Elif Nur Akalın
2018400312

CMPE 300 Project 1

Introduction

In this project, we are required to simplify a data using Relief algorithm and MPI. Since the data might be very large, we need to utilize multiple processes so that the job is done more effectively. To simplify, we remove the features of a data that is less related, which is done through Relief.

Execution

To compile the code, input format is like the following:
```
mpic++ cmpe300_mpi_2018400312.cpp -o cmpe300_mpi_2018400312
```
And to run the code, input format is like the following:
```
mpirun -oversubscribe -np [input_size] ./cmpe300_mpi_2018400312 [input_file]
```

In my case, I used the following commands:
```
mpic++ cmpe300_mpi_2018400312.cpp -o cmpe300_mpi_2018400312
mpirun -oversubscribe -np 3 ./cmpe300_mpi_2018400312 ./tests/test1.txt
```

test1.txt is the test case that is given in the project description.

Program Structure

For this algorithm, my code is parted into an if-else block. On top of the if-else block, rank, size, initial input values, (which are the process count, feature count, iteration count and desired number of features), an integer called slice size (which is the number of rows to be sent to each slave), and a 2D double vector called slice (which holds the data segment to be sent to the slave processes) are initialized.

After that, there is an if block, and all the code inside it is only going to be run by the master process.

Master process creates a 2D vector of double called V, and reads all of the data that is inside the given file, of which name is passed as an argument. The initial parameters are broadcasted to all of the slave processes through `MPI_Bcast` function. After that, the segments are sent to the slave processes one by one as a 1D vector, and they will be combined into a single 2D array in the slave processes.

After all the segments are sent to the corresponding slaves by the master process, the master process creates a vector of size slave count times T called result, and waits for the slave processes's answer. The slave processes send the desired number of features. Master process saves what it receives from the slaves into the result vector. This result vector is the concatenation of the result that the slaves have found from their Relief algorithms. So, it might contain duplicates. To prevent them, the result is sorted and the duplicates are erased using unique function. The final list of features is ready, and it is printed.

In the else part of the main if-else block, the slave processes's behavior is coded. Each slave receives the parameters using the MPI_Recv function. Then, they allocate space for their data segment which is a 2D array called slice, using resize function. After space allocation, the slave processes receive the corresponding data segment. Then two new vectors of pairs are created, which are called weights and distances. The second part of the pairs are their indices, they are held so that after sorting they remember their initial index. For each iteration, the distances between the row at the iteration index and all the other rows are found and the distances array is updated. Then the distances array is sorted, and the nearest hit and miss indices are found.

After the hit and miss indices are found, all of the weights array is updated. The diff between the current index and the hit index is subtracted from, and the diff between the current index and the miss index is added to the corresponding index, after they are divided by the iteration count.

After all the iterations are done, the weights vector is sorted by ascending. The elements besides the last T are deleted. The indices of the remaining features are found from the second element of the pairs of the weights array. They are printed, and then pushed to a vector called result, and sent back to the master process, and after the master prints the unified result, the program is finalized.

Example Output

Test Input: mpi_project_dev2.tsv

Output:

```
mpirun -oversubscribe -np 11 ./cmpe300_mpi_2018400312 ./tests/mpi_project_dev2.tsv
Slave P1 : 4 5 8 11 18 21 30 32 44 49
Slave P2 : 0 3 4 8 11 13 21 26 32 39
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 P6 : 0 3 5 8 16 21 26 30 35 47
Slave P7 : 0 2 3 4 5 16 18 21 32 45
Slave P8 : 0 3 11 18 21 24 26 39 44 46
Slave P9 : 0 3 5 11 18 21 26 30 35 47
Slave P10 : 0 5 8 11 16 18 20 21 30 46
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
```

Difficulties Encountered

The project was instructive, and it broadened my horizon about multiprocessor programming. However, it was difficult to debug. I did many mistakes in my code, but the output error was segmentation fault most of the times, it didn't give exact information on which line the error was found, therefore it was difficult to trace my mistakes.

Conclusion

All in all, the project was very enlightening, and my code works correctly with all af the given inputs. I believe it will run correctly in the upcoming test cases also.