Assignment 1

Mahi Agrawal (20111029) | Tamal Deep Maity (20111068)
maahi20@iitk.ac.in | tamalmaity20@iitk.ac.in

Contents

1. Running the Jobscript

To run the jobscript use the following command :-

```
python3 run.py
```

2. Code Explanation

For performing stencil computation, a process had to communicate to the neighbouring processes for taking the data for the computation of the value of the boundary cells.

We have taken two arrays, one with the name "arr" to store data points and the other with the name "new_arr" as auxiliary array to perform the updation of data points. It was required to take the auxiliary array because if we had updated the data points in "arr" itself then updating one cell with its neighbors would mean that when that neighbor would get updated it would use the new updated value in place of the old time step value of its neighboring cell.

For Communicating with the neighbouring process we divided the communication in four steps.

- **Step 1** :- Top to bottom communication :- In this the processes in a row sends data (last row of the data point matrix) to the process in the cell which is just below it.
- **Step 2**: - Bottom to top communication :- In this the processes in a row sends data (first row of the data point matrix) to the process in the cell above it.
- **Step 3** :- Left to right communication :- In this the processes in a column sends data (last column of data point matrix) to the process in the cell just right to it.
- **Step 4:**- Right to Left communication :- In this the processes in a column sends data (first column of data point matrix) to the process in the cell just left to it.

For derived datatype we have used MPI_Type_vector, and we have created two different datatypes for vector, one for row and the other for column.

For computation the steps are as followed :-

- **Step 1** :- For one particular data matrix of a process, if the cells for which we are computing are not boundary cells, then to compute the value there is no data required from the neighbouring process. So, we have calculated them first.

- **Step 2** :- Then all the corner processes will have only two neighbours, so accordingly computing the new average value of the data points belonging to those corner processes.

- **Step 3** :- Then all the boundary processes except the corner processes will have 3 neighbouring processes, so accordingly computing the new average value of the data points belonging to those boundary processes. (First for the boundary processes which are the part of first and last row and then the boundary processes of first and last column of the process matrix, where the process matrix is a N*N 2-d process layout)

- **Step 4**:- Then computing the new average value of all the boundary data points of all the remaining processes which have 4 neighbouring processes each.

3. Problems/Issues Faced

- Initially our code for multiple MPI_Send() and MPI_Recv() was going into deadlock, as we were passing wrong tag values.

- We realised that the code was failing for N=1024 if the arrays for data points were statically allocated and so we dynamically allocated the array as it allocates memory in the heap rather than the stack memory.

- For Pack and Unpack we didn't realise that buffer size is in bytes, so our buffer was overflowing. We changed count to reflect the total number of bytes received (number of doubles * size of double) which solved the issue.

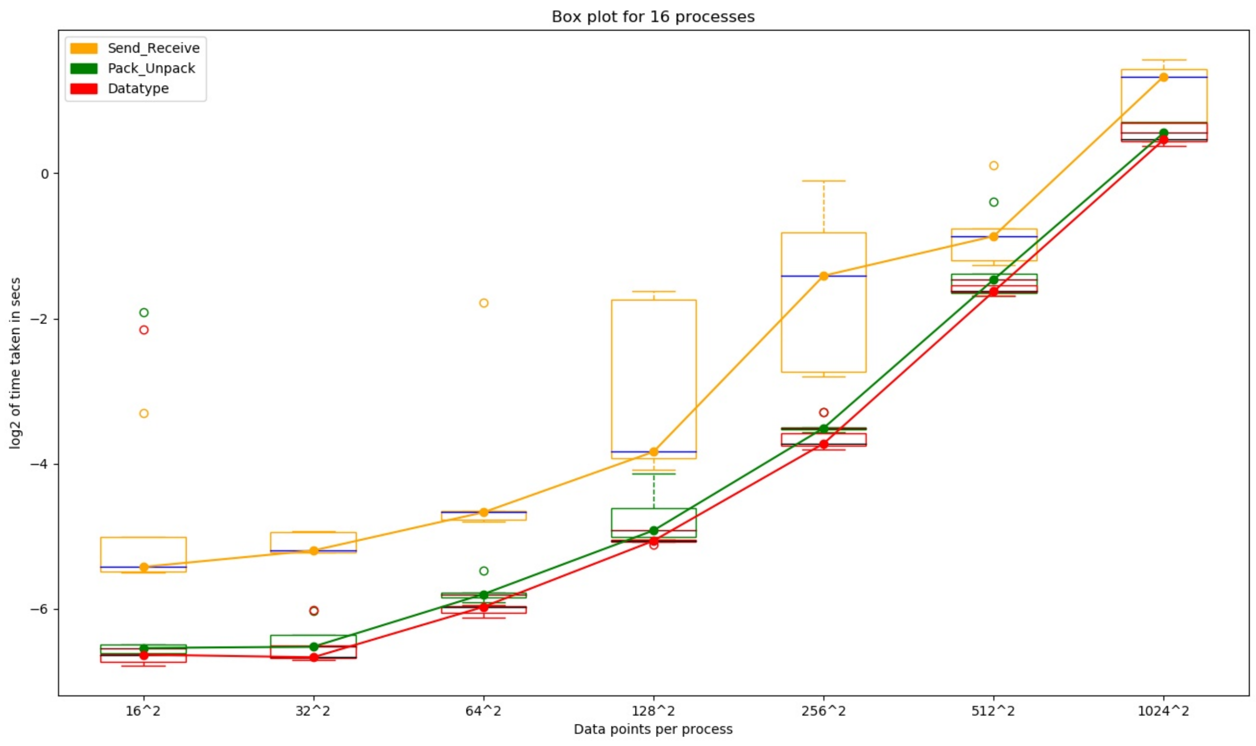
- We were not assigning position parameter to begin with. The buffer space can be reused by assigning 0 to the variable position before each MPI_Pack() and MPI_Unpack().

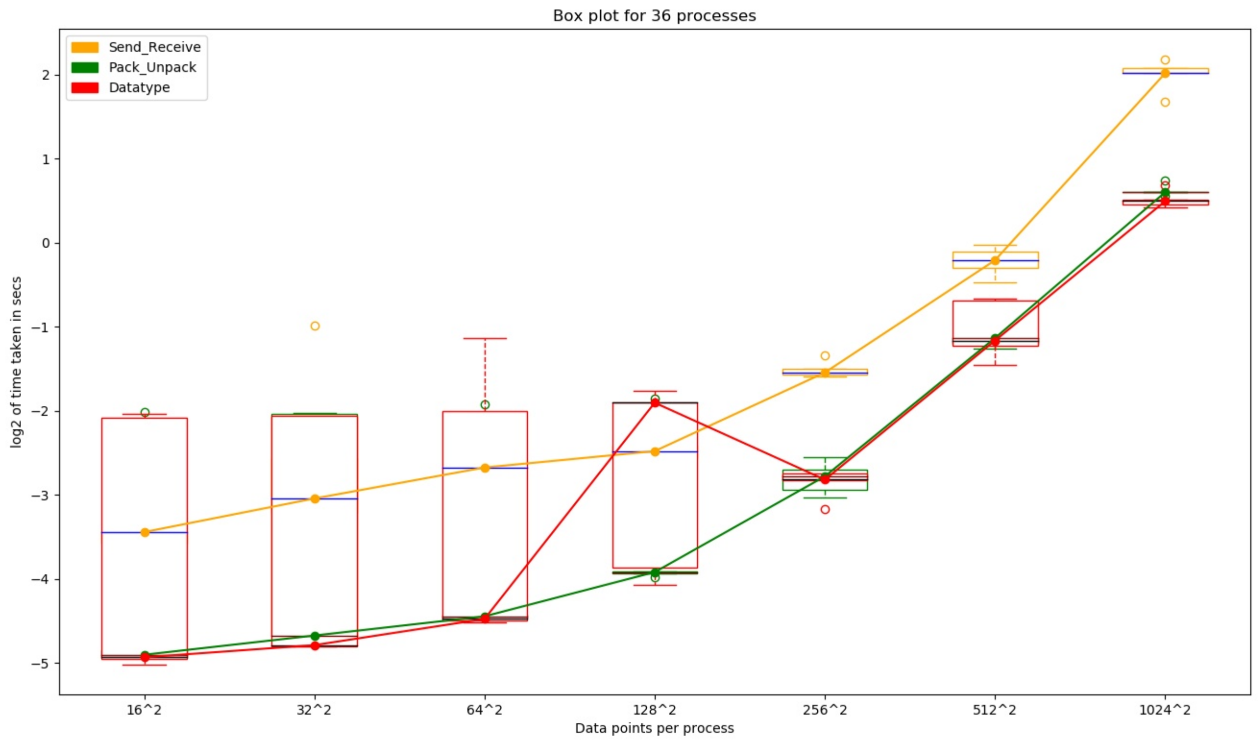
- The major problem we faced was in derived datatype MPI_Type_vector. Initially the dynamic allocation of 2D matrix for data points didn't work for derived datatype vector, as when we were sending columns then the different rows were not contiguous in normal dynamic allocation of 2D matrix, and so when we were performing left to right and right to left communication, the correct values of columns were not being communicated. We realised that in the derived datatype "columntype" in our code, it considered each element to be inserted in the vector after every N doubles, and as the matrix was not contiguously allocated, it gave us some garbage value. To correct this we found out a way to dynamically allocate a 2D matrix contiguously.

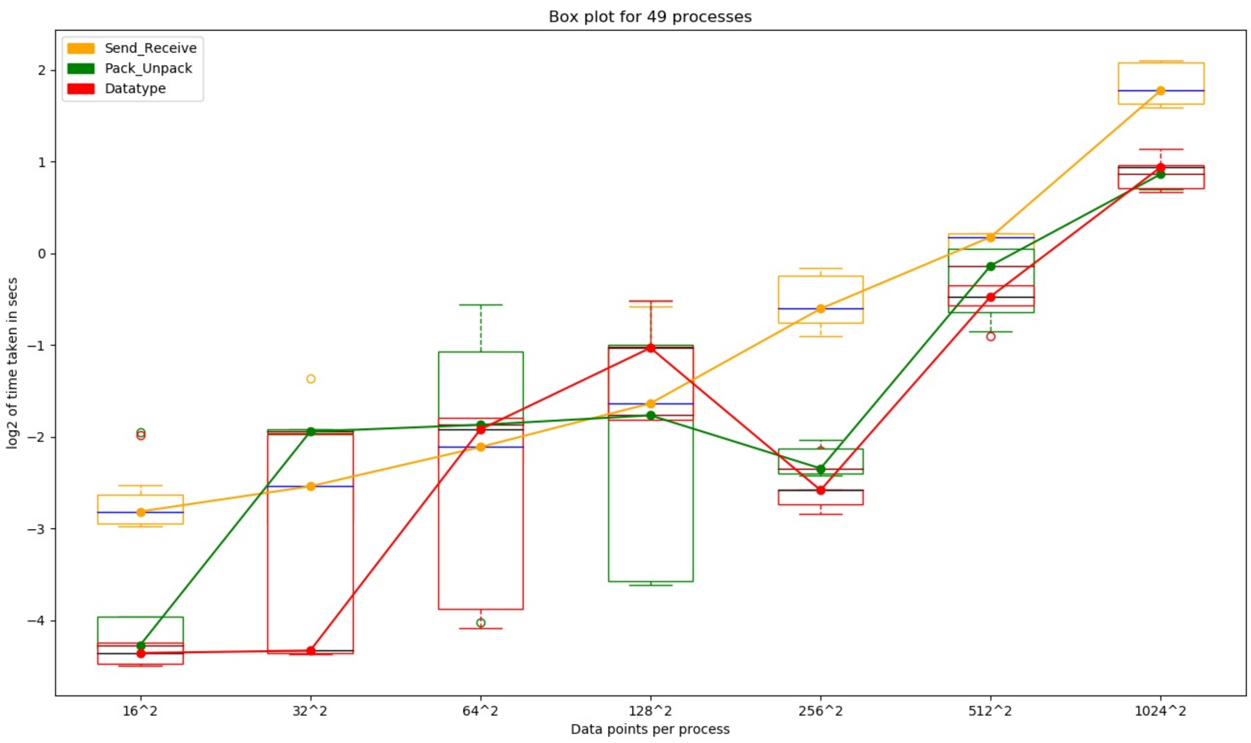
- We were using `memcpy()` to copy data values from "new_arr" to "arr" in each iteration. But because we were allocating those arrays in different ways, garbage values were getting copied. So, we dropped its usage and copied data values using 2 nested for loops.

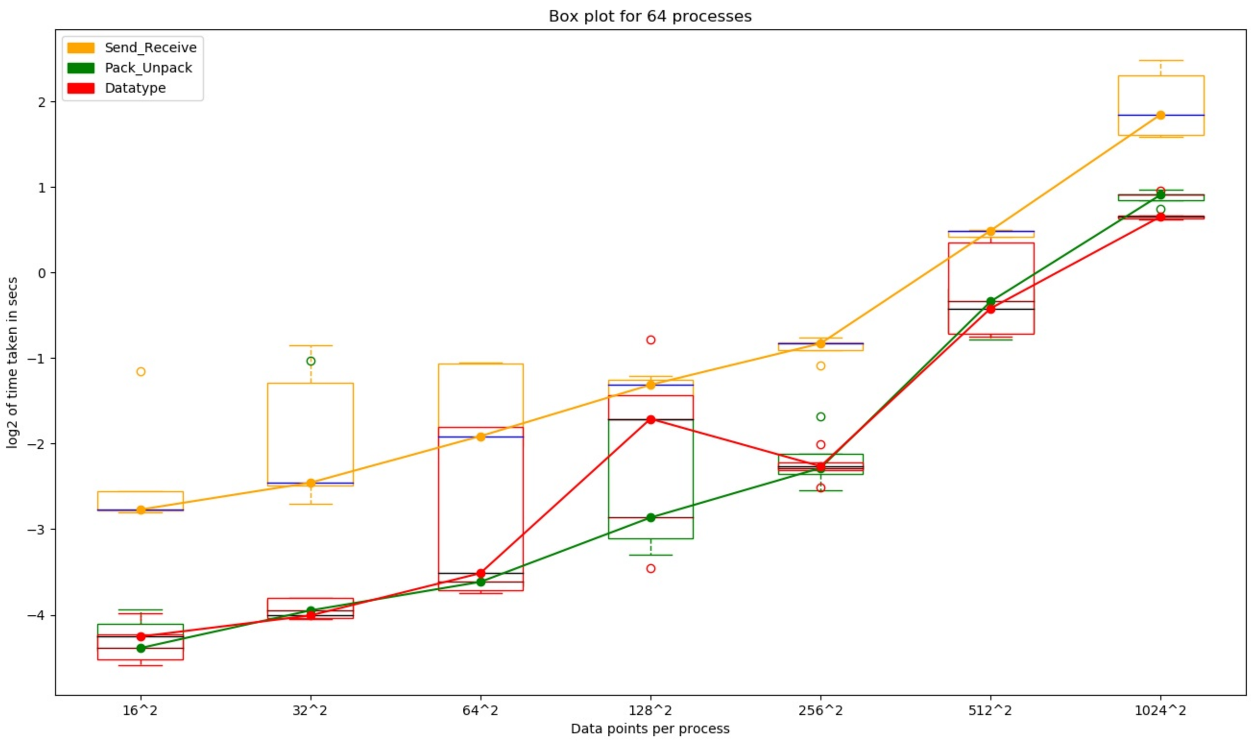
- One problem that we encountered during the implementation was that matplotlib wasn't generating the plot when we were running the script from terminal using ssh. Changing the backend to `Agg` backend which writes to files solved the problem.

4. Plots


Box plot for 16 processes
Send_Receive
Pack_Unpack
Datatype
log2 of time taken in secs
Data points per process
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
0
−2
−4
−6


Box plot for 36 processes
Send_Receive
Pack_Unpack
Datatype
log2 of time taken in secs
Data points per process
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
2
1
0
−1
−2
−3
−4
−5


Box plot for 49 processes
Send_Receive
Pack_Unpack
Datatype
log2 of time taken in secs
Data points per process
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
2
1
0
−1
−2
−3
−4


Box plot for 64 processes
Send_Receive
Pack_Unpack
Datatype
log2 of time taken in secs
Data points per process
16^2
32^2
64^2
128^2
256^2
512^2
1024^2
2
1
0
−1
−2
−3
−4

5. Observations

- We observed that mostly the time for multiple MPI_Send()/Recv() has the highest time for all the configurations as each element is sent separately. If the communication is done by using MPI_Pack()/ MPI_Unpack() or MPI derived datatypes then it requires less time as multiple elements are sent in one go.

- The number of messages sent while using the methods MPI_Pack()/Unpack() and MPI Derived datatypes are the same. This might be a possible reason for the plots showing that both those methods' time are close.

- Over multiple runs of the jobscript, we observed that for N=128 and for various values of P (mostly ≥ 36), MPI derived datatype performs worse than MPI_Pack()/Unpack() and sometimes even worse than MPI_Send()/Unsend().

- In most of the cases, the communication done using MPI derived datatypes required lesser time as compared to the method when communication was done using MPI_Pack()/Unpack(). Although both methods send multiple elements(and the same number of elements in this case) in one go, when we use MPI_Pack/Unpack() then it copies data into a buffer(additional 'memory to memory' copy of data) which costs additional time whereas there is no copying of the data when communication is done using MPI derived datatypes. Moreover, MPI_Pack() requires one function call for each packed item where as MPI Derived datatypes requires fixed number of function calls and the data layout is also regular(like arrays) in this case. By this we can conclude that time elapsed between sending data and receiving data is same for both. But the time required to Pack data before sending them and Unpack data after receiving it costs additional time.

- We noticed in various runs of the jobscript that for N= 1024^2, P =16 and P =64 when MPI_Pack()/Unpack() was used, the communication plus computation time were more or less the same. The same was noticed when MPI derived datatype was used. This might be an indicator of good scalability of the codes for MPI_Pack()/Unpack() and MPI derived datatypes.

- For several runs of the jobscripts almost for every plot, there were multiple occurences of outlier data points. A possible reason could be lack of high number of executions which would have taken into account the variability of the statistical data better.

- We also noticed that when there was high load in the cluster it took 50 mins to run the jobscript whereas when there was less load it took around 6 to 7 minutes to run the jobscript.

6. Experimental Setup

(1) **Headers included in C program:**

- stdio.h, stdlib.h, math.h, string.h, mpi.h (MPICH version 3.3.2)

(2) **Python packages/modules used:**

- matplotlib (version 3.3.4)
- math
- os
- sys (version 3.8.5)