Assignment3

- script.py generates the hostfile on-the-fly based on the node status. For 2 nodes, it can generate 2 different categories of hostfile, category 1 containing both nodes from different group and category 2, containing both nodes from same groups. Program to generate both categories are mentioned in script.py. (Category 1 is commented and Category 2 is default one. Can swap the comments to check for category 1).
- run.sh calls script.py and generates hostfile.

Code

- After MPI_Init(), first reading the file at rank 0 process to know the number of columns (assuming format remains same i.e. first two columns of latitude and longitude and rest columns as years) and number of rows (number of Stations).
- Now, this number of rows and columns are broadcasted to all processes, so that, they can declare their receive buffer.
- Then, creating array ”data” to store the values from the file.
- Now, reading the file again at rank 0 process to fetch values and storing it in ”data”.
- Before distributing, applied MPI_Barrier(). Otherwise, rest of processes will start the timer before rank 0 process will finish reading file
- Now, distributing data from rank 0 process to all processes using **Scatter** .
- Every process finds year-wise minimum temperature for each year.
- Calling MPI_Reduce() from rank 0 process, getting overall across all stations year-wise minimum temperature for each year.
- Now, calculating Global minimum temperature from year-wise minimum temperature we get in previous step on rank 0 process by iterating over minimum of all years we get.
- Writing year-wise minimum temperature in output.txt is performed during calculation of Global minimum temperature.
- At last, writing Max time across all processes and Global minimum temperature in output.txt file.

Data Distribution Strategy

- MPI_Scatter() used to distribute the data across all processes.
- As MPI_Scatter Algorithm uses Recursive Halving (Vector halving and Distance halving), it takes time to scatter n bytes equal to

  $$(\log p)*L + (p-1)*(n/p)*(1/B)$$

  where p is number of processes.
- As we want every process to find local minimums and for that sending particular amount of data to the processes on which they can perform their task will be better, instead of sending whole data to every process. Hence, we used Scatter for the distribution so that every process should get that particular amount of data.
- Also, Latency term is log p only and with every step, vector and distance gets halved.

Other Experiments

- We tired MPI_Bcast(), broadcasting whole data to each process and then making them to compute local minimums on their particular amount of data. But, this was increasing the time as every Bcast algorithm is having larger latency term than Scatter. Also, Van de Geijn et al., itself uses scatter.
- We also tried MPI_Pack(), but this also degrades the performance. As, packing and unpacking of data was consuming time. Also, sending packed data to each process requires linear time.
- Hence, it gets clear that scatter is performing better.

Observations

- Running on single node, it can be clearly observed that with increase in cores per node, time decreases.

Cores Per Node	**time (in sec)**	**Speedup**
1	0.125744	1
2	0.09763	1.28
4	0.059663	2.10

Table 1: Processes Running on Single Node

- Now, first we took 2 nodes from same group. In this case also, with increase in process per node, we are getting better performance.

Cores Per Node	**time (in sec)**	**Speedup**
1	0.371486	1
2	0.356624	1.04
4	0.352981	1.05

Table 2: Processes Running on 2 nodes of Same Group

- On selecting nodes from different groups (total 2 nodes), it is observed that with increase in cores per node, time remains almost constant. Because of inter-group communication, we are not getting desired performance.

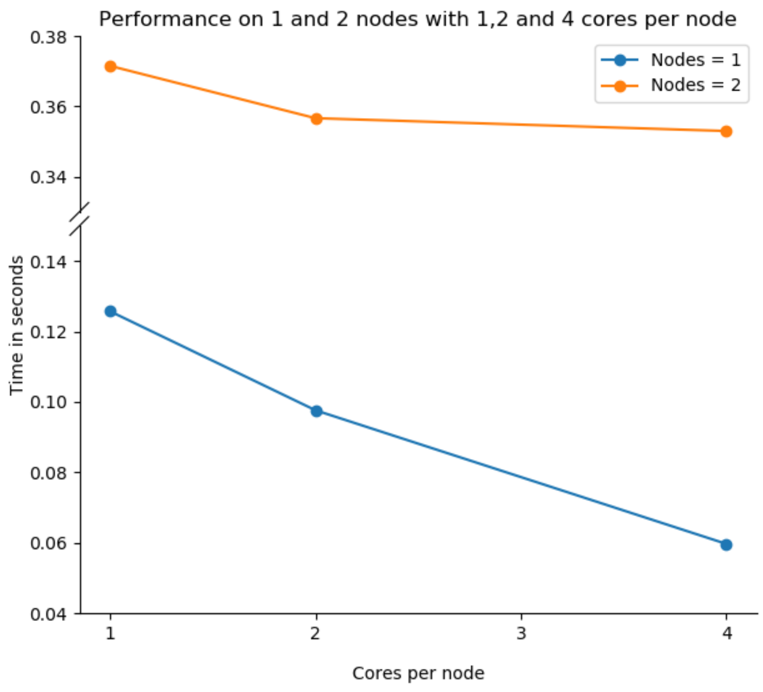
Cores Per Node	**time (in sec)**	**Speedup**
1	0.346730	1
2	0.344033	1.007
4	0.344388	1.006

Table 3: Processes Running on 2 nodes of Different Group

- It can also be observed that with increase in nodes, the performance degrades because of internode communication.

Plot

Below plot shows the performance of node 1 and node 2 (from same group) with 1,2 and 4 cores per node.


Performance on 1 and 2 nodes with 1,2 and 4 cores per node
Nodes = 1
Nodes = 2
Time in seconds
Cores per node