27.01.2021

**Adalet Veyis Turgut**

# CMPE 300 – ANALYSIS OF ALGORITHMS

# MPI PROGRAMMING PROJECT

## Introduction

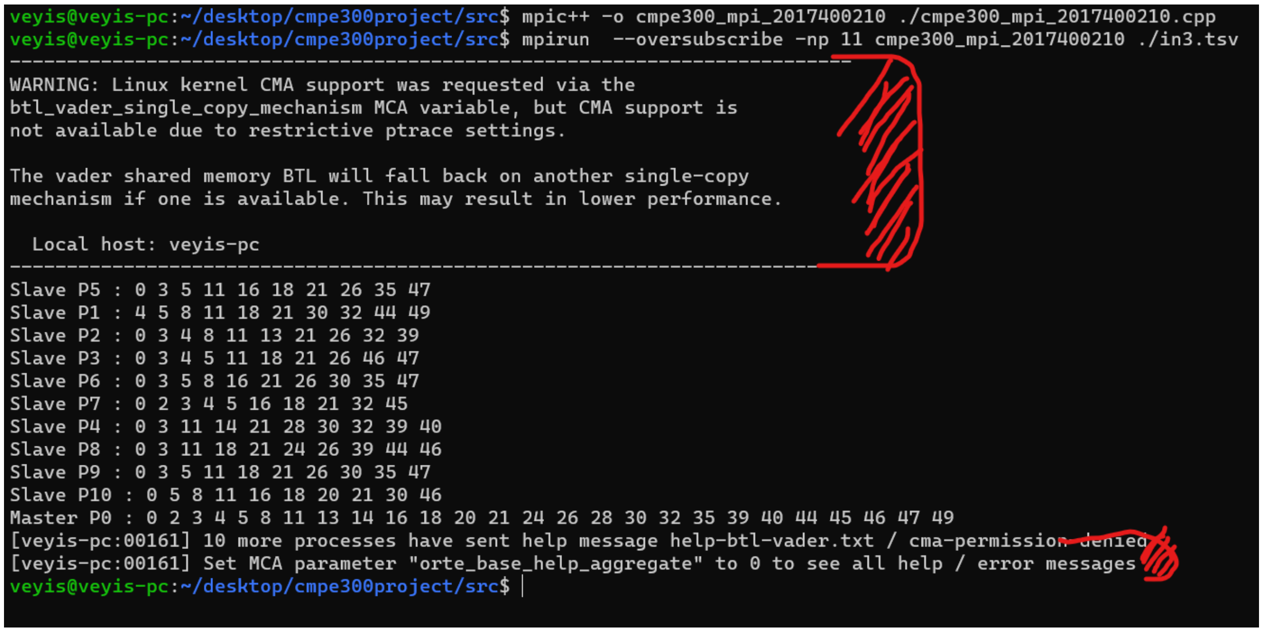
Data processing is a very popular and important topic in computer science. Since data is “big” and not ready to process directly, we should clean and select that data very carefully. In that sense I implemented feature selection algorithm Relief, widely used in machine learning, using C++ and MPI library, in this project. Project was straight-forward. Messaging between slave and master processors took only a few lines. So, parallel part of the algorithm did not take much time. Remaning part was the implementation of relief algortihm. Finding min-max features, using diff function, finding nearest hit and etc. was simple C++ codes.

## Program execution

*<u>To compile:</u>* mpic++ -o cmpe300_mpi_2017400210 ./cmpe300_mpi_2017400210.cpp

*<u>To run:</u>* mpirun  --oversubscribe -np <P> cmpe300_mpi_2017400210 <inputfile>

If you encounter the lines that covered in red in the screenshot below, please add the following command while running: --mca btl_vader_single_copy_mechanism none



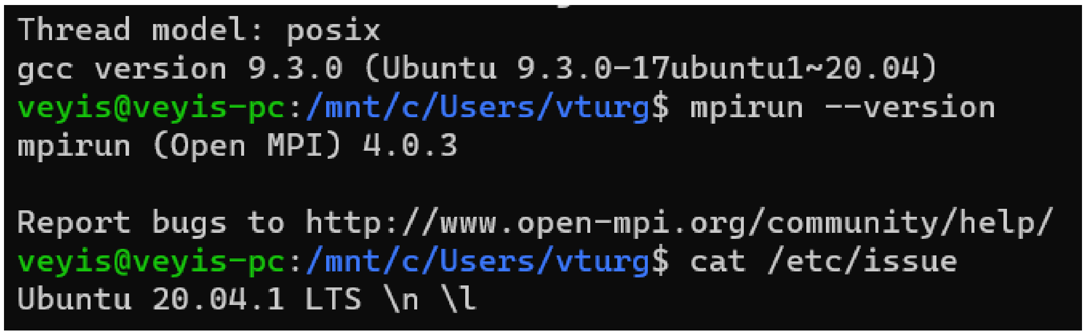
```
veyis@veyis-pc:~/desktop/cmpe300project/src$ mpic++ -o cmpe300_mpi_2017400210 ./cmpe300_mpi_2017400210.cpp
veyis@veyis-pc:~/desktop/cmpe300project/src$ mpirun  --oversubscribe -np 11 cmpe300_mpi_2017400210 ./in3.tsv
--------------------------------------------------------------------------
WARNING: Linux kernel CMA support was requested via the
btl_vader_single_copy_mechanism MCA variable, but CMA support is
not available due to restrictive ptrace settings.

The vader shared memory BTL will fall back on another single-copy
mechanism if one is available. This may result in lower performance.

  Local host: veyis-pc
--------------------------------------------------------------------------
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 P2 : 0 3 4 8 11 13 21 26 32 39
Slave P3 : 0 3 4 5 11 18 21 26 46 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 P4 : 0 3 11 14 21 28 30 32 39 40
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
[veyis-pc:00161] 10 more processes have sent help message help-btl-vader.txt / cma-permission-denied
[veyis-pc:00161] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages
veyis@veyis-pc:~/desktop/cmpe300project/src$ |
```

I designed the project with these versions of programs in WSL environment.

27.01.2021
**Adalet Veyis Turgut**



```
Thread model: posix
gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)
veyis@veyis-pc:/mnt/c/Users/vturg$ mpirun --version
mpirun (Open MPI) 4.0.3

Report bugs to http://www.open-mpi.org/community/help/
veyis@veyis-pc:/mnt/c/Users/vturg$ cat /etc/issue
Ubuntu 20.04.1 LTS \n \l
```

**Program structure**

I used three functions other than main. relief function is the function that slaves call once to execute algoritm. manhattanDistance function is a helper to calculate Manhattan distance and lastly topT function extracts top features after calculating feature weights.

In main, I initialized the Mpi environment and opened the input file. Then, I extracted parameters of the program ( P, N, M, A, T) from first two lines of the input file at master using built-in stoi and substring functions. I broadcasted these values to slaves using MPI_Bcast function. After these lines, I initialized arrays of master and slaves that will be used in MPI_Scatter and MPI_Gather methods.

There are N lines at each file and each line has A+1 values. Since I use 1 dimensional array, I should initialize the array in master as N*(A+1) length, right? But scatter does not work that way. It scatters first part of the data to itself. Therefore I added N/(P-1) *(A+1) dummy length to inputData array. Before scattering the data, I stored it in inputData array. Each consecutive A+1 elements represents a line in input file. Scatter method takes 8 parameters, last two are environment parameters and straight-forward. First three are about sender and next three are about receiver processors. inputData is the data to be sent and instances is the array that will be receiving data. (A+1) * N / (P-1) is the amount of data to be sent to each slave and MPI_DOUBLE is the datatype of the array.

After sending the data, I called relief function in each slave. Since relief function returns a vector, I copied it to array so that I could send it back to master easily. Slaves holds their top T features at topT array and master collects them to topFeatures array. I used gather function to collect the data back from slaves. MPI_Gather is the reverse of scatter function. It, again, collects first part of the data from itself. I, again, added dummy values and disregarded them while calculating at the end.

After gathering all the features from slaves, I must sort them in increasing order and delete the repeated items. Firstly, I used built-in sort function. Since I added T dummy values to the beginning, I disregarded them. After sorting, I extracted unique elements to extractedFeatures array. The main logic behind this extraction is comparing every element in topFeatures array to extractedFeatures. If it exist, continue; else, add it to array and increment index. Before terminating program, I printed out the top features in increaing order.

Relief function takes N/P-1 lines of data in 1 dimensional array form, N, A, M, T and rank of the slave as parameters. First thing it does is converting array to two dimensioanl vectors. N/(P-1) lines and A+1 element in each line. I found minimum and maximum of each feature with nested loop and if else statements. It was very simple. After that, I calculated nearest hit and miss. For every line other than current, calculate the manhattan distance using helper function. Then, if class value is same as current line, it is a possible hit, compare the distance with current minimum distance. Else if class value is not the same, it is a possible miss, compare the manhattan distance with current nearest miss. Then, to calculate feature weights, diff method in relief algorithm is used. Rather than writing a seperate function, I just used a for loop to calculate it. Here I used all the helper values I calculated before such as maxFeature, nearestHit,etc. The return value of this function is the vector of top T features, I calculated it in other function named topT.

Another helper function is manhattanDistance. It takes two vectors as parameters and calculates the sum of absolute values of difference of each indeces of these two vectors. Since parameter vectors' last element is class value (0 or 1), we do not need to add it to calculation.

Last function is named topT. It sorts the feature weights, prints out and returns their indices in increasing order. To not losing indices of features, I made a pair. Then I sorted the features using built-in sort function. Since each element is a pair, I also have indeces. I extract top T of them and returned it.

**Difficulties encountered**

At first I tried to send the data in vector / matrix (2d array) form, but I could not managed it. So, I scattered and gathered data using 1 dimensional array. Keeping track of indices was really frustrating. I mostly worked with vectors, since it is easy to work with. Copying the coming data from array to vectors in slaves was a bit cumbersome as well. I also wanted to send raw strings to slave processors in order to make the program more parallel. My motivation was processing the raw (tsv) input in slaves parallelly and consequently reducing time, but I failed to send string again. So, in the end, I converted raw string lines to 1 dimensional double array in the master processor.

**Conclusion**

With the increasing number of computing units and power, parallel programming became a must. In order to maximize the efficiency an minimizing time, dividing the total work among processors is a important. In this project, I learned the basics of these. On the other hand, relief is an easy algorithm to implement, I will try to use this algorithm in ML problems. With the things I learned in Cmpe322 course about parallel computing, I am very excited and looking forward to make more projects about parallel programming in real life.