Cmpe 300

Berkay Demirtaş

Mpi Project

Programming project

Submission Date: 27.01.2021

Introduction:

In this project, we are asked to apply relief alghorithm in parallel manner to given data set with Mpi interface. There is 1 master processor and (P-1) slave processors. (P is the total number processors). Master procesor handles I/O operations and send the given data to slave processor (same number of instance to each of them, number of iterations and number of feature need to be detected by each slave). Slave processors applies relief alghorithm. Relief is an algorithm that enables us to detect most important and varying features in our data set. I will explain relief alghorithm in detail incoming sections. At end slaves prints the feature ids they find and sends this ids to master. Master takes this ids, removes dublicates and print result.

Program Execution:

my mpi version (2.1.1)

gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0)

There should be a cpp compiler in your computer to run this program. Following comments will be explain the setup needed in lubuntu environment. Then download libopenmpi-dev library with "apt install" command (version 2.1.1). After this you are ready to run this program. Open command line and go to file that includes source file. After that run the following command to create to object file.

1. mpic++ -o hellopp ./cmpe300_mpi_2017400234.cpp

Let's say name of our input file is "input.txt" and we have 11 processors. Then to run this object file

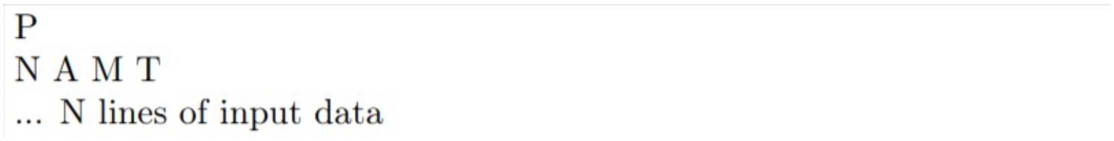
2. mpirun -oversubscribe -np 11 ./hellopp input.txt

If program does not stop, you can use Ctrl^c to stop.

cmpe300_mpi_2017400234.cpp

Input and Output:

Input file should have following format;
*1



P
N A M T
... N lines of input data

P is the number of processor available. (1 Master and P-1 slave). (same with the given with command line)

N is the number of instances and N is always divisble by P-1.

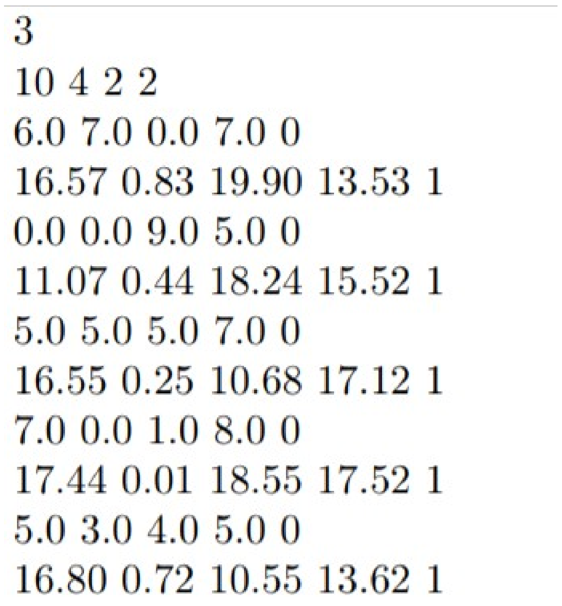
A is the number of features each instance contains and for each instance there is a class binary number.

M is the iteration count will be used in relief algorithm.

T is the number of features ids that each slave should print.

At the end there are N line of data each corresponds an instance.

*2



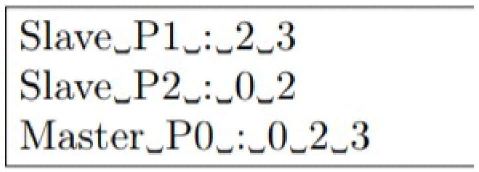
```
3
10 4 2 2
6.0 7.0 0.0 7.0 0
16.57 0.83 19.90 13.53 1
0.0 0.0 9.0 5.0 0
11.07 0.44 18.24 15.52 1
5.0 5.0 5.0 7.0 0
16.55 0.25 10.68 17.12 1
7.0 0.0 1.0 8.0 0
17.44 0.01 18.55 17.52 1
5.0 3.0 4.0 5.0 0
16.80 0.72 10.55 13.62 1
```

You can see an example input file.

As an output you should expect one line output for each slave processes that contains detected features ids by relief algorithm(T feature ids). At the end master process take all feature id's removes dublicates and print all of them once again in ascending order. (order of output lines of slaves can be unordered. For example in the first line, there might be outputs of second slave).

Output of example input file is ;

*



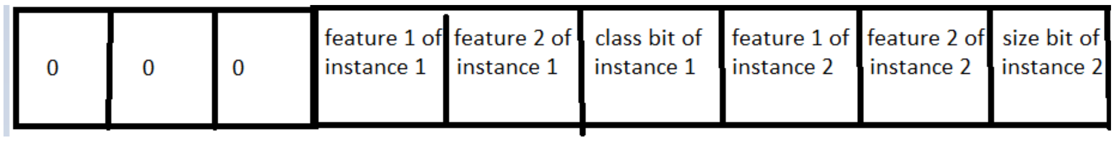
```
Slave␣P1␣:␣2␣3
Slave␣P2␣:␣0␣2
Master␣P0␣:␣0␣2␣3
```

Program structure:

In my program there are 3 important array:

1. double arr[(P*N*(A+1))/(P-1)]

This array stores features of all instances (with size bit).



0	0	0	feature 1 of instance 1	feature 2 of instance 1	class bit of instance 1	feature 1 of instance 2	feature 2 of instance 2	size bit of instance 2

For example , we see arr of an input that have 2 features, 3 process and 2 instances.

In that case this array is divided into 3 part and each process takes one of them which means each slaves handle one instance.

[(P*N*(A+1))/(P-1)] corresponds the number of instances + part of master.

2. double pref[N*(A+1)/(P-1)]

Stores data sent by master(with scatter) for each slave.

N*(A+1)/(P-1) corresponds the number of instances/(P-1)

3. int masterIndexes[T*P]

Gathers all indexes that finded by slaves. (With gather operation).

Relief algorithm:

This algorithm aims to weight the quality of attiribitues by considering dependincies between them.

pseudocode for Relief algorithm: *

```
Input: dataset (for each training instance a vector of feature values and the class value)
        n ← number of training instances
        a ← number of features (not including class variable)
        m ← number of iterations

initialize all feature weights W[A]=0.0
for i=1 to m do
        randomly select a target instance Ri
        find a nearest hit H and nearest miss M (instances)
        for A=0 to a-1 do
                W[A] = W[A] - diff(A,Ri,H)/m + diff(A,Ri,M)/m
        end for
end for

Output: the vector W of feature scores that estimate the quality of features
```

*

$$\mathrm{diff}(A, I_1, I_2) = \frac{|value(I_1,A) - value(I_2,A)|}{max(A) - min(A)}$$

Smaller manhattan distance means hit and if same class then nearest hit else nearest miss

*

Manhattan distance formula:

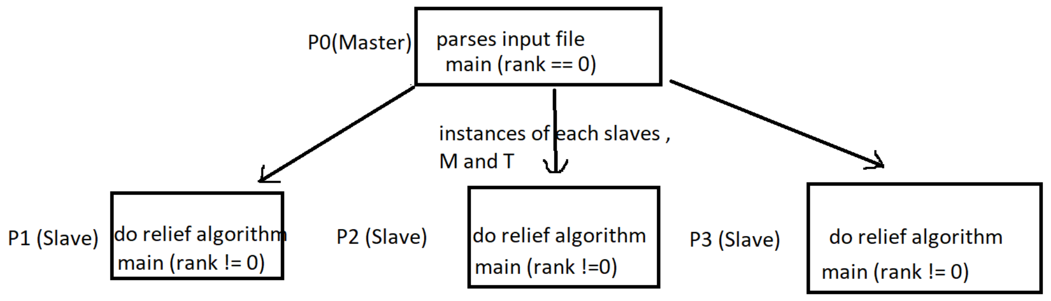
$$\boxed{\text{ManhattanDistance}([x_1, y_1, z_1], [x_2, y_2, z_2]) = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|}$$

Functions :

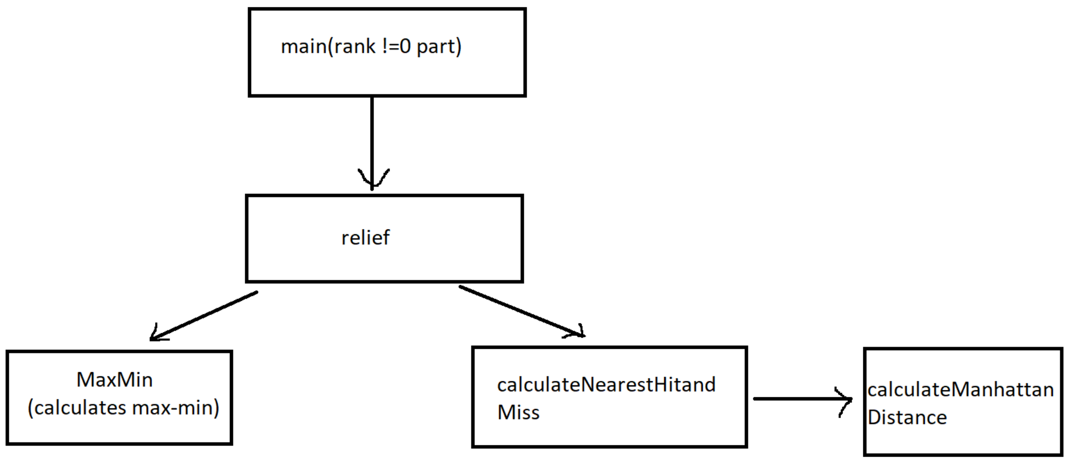
1. Main :

At the start of the main, Mpi_init is called and ranks are stored in "rank" integer.

In rank == 0 part of the main, input file is parsed and arr array is filled as explained above. M and T values is sent to slave processors by Mpi_send.



In rank != 0 part, slave processors recieves M and T values with Mpi_recieve. Then fills W array(calculated values of relief algorithm) with 0. After does the function calls in the following diagram and prints the ids of the needed features by checking updated W vector.



2. Relief: (Arguments : W vector and pref vector)

This function implements 2 for loop in psedocode of relief algorithm. Takes nearest hit and miss from calculateNearestHitandMiss function as a vector. By calling MaxMin function gets the denominator of "diff". By using this values calculates new W[a]'s for each iterations.

3. MaxMin: (Arguments : pref vector and featureNum int)

Checks all values in pref array with id featureNum and returns Max-min.

4. CalculateNearestHitAndMiss(Arguments : pref vector , returnValues vector and selected int)

Finds the manhattan distance between instance with selected id and all other processors by considering their class and returns the ids of nearest hit and nearest miss instances by putting them in a vector.

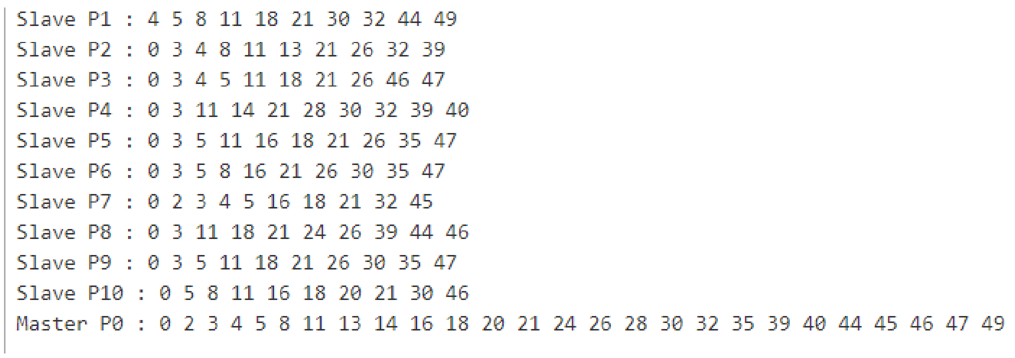
5. CalculateManhattanDistance(Arguments: v1 and v2 vectors)

It is a simple class that calculates manhattan distance between v1 and v2 vectors.

For messaging I use scatter, gather , send and recieve functions. They are explained at the top of this part.

Compile-Run output:

for input3



```
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 :

It was hard for me to realize how to use Mpi_scatter. When I first implement my project I does not realize that one part of the source array is sent to master processor. Therefore my last processor took wrong inputs. Another bug that I encounter is that sometimes iteration count is higher than number of slave processor so we need to take modulo.

Conclusion:

It was my first paralel programing project and I learned a lot. I realise that I need to change my way of thinking when writing paralel algorithms compared to sequential algorithms. However I think finding resources about Mpi is an issue.

* taken from announcment sheet of project