CMPE 300 MPI PROJECT 18/01/2021

Bilal Tekin
2017400264

-MPI Programming Project

Introduction

Nowadays, we have more than one processor. Therefore, we write our programs using parallel programming. Instead of taking the whole data and executing relief algorithm to reach the result. In this problem, we divide this data into smaller data parts. Each processor takes these divided and smaller data parts and all of them executes the algorithm in parallel. With these, the relief algorithm completes in a smaller amount of time.

Program Execution

I used “Open MPI” library.
mpirun –version => mpi (Open MPI) 4.0.3

The compiler that I used is “mpic++”
mpic++ --version => g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0

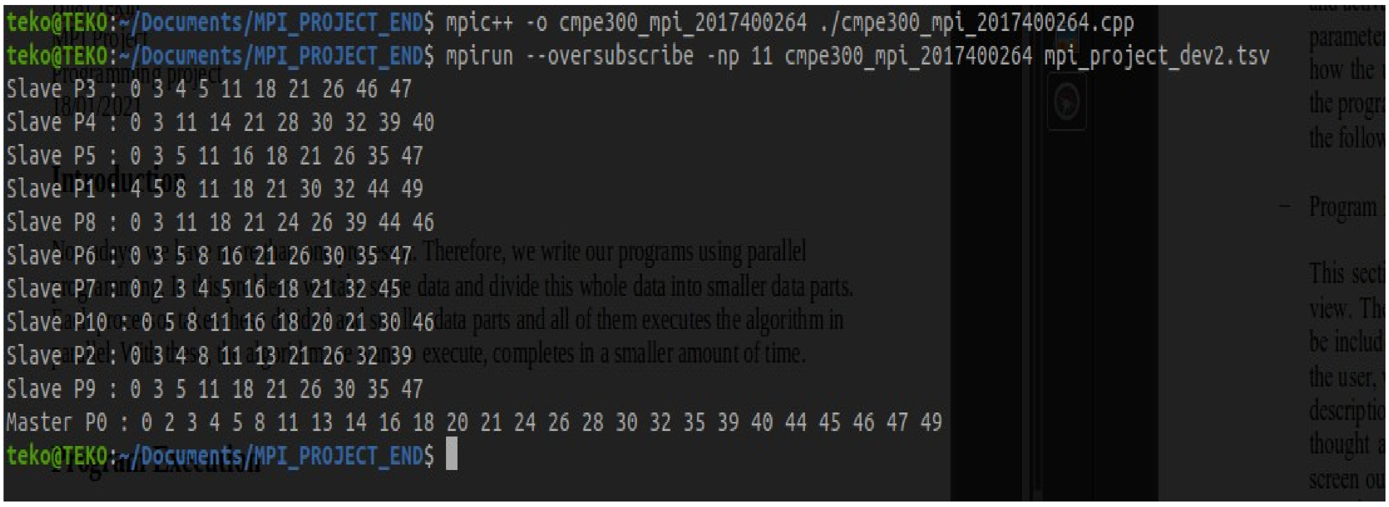
First of all, we should compile code with :

“mpic++ -o cmpe300_mpi_2017400264 ./cmpe300_mpi_2017400264.cpp”

which creates an executable file. After compiling the code, we should run this file by giving the necessary parameters. The executable can be executed by the command which is :

“mpirun --oversubscribe -np <P> cmpe300_mpi_2017400264 <input file path>”

In this command we use some parameters which determine the program’s life style. One of the very important parameter is <P> which determines the number of processors that we want to use in our parallel programming problem. The other parameter is “-np” which is used to work with more than one processor, which is necessary for the parallel programming. “cmpe300_mpi_2017400264” is the name of the executable file which is created by compiling the code.
“<input file path>” is the path of the input file which contains the data we want to divide into smaller parts and use in our problem. This input file can be like “.txt, .tsv etc.” formats. The number of processors, the number of data lines, amount of features, iteration count, and the desired top features number are available in this file. After executing the program, we can have the output which shows the top features that are selected by slave processors and all selected features taken by master processor in increasing order. An example of output can be seen below.



```
teko@TEKO:~/Documents/MPI_PROJECT_END$ mpic++ -o cmpe300_mpi_2017400264 ./cmpe300_mpi_2017400264.cpp
teko@TEKO:~/Documents/MPI_PROJECT_END$ mpirun --oversubscribe -np 11 cmpe300_mpi_2017400264 mpi_project_dev2.tsv
Slave P3 : 0 3 4 5 11 18 21 26 46 47
Slave P4 : 0 3 11 14 21 28 30 32 39 40
Slave P5 : 0 3 5 11 16 18 21 26 35 47
Slave P1 : 4 5 8 11 18 21 30 32 44 49
Slave P8 : 0 3 11 18 21 24 26 39 44 46
Slave P6 : 0 3 5 8 16 21 26 30 35 47
Slave P7 : 0 2 3 4 5 16 18 21 32 45
Slave P10 : 0 5 8 11 16 18 20 21 30 46
Slave P2 : 0 3 4 8 11 13 21 26 32 39
Slave P9 : 0 3 5 11 18 21 26 30 35 47
Master P0 : 0 2 3 4 5 8 11 13 14 16 18 20 21 24 26 28 30 32 35 39 40 44 45 46 47 49
teko@TEKO:~/Documents/MPI_PROJECT_END$
```

Program structure

First of all, in main part of the code, some important functions of the parallel programming was executed. They are MPI_init which initializes the MPI environment, MPI_Comm_size which gets the rank of the calling MPI process in the communicator given, MPI_Comm_size which gets the number of MPI processes in the communicator given. With these functions, we say that we initialize the processors and get the rank and number of the processors. Later, the reading process of the input file takes place. Three important arrays which are the types of data structures in c++ are created before reading the input file. One of them is “arr” which is for saving the data coming from input file, the another one is pref which is for slave processors to get their own data, the last one is weightIds that holds the weight of features selected. When we proceed, we start reading the input file and saving the whole data into “arr” and execute the MPI_Scatter method to divide the whole data in arr and distribute it to slave processors. The slave processors takes their own data and executes the algorithm which is written in the code. The algorithm, firstly, finds all maximum and minimum values of the columns of data via “vector<pair<float,float>> getMaxsAndMinsInMatrix(vector<vector<float>> matrix)” method. In this method, All columns are visited and the minimum as well as the maximum values are put into a vector<pair<float,float>> called “maxsAndMins” for each column of the matrix that comes as a parameter to this method. After finding of these values, for M iteration, the algorithm finds the index of hits and misses by “pair<int,int> findIndexOfHitAndMiss(vector<vector<float>> matrix, int index)” method. In this method, the target is selected in sequential order and the matrix is iterated row by row. According to hit and miss definition given in the description of the project, every row is compared with the target row to find hit and miss rows. The found hit and miss indexes were returned as a pair of integers. After finding these properties, with the relief algorithm, the weight is calculated. When the slave completes finding the weights, It calls “void printSingleMatrix(int matrix[], int size, int rank)” to writes the selected weight ids to the screen. After that, It also sends the weight array to the master processor like all other slave processors. Master processor takes all weights that comes from the slave processors, sort the them according to their feature ids and print the sorted feature ids to the screen via “void printResult(int matrix[], int size)” method that takes an array and writes it to the screen.

Difficulties Encountered

I didn’t know the parallel programming libraries. Therefore I encountered some difficulties that took too much time to become familiarize. When I send the whole data to the slave processors, I wanted to send a vector<vector<float,float>> structure but even if I have searched for some hours and tried many ways I have not succeed in this. The only way that I found was to send the one dimensional array that includes the whole data. After this, The receiving the data from slave processors was also hard for me to implement. I looked the PS codes and found some examples that do what I want. With these examples’ code, I find a way to achieve the goal. Another problem was understanding the algorithm. I did not encountered such an algorithm which is, I think, very complicated. Writing the code was very easy, understanding the problem and algorithm was the thing that takes too much time.

Conclusion

In this project, We learned that when we use more than one processor for the problems that supports parallel programming, the problem can be solved in a very short amount of time. Instead of giving the whole data to one processor, dividing the burden saves lots of time.