IT506

Accelerated Computing

COURSE PROJECT
K MEANS CLUSTERING

Instructor:
Dr.Bhaskar Chaudhary

Author 1 :
Bhat Shripad Anant
201911003

Author 2:
Soni Maulik Rameshbhai
201911009

November 11, 2019

Contents

1 Problem Description

K-means clustering is one of the most popular unsupervised machine learning algorithms. K-means clustering as the name suggests creates k clusters of data points from the dataset. K means clustering assumes that data points in the same neighborhood belong to the same cluster.

As the size of the dataset increases, the time required to cluster the data increases and the performance also decreases significantly in case of serial computation. Hence there is a need to convert the serial K-means algorithm to a parallel algorithm so as to improve the performance and the computation time.

2 K-means Clustering Algorithm

For every cluster a centroid is computed by taking the mean of all the data points belonging to the cluster, hence the word 'means' in K-means clustering. K refers to the number of such clusters in the dataset. Each data point is assigned to a cluster based on the distance of the data point from the centroids of each clusters. If the centroid of a cluster is closest to the data point, then the data point is allocated to that particular cluster.

K-means clustering is an iterative algorithm which consists of below 4 steps.

1. Initialise k centroids for each of the k clusters randomly.
2. Compute the distance of all the data points with each of the k centroids and assign the data point to the cluster with closest centroid.
3. Compute the new centroid for each cluster by taking mean of all data points assigned to the cluster in the previous step.
4. If the new centroids are same as the previous centroids, algorithm has converged else go to step 2.

3 Scope of Parallelism

In K-means clustering algorithm there are 2 major tasks. First task is to compute distance between the centroids and the data points and assign the points to one of the clusters. Second task is to compute the centroids of the clusters formed in the previous task.

Computation of the distance and assigning a data point to a cluster can be done independently for each data point. So the first task can be completely done in parallel.

For the second task we need to find the mean of all the points in each cluster, and in these case sum of points in each cluster can be computed in parallel.

4 Theoretical Analysis

Serial K-Means Clustering
If N is the number of data points to be grouped into K clusters, and each data point is of D dimensions and takes I iterations, then the time complexity is *O(N*K*D*I)*

Parallel K-Means Clustering
If N is the number of data points to be grouped into K clusters, and each data point is of D dimensions and takes I iterations, then the time complexity for distance computation and assigning to clusters(assuming N processors) is *O(K*D*I)*

For calculating centroids by computing mean of all the clusters, the time complexity is *O(log(N*K*D)*I)*

So the total parallel time complexity is *O(K*D*I+log(N*K*D)*I)*

$$SpeedUp = \frac{O(N*K*D*I)}{O(K*D*I+log(N*K*D)*I)} = \frac{O(N)}{O(log(N))}\; if N >> I, K, D$$

$$CGMA = \frac{No.of Computations}{No.of Memory Accesses} = \frac{O(N)}{O(N)} = 1$$

5 Parallelization Strategy

As discussed in the section scope of parallelism, there are two major tasks in K-means clustering that can be parallelized. Hence there are 2 CUDA kernels each performing one of these tasks.

Kernel-1 is doing the following operations for each data point parallely.

- Compute distance between each data point and all the cluster centroids.
- Compare the distance between the point and the cluster centroids of all clusters and assign to the closest cluster.
- Count the number of data points that are assigned to each cluster.

Kernel-2 is performs summation of all points in each cluster using reduction.

Final step is to compute the new centroids by dividing the sum of all points in each cluster by the number of points in each cluster calculated in Kernel-1.

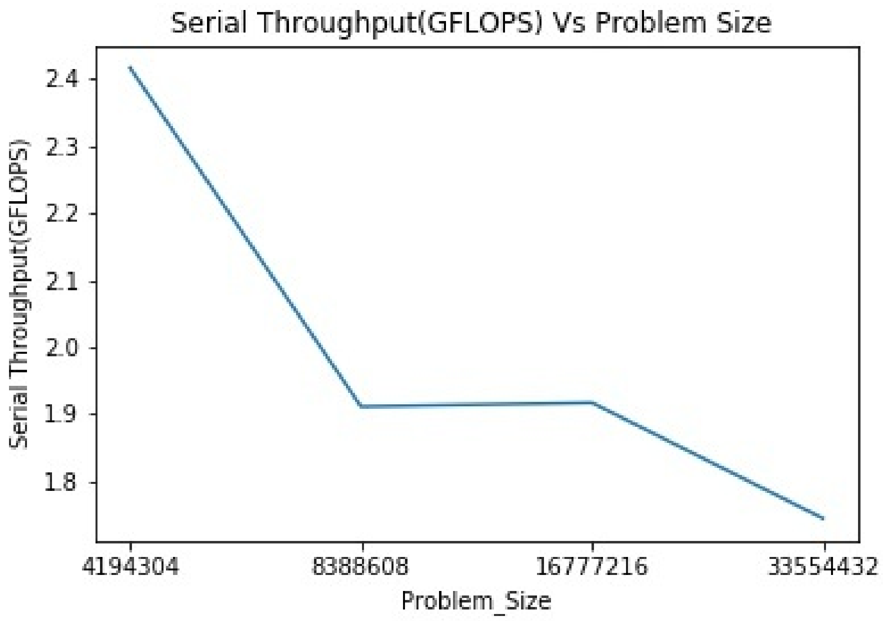
6 Optimizations

- Reduced control divergence in Kernel-1 by replacing nested if else blocks by ternary operator.
- Implemented privatization to calculate number of cluster points using shared memory and atomic operations in kernel-1.
- Implemented reduction in kernel-2 by using shared memory and sequential addressing to eliminate bank conflict and reduce control divergence.
- Reduced data transfer from CPU to GPU and vice versa by performing in place reduction in kernel-2.

7 Serial K-Means Clustering

Serial K-Means Clustering is performed on two dimensional points of size 4194304 to 33554432, and the corresponding time to execute and throughput is recorded.

Problem Size	**Time(ms)**	**Throughput(GLFLOPS)**
4194304	1500	2.4591
8388608	3370	1.91168
16777216	6720	1.91736
33554432	22150	1.74512

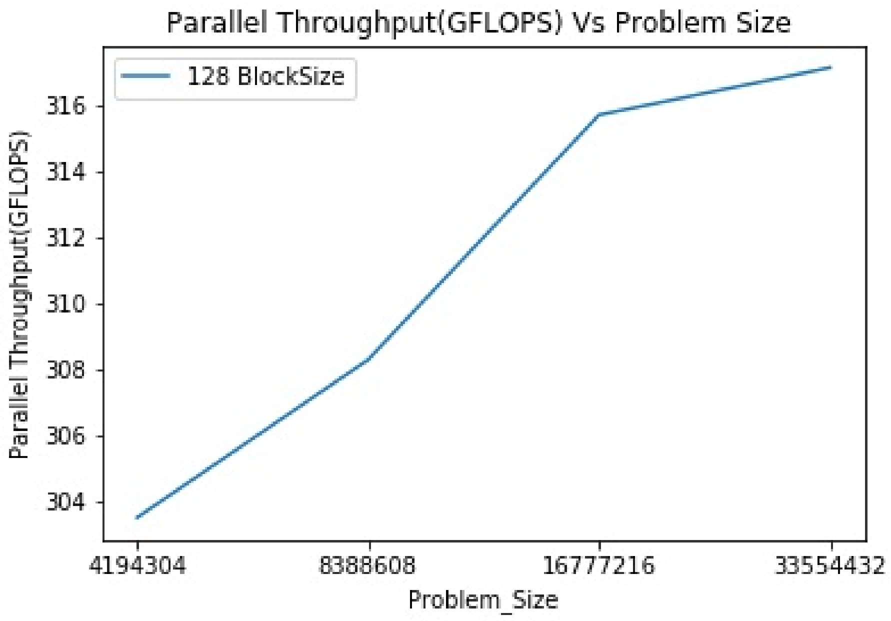


We can observe from the above table and the plot that the throughput is decreasing as the problem size is increasing. Reason for this behaviour is the increase in time for higher problem sizes. More time is spent on accessing the main memory as the cache sizes are exceeded more frequently in higher problem sizes.

8 Parallel K-Means Clustering

Parallel K-Means Clustering is performed on two dimensional data points of size 4194304 to 3355443, for 128 block size which gives the best time and throughput.

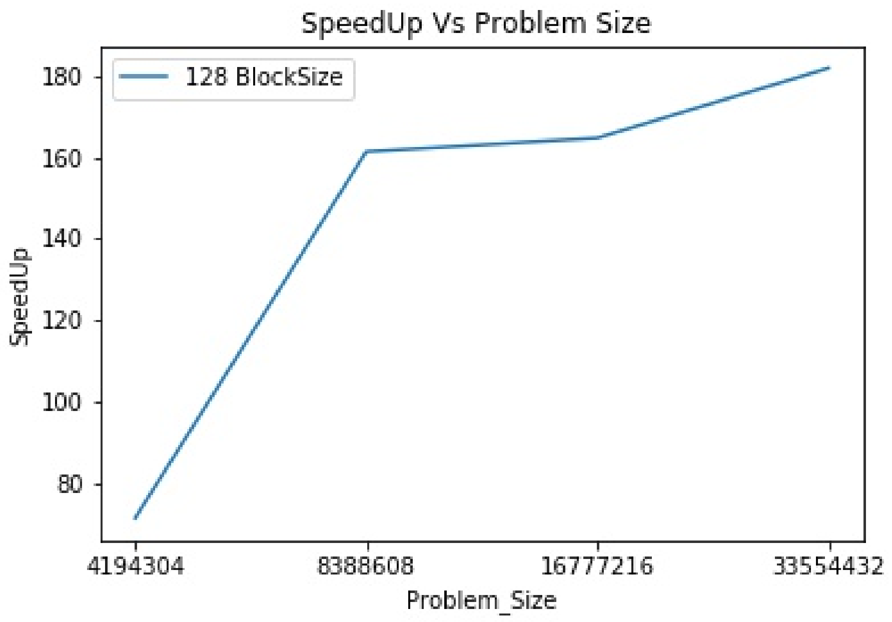
Problem Size	Time(ms)	Throughput(GFLOPS)
4194304	15.919392	303.51
8388608	20.897951	308.28
16777216	40.815201	315.68
33554432	121.89593	317.11



In case of Serial K-Means clustering we observe a decrease in throughput as the problem size increases, but in case of Parallel K-Means clustering throughput increases as the problem size increases.This is due to the fact that as the problem size increases the utilization of GPU Hardware increases due to large amount of computations that are done in parallel.
Also block size of 128 gives the best performance because there are 16 blocks in each MP which is the maximum allowed for a MP as per the GPU specification.

9 Performance Analyis

Speed Up Analysis
Speed Up plot for all the problem sizes is as shown below.



Speed Up increases as the problem size increases and the maximum value achieved is 180.

Below Analysis for time taken for problem size 33554432 for parallel execution.
Memory bandwidth of GPU is 288Gb/s and bandwidth between CPU and GPU is 32Gb/s(PCI3.0x16) and peak throughput of Tesla k40c is 1.43TFLOPS. Memory Access time assuming the above values of bandwidth is 6.94ms for kernel-1 and 5.2ms for kernel-2. Computation time for both the kernels combined is 4.2ms.
Further for calculating the number of cluster points atomic operations are performed for which 105.77ns time is consumed. So the total calculated time is approximately 49 ms for 3 iterations and the observed time is 121ms.

Throughput Analysis
Peak throughput for the GPU is 1.43TFLOPS and we achieved a maximum throughput of 0.317TFLOPS which is 22.16 percent of the peak throughput of the GPU.This is due to the fact that while performing reduction in kernel-2 more and more threads get inactive and the hardware is underutilised. This was verified by using Nvidia profiling tool output as shown below.

Kernel-2
Device Tesla K40c

Kernel	Average Multiprocessor efficiency
distanceCalculation	99.89 %
sumCluster	73.07 %

Another reason for achieved throughput to be lesser than the peak throughput is due to stalls happening in the kernels when syncthreads() is used.

Kernel	Average Stalls
distanceCalculation	11.29 %
sumCluster	4.58 %

There is control divergence in both kernels which leads to serial execution of those threads which also leads to the decrease in throughput.

Atomic operations are used for calculating the number of cluster points in kernel-1 which also leads to serial execution of threads and leads to decrease in throughput.

NVIDIA PROFILING OUTPUT

```
Time(%) Time     Calls  Avg       Min       Max       Name
56.61%  320.50ms   8  40.063ms   960ns    320.50ms  [CUDA memcpy HtoD]
27.00%  152.89ms   3  50.964ms  48.219ms  56.430ms  computeDistance
16.39%  92.798ms   9  10.311ms  5.3760us  30.892ms  sumCluster
0.00%   18.592us   12  1.5490us  1.3440us  1.7920us  [CUDA memcpy DtoH]
```

10 Conclusion

We observe that Parallel K-Means Clustering is efficient in terms of time required to compute the clusters as compared to the Serial K-Means clustering specially in case of large problem sizes. We achieved a significant Speed Up of 180 and throughput of 0.3TFLOPS.

11 Further Improvements

Atomic operations are used to compute number of cluster points by adding the number of points in a cluster from each block. But reduction can be used to achieve the same and use the GPU more efficiently.