Accelerated Kth Nearest Neighbor Algorithm Final Report

Sanshan Guo Andrew id: sanshang
Xinhao Zhou Andrew id: xinhaoz

May 2020

1 Summary

We implemented a sequential version of Kth Nearest Neighbor Algorithm and accelerated it with Kmeans on sequential version. We also implemented the parallel version of KNN[4] and Kmeans[6] with CUDA[1]. We compared the performance improvement of each stage optimization.

2 Background

2.1 What are the key data structures?

Sequential Version KNN For sequential version, we use priority queue as the primary data structure to implement the sorting process. We abstract each data point as a class DataPoint with property id, a vector of attributes and label.

Sequential Version Kmeans For kmeans, each cluster contains a central datapoint, a list of all the points and the size of the cluster, which is abstracted as a class named Cluster. Another class named Kmeans contains the attribute k and a list of clusters.

Parallel Version For parallel version, we use array as the primary data structure. We abstract the id of all the data points as a one-dimensional array. We abstract the attributes of all the data points as a two-dimensional array but store in a one-dimensional array for convenience. We abstract the label of all the data points as a one dimensional array.

2.2 What are the key operations on these data structures?

Sequential Version KNN The data points are parsed from the train and test data sets and stored to vector. The vector of data points then enter a

priority queue to get the smallest distance between a test data point and a train data point.

Sequential Version KMeans Firstly, the central points of k clusters are randomly selected and stored in a vector of data points. And the Kmeans and Clusters are initialzed randomly accoding to the central points. Then all the data points are iterated to be assigned to different clusters according to the distance between that data points with the central data point. Then we get a new central data point based on the current cluster.

2.3 What are the algorithm's inputs and outputs?

algorithm input The input of the algorithm will be the training data set, the test data set, the distance calculation function, the number of k.

algorithm output Th output of the algorithm will be the predicted label of all the test data and the accuracy of the prediction.

2.4 What is the part that computationally expensive and could benefit from parallelization?

We use array to store data point id, attributes and labels for both train and test data sets. Thus, when computing the distance, this action involves the large volume of array data access.

KNN Algorithm

- **Distance Calculation** For KNN algorithm, we need to calculate the distance between the target test data set and all the data points in train data set, which is really computation expensive.
  - Firstly, we introduce Kmeans algorithm to partition the data points in train data sets. In this way, we compute before to narrow down the potential data points that could contribute to the kth nearest neighbors. In this way, we limit the access time of the train data point array.
  - Secondly, instead of doing the computation sequentially, we compute the distance between a test data point and the train data points in a cluster in parallel to increase the computation performance.
- **Sort** For KNN algorithm, we need to sort the data point by the distances. This action is performed with a priority queue in sequential version, which involves reading the data multiple times. We also try to use bubble sort to sort the data points in sequential version. It turns out that this action is really computation expensive. This action can be performed in parallel to improve the program performance.

- **Assign Labels** For KNN algorithm, we need to use the computed kth nearest neighbor and assign the label to all the test data points. We can introduce parallelism in this part to increase the program performance. We parallel all the text data points to ease that computation complexity.

KMeans Algorithm

- **Compute Distance** The distance between the central points and all the train data points in train data set is needed. The data attribute is read to device in parallel. The distance is calculated based on the attributes of two data points. The sequential version requires large volume of data calculation and huge delay. This can be improved by performing the distance calculation in parallel.
- **Get Central Points** The calculation of the central points of each cluster requires the iteration of the nodes in each cluster. This behaviour has less data dependency and could be done in parallel. In this way, the computation complexity is decreased and the program performance is improved.
- **Compare New and Old Cluster** The comparison between the new and the old data central point requires the iteration of all the clusters. This could also be done in parallel. There is no data dependency among clusters thus parallization can really help increase the program performance.
- **Assign Cluster** The data point in test data set has to be assigned to a specific cluster. This operation requires the iteration of the whole array and the whole cluster. The distance is calculated between the current data point and the k data center of the clusters. This requires a large amount of computation and the data dependency between clusters and data points is really low. Thus, we could use parallelism in the process to increase the speed of this process and improve the program performance.

2.5 Break down the workload

2.5.1 Where are the dependencies in the program?

KNN Algorithm There is relatively no dependencies existed in this process. The prediction process is based on the distances between the test data point and the train data point, which is fixed. Thus, the dependency between data point is really small.

KMeans Algorithm

- The distance computation between the central data point and the train data point and the cluster initialization process has to be done before the assign Cluster operation.
- The operation of updating central points attributes has to be implemented before the operation of get new central point.

- The operation of getting new central point has to be implemented before the operation of compare the old and new central points.
- The operation of comparing old and new central points have to be done before the operation of getting new central points for the clusters.
- The assignment of central points has to happen before the computation between data points and the central points.

2.5.2 How much parallelism is there?

There are many potential places where we can use parallelism to speed up. The data points of train data set and test data set are independent. There are relatively few dependency between data points.

2.5.3 Is it data-parallel?

Yes, there are a lot of data parallelism existed in this process. The data points in the train and test data sets are relatively independent. Thus, there a potential places where we can do data parallelism to increase the speed of the program.

KMeans Assign Cluster During the process of assigning clusters to a data point, we can do data parallelism by assigning a data point to a single thread in GPU to carry out the parallelism.

KNN Assign Label During the process of assigning labels to a data point, we can do the parallelism. The prediction part is really independent and we have done the computation and sorting preparation work before the actual prediction process. Thus, the data parallelism is existed in the prediction part.

2.5.4 Where is the locality?

Array Representation We abstract the data point as class DataPoint, with property id, vector of attributes and label. Thus, it is more suitable to abstract the attributes of the train and test data sets as two dimensional array. We make the attributes on dimensional array to try to increase the locality when accessing the attributes.

3 Approach

3.1 Describe the technologies used.

Sequential Version For sequential version, we implement the whole process with **C++**. We use **priority queue** data structure to handle the sorting process of the prediction process. We read the data points to the priority queue and overwrite the operator function as comparing the distance between current node and the target node.

Optimized Sequential Version In order to limit traversing all the train data points, we introduce **KMeans algorithm** by partitioning the data points before the prediction.

Parallel CUDA Version For parallel CUDA version, we use **CUDA** as the programming language and try to use **NVIDIA GPU** on **GHC Machines** to increase the speed of the process. We use **thrust library** during this process, which is a C++ Standard Library that can do the interaction between GPU and CPU.

Parallel OpenMP Version We also tried to use **OpenMP** to parallel the program and increase the program execution speed. We simplely add the pragma before the potential parts that could benefit in parallel.

3.2 Describe how you mapped the problem to your target parallel machine(s).

KNN Compute Distance, KNN Initial Index For KNN Compute Distance function, we map the computation to the 3D version of grid, with each layer as on cluster. We map block (y, z) in each layer x computing the distances between train data set and test data set.
Within the block, we map each thread in a block at position (x, y) to compute the distance between data point x in train data set and data point y in test data set.

KNN Assign Label For KNN Assign Label function, we map the computation to a 2D version of grid. Each line of the grid represents a single cluster. Each thread of the line represents the computation of a single test data point.

KMeans Compute Distance For KMeans Assign Cluster function, We map the computation to a 1D grid. Each thread in block performing the operation of computing the distances between several potential central points and single current train data point.

KMeans Update Central Points Attribute For Kmeans update central points attribute function, we map the computation to a 1D grid. Inside a block, each thread carries out the computation of a single train data point.

Kmeans Get New Central Point, KMeans Compute Old And New Central Point For this function, there is only one grid, with multiple block. Each block corresponds to the computation inside a single cluster.

KMeans get New Cluster Center For this function, we map the 1D grid, with each thread in the block representing one data point in train data set.

KMeans assign cluster For assigning cluster function, we map to 1D grid, with each thread in the block representing on data point in test data set.

3.3 Did you change the original serial algorithm to enable better mapping to a parallel machine?

Computational Goals Stay Same. The original goal and computational steps does not change. We follow the steps:

- sorting all the data points in train data set by the distance of the train data point and the current test data point;
- get the nearest kth data points
- assign the label of the test data point

Representation Changes. The data structure changes for better get parallelism and locality.

- The data structure used for sorting process changes from priority queue to array representation to better map to the thread on GPU machines.
- The data point representation changes from Class DataPoint to multiple one dimensional array to increase locality.

3.4 If your project involved many iterations of evaluation and optimization, please describe this process as well.

- We tried to use two dimensional array to optimize the sequential version. But it turns out the sorting process of the two dimensional array costs a lot. Thus, we remain the original design of using a priority queue to implement the sequential version.
- We tried to use openmp by adding pragma to parallel the program. It turns out the parallel effect of openmp is not as good as CUDA. We remained the coding of CUDA in our repository.
- We tried to use trust library to implement the C++ and CUDA. It turns out that it actually works in this process. Detail code can be seen in your repository.

4 Results

We test our results based on the UCI Census Income Data Set[2]. The data set has more than 30,000 instances, and we divide them into train set with 30000 instances and test set with 2561 instances. In the meantime, we filter the nominal attributes and normalize the numeric attributes for convenience.

One thing we discovered during our experiments is that 74% of the labels are negative, which means if our accuracy is around 74%, the result is not valid. We had a pilot experiment on this data set on Weka[3], which is a machine learning tools, and the accuracy of KNN prediction based on this split is around 79%. The accuracy is shown in figure 1.

Correctly Classified Instances	2067	79.3474 %
Incorrectly Classified Instances	538	20.6526 %
Kappa statistic	0.4288	
Mean absolute error	0.2223	
Root mean squared error	0.4025	
Relative absolute error	58.9951 %	
Root relative squared error	91.1597 %	
Total Number of Instances	2605	

Figure 1: Weka KNN accuracy: Our validation baseline

4.1 Kmeans speedup

In this part we get the partial speedup of our parallel kmeans algorithm. We compare the cuda version of kmeans and C++ sequential version of kmeans to get a performance of our kmeans algorithm. To measure the performance of the computation, we change the number of clusters since it has a significant effect on the computation intensity. The result is in figure 2. The time of computation fluctuates because as the number of clusters varies, the iterations for the centers to converge are not linear.

Performance analysis For the Kmeans algorithm, we don't have an amazing speedup. We think the main reason is that it takes a long time to obtain the new centers. For each iteration, we need to sum up all instances in a cluster to find out a new center. We use the prefix sum function offered in assignment 2 to enhance the parallelism within a block. By doing that, each block only needs to atomically add once for one attribute. However, this is still slow.

4.2 Knn speedup

The CUDA Knn algorithm gets a comparatively better speedup to the sequential Knn algorithm. The speedup is presented in figure 3. In the beginning, we scaled the value of k in Knn algorithm, however, the number of k doesn't have an obvious impact on the computation time, so we scale the test instances.

Performance analysis The Knn algorithm gets a relatively good speedup over sequential Knn. And the size of test instances have a linear effect on the computation time, while it doesn't impact the speedup much. One noticable thing is that the sequential Knn algorithm use the priority queue to sort the distance vector, while in the CUDA Knn, we use Thrust sort library to sort the

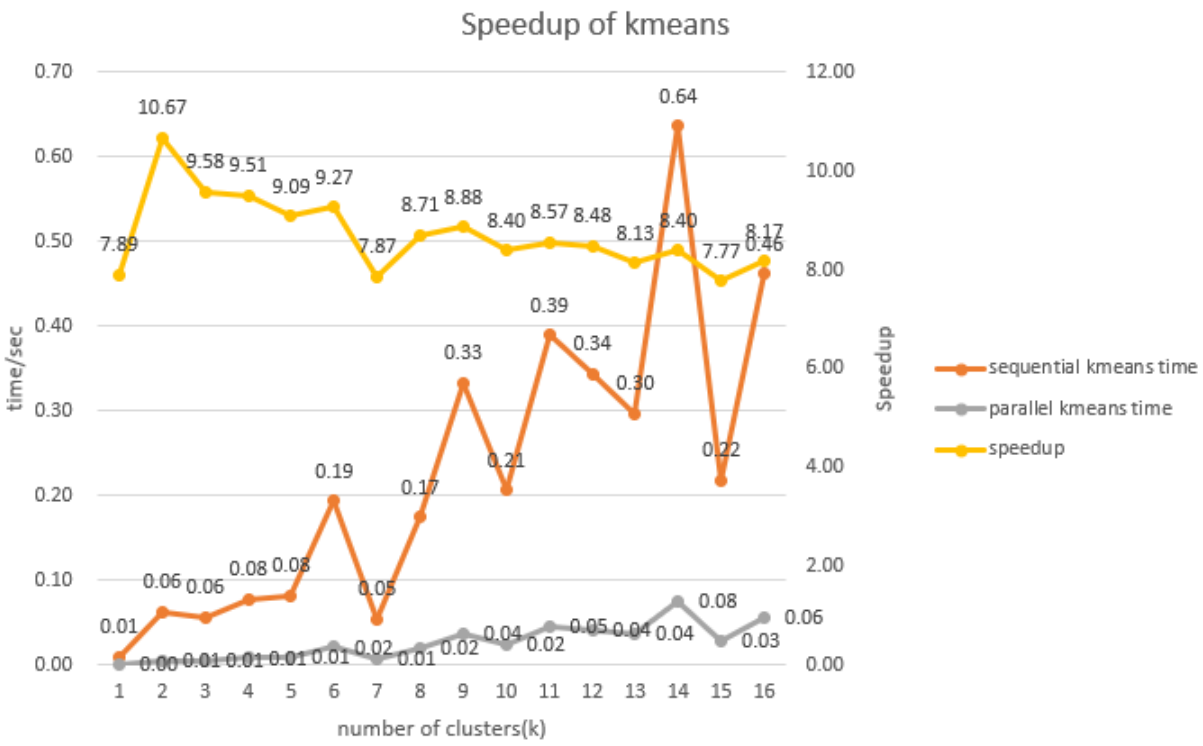

Figure 2: CUDA Kmeans speedup over sequential Kmeans

distances. The Thrust library use radix sort[5]. This may make some difference for the speedup.

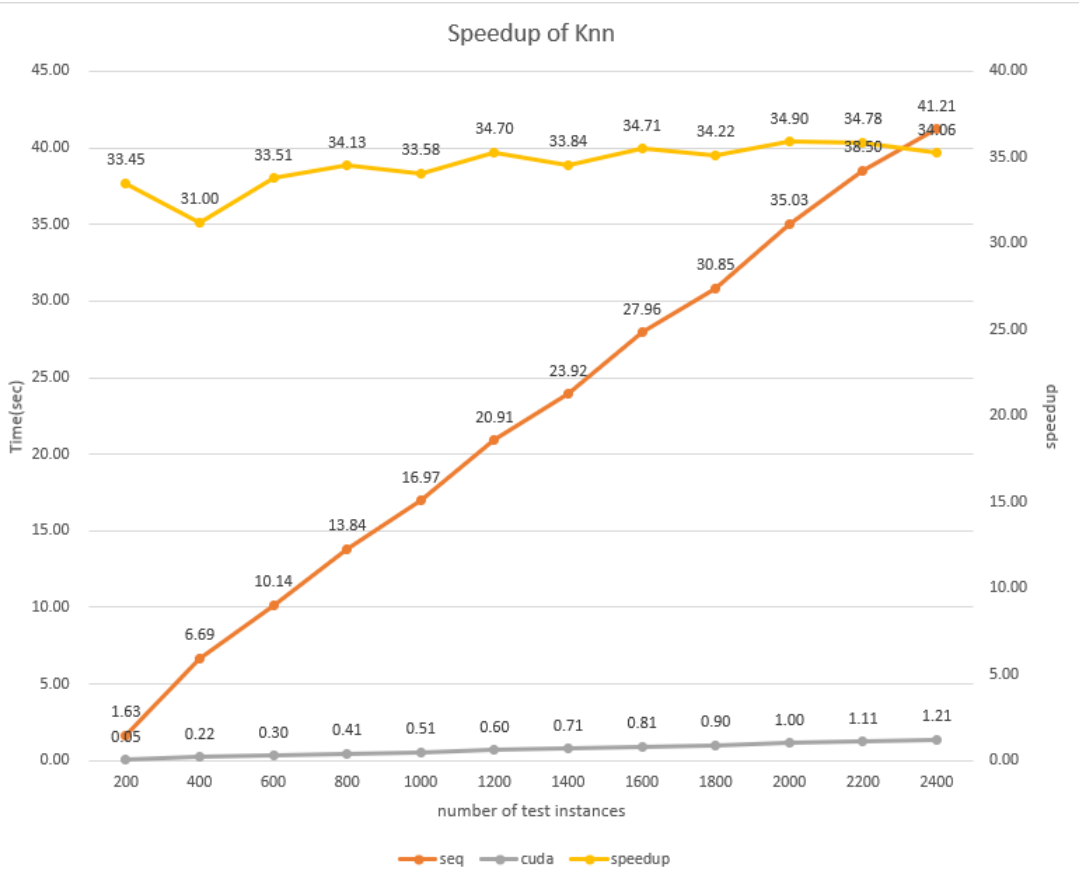

Figure 3: CUDA Knn speedup over sequential Knn

4.3 Kmeans and Knn Speedup

We combined the Kmeans algorithm and Knn algorthm. The Kmeans separates the train instances into several clusters and the Knn first reference to the each cluster's center points and chooses a best cluster for test instance. By doing this, we can reduce the computation intensity in Knn algorithm. The performance is presented in figure 4.

Performance analysis The speedup is linear to the number of clusters. The division of train instances is helpful for getting a much smaller computation size. And the good point is that division of clusters have a minor effect on the

accuracy. The accuracy is shown in figure 5. From the figure 5 we can see that division will always lead to accuracy loss, but it's accecptable since the loss is generally less than 2%. When the number of clusters is 16, the Kmeans and Knn algorithm can get a speedup of 96 over the sequential Knn algorithm while has almost the same accuracy.

4.4 Result summary

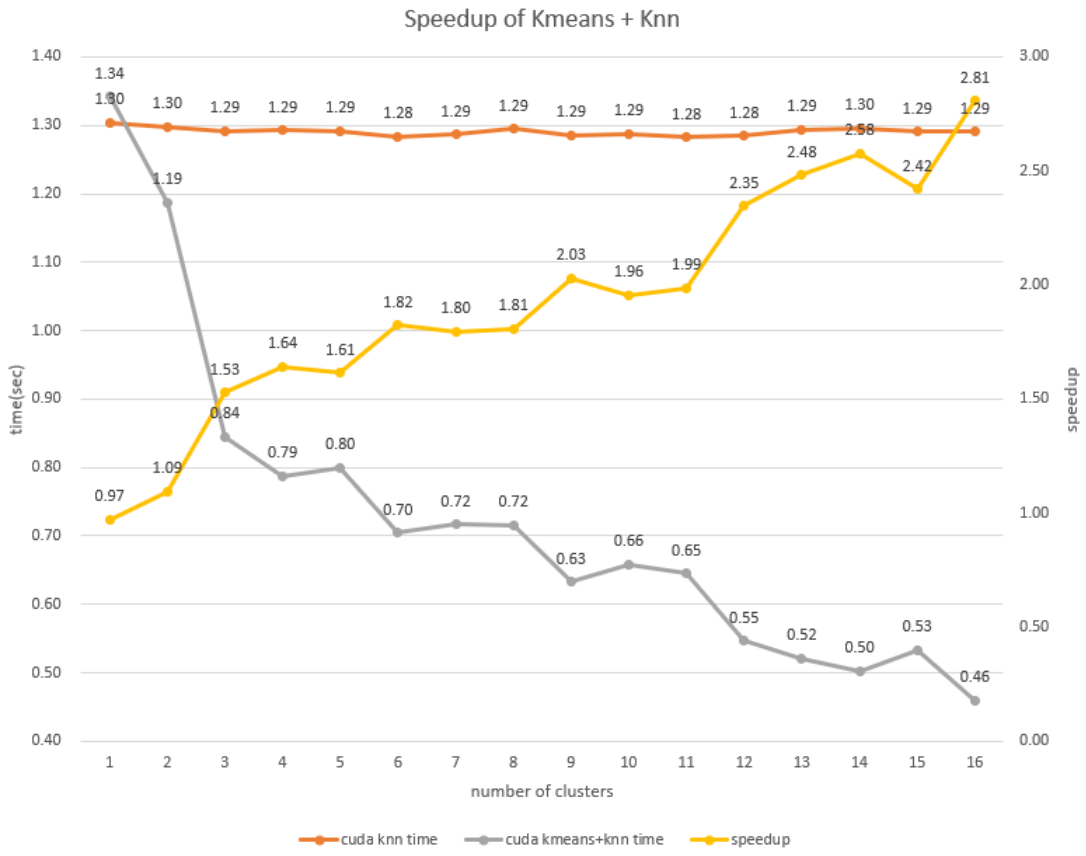

Figure 4: CUDA Kmeans and Knn speedup over CUDA Knn

5 Division of Work

Sanshan Guo(50%):

- Sequential version of Kmeans algorithm.

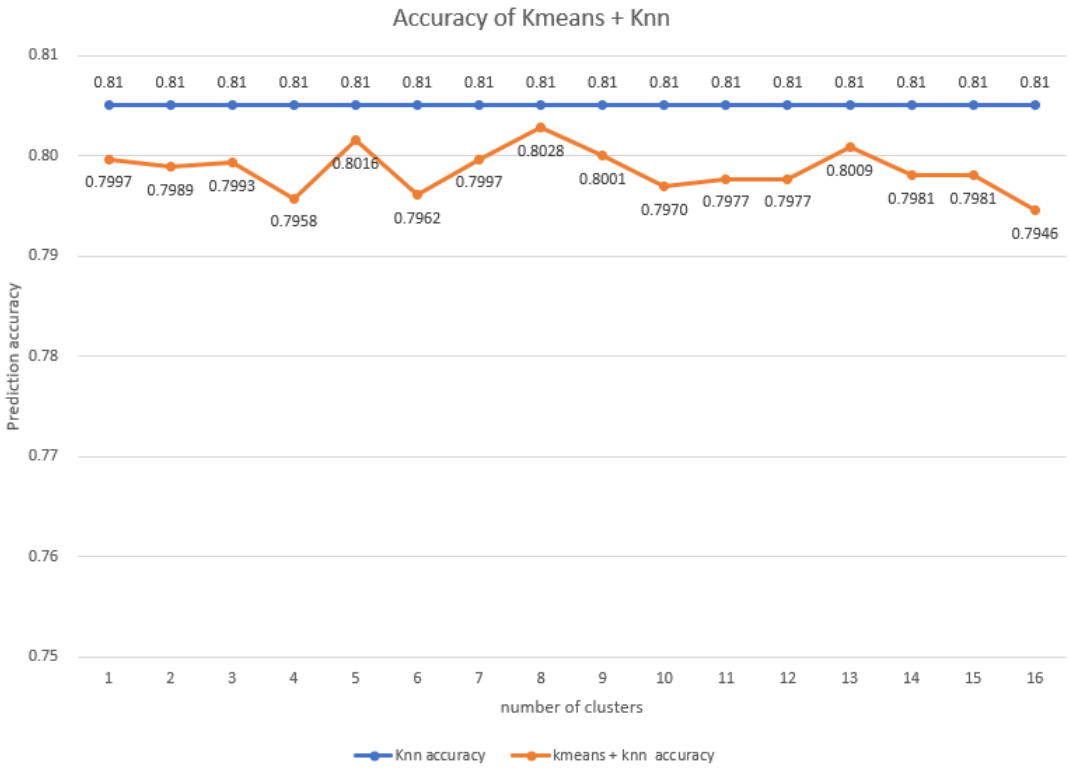

Figure 5: Kmeans and Knn accuracy

- Data parser.
- CUDA version of Kmeans. Combination of Kmeans and Knn algorithm.

Xinhao Zhou(50%):

- Sequential version of Knn algorithm.
- CUDA version of Knn. Combination of Kmeans and Knn algorithm.

6 Code Repository

https://github.com/Xinhao-Zhou/418-finalproject-knn

References

[1] Zhenyun Deng et al. "Efficient kNN classification algorithm for big data". In: *Neurocomputing* 195 (2016), pp. 143–148.

[2] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[3] Mark Hall et al. "The WEKA data mining software: an update". In: *SIGKDD Explorations* 11.1 (2009), pp. 10–18.

[4] S. Liang et al. "A CUDA-based parallel implementation of K-nearest neighbor algorithm". In: *2009 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 2009, pp. 291–296.

[5] Dhirendra P. Singh, Ishan Joshi, and Jaytrilok Choudhary. "Survey of GPU Based Sorting Algorithms". English. In: *International Journal of Parallel Programming* 46.6 (Dec. 2018). Copyright - International Journal of Parallel Programming is a copyright of Springer, (2017). All Rights Reserved; recent update - 2018-11-13, pp. 1017–1034. URL: `https://search-proquest-com.proxy.library.cmu.edu/docview/2132193913?accountid=9902`.

[6] M. Zechner and M. Granitzer. "Accelerating K-Means on the Graphics Processor via CUDA". In: *2009 First International Conference on Intensive Applications and Services*. 2009, pp. 7–15.