Student Name: Gökhan UYSAL
Student Number: 2017400024

CMPE 300 MPI PROJECT

Introduction

The goal of this project is to experience and learn how to use the MPI library. With the usage of MPI, the concept is to implement relief algorithm. Briefly the algorithm finds the important features and eliminate the unnecessary features from a given data set. At these days the size of data is become higher, for this reason there is a need for parallel processing and in this project the goal is implementing this algorithm with parallel processing with the help of MPI library.

Program Execution

Compile and run commands are the same specified in the description.

```
> mpicc -o cmpe300_mpi_2017400024 ./cmpe300_mpi_2017400024.c

> mpirun --oversubscribe -np <P> ./cmpe300_mpi_2017400024 <inputFile>
```

My Open MPI version is 4.0.3

My ubuntu version is 20.04.1 LTS.

Program Structure

Firstly, my program uses mpi, stdio, stdlib, limits, math libraries.

My program consists of 3 function 2 global variable and a main function. So, I explain these topics.

The first function is mannhattanDistance(double A[], double B[], int size). This function returns a double value. The aim of this function is calculating the mannhattan distance for given to double type array and returning the distance. The size parameter is used for loop count. In the function there is a local variable which named as result. Result is the calculated distance. There is a loop in the function to calculate distance. After calculating the distance function returns the result variable.

The second function is diff(double target[], double source[], double max[], double min[], int index). This function returns a double value. The goal of this function is calculating the new weight value for weight array with given index and returning this value. The target parameter is the target instance, and the source parameter is the nearest instance. Max and min parameters are same as the max min in the algorithm and index parameter is which index to calculate. In the function there is only one statement and this statement is calculating the difference with given algorithm and returning it with type double.

The third function is compare(const void *a, const void *b). The goal of this function is to help qsort() function which is used in main function. This function takes two value and compare them after comparing it return 1, -1 or 0 according to compare result.

There are two global variable which are named M and T. These variables are same as specified in description in the project. M is the iteration count and T is the count for chosen features.

Now, I will explain the main function from top to bottom. The first statement of the main function is setvbuff function. This function disables the buffered writing for stdout. The usage of this function is to prevent some synchronization problem during the printing results of slaves. Afterwards, there are some required MPI functions. Firstly, my program declares the variables for total number of processors named by P, the number of instances named by N, and the number of features in a instance named by A and reading them with the given input file.

Nextly, the array including all input data which will be distributed late by master to salve processors named as arr[] with the type of double and the second array including data which is received by each slave named as pref[] with the type of double are declared. Each slave receives its part of the data in pref[] array. The arr[] array size is defined which size will be distributed.

My program starts the first part of master process here. In this part program understand that it is a master process with the check of rank. If the rank equals 0 it is master process. Afterwards, master process reads M and T value from given input file and put them to global variables and then it starts reading the data and put them to arr[] array. Before reading the data it fills the some of the arr[] array because while distributing it master process will distribute some part of the array to itself so it fills with 0 for this part of array. After reading and putting data to array it closes the file and sends M and T values to salve processes with MPI_Send() function. After this, by using MPI_Scatter() function it distributes the arr[] array to slave processes. By using MPI_Scatter() function arr[] scatter parts of the data belonging each slave to its pref[] array. From now, each slave has its data to process in its pref[] array.

There is a masterSignal and while loop to finish the program successfully and the section for slave processes starts here in the while loop.

Firstly slave process initialize their own M and T value and receive these values with MPI_Recieve() function from master process. Then, slave process initialize max[], min[], nearestHit[], nearestMiss[], targetInstance[] and W[] arrays with the type of double and result[] array with the type of int.

max[] and min[] arrays are used for storing max and min features values. nearestHit[] and nearestMiss[] are used for storing nearestHit instance and nearestMiss instance in the algorithm. targetInstance[] is sued for storing chosen instance. W[] is used for storing weight values and the result[] is used for storing results.

Firstly, W[] array is filled with 0. Then there are two more declarations which are int targetID and double targetClass. The targetID is the chosen instance id in each iteration and targetClass is the chosen instance target class in each iteration.

Afterwards, there is a for loop to finding and filling max[] and min[] array. In this loop there are two more local variable declare. These are double maxTemp and double minTemp. These two variable is used for finding max and min of a feature and then storing it in the min[] and max[] arrays.

After finding the max[] and min[] arrays the M iteration parts of the algorithm starts in here. There is a big for loop for iterate M times the algorithm. In this loop firstly there is a another loop for filling the targetInstance[] array. In this loop we chose the target instance and fill the targetID and targetClass. Then, program initialize two local variable double minnearestHit and double minnearestMiss with INT_MAX value. These two variables used for storing temp min Manhattan distance. Then, there is another loop for storing nearestHit[] and nerestMiss[] arrays. In this loop there is a if else statement which controls the target class of the instances if target class is same for two instance it fills nearestHit[] array else it fills nearestMiss[] array. While filling these array slave process uses the manhattanDistance() function to calculate distance of instances. After finding the nearestHit and nearestMiss it calculates the relief difference and changing the W[] array with new values with the help of diff() function. These algorithm goes for M times and fills the W[] array.

Nextly, there is a loop for finding the top T weights. In this loop two local variables are initialized which iare double tempMax with INT_MIN and tempIndex with -1. tempMax variable stores temp max values of weights in loop and tempIndex is stores the temp index of this max weights. After the loop result[] array will filled with tempIndex value. After T times the result[] array stores the top T features. Then it sorts these values with c library function named qsort(). qsort() function uses the compare function which is explained before. After sorting, slaves must print the result to stdout in ascending processor order. For this reason, there is a control mechanism. The control mechanism is like queue. The first slave prints results to stdout and then sending message to second slave with MPI_Send() function and sends results message to master process. Then the second slave waits for receiving the message from first salve then it prints its own result and then send message to third slave and the result message to master. This process goes on like this. After last slave receives the message it prints its output and send message to master process and the slave process part of project is finishes in here.

Then again master process works. Firstly, it initializes temp[] and result[] arrays. The temp[] array is used for storing the received result messages with the help of MPI_Recieve() function and puts them in result[] array. After filling the result[] array master process sort it with qsort() function. After sorting the results master process use a duplicates elements in a sorted array algorithm to get rid of these duplicates in result[] array. After this process master process prints the result to stdout and broadcast mastersignal and end all the processes by using MPI_Bcast(), MPI_Barrier(), MPI_Finalize() functions and the program terminates successfully.

Difficulties Encountered

The main and only difficulty I encountered was the problems while trying to synchronize the output of the slave processes. But in the end I come up with good idea to solve this problems and the idea was implementing message queue in slave process for printing output.

Conclusion

I think it was an instructive project to meet MPI and parallel programing. The focus was on synchronization and parallel programing. Moreover, relief is up to date and very useful algorithm to use. For this reason, implementing something very useful and up to date was very satisfying.