

KMeans Clustering Algorithm Parallelization With

CUDA Task 4

Presentation of Report:

From our implementation so far, it is pretty obvious that clustering is a search method that reveals hidden patterns that exist in datasets. A process of grouping data objects into disjoint clusters so that the data in each cluster are similar, yet different to the other clusters. In k-means, a data point consists of several values, called features. By dividing a cluster of data objects into k sub-clusters, k -means represents all the data objects by the mean values or centroids of their respective sub-clusters. Since the complexity and time taken by the sequential compiler is high, we proposed the use of GPU for parallelization to the maximum extent possible.

The objective of the project is to implement a parallel version of k-means clustering algorithm using CUDA parallel programming language. The project also aims to compare the efficiency between serial C++ and Cuda parallel versions of K-means clustering algorithm.

Implemented Planned Actions

Task I: Problem Analysis and Design of Solution

Project execution schedule and outline of its objectives. To study and understand the k-means clustering algorithm and the mathematical model.

Task II: Presentation of Solution with required Scenarios

Centroid approach to k-means algorithm. Validation of the Mathematical model. Implementation of the K-means Algorithm with Cuda programming.

Task III: Presenting a Working Solution for Various Devices:

An Implementation of K-Means Algorithm using a Sequential C++ as a benchmark for comparing the speedup of the Cuda version. An implementation of parallel k-means using **global memory** and **shared memory** to optimize the speed of the algorithm. Comparing the results timing with serial and parallel algorithms. Execution of both the sequential code and the parallel code comparing different data sets.

Task IV: Reporting

Comparison of Execution Time vs Input Data size : *Putting simple `time.time()` calculations immediately around the function invocations, I get*

the following results for a dataset of $n \in 100, 100000$ points, $T=300$ iterations and $k = 3$ clusters, using the average time of 5 runs:

Size of Data (N)	No. of Clusters (K)	Serial Time (C++)	Parallel Time (Cuda)
100	3	0.00054s	0.00956s
100000	3	0.26804s	0.0752s

Optimization of Code by fixing global memory load

The simple fix for the global memory loads is to place the means into shared memory and have the threads load them from there. Here is the noticeable improvement in the speedup.

Size of Data (N)	No. of Clusters (K)	Serial Time (C++)	Parallel Time (Cuda)	Parallel Time after optimization
100	3	0.00054s	0.00956s	0.00878s
100000	3	0.26804s	0.0752s	0.0611s

Performed the assignment and block-wise reduction in the same kernel to further obtain a speed up of 0.00822s for $N = 100$ and 0.0171s for $N = 100,000$. *Now, that's fast! Our CUDA implementation is nearly 16 times faster than plain c++.*

```
student@des10: ~/s183179
Iteration 9: centroid 1: nan
Iteration 9: centroid 2: 0.773515

real    0m0.137s
user    0m0.024s
sys     0m0.076s
student@des10:~/s183179$ nano kmeans.cu
student@des10:~/s183179$ nvcc kmeans.cu
student@des10:~/s183179$ time ./kmeans
0.712785
0.591974
0.136849
Iteration 1: centroid 0: 0.712785
Iteration 1: centroid 1: 0.591974
Iteration 1: centroid 2: 0.136849
Iteration 2: centroid 0: 0.768799
Iteration 2: centroid 1: 0.505655
Iteration 2: centroid 2: 0.178111
Iteration 3: centroid 0: 0.731632
Iteration 3: centroid 1: 0.499594
Iteration 3: centroid 2: 0.175375
Iteration 4: centroid 0: 0.751047
Iteration 4: centroid 1: 0.473530
Iteration 4: centroid 2: 0.163592
Iteration 5: centroid 0: 0.757344
Iteration 5: centroid 1: 0.464909
Iteration 5: centroid 2: 0.161756
Iteration 6: centroid 0: 0.756990
Iteration 6: centroid 1: 0.462935
Iteration 6: centroid 2: 0.153024
Iteration 7: centroid 0: 0.756990
Iteration 7: centroid 1: 0.458507
Iteration 7: centroid 2: 0.148406
Iteration 8: centroid 0: 0.777439
Iteration 8: centroid 1: 0.467869
Iteration 8: centroid 2: 0.145285
Iteration 9: centroid 0: 0.759048
Iteration 9: centroid 1: 0.461065
Iteration 9: centroid 2: 0.147883

real    0m0.122s
user    0m0.017s
sys     0m0.078s
student@des10:~/s183179$ time ./kmeans
```

Data size: 100

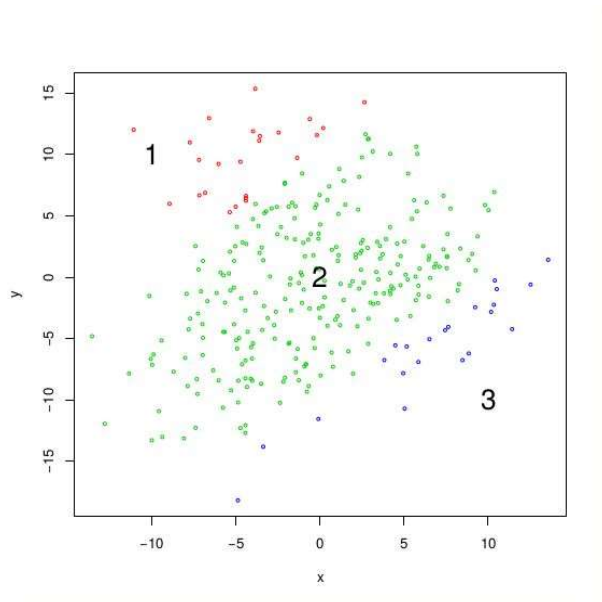
```
student@des10: ~/s183179
student@des10:~/s183179$ time ./kmeans[B
-bash: ./kmeans[B: Nie ma takiego pliku ani katalogu

real    0m0.001s
user    0m0.000s
sys     0m0.000s
student@des10:~/s183179$ nano kmeans.cu
student@des10:~/s183179$ nvcc kmeans.cu
student@des10:~/s183179$ time ./kmeans
0.287968
0.888427
0.352710
Iteration 1: centroid 0: 0.287968
Iteration 1: centroid 1: 0.888427
Iteration 1: centroid 2: 0.352710
Iteration 2: centroid 0: 0.139029
Iteration 2: centroid 1: 0.804230
Iteration 2: centroid 2: 0.471464
Iteration 3: centroid 0: 0.135753
Iteration 3: centroid 1: 0.859625
Iteration 3: centroid 2: 0.479090
Iteration 4: centroid 0: 0.137370
Iteration 4: centroid 1: 0.836495
Iteration 4: centroid 2: 0.482217
Iteration 5: centroid 0: 0.140618
Iteration 5: centroid 1: 0.842850
Iteration 5: centroid 2: 0.493124
Iteration 6: centroid 0: 0.139029
Iteration 6: centroid 1: 0.831847
Iteration 6: centroid 2: 0.496731
Iteration 7: centroid 0: 0.139029
Iteration 7: centroid 1: 0.830268
Iteration 7: centroid 2: 0.495296
Iteration 8: centroid 0: 0.139029
Iteration 8: centroid 1: 0.829741
Iteration 8: centroid 2: 0.494818
Iteration 9: centroid 0: 0.139029
Iteration 9: centroid 1: 0.829741
Iteration 9: centroid 2: 0.494818

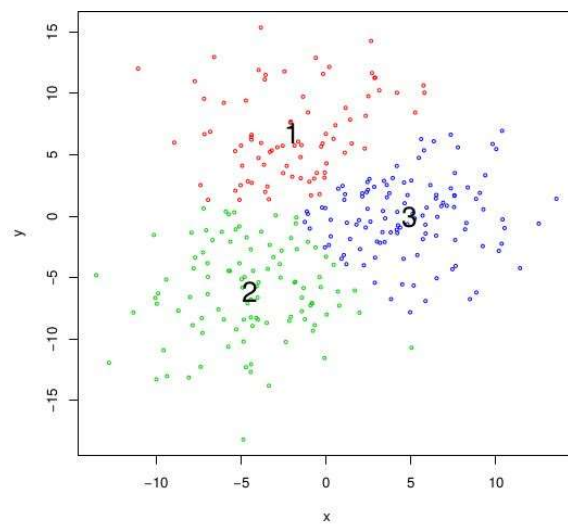
real    0m0.095s
user    0m0.046s
sys     0m0.045s
student@des10:~/s183179$
```

Datasize: 100,000

Visual Representation of *Points before algorithm*



Visual Representation of *Points after algorithm*



Objectives Achieved

Following is the list of objectives achieved during the course of the project:

- Implementation of sequential k-means clustering algorithm.
- An Implementation of K-Means Algorithm using a Sequential c++ as a benchmark for comparing the speedup of the Cuda version.
- An implementation of parallel k-means using global memory and shared memory to optimize the speed of the algorithm. Also performed the assignment and block-wise reduction in the same kernel for more optimization.

In Conclusion:

This algorithm can be used in the variety of application like grouping of same colors based on the RGB value in image, It can handle numeric weather forecast as well as abnormal

climate event identification, the Parallelism used reduces the execution time and so other application like wireless network, sensor based application and search engine can equally use it.