Kmeans clustering on GPU

Chinmai *cchinmai@ur.rocheter.edu*

***Abstract*—In this project an optimized k-means implementation on GPU is compared with a sequential k-means implementation on CPU. The project is intended to analyse the performance improvement and bottlenecks of parallel programs to implement machine learning algorithm on a GPU. Focus of the project is to exploit the computational capabilities and hence the algorithm is realized in hybrid manner, parallelizing distance calculations on the GPU while sequentially updating cluster centroids on the CPU based on the results from the GPU calculations.**

I. Introduction

Data mining is the process of analyzing data on specific dimensions and finding correlations in large databases to establish relationship and extract essential information. Due to immense growth in data Machine learning algorithms have been adapted to obtain optimization and scalability.One of the many data mining methods widely in use is partitional clustering as know as k-means which was proposed by Lloyd. Popularity of kmeans is due to its low implementation complexity and well understood mathematical properties. However as the data grows performance of the k-means algorithm deteriorates, since is requires large number of iterations for every cluster and initial seedings.

GPUs have become one of the most popular computing platforms for high-throughput computing applications. GPUs are extremely conducive in the field of Data Science as is it possible to achieve high throughput by exploiting thousands of threads.In This project we discuss the k-means implementation on GPU and compare the results and speed up achieved with the serial implementation of K-means.

II. K-means clustering

K-means clustering is a method of partitioning data-set into k clusters based on a particular dimension. It proceeds by selecting K initial centroid values for each cluster. Each data value is then assigned to the cluster which has the closest distance from its centroid. These centroid values are updated by the mean of its constituent instance. Data-set are again classified into different clusters which has the closest distance from the updated centroids. This process of distance calculation and updating new centroids are repeated until there is no change in the new centroids.

K-means Algorithm

D be the Dataset that has to be clustered.
1. Let C1 . . . Ck be the initial cluster centers.
2. For each point xi in D, assign it to the closest cluster Cj.
3. For each cluster Ci, update its center by averaging all of the points xj that have been assigned to it.
4. Iterate between (2) and (3) until convergence.
5. Return C1 . . . Ck.

III. Implementation

In this project Image segmentation is performed to demonstrate K-means clustering. Dataset required to perform K-means clustering is extracted from image i.e. intensity values of every pixel in an image. The image is then clustered based on the intensity. As a part of this project, an iterative version of the algorithm was implemented. The algorithm takes a 2 dimensional image as input.It initializes random centroids with R G B intensities forming a pixel. For each pixel euclidean distance is calculated and a label is assigned to the nearest centroid. New centroid is calculated until convergence.

Algorithm 1: K-means clustering Algorithm

Input : Read the image with D dataset
Output: L
Initialize k random centroids with $R\ G\ B$ intensities forming a pixel.
repeat
  For each pixel of an image, calculate the Euclidean distance d, between the center and each pixel of an image.
  for all $\mathrm{x}_i \in D$
  Assign label to all the pixels to the nearest centre based on distance d.

$$\mathbf{L_i} = \quad arg\ min \|\mathbf{x_i} - \mathbf{C_k}\|^2$$

  for all $\mathbf{C_k} \in \mathbf{C}$
  Recalculate new centre.

$$\mathbf{C_k} = \frac{1}{n_k} \sum_{x_i \in C_k} x_i$$

until *Convergence*;

Although k-means has the great advantage of being easy to implement, it has some drawbacks. The quality of the final clustering result depends on the arbitrary selection of initial centroid. So if the initial centroid is randomly chosen, it will get different result for different initial centers. So the initial center has to be carefully chosen to get desire segmentation. And also computational complexity is another term which we need to consider while designing the K-means clustering. It relies on the number of data elements, number of clusters and number of iteration.

A. Sequential Implementation

Sequential implementation of K-means clustering was performed to analyze the computational complexity of the algorithm. Since there is only one master thread to perform clustering, computational intensity increases with increase in

dataset and clusters. The master thread iterates through every data-set to calculate euclidean distance,thus assign label of the closest centroid for every pixel. In the next stage centroids are updated by iterating over all labels partially calculating the new centroids. A k dimensional vector n is updated in each iteration where each component n_k holds the number of data points assigned to cluster C_k . Next another loop over all centroids is performed scaling each centroid C_k by n_k giving the final centroids. Convergence is also determined by checking whether the last labeling stage introduces any changes in the clustering.

B. Parallel Implementation on GPU

The CPU takes the role of the master thread. First seeds of centroids chosen by the CPU. Centroids and data points are uploaded to the GPU. Since the data points do not change over the course of the algorithm hence they are transferred only once. Master thread launches the GPU kernel called kernel_euclidean_distance.GPU threads act as the slave threads of the CPU, each GPU thread execute this kernel_euclidean_distance. The iterative process of calculating euclidean distance and labeling the nearest centroid to the centroid is performed by the GPU threads in parallel. The results from the labeling stage, namely the membership of each data point to a cluster in form of an index, are transferred back to the CPU. Finally the CPU calculates the new centroid of each cluster based on these labels and performs a convergence check. Convergence is achieved in case no label has changed compared to the last iteration.

IV. Limitation on exploiting GPU parallelism

The GPU is viewed as a set of multiprocessors executing concurrent threads in parallel. Threads are grouped into thread blocks and execute the same instruction on different data in parallel. One or more thread blocks are directly mapped to a hardware multiprocessor where time sharing governs the execution order. Within one block threads can be synchronized at any execution point. A certain execution order of threads within a block is not guaranteed. Blocks are further grouped into a grid, communication and synchronize among blocks is not possible,execution order of blocks within a grid is undefined. Threads and blocks can be organized in three and two dimensional manners respectively.A thread is assigned an id depending on its position in the block, a block is also given an id depending on its position within a grid. Thread and block id of a thread is accessible at run time allowing for specific memory access patterns based on the chosen layouts. Each thread on the GPU executes the same procedure known as a kernel. In this project calculation of euclidean distance can be can be performed by the GPU threads. Each thread calculates the euclidean distance and iterates through k cluster in order to label the nearest centroid. However updating a new centroid is via reduction.

Reduction operation requires threads to be synchronized to avoid the race conditions. However this does not ensure to have achieved complete parallelism since all the other blocks are waiting for the threads to synchronize. GPU also offers atomic functions, however they do not guarantee all of the blocks are running simultaneously.

V. Experimental Results

The sequential code and parallel code was executed for images lena.bmp and caterpiller.bmp. In order to observe the speed up and utilize GPU threads it is necessary to perform clustering on a huge file. Hence Caterpiller.bmp is used to observed the speedup. Fig 1 shows the experimental images used i.e lena.bmp and caterpiller.bmp. Fig 2 and 3 are the images obtained by CPU for K=2 and K=4. Fig 4 and 5 are the images obtained by GPU for K=2 and K=4. Table 1 depicts the execution time received by CPU and GPU for K=2,4,6. Although the speedup depends on initial seeding and number of iterations ran,but the execution time of GPU is way lesser on CPU.

TABLE I. Execution time for Caterpiller.bmp on CPU and GPU.

	N=2	N=4	N=6
CPU Execu. time	25934.9ms	19013.68ms	37407.47ms
GPU Execu. time	2968.87ms	9629.31ms	6213.29ms

Fig. 1. Following images were used to test a)lena b)caterpiller

Fig. 2. Results produced from sequential code for K=2

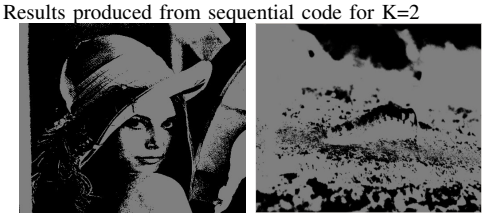


Fig. 3. Results produced from sequential code for K=4

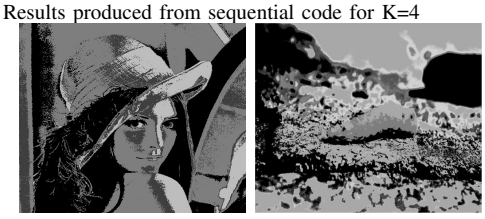


Fig. 4. Results produced from Parallel code for K=2

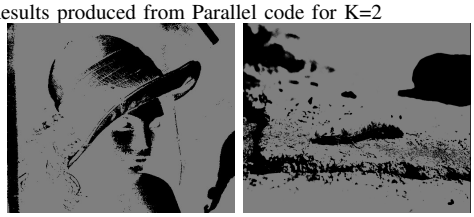
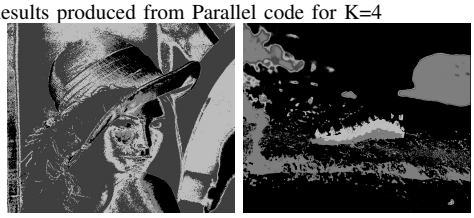


Fig. 5. Results produced from Parallel code for K=4



VI. Conclusion

Exploiting the GPU for the labeling stage of k-means proved to be beneficial especially for large data sets and high cluster counts. The presented implementation only performs a part of the algorithm in GPU hence it still has a scope of improvement. Many real-life data sets like document collections operate in very high dimensional spaces where document vectors are sparse.The implementation presented does not harvest all of the GPUs computational power.

References

[1] Suman Tatiraju,Avi Mehta. Image Segmentation using k-means clustering, EM and Normalized Cuts.

[2] Mario Zechner, Michael Granitzer.Accelerating K-Means on the Graphics Processor via CUDA.

[3] Nameirakpam Dhanachandra, Khumanthem Manglem and Yambem Jina Chanu.Image Segmentation using K-means Clustering Algorithm and Subtractive Clustering Algorithm *National Institute of Technology, Manipur 795 001, India.*

[4] http://www.mathworks.com/.

[5] Nvidia cuda site