REPORT

Ankita Dey, 20111013 P J Leo Evenss, 20111038

This assignment is about finding the minimum temperature among all the latitudes and longitudes given in tdata.csv by distributing the data among processes and running them in parallel, comparing the speedup over 6 combinations.

Code Explanation:

1. The tdata.csv file is initially read and the number of rows and columns are computed.

2. The tdata.csv is again read by root process(rank 0), into an 1D array of size rows*columns.

3. Now the root process distributes the array to other processes using the following methodology.

    a) The number of rows in the tdata.csv file is split equally among the processes from 1 to P-1 (number of processes) using floor(rows/P).

    b) The remaining last set of rows are held by root process for computation.

    c) The derived datatype MPI_Type_contiguous is used to define a new datatype of the size rows/P.

    d) The point-to-point MPI_Send call is used to send the data to the process 1 to P-1.

    e) The ranks other than root receives the data using MPI_Recv call.

    f) This completes the distribution of data.

4. Now the yearly minimum data is locally calculated among all process in parallel with their set of data.

5. This local minimum is now used to obtain global yearly minimum using the MPI_Reduce.

6. Now from the yearly minimum the global minimum over all the years across all the latitudes and longitudes are calculated at by root process.

7. The entire code from data distribution till the computation of the global minimum is timed using MPI_Wtime, and maximum time among all the process is obtained using MPI_Reduce.

8. The yearly minimum (excluding latitude and longitude), global minimum, maximum time is written into 3 lines in an output.txt file (overwritten each time), for all the 6 combinations of PPN given.

9. The hostfile is generated using the script.py file provided to us during the Assignment2 in helper scripts.

10. The run.sh and Makefile is written in accordance with the configuration requirement of the assignment.

11. The timings of the different configuration is plotted into a linear plot using plot.py.

12. (P=nodes ; PPN=cores/process per node)

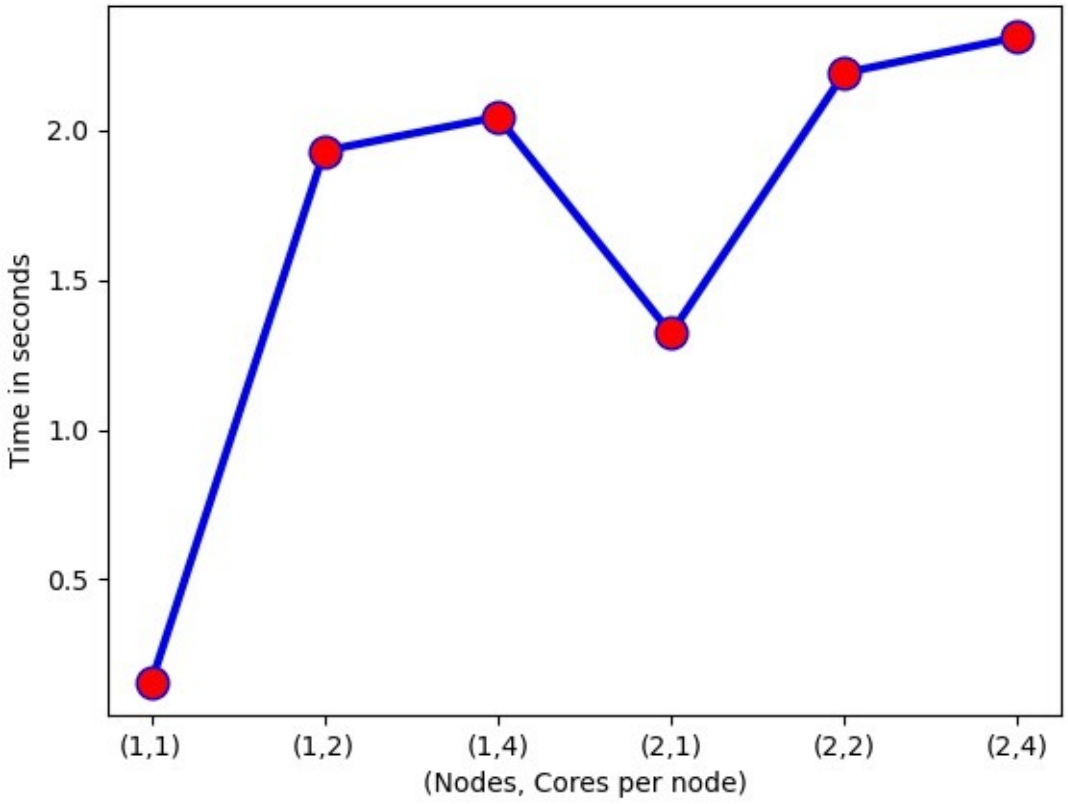
Distribution strategy - Reason:

1. The entire data had huge number of rows than the columns, so if we split column wise, the usage of mpi_vector datatype or mpi_pack leads to redundant traversal of entire set of rows for (columns/ number of process) times for creating new datatype and packing respectively.

2. If we split row wise, there will be a single traversal for all processes. And so we used row wise split. Due to this, we don't have to use other mpi collectives like gather to communicate the (columns/process) data back to the root which had to be done in former column wise split case.

3. We used mpi derived datatype – contiguous and used P-1 amount of sends for achieving the desired output.

Plot:



1. The plot for the maximum time among the processes is show above for P 1,2 and PPN 1,2,4.

Observations:

1. The line graph above shows the speedup and scaling among the number of cores and the number of processes per core.

2. We can see the time increases as we increase the number of processes rather than decreasing. This is due to the time involved in distribution of data, a pure overhead.

3. The speedup in turn reduces to further negative as we scale up among processes of different nodes, due to the communication time involving inter node hops.

4. In short the computation time is very much less than the communication time and hence the data distribution overhead dominates the exploited parallelism among the nodes.

5. If we only consider the computation time among 1 process vs parallel processes, then we there will be a speedup gain.

6. We ran 5 iterations and found the timing values to be almost similar.

Experimental Setup:

1. The hostfile is being generated using the helper scripts provided script.py, based on the P and PPN.

2. The filename (tdata.csv) is passed as argument inside run.sh.

Code Documentation:

1. Download the git Assignment3 folder in appropriate location.

2. Include the tdata.csv in Assignment3 folder.

3. Run chmod u+x run.sh

4. Now run ./run.sh (it starts running whole configuration, time to complete depends on the hostfile generation from the helper scripts provided, script.py; usually 7-10 mins or maximum of 15mins).

5. The code starts compiling using make command and output of src.x is obtained.

6. The hostfiles is generated on the fly using script.py from the helper scripts in the resources section.

7. The src.x is executed using mpiexec command with for loop execution sequence as specified in assignment.

8. The output.txt file is generated which captures the yearly minimum (excluding the lat and long), global minimum, and maximum time.

9. The plot script is executed using python3 plot.py which uses time.txt to plot the plot.jpg.

NOTE: The entire code takes approximately 10 to 15 minutes due to the helper script for hostfile generation as per the configuration, so submitting only 1 iteration , as 5 executions takes more time and the results were similar. Include the tdata.csv inside Assignment3 folder.