Results, observations and required experimental setup for assignment 3

July 31, 2021

1 Distribution strategy

1.1 How

We have divided the data row wise. We tried to give roughly equal portion of data to each process by first dividing the total rows available by total number of processes. If the division results in remainder then all those rows are given to last process. As in our configuration maximum processes can be 8 therefore last process can have at-most 7 extra rows than other processes.

1.2 Why

As our given data file has more number of rows than columns we divided the data row wise and gave each process roughly same chunk of rows for computation. We found this type of division is more intuitive and this can be easily done using `MPI_Scatterv` as the data was stored contiguously after reading the data from csv file.

2 Code explanation

1. First we calculate the number of rows and number of columns by reading the csv file.
2. We create 2D matrix to hold the data from csv file. Each value except lat and long values i.e. the data corresponding to each year is stored in matrix.
3. After reading the whole file, timer is started to calculate the maximum time required for execution of data distribution and obtaining the result.
4. `MPI_Bcast` is used by root process(rank = 0) to broadcast the number of rows and columns to each process.

5. Whole data is divided into chunks of rows. If the data cannot be evenly distributed in processes, last process will get the remaining portion.
6. Root process uses `MPI_Scatterv` (as data may not be evenly distributed) to send data to all processes.
7. A 1D array is created in order to store minimum value across each portion and for each year.
8. `MPI_Reduce` is used (by rank = 0) on all such 1D array that hold the year wise minimum temperatures for each process to calculate overall year wise minimum temperatures.
9. Comparison is carried out to calculate the global minimum.
10. Maximum time is stored in 'maxtime'.

3 Generated Plots

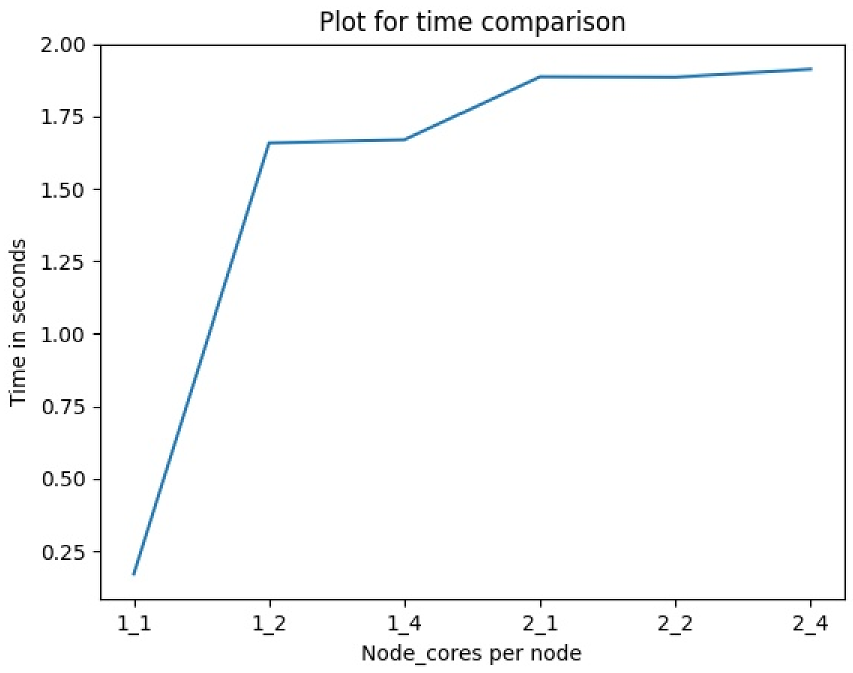

Figure 1: performance including data distribution time

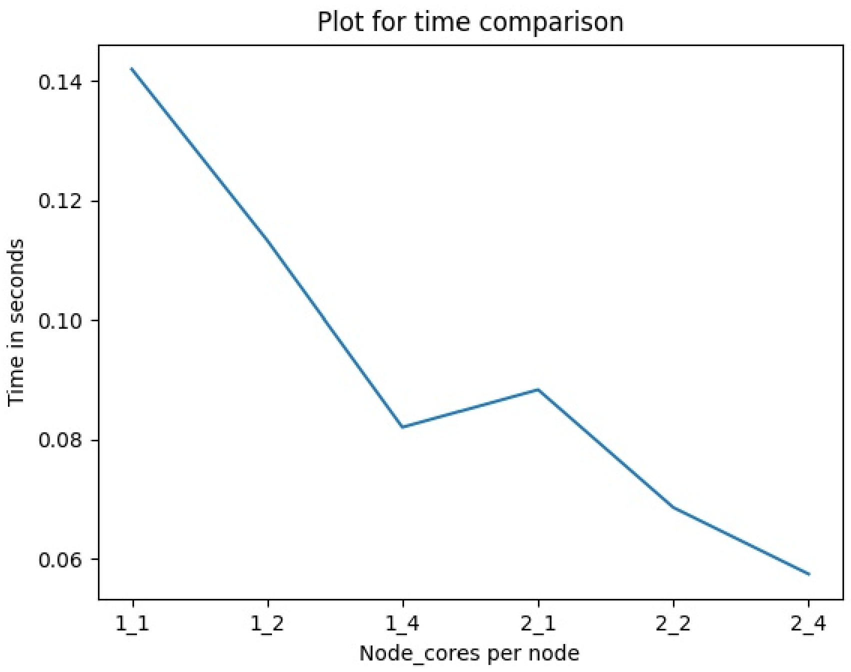

Figure 2: performance excluding data distribution time

4 Observation

4.1 Figure 1

1. In this figure we have measured the time starting right after reading the csv file and ending just before `MPI_Finalize()`, as stated in the assignment pdf.
2. The execution time is least for nodes=1 and process per node=1 as there won't be any data division.
3. As shown in figure 1, if we compare the execution time across same node (say node=1) as the number of cores per node increase, execution time increases which justifies the fact that the communication across processes/cores increase the time of execution.
4. Also when number of nodes are increased it adds the overhead of inter process communication and hence the execution time increases. Hence in our code performance decreases by increasing the number of processes and cores per node.

4.2 Figure 2

1. In this figure we have not included the data distribution time, i.e. when all the processes have their share of data.
2. We see that time decreases as we increase the number of processes, which is expected according to Amdahl's law.
3. We can also conclude that our data distribution strategy is consuming a lot of time as compared to computation time.
4. Communication to computation ratio (C-to-C ratio), a lower C-to-C ratio is desirable and usually indicate better parallel performance. Which is not in our case.

5 Running the code

1. To run the job script use the following command: `sh run.sh`
2. This script makes script.sh as executable which in turn helps to generate hostfile by pinging to different csews nodes. It compiles `src.c` code and generated the `output.txt` file.
3. The format of `output.txt` is: First line contains the yearly minimum values in a csv format, the second line contains the global minimum and the third line contains the maxtime required across all processes.

6 Experimental setup

1. We have used `script.sh` to generate host file on-the-fly by pinging the different csews nodes.

Note : *Output will only be present in output.txt, nothing is printed in standard output.*