Assignment 1

Aniket Sanghi (170110)
Sarthak Singhal (170635)

23th February 2021

We pledge on our honor that we have not given or received any unauthorized assistance.

1 How to Run

```
>>> ./run.sh
```

2 How to make plots

```
>>> python3 plot.py data
```

3 Explanation of code

We have commented the code so it should be readable. Here we mention broadly what it does

- Initialise dynamically data matrix of size $(n+2) \times (n+2)$, where the main data is stored in the sub-matrix $(1,1)$ *to* (n,n) (randomly initialise). The outer rows and columns will be used to store the data that will be received from the neighboring processes
- Then for every time stamp the process does the following
  - After checking if the corresponding neighboring process exist in the process matrix send the row/column to them and receive their row/column
  - Asynchronous Send/Recv is been used for all the methods
  - The row is sent/received using only one ISEND/IRECV call in all the three methods while for the columns, data is sent/received with
    - Multiple sends/recv for method 1
    - Sent/Recv in one call after clubbing the data using MPI_PACK in method 2
    - Made a vector data type that will send each column in one go, used it to send/recv data in one call in method 3
  - The computation of the inner grid(non-halo region) is done after issuing the Sends and Recvs and before WaitAll is called. This is done in order to overlap the computation and communication.
- Each of the method is timed and the max time taken by all the processes is found using MPI_Reduce

4 Observation from Plots

NOTE: Mean of the 5 observations was taken as the data point for the line graph.

- In all the 4 plots it was observed that the method1 was performing worse than both the method2 and method3. Similarly, method3 was performing (slightly) better than method2.

  - This outcome was expected because in the method1 we were sending the elements of the columns one by one which would lead to a large number of MPI Sends(also corresponding MPI Recvs) and hence degradation in the performance. While in the method2 and method3 there was just 1 MPI Send for the complete column leading to a better performance.
  - In the method2, we were explicitly iterating over the whole column to pack the data into a single buffer, while in the method3 the vector data type handled the task of sending the whole column in one go. This suggests that library handles the task of packing and sending much more efficiently.

- Another key observation is that the variability in values is particularly high for $P = 49$ and $P = 64$ as compared to other two which can be attributed
  - **Node Quantity**, Since the number of nodes have to communicate is more for these (since ppn = 8), the network overhead must have added to the variability
  - **Network Bandwidth and Core Usage**, Now since for every iteration to occur, all processes have to complete the previous time-stamp iter (need core to run) and more network bandwidth hence leading to more overhead.
- There were some outliers in the data also which can be observed in the plots as circles. These were mostly due to the high variability in nodes on some occasions due to heavy usage.

5 Problems and Solutions

5.1 NodeAllocator

- **Problem** It was mostly allocating hosts with [csews1-csews10] and the code seemed to perform poorly on them suggesting heavy load
  - **Solution** In order to avoid using the starting nodes which seemed heavily loaded we removed hosts csews1-csews15 from the /.eagle/hosts.txt file so that NodeAllocator allocated node only in range csews16-csews32. Here the performance of code increased by more than 10 times.
  - *Note:* NodeAllocator's monitor has to be restarted to see the effect of hosts.txt change

5.2 Testing

- **Problem** Testing the code was a problem as the printed output doesn't appear in an order
  - **Solution** So, in order to print the output in a specific order we used MPI_Barrier in a loop to allow only one process to print at a time
  - *Note:* The testing code is written in the script. In order to use it, just un-comment it. The commented code section has a comment **"// TESTING"** above it