Laboratory Exercise 06

K-Means Clustering

Harshdeep Singh Chawla
Performed September 23rd, 2016
Submitted September 23rd, 2016

Instructor: Stephen Moskal
TA: Tejaswini Ananthanrayana

Lecture Section 06
Professor: Sonia Lopez Alarcon

HPA LAB 06

Abstract:

In this lab, we had to do classification of unknown data. For this purpose, we used K-Means algorithm. The K-means algorithm is a common algorithm which is used for finding optimal groupings of data into fixed number of clusters. In this lab, we classified the data into three clusters. The data used for the clustering was two-dimensional. The algorithm had two stages one was to assign the cluster to the data whose centroid distance was the shortest. This part was parallelizable. The other part was to calculate the new centroid after each iteration in the algorithm. This part was done sequentially. Constant memory was also used to store the values of centroids which remained constant for one cycle of kernel launch. The data was organized in clusters and can be seen in result section.

Design Methodology:

In this lab we were given to files. The main.cpp file generated the data set to be clustered and the other file was the header file. The algorithm for K-Means was implemented in the same way as it was mentioned in the lab manual. First each point's distance was measured from the cluster assigned to it and then from the other two clusters. And then the cluster which was the nearest to the point was assigned to the point. After checking this for all the points in the space, the new centroids were calculated by adding the x and y coordinates of all the points and dividing that by number of points present in that cluster. This gave us the new centroids values. The new centroid values were updated and the whole process was repeated till all the points were assigned to the cluster nearest to them. This was checked by the help of bool function (cluster[i]. altered). For repeating the process as many times as there was any alterations in the assignment of clusters, while loop was used.

When working on GPU, the same algorithm was used. Since in this lab we were using constant memory, there was data which was needed to be transferred to constant memory from global memory. For this purpose we made use of cudaMemcpyToSymbol() function. The values of centroid were stored in the constant memory. The kernel was only responsible for assigning the correct cluster to each point in the space. The new centroids were calculated and were updated sequentially.

HPA LAB 06

Result: The Results for this lab are as follows:

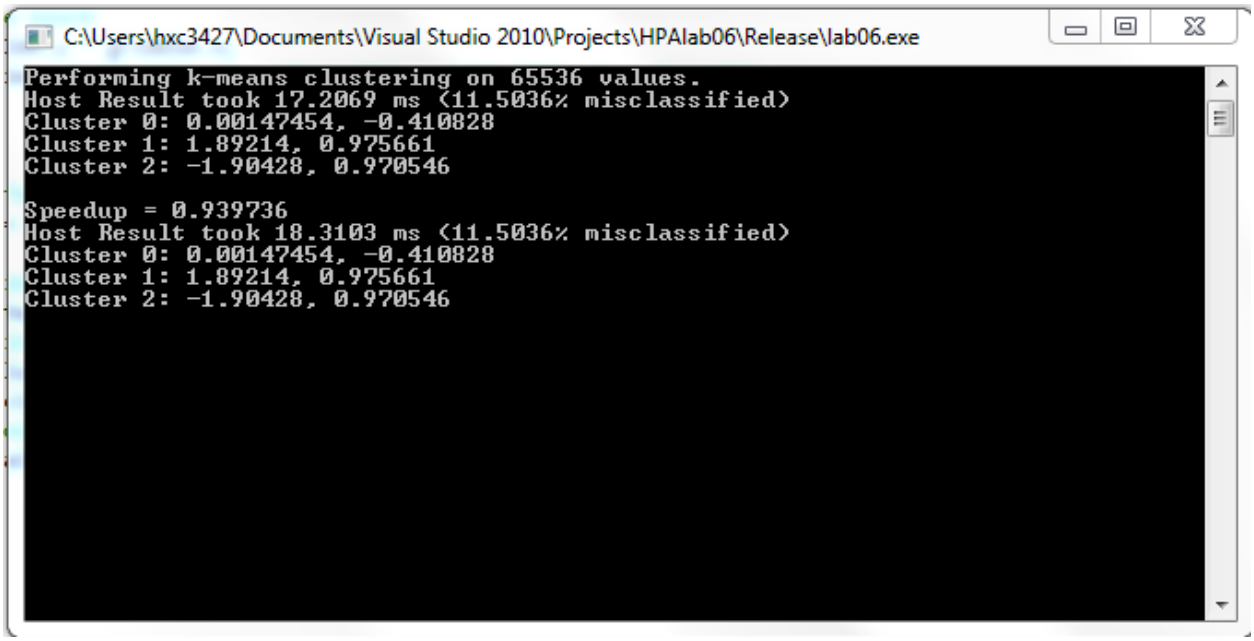


Figure: Sample Output.

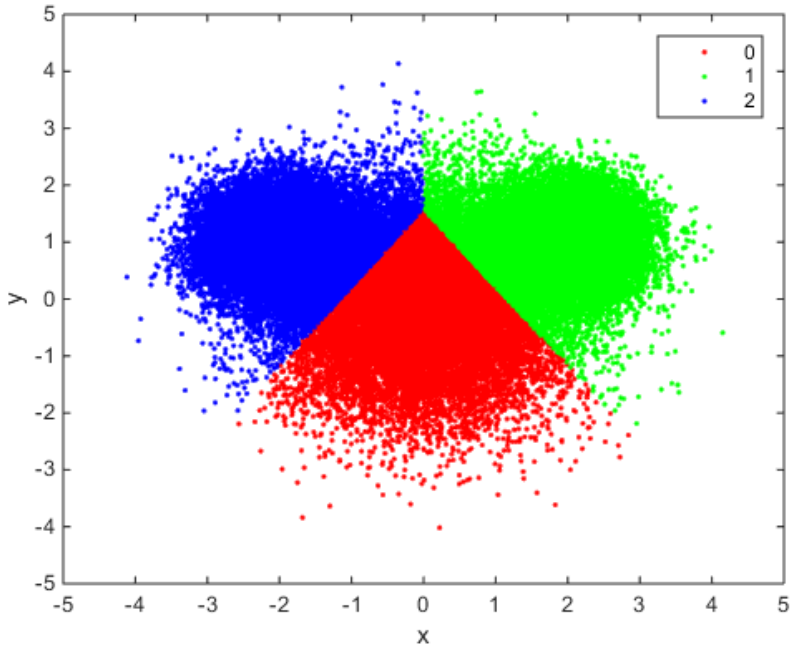


Figure: K-Means Clustering done by CPU for 65536 points in space.

HPA LAB 06

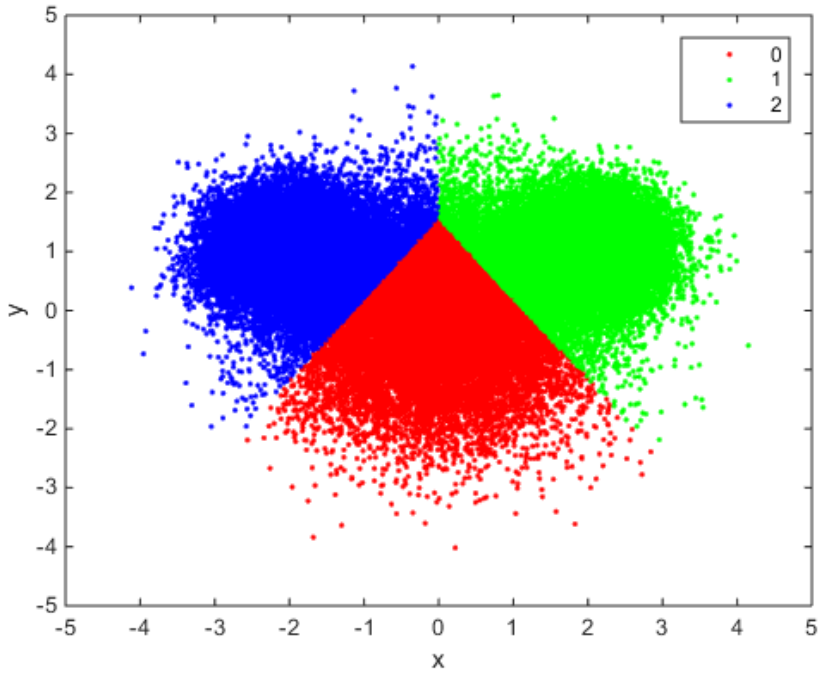
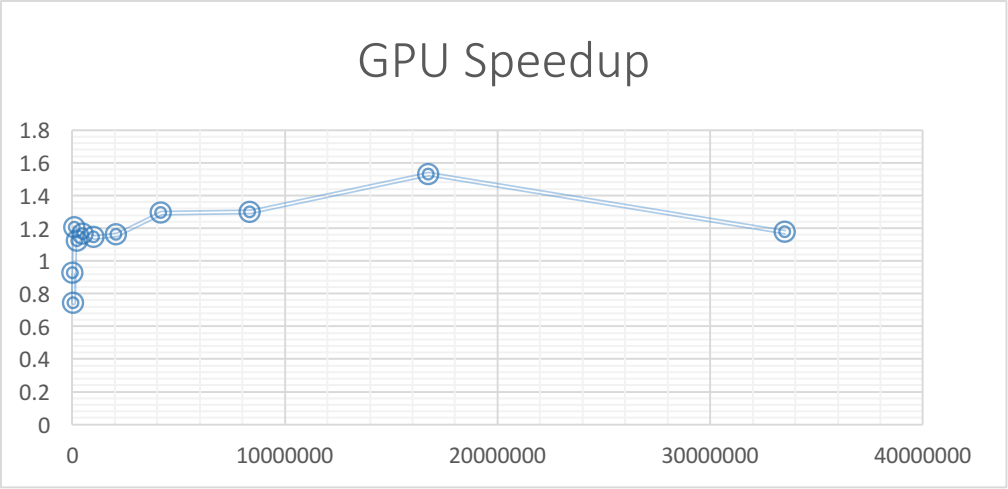


Figure: K-Means Clustering done by GPU for 65536 points in space.

Data Size	CPU Time (ms)	GPU Time (ms)	GPU Speedup	% Misclassified
32768	8.96552	9.68966	0.925267	11.3617
65536	17.5172	18.6552	0.743666	11.5036
131072	33	27.4483	1.20226	11.4357
262144	82.2759	73.3448	1.12177	11.5185
524288	160.931	138.069	1.16558	11.5547
1048576	367.414	320.897	1.14496	11.5608
2097152	867.207	747.655	1.1599	11.5922
4194304	1897.72	1466.83	1.29376	11.5828
8388608	3614.83	2784.1	1.29838	11.5779
16777216	8988.24	5878.97	1.52888	11.5828
33554432	14497.8	12338.4	1.17501	11.5679

Table: All the observations that were taken during the lab.



Graph: GPU Speedup vs Number of points in space

Conclusion:

The objective of the lab was to perform K-Means Clustering on a randomly generated data by the means of CPU and GPU and compare their performance. As we can see in the result section the GPU performs better when the number of points in the 2-D space increases. But this speed up is small as compared to what we got in other labs. This is due to the fact that some part of calculation for GPU is done by CPU. For this to happen there was data transfer happening between the CPU and GPU. Which cost a lot of time. We made use of constant memory for this lab. Constant memory was helpful for storing the values of centroid. As for per iteration the centroids remained constant. The performance graph of GPU speedup vs number of elements ca be seen in result section. And this shows that the algorithm was parallelizable.

Questions:

Q1:

You used constant memory as part of this implementation. Why does it make sense to use constant memory for the clusters? What is the maximum number of clusters that you could store in constant memory?

It makes sense to use constant memory for the clusters because, for a particular iteration the values of the clusters are same. And hence using constant memory will save a lot of global memory access. The maximum number of clusters that we can store in constant memory is 2665. Since single cluster takes 24bytes of memory and we have 64kb of constant memory.

Q2: Could you have used constant memory for the data points as well? Make sure that your answers are **very** specific and include concrete numbers to back up your claim.

Yes, we can use constant memory for the data points. But using constant memory would limit the number of elements in our data set highly. We have limited amount of constant memory (in GTX 480 it is 64kb). So, if the value of data is very large we cannot use constant memory. Each point in a cluster has 8 bytes of space. So, the total number of points that we can have in our data set is limited to 8000.