Perceptron Parallel Binary Classification Algorithm

Course 10324 - Parallel and Distributed Computation

Final Project - Afeka 2019

Eden Dupont 204808596

Instructions:

Input location is in C:\input.txt
Output location is in C:\output.txt

Run exe with MPI -

Running with 1 host will run the algorithm using OpenMP and CUDA

Running with more than 1 host will run the parallel version with unlimited threads (as the number of cores in the CPU)

Algorithm:

Reading data:

Master sends the data which he reads from input.txt to other hosts via broadcast in MPI.
Each host saves the data points into memory and copies it to the GPU for future calculations.
This process takes an average of 1-2 seconds.

Algorithm:

Master process

Master process sends out alphas for calculation to other hosts via MPI (lowest to highest), one alpha each.
The return with a result for their alpha (q,W,alpha), if it reaches q<Q, master verifies that no lower alpha is still being calculated, if there is, it will save the data and give the process a new alpha to calculate.
If there is no lower alpha being calculated it will print the solution to output and send finish tags to all processes.
If no result reached until all alphas have finished, send finish tags to all hosts and print output.

Host process

Each host receives an alpha to calculate, runs the function on all points until reaching a faulty point (serial), adjusts W and runs again until reaching the LIMIT, stop if no faults, calculates q with CUDA by running f on all points, counting the number of points not in their place and returning q, W and received alpha to master
The host will now wait for a new alpha to calculate or a finish tag to finish the process.

Parallelization implemented with:

OpenMP

- Master process, one core is responsible for dynamic alpha sending to hosts, second core responsible for doing calculations with alphas along with other slaves
- Vector operations - small operations with a for loop
- Malloc/free operations - for loops
- Sending to hosts initial alphas and FINISH tags - for loop

MPI

- Sending to hosts the data which MASTER reads
- Sending to hosts which alphas to calculate (dynamic)
- Sending back to host the result of the calculations

CUDA

- Calculation of q

Why I chose this architecture -

I chose the dynamic scheduling of the tasks (calculating q for each alpha) because there is no need to calculate anymore once we find a solution that satisfies the requirements for a given alpha, and we need the lowest alpha possible.
This is why there is one host (Master) who is responsible for the scheduling and determining whether the returned solution is satisfying the requirements.

For the q calculation for each alpha, there are two main steps:

1. Executing f function on all points in their order, and adjusting W according to the first misplaced point.
   This is ran a LIMIT value amount of times. Which makes it inefficient for CUDA calculation, because usually the first misplaced point is located within the first block of the calculation in CUDA, and W has to be adjusted again for another calculation.
   Time measurements were compared using CUDA versus serial implementation, and time was significantly reduced when using the serial version (17 seconds to 0.03 seconds - average)

2. Calculating the number of misplaced points given W, and returning the ratio q.
   CUDA was used for calculating q, which involves performing f function on all points, which is perfect for it because W is the same for all the points, and no adjustments are required. The function is performed, then the results are summed into smaller chunks, and reduced again for a final solution.

Time results:

Time is measured since the beginning of the algorithm, excluding data reading time.

Data1.txt

N= 150000 K=2 alpha_zero = 0.1 alpha_max = 0.9 LIMIT=100 QC = 0.3

Time Table (seconds)	1 Host	2 Hosts	3 Hosts	4 Hosts	5 Hosts	6 Hosts
Serial 1 Core	0.035720					
Parallel 4 Cores	0.003378	0.014506	0.009739	0.028858	0.019856	0.035134

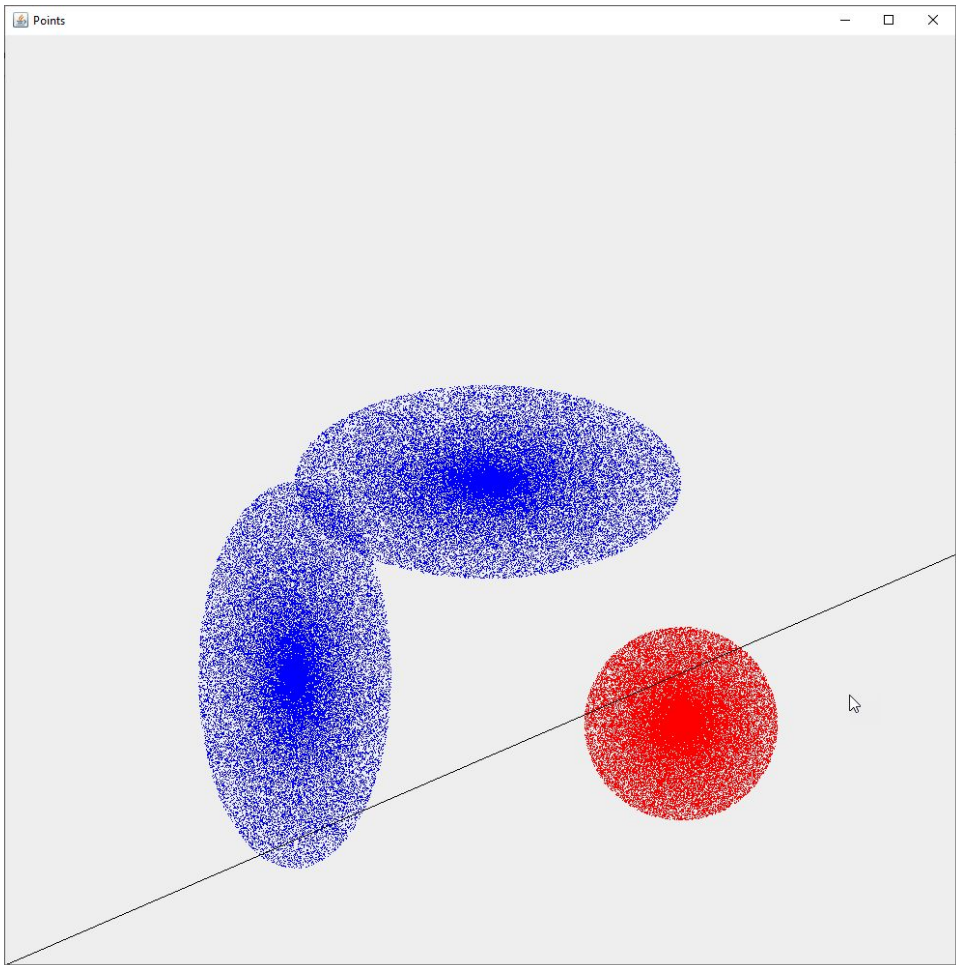
Result is :

Alpha =0.1 , q = 0.051747

W[0] = -144.159438

W[1] = 334.312856

W[2] = 3.4

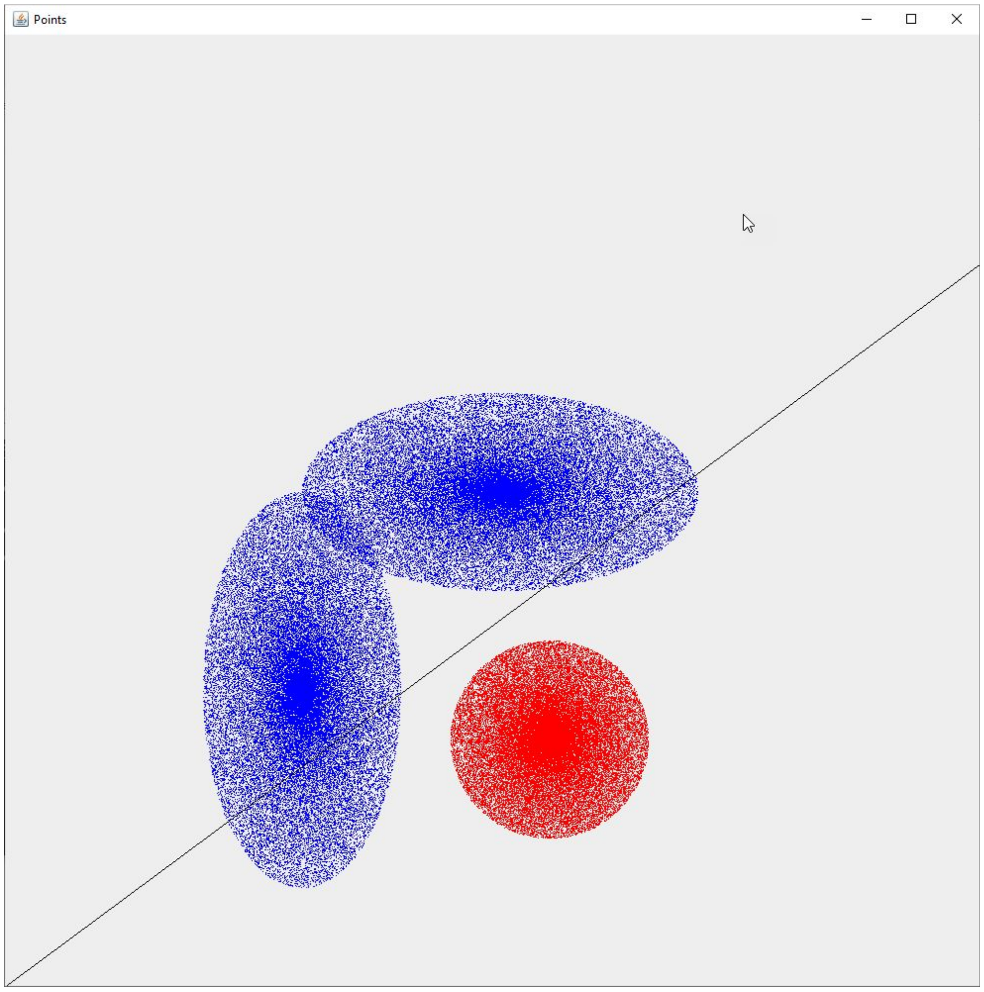


Using Data2.txt

N= 150000 K=2 alpha_zero = 0.1 alpha_max = 0.9 LIMIT=100 QC = 0.3

Time Table (seconds)	1 Host	2 Hosts	3 Hosts	4 Hosts	5 Hosts	6 Hosts
Serial 1 Core	0.033837					
Parallel 4 Cores	0.004219	0.014528	0.011569	0.016249	0.016536	0.036475

Result is :
Alpha =0.1 , q = 0.079240
W[0] = -211.073480
W[1] = 284.483084
W[2] = 1.6



Using Data1.txt (modified)

(KNOWN BUG - when no alpha is found, output can only be viewed from output file)

N= 150000 K=2 alpha_zero = 0.1 alpha_max = 0.9 LIMIT=100 QC = 0

Time Table (seconds)	1 Host	2 Hosts	3 Hosts	4 Hosts	5 Hosts	6 Hosts
Serial 1 Core	0.324613					
Parallel 4 Cores	0.003344	0.070433	0.031534	0.04821	0.034069	0.054321

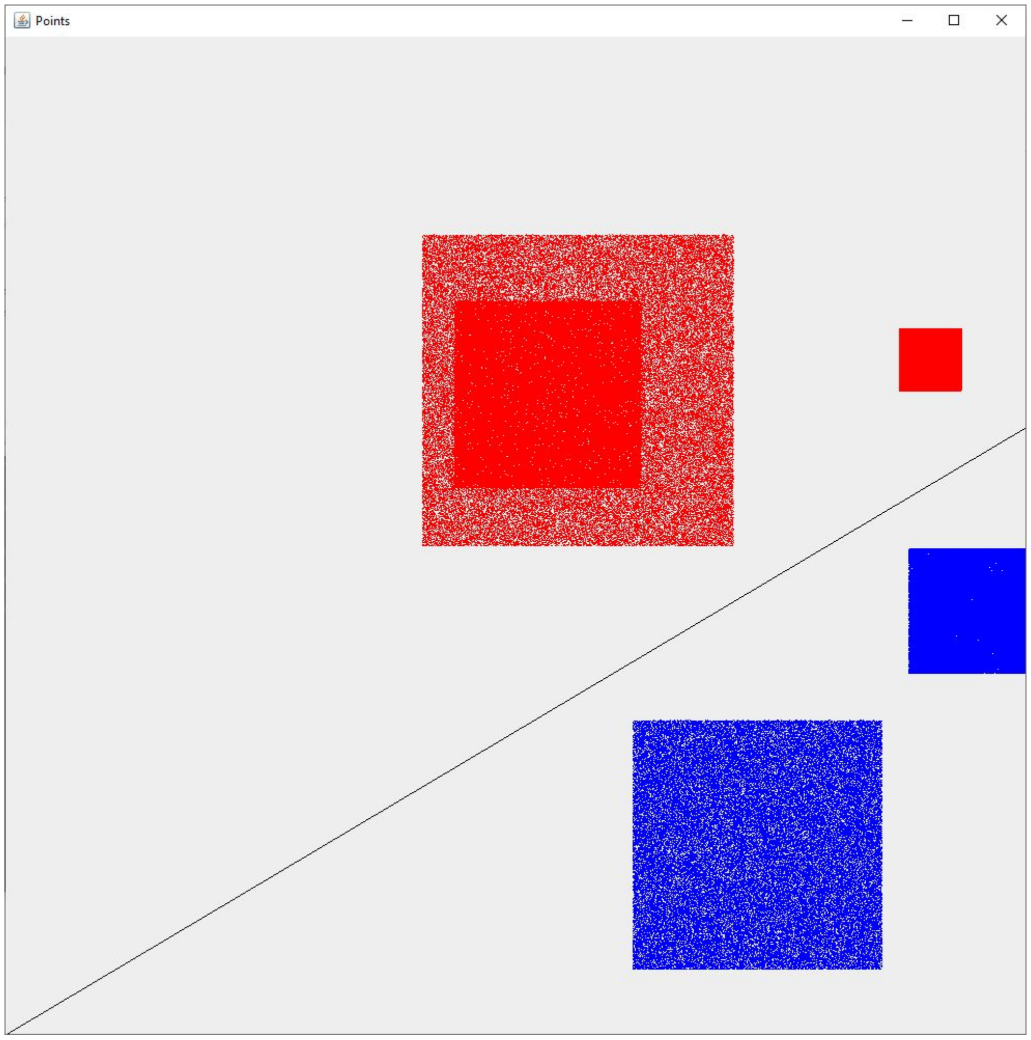
Result is :
No alpha is found

Using myData1.txt (generated with datasetMaker.exe)

N= 500000 K=2 alpha_zero = 0.1 alpha_max = 1.0 LIMIT=600 QC = 0

Time Table (seconds)	1 Host	2 Hosts	3 Hosts	4 Hosts	5 Hosts	6 Hosts
Serial 1 Core	0.222321					
Parallel 4 Cores	0.116887	0.809789	0.792426	0.785125	0.820189	0.777019

Result is :
Alpha =0.1 , q = 0
W[0] = 9.781100
W[1] = -16.4411
W[2] = 0.01

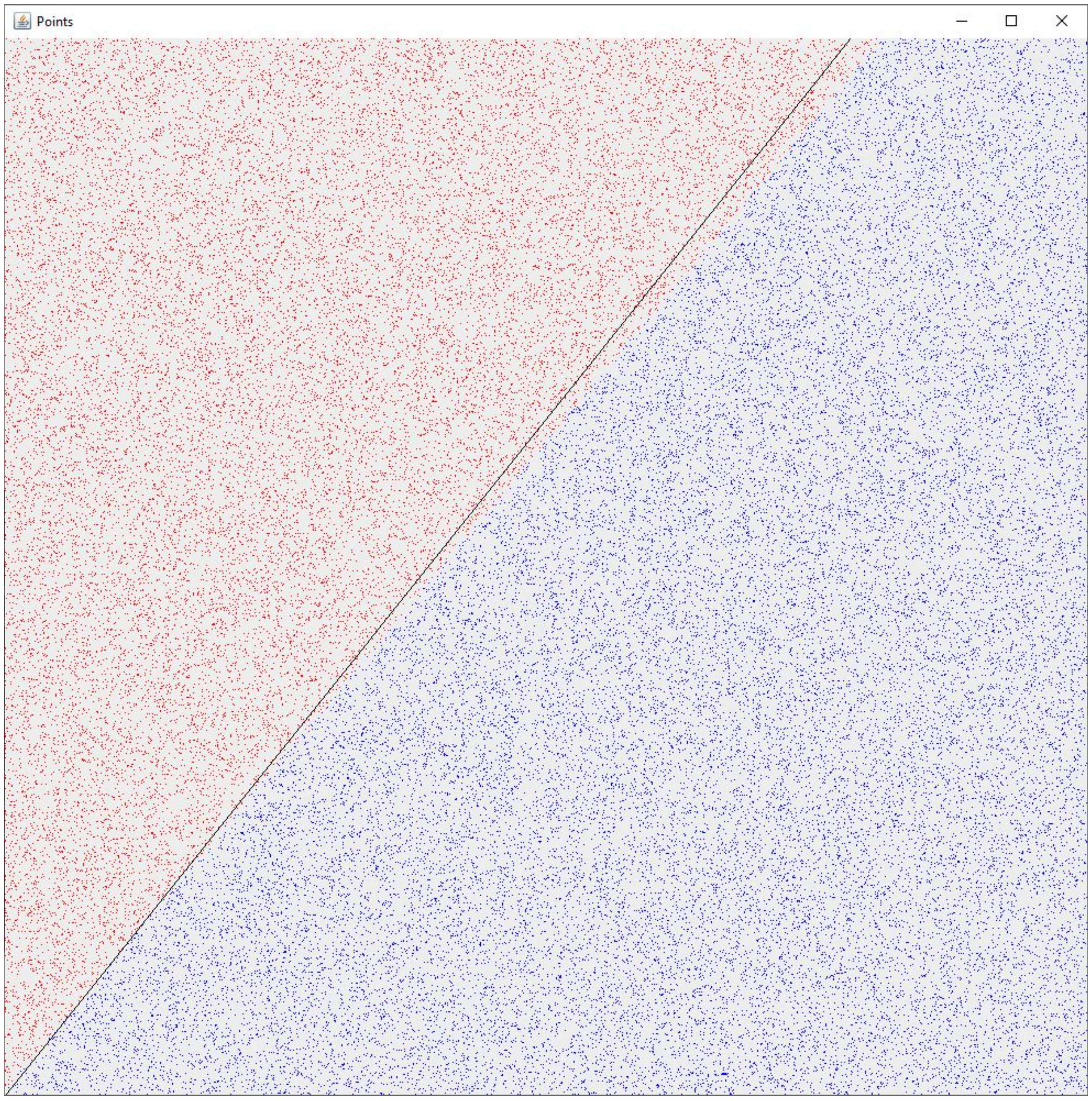


Using myData2.txt (generated with datasetMaker.exe)

N= 250000 K=2 alpha_zero = 0.01 alpha_max = 1.0 LIMIT=100 QC = 0.1

Time Table (seconds)	1 Host	2 Hosts	3 Hosts	4 Hosts	5 Hosts	6 Hosts
Serial 1 Core	0.060250					
Parallel 4 Cores	0.007008	0.025466	0.031769	0.028717	0.026348	0.050446

Result is :
(W should converge to 24,-20,100)
Alpha =0.01 , q = 0.009336
W[0] = 26
W[1] = -20.7686
W[2] = -0.06



Data Maker (included in zip)

This program can create data files with two options:

1. Choose W as an input, the program will create a dataset of points such that when the Perceptron algorithm is executed on the dataset, the solution should converge to the same W, generated point
2. Choosing the group choice, the program should create 6 squared clusters in different sizes.

BinaryClassification.jar (provided by lecturer Dr. Boris Morose)

Run jar file in CMD like so:

```
Java -jar BinaryClassification.jar input.txt
          OR
Java -jar BinaryClassification.jar input.txt 1 2 3
```

In order to print the line -ax+by-c=0