If every mpiexec call with hostfile “hosts” has been given timeout of 25 seconds in case of overload of servers and corresponding new mpiexec call is generated with hostfile as “hostsimproved” to ensure smoooth execution. Please note there are such 5*4*7=140 calls, so this may take a slightly large amount of time (timeout may occur for 1-2 calls). After execution of each call, code prints “**ok**” on stdout for giving sense of progress.
- In the job script, we observed that mpiexec throws “not able to parse hostfile error” after “make” call. On running the mpiexec with generated hostfiles separately on terminal worked fine. We were not able to understand the nature of error. We have handled this error with try except call in python by giving 5 seconds of sleep time for ‘except’ part of the code. This way code finally worked. So, **Please wait for 15-20 seconds before printing of first “ok”**.