CA Lab Project

Ishaan R Dharamdas
Department of Computer Engineering
National Institute of Technology Karnataka
Surathkal, Mangalore 575025
Email: ishaanrd6@gmail.com

M R Ashrit
Department of Computer Engineering
National Institute of Technology Karnataka
Surathkal, Mangalore 575025
Email: ashuabhi260898@gmail.com

I. Problem Statement

Accelerating K-Means on the Graphics Processor via CUDA

A. Description of K-Means clustering algorithm

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

The centroids of the K clusters, which can be used to label new data

Labels for the training data (each data point is assigned to a single cluster)

II. Project Execution Plan

Implement the K-means clustering algorithm in serial and parallel to determine the performance difference between the two methods.
Implementation will be done using the NVIDIA CUDA Architecture in C++. The input will be taken from a file and will consist of:

1. A set X of n data points $x_i \epsilon R^d$, i = 1, . . . , n.

2. Number of clusters k which belongs to the set of positive integers and is less than n.

A cluster C_j which is a subset of X, j = 1, . . . , k with a centroid $c_j \epsilon R^d$ is composed of all points in X for which c_j is the nearest centroid using euclidean distance.

The Output file will consist of k lines indicating the co-oridnates of the cluster center.

III. Project Timeline

31/10/18 - Finish implementation of the serial code for K-means clustering
09/11/18 - Finish implementation of the parallel code for K-means clustering
15/11/18 - Complete comparing the results and submit the project and relevant files

IV. Work Distribution

Both the members contributed to the project equally and shared all the work involved like gathering resources, input data sets for testing, and the algorithms and their implementation.

V. Objectives Achieved

1. The K-means clustering algorithm was implemented in normal c serially and the outputs were as desired. The code was run for different values of 'k' and gave appropriate results.
2. The algorithm was then implemented using the NVIDIA CUDA architecture to run on the GPU. The results were again as expected for different values of 'k'. It was observed that as k increased, the execution time relatively decreased which was desired.
3. Both the CPU and GPU implementation of the K-means clustering algorithm were run and compared. For small values of k, the sequential code ran faster than the parallel code but as k increased, the speedup by the parallel code dramatically increased which was expected.

VI. Results

Depending on the hardware, data set and the value of k, dramatic improvements were seen in the performance of the GPU implementation over the CPU implementation.
Here is a sample output for the same data set which was run on our institute's server. We can see that as k increases, the speedup given by the parallel code increases. For k = 128, the speedup is nearly 25x!

k	seqTime	cudaTime	speedup
2	0.2199s	0.4342s	.5x
4	0.0509s	0.3872s	.1x
8	0.2857s	0.3843s	.7x
16	1.1846s	0.4485s	2.6x
32	1.7239s	0.4122s	4.1x
64	4.6930s	0.4745s	9.8x
128	12.5834s	0.5165s	24.3x

VII. Algorithm

A. Sequential Algorithm

The following methodology was used for solving the K-means clustering algorithm.
An initial clustering C is created by choosing k random centroids from the set of data points X . This is known as the seeding stage. Next a labeling stage is executed where each data point $x_i \ \epsilon \ X$ is assigned to the cluster $C_j forwhichD(x_i, c_j)$ is minimal. Each centroid c_j is then recalculated by the mean of all data points $x_i \ \epsilon \ C_j$ via $c_j = 1/ \mid c_j \mid \Sigma_{x_j \ \epsilon \ C_j} x_i$.
The labeling and centroid update stage are executed repeatedly until C no longer changes. This procedure is known to converge to a local minimum subject to the initial seeding.

B. Parallel Algorithm

The labeling stage is identified as being inherently data parallel. The set of data points X is split up equally among p processors, each calculating the labels of all data points of their subset of X . In a reduction step the centroids are then updated accordingly. It has been shown that the relative speedup compared to a sequential implementation of k-means increases nearly linearly with the number of processors. Performance penalties introduced by communication cost between the processors in the reduction step can be neglected for large n.
Since the GPU is a shared memory multiprocessor architecture this section briefly outlines a parallel implementation on such a machine. Processors are now called threads and a master-slave model is employed. Each thread is assigned an identifier between 0 and t 1 where t denotes the number of threads. Thread 0 is considered the master thread, all other threads are slaves. Threads share some memory within which the set of data points X , the set of current centroids C as well as the clusters C_j reside. Each thread additionally owns local memory for miscellaneous data. It is further assumed that locking mechanisms for concurrent memory access are available.

C. Parallel K-means via CUDA

1) Program Flow: The CPU takes the role of the master thread. As a first step it prepares the data points and uploads them to the GPU. As the data points do not change over the course of the algorithm they are only transferred once. The CPU then enters the iterative process of labeling the data points as well as updating the centroids. Each iteration starts by uploading the current centroids to the GPU. Next the GPU performs the labeling The results from the labeling stage, namely the membership of each data point to a cluster in form of an index, are transfered 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. Optionally a thresholded difference check of the overall movement of the centroids can be performed to avoid iterating infinitely for some special cluster configurations.

D. Conclusion and Future Work

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 is only limited in the available memory on the GPU and therefore scales well. However, some drawbacks are still present. Many real-life data sets like document collections operate in very high dimensional spaces where document vectors are sparse. The implementation of linear algebra operations on sparse data on the GPU has yet to be solved optimally. Necessary access patterns such as memory coalescing make this a very hard undertaking. Also, the implementation presented is memory bound meaning that not all of the GPUs computational power is harvested. Finally, due to rounding errors the results might not equal the results obtained by a pure CPU implementation. However, our experimental experience showed that the error is negligible. Future work will involve experimenting with other k-means variations such as spherical or kernel k-means that promise to increase the computational load and therefore better suit the GPU paradigm. Also, an efficient implementation of the centroid update stage on the GPU will be investigated.

References

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