Assignment 3

Aniket Sanghi (170110)
Sarthak Singhal (170635)

20th April, 2021

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

1 How to Run

```
>>> bash run.sh
```

2 How to make plot

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

3 Data Distribution Strategies

- **Partitioning based on the rows**

  Strategy

  1. Data is partitioned w.r.t to the stations i.e. complete data of continuous stations is distributed to the processes.
  2. After distribution each process computes year-wise minimum temperature for its local data.
  3. Then per process year-wise minimum temperatures are collected at root and overall year-wise minimum temperature is computed.
  4. Finally root computes overall minimum temperature using year-wise minimum temperatures.

  Implementation

  1. Implemented in *strategy1* function in src.c.
  2. Input file is read by the root process.
  3. Number of years and stations are broadcasted to each of the process.
  4. $floor(num\_stations/num\_procs)$ rows are distributed to each process. If the number of stations is not a multiple of number of processes then the root process handles the computation of the last remaining stations which are not scattered. For doing this distribution MPI_Scatter is used instead of its vector version. This works because number of stations is way to large than the number of processes due to which a slight increase in the data for root process doesn't affect the performance.
  5. Then each process computes year-wise minimum temperatures for the data received by the process.
  6. After computation MPI_Reduce is used to gather overall year-wise minimum temperature at the root process.
  7. Finally root computes overall minimum temperature from the year-wise minimum temperatures data.

Why?

1. Since the number of columns i.e. the numbers of years are limited to a few whereas the stations are quite high in number, it seemed a better idea to distribute data along the dimension that can increase/scale at a higher pace

Observations

1. The code scaled well when the number of processes were increased but all were confined to a single node.
2. The code didn't scale (didn't get a speedup) when the number of processes spanned over 2 nodes instead of 1. The primary reason for this that we suspect is that the inter-node communication overhead surpasses the computation overhead in this case due to small set of data.

P	PPN	Speedup
1	1	1
1	2	1.56
1	4	2.07
2	1	0.39
2	2	0.40
2	4	0.42

Table 1: Speedup and Scalability

- **Partitioning based on the columns**

  Strategy

  1. Data is partitioned w.r.t to the years i.e. every process gets data corresponding to few years to process
  2. After distribution each process computes year-wise minimum temperature for its local data.
  3. Then per process year-wise minimum temperatures are collected at root using Gatherv and overall year-wise minimum temperature is computed.
  4. Finally root computes overall minimum temperature using year-wise minimum temperatures.

  Implementation

  1. Implemented in *strategy2* function in src.c.
  2. Input file is read by the root process.
  3. Number of years and stations are broadcasted to each of the process.
  4. Columns are distributed to each process using asynchronous send and recv calls of a user-defined vector datatype just the last process may receive a different number of columns to fit in.
  5. Then each process computes year-wise minimum temperatures for the data received by the process.
  6. After computation MPI_Gatherv is used to gather overall year-wise minimum temperature at the root process.
  7. Finally root computes overall minimum temperature from the year-wise minimum temperatures data.

  Why?

  1. In the row-distribution strategy after scatter, reduce was called. We wanted to try and reduce the overhead in the case of reduce (though we suspected it to be small). So we decided to distribute columns, but since we couldn't find any way to use vector data type with MPI_Scatter, we used Isend/Irecv calls to do the same. Instead of Reduce, in this case we used Gatherv which should be faster due to less data that it has to communicate in P2P communication in it.

  Observations

  1. The code scaled well when the number of processes were increased but all were confined to a single node.
  2. The code didn't scale (didn't get a speedup) when the number of processes spanned over 2 nodes instead of 1. The primary reason for this that we suspect is that the inter-node communication overhead surpasses the computation overhead in this case due to small set of data.
  3. The speedup/scaling was just a little poorer than the row-distribution strategy which we suspect is due to the use of Send/Recv calls instead of Scatter.

P	PPN	Speedup
1	1	1
1	2	1.44
1	4	1.87
2	1	0.37
2	2	0.39
2	4	0.40

Table 2: Speedup and Scalability

4 Plots

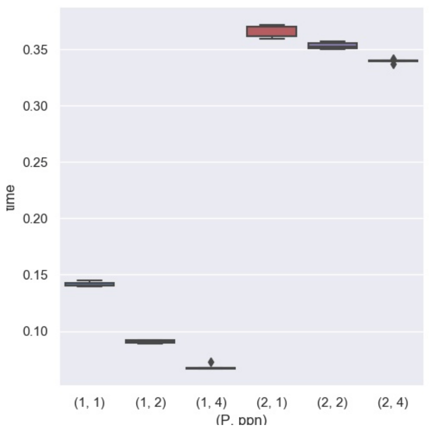

Figure 1: Partitioning based on the rows

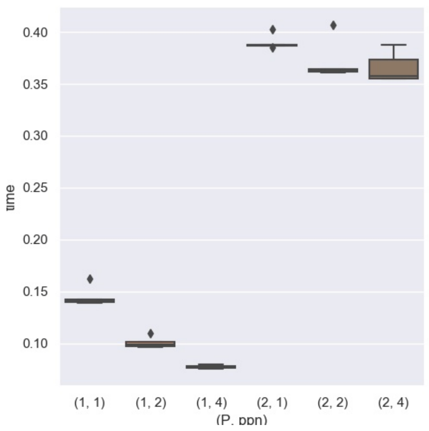

Figure 2: Partitioning based on the columns