Ufuk UYAN N17133297
Hasan MUTLU N17232403
Necati ÇAĞAN N18148806

21/01/19

CMP 674: PARALLEL COMPUTING USING GPUs

PROJECT FINAL REPORT

1. INTRODUCTION

Clustering is one of the most commonly dealt learning algorithms. The main purpose of clustering is labeling among a collection of data. In this manner, clustering can be defined as the process of organizing objects into groups having similarities. Since large groups of data must be handled while clustering, computation time can be too large. In this project, a type of clustering algorithm, namely k-means clustering algorithm, is going to be implemented using three different computing methodologies, which are serial execution on CPU, parallel execution on CPU and parallel execution on GPU, and performance indices are going to be discussed. Throughout the report, the problem is going to be defined and the related algorithm is going to be described in a brief way, and then implementations are going to be explained. Following that, performance index of each implementation is going to be handled. Finally, necessary discussions are going to be made and the report is to be wrapped up.

2. PROBLEM DEFINITION

As mentioned above, clustering is one of the most essential problems among the engineering community [1]. Clustering is defined as grouping a set of data points so that points within each cluster are like each other while points from different clusters are dissimilar. The points under investigation are in a high-dimensional space, and similarity of these points is mostly defined in terms of a distance measure. A well-known distance measure can be exemplified as the Euclidean distance. Clustering problem can be applied to any field on daily life. Customers having purchased similar products, products based on the sets of customers, documents having similar words are small portions of cluster examples.

Methods of clustering can be divided into several categories, such as hierarchical and point assignment [2]. In hierarchical clustering, each point is itself a cluster, and iteratively two nearest clusters are combined into one cluster. On the other hand, in point assignment clustering, a set of clusters is maintained and then points are placed into the nearest cluster [2]. K-means clustering is another method of clustering, which is going to be covered in this report.

3. K-MEANS CLUSTERING ALGORITHM

K-means clustering algorithm is a popular clustering algorithm used in unsupervised machine learning [1]. Given n samples on space, k-means algorithm partitions these points into k clusters such that each sample belongs to a certain cluster which has a centroid. The centroid is commonly referred to a mean value.

The algorithm flow is defined as follows [3]:

- **i.** Randomly choose k points from the given dataset as initial centroids.
- **ii.** Create the cluster such that all points are the closest to a centroid. A simple way to make a decision of points is constructing a boundary line which may be a bisector between two centroids.
- **iii.** Move the centroids by taking mean of the points in a certain cluster.
- **iv.** Check if maximum iteration number is reached or centroid movement is finished. If true, stop the algorithm. Otherwise, go to step (ii).

4. IMPLEMENTATION

When we first created our algorithm in serial and parallel on CPU, we were able to use the object-oriented programming approach provided by the C++ language. But we predicted that the object-oriented approach would make our CUDA solution very difficult and therefore, we did not use object-oriented fashion in our CUDA solution, so we tried to apply our serial and parallel implementation to CUDA only using array structures.

Although there are several other implementations of k-means clustering algorithm as in [4] and [5], in our CUDA solution we divide parallelization problem into two parts. First part is finding cluster of each points and second part is updating center of each centroids according to their points.

For the first part of the solution, we calculate a block size according to point count (to calculate block count, Thread count is fixed and used as 32) so we used a thread to calculate centroid of each points. After that we pass all the points to our kernel function which calculates closest cluster and sets mapping array (This array is used to hold which number belongs to which cluster). As mentioned before, we used Euclidean distance formula to calculate distance between the clusters and the given points.

For the second and last part of our solution, we calculated the new positions of each centroids with using our mapping array and the points and we have repeated our algorithm until the last two digits of each position of the centroids remain unchanged.

5. RESULTS AND DISCUSSION

K-means clustering algorithm is implemented as a serial execution on CPU, parallel execution on CPU and parallel execution on GPU as described in the introduction part. Serial implementation of algorithm on CPU is tested with different cluster size and data count. The results we have obtained are shown in Table 1.

Table 1. Algorithm Execution Times of Serial Implementation on CPU

Cluster Size \ Data Count	**1.000**	**10.000**	**100.000**	**1.000.000**
5	7 ms	30 ms	417 ms	2.29 s
10	10 ms	99 ms	1.42 s	12.80 s
15	18 ms	175 ms	1.55 s	18.96 s
20	16 ms	165 ms	1.39 s	16.01 s

As shown in the table, when data size increases, computation times increase tremendously. In the case of 1000 data, computation time is about doubled when cluster size is increased from 5 to 20. However, computation time is increased by 7 times in case of 1 million data.

Parallel implementation of algorithm on CPU is tested with same cases with previous computing methodology. The results are shown in Table 2.

Table 2. Algorithm Execution Times of Parallel Implementation on CPU

Cluster Size\ Data Count	**1.000**	**10.000**	**100.000**	**1.000.000**
5	4 ms	47 ms	267 ms	3.92 s
10	12 ms	63 ms	395 ms	6.70 s
15	16 ms	114 ms	859 ms	9.17 s
20	23 ms	137 ms	1.10 s	13.99 s

Parallel implementation of algorithm on CPU has shown worse results than expected in the cases of small number of data and cluster size. However, 2 seconds improvement was observed in the case of 1 million data and 20 clusters.

Finally, implementation of algorithm with parallel manner on GPU shows reasonable results as expected. Memory operations are relatively time-consuming and in case of small number of data and cluster size, memory operation outweigh execution time. However, in the extreme cases, execution time performance is decreased by one seventh.

Table 3. Algorithm Execution Times of Parallel Implementation on GPU

Cluster Size\ Data Count	**1.000**	**10.000**	**100.000**	**1.000.000**
5	79 ms	83 ms	130 ms	620 ms
10	79 ms	104 ms	168 ms	1.12 s
15	81 ms	93 ms	257 ms	2.04 s
20	78 ms	111 ms	395 ms	2.28 s

Ufuk UYAN N17133297
Hasan MUTLU N17232403
Necati ÇAĞAN N18148806

21/01/19

Results shown in Table 1, Table 2 and Table 3 are illustrated in comparative graphs as in Figure 1, Figure 2, Figure 3 and Figure 4. In addition, screenshots obtained from profilers, both in nvprof and nvvp are shown in Figure 5 and Figure 6.

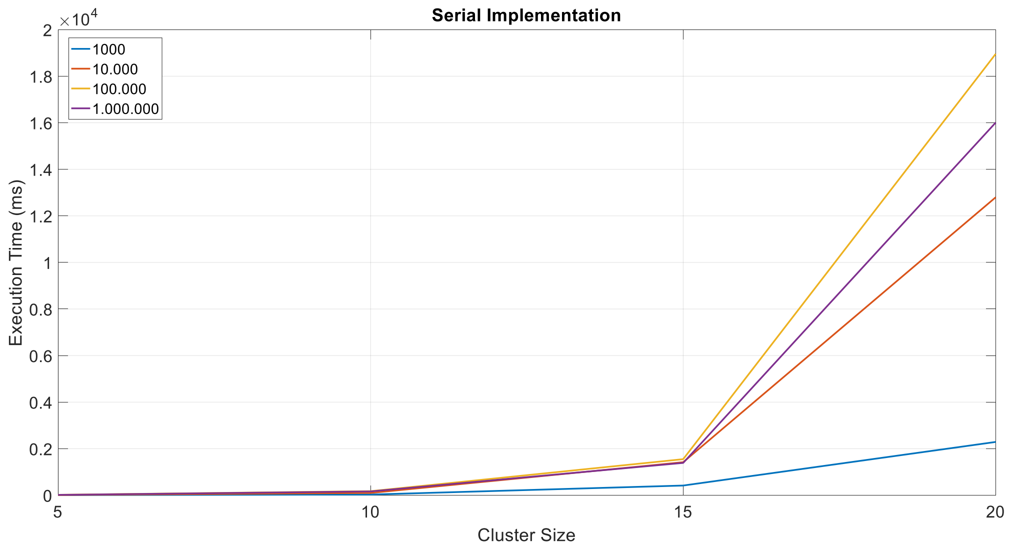


Figure 1. Execution Times of Algorithm with Serial Implementation on CPU

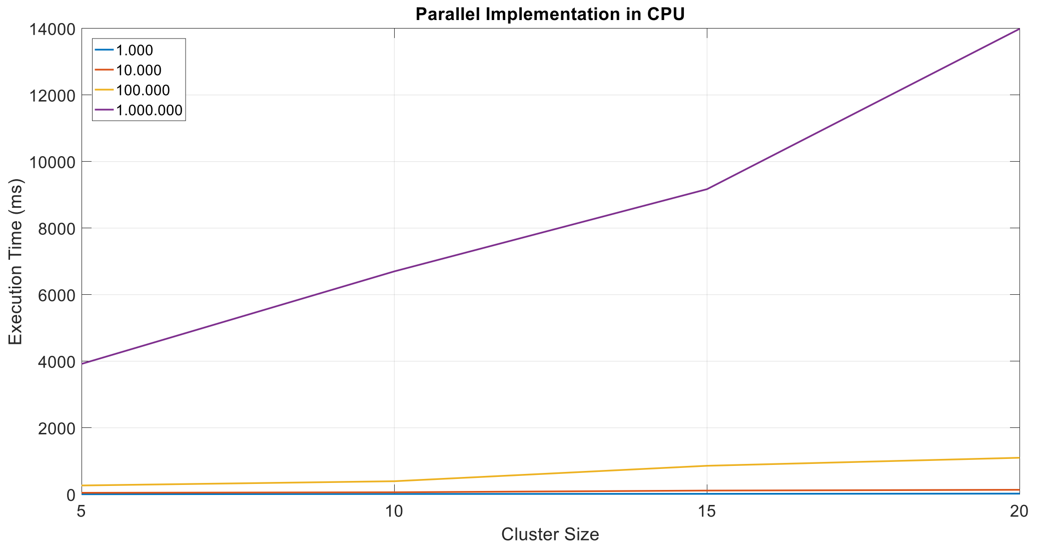


Figure 2. Execution Times of Algorithm with Parallel Implementation on CPU

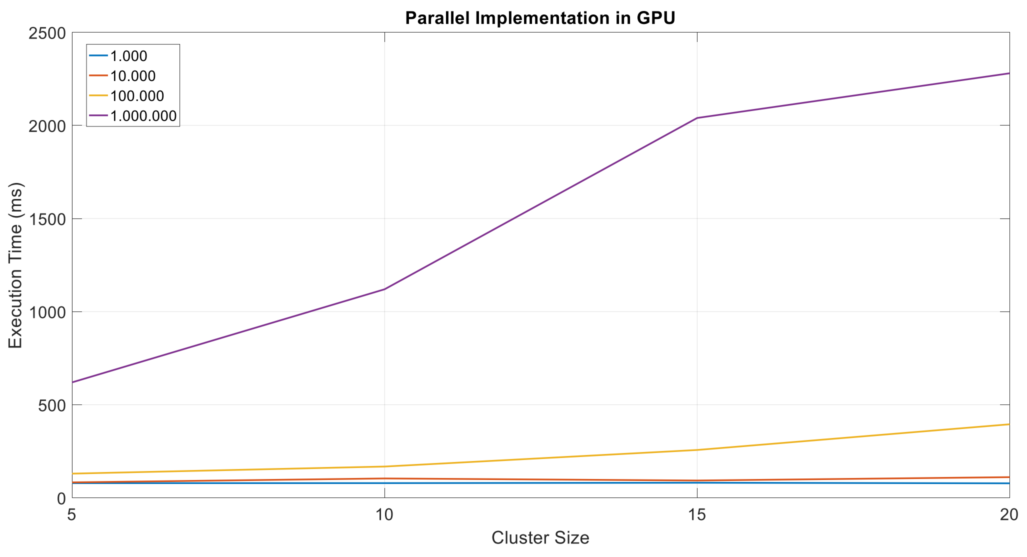


Figure 3. Execution Times of Algorithm with Parallel Implementation on GPU

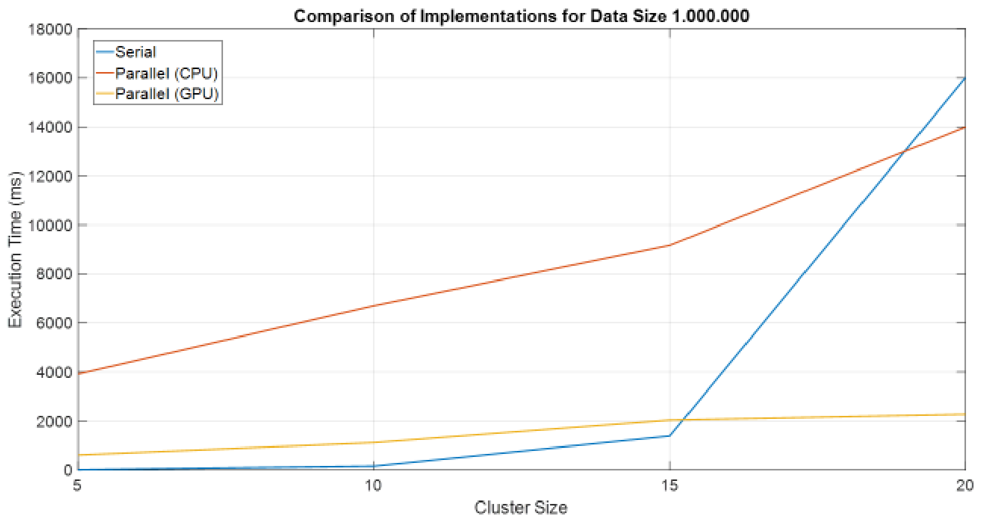
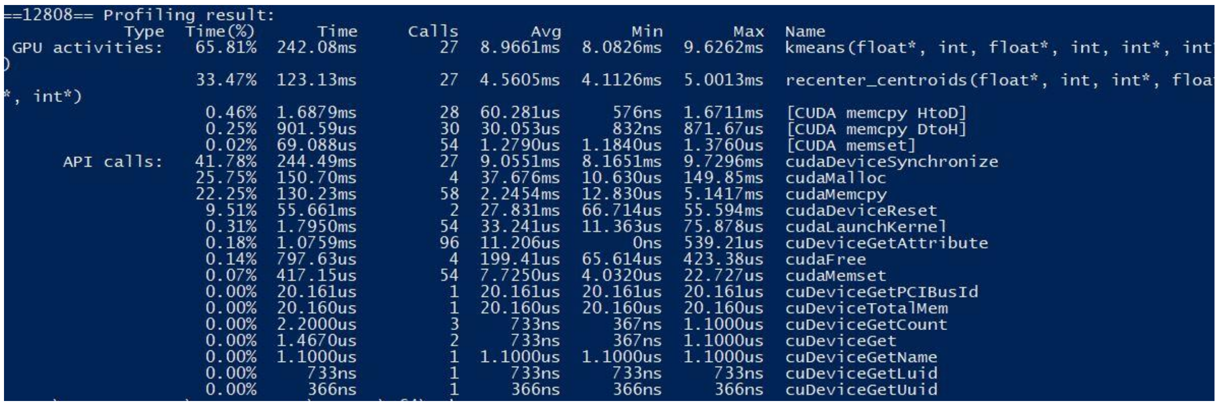
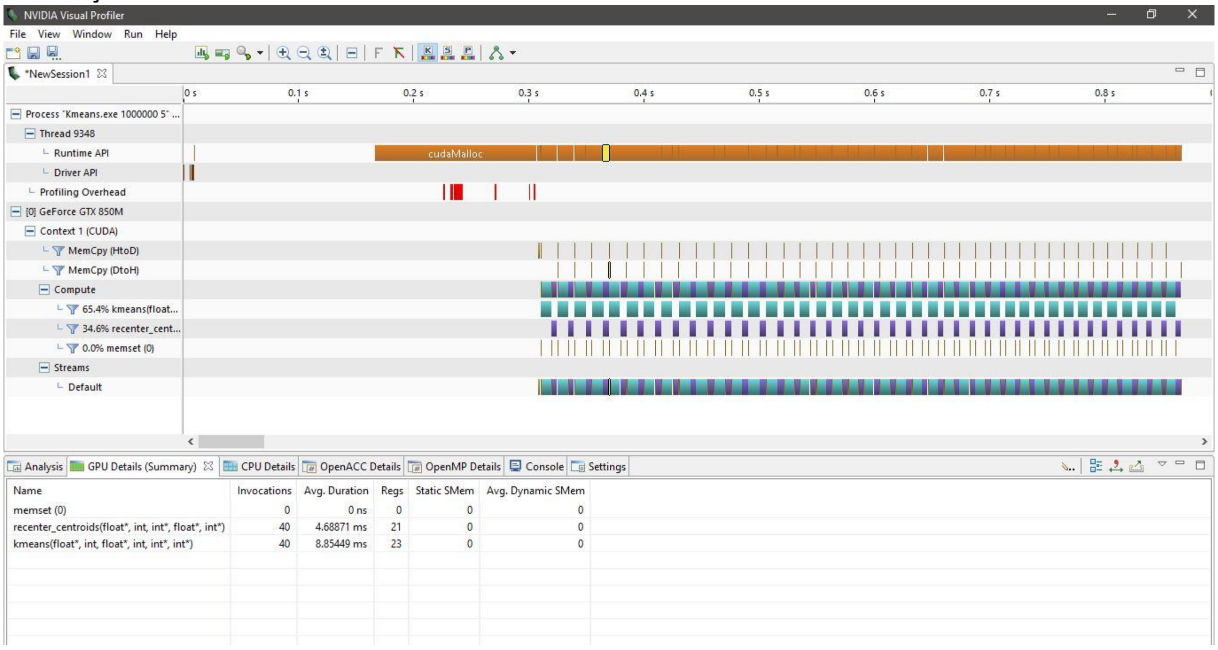


Figure 4. Comparison of Implementations for Large Data (1.000.000)



```
==12808== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   65.81%  242.08ms        27  8.9661ms  8.0826ms  9.6262ms  kmeans(float*, int, float*, int, int*, int
)
                   33.47%  123.13ms        27  4.5605ms  4.1126ms  5.0013ms  recenter_centroids(float*, int, int*, floa
*, int*)
                    0.46%  1.6879ms        28  60.281us     576ns  1.6711ms  [CUDA memcpy HtoD]
                    0.25%  901.59us        30  30.053us     832ns  871.67us  [CUDA memcpy DtoH]
                    0.02%  69.088us        54  1.2790us  1.1840us  1.3760us  [CUDA memset]
      API calls:   41.78%  244.49ms        27  9.0551ms  8.1651ms  9.7296ms  cudaDeviceSynchronize
                   25.75%  150.70ms         4  37.676ms  10.630us  149.85ms  cudaMalloc
                   22.25%  130.23ms        58  2.2454ms  12.830us  5.1417ms  cudaMemcpy
                    9.51%  55.661ms         2  27.831ms  66.714us  55.594ms  cudaDeviceReset
                    0.31%  1.7950ms        54  33.241us  11.363us  75.878us  cudaLaunchKernel
                    0.18%  1.0759ms        96  11.206us       0ns  539.21us  cuDeviceGetAttribute
                    0.14%  797.63us         4  199.41us  65.614us  423.38us  cudaFree
                    0.07%  417.15us        54  7.7250us  4.0320us  22.727us  cudaMemset
                    0.00%  20.161us         1  20.161us  20.161us  20.161us  cuDeviceGetPCIBusId
                    0.00%  20.160us         1  20.160us  20.160us  20.160us  cuDeviceTotalMem
                    0.00%  2.2000us         3     733ns     367ns  1.1000us  cuDeviceGetCount
                    0.00%  1.4670us         2     733ns     367ns  1.1000us  cuDeviceGet
                    0.00%  1.1000us         1  1.1000us  1.1000us  1.1000us  cuDeviceGetName
                    0.00%     733ns         1     733ns     733ns     733ns  cuDeviceGetLuid
                    0.00%     366ns         1     366ns     366ns     366ns  cuDeviceGetUuid
```

Figure 5. nvprof Screenshot



Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
memset (0)	0	0 ns	0	0	0
recenter_centroids(float*, int, int*, float*, int*)	40	4.68871 ms	21	0	0
kmeans(float*, int, float*, int, int*, int*)	40	8.85449 ms	23	0	0

Figure 6. nvvp Screenshot

6. CONCLUSION

K-means clustering is a well-known clustering algorithm used in unsupervised machine learning application. Computational cost is one of the most important obstacles in clustering algorithm in case of excessive amount of data and cluster size. In this study, we implement K-means clustering algorithm with parallel implementation methodologies on GPU by means of CUDA. Obtained results shows that memory operation in GPU implementation dominates execution times when there is few data and cluster. The best improvement in execution time is obtained in case of clustering excessive amount of data to many clusters.

7. REFERENCES

[1] Clustering - K-means. [Online]. Available: https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/. [Accessed: 21-Jan-2019].

[2] Clustering Algorithms. [Online]. Available: https://web.stanford.edu/class/cs345a/slides/12-clustering.pdf. [Accessed: 21-Jan-2019].

[3] Clustering Using K-Means Algorithm. [Online]. Available: https://towardsdatascience.com/clustering-using-k-means-algorithm-81da00f156f6. [Accessed: 21-Jan-2019].

[4] Li, You, et al. "Speeding up k-means algorithm by GPUs." Journal of Computer and System Sciences 79.2 (2013): 216-229.

[5] Cuomo, S., et al. "A GPU-accelerated parallel K-means algorithm." Computers & Electrical Engineering (2017).