PARALLELIZING K-MEANS CLUSTERING USING GPU

Harshal Arun Bhavsar

M.E. Computer Engineering Student, Matoshri College of Engineering and Research Center, Maharashtra, India

ABSTRACT

The system with an optimized k-means implementation will be performed on the graphics processing unit (GPU). NVIDIA's Computing Unified Device Architecture (CUDA), available from the G80 GPU family onwards, is used as the programming environment. Emphasis is placed on optimizations directly targeted at this architecture to best exploit the computational capabilities available. Clustering involves partitioning a set of objects into subsets called clusters so that objects in the same cluster are similar according to some metric. Clustering is widely used in many fields like machine learning, data mining, pattern recognition and bioinformatics.

K-means clustering is very popular clustering method used which uses distance as the similarity measure. K-means algorithm uses a set of K random objects from the available data set and performs distance computations. K-means chooses multiple models with the goal of refinement and faster convergence. The focus of the proposed system is to investigate different approaches to parallelism and then ensemble K-means algorithm on modern many core hardware. The many core hardware involves GPUs and CPUs backed by the CUDA software stack. The main feature of the proposed system is that it makes use of the large computing capacity of the hardware by minimizing the amount of data access.

Keyword: - *GPU, CPU, K-Means, CUDA.*

1. INTRODUCTION

1.1 Clustering

Clustering can be considered the most important unsupervised learning problem. So, as every other problem of this kind, it deals with finding a structure in a collection of unlabelled data. A cluster is therefore a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters. Clustering is very crucial field of research having many real time applications. These applications include Data Mining, Pattern recognition, Image analysis, Bio-informatics, Machine Learning, Voice mining, Image processing, Text mining, Whether report analysis etc.

1.2 K-Means Clustering

K-means is a typical clustering algorithm and it is widely used for clustering large sets of data. A good clustering method produces high-quality clusters to ensure that the inter-cluster similarity is low and the intra-cluster similarity is high, in other words, members of a cluster are more like each other than they are like members of a different cluster.

1.3 GPU

A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics and image processing, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms

where processing of large blocks of data is done in parallel. In a personal computer, a GPU can be present on a video card, or it can be on the motherboard or in certain CPU's on the CPU die.

1.4 CUDA

CUDA stands for Compute Unified Device Architecture, it is a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce. CUDA gives developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs.

Using CUDA, the GPUs can be used for general purpose processing (i.e., not exclusively graphics); this approach is known as GPGPU. Unlike CPUs, however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly.

2. RELATED WORK

S.A. Arul Shalom et al. [1] present an efficient implementation of the k-means clustering algorithm in the GPU. They realize this by using the multipass rendering and multi-shader capabilities of the CPU. This is done by maximizing the use of textures and minimizing the use of shader program constants. In this implementation they have minimized the use of GPU shader constants thus improving the performance as well as reducing the data transactions between the CPU and the GPU. Handling data transfers between the necessary textures within the GPU is much more efficient than using shader constants. This is mainly due to the high memory bandwidth available in the GPU pipeline. Since all the steps of k-means clustering could be implemented in the GPU, the transferring of data back to the CPU during the iterations is avoided. The programmable capabilities of the GPU have been thus exploited to efficiently implement k-means clustering in the GPU. Implementation is done using OpenGL as the Application Programming Interface (API), and the operational kernels are invoked via shader programs, using the Graphics Library Shading Language (GLSL). The Single Instruction Multiple Data (SMID) technique is employed to achieve data or vector level parallelism in the fragment processor.

Reza Farivar et al. [2] have used two types of data structures. They have implemented the CUDA-accelerated k-means algorithm in three distinct stages of operation. The 1st stage initializes the CUDA hardware, allocates the appropriate host and device memory storage areas, estimates the initial set of centroids and loads the data set into the on-board memory of the graphics card. The second part, which is the workhorse of the program, is the kernel running on the GPU device. Each thread will process a single data point, and compute the distance between the point and each centroid. The third part of the program relabels points to the nearest centroid, and computes the next centroid estimation. This part is executed serially in the host. The data set consists of a one dimensional array of unsigned long integers, which are 4 bytes in size per point.

BAI Hong-Tao et al. [3] have proposed a novel Single Instruction Multiple Data (SIMD) architecture processors (GPUs) based k-means algorithm. In this algorithm, in order to accelerate compute-intensive portions of traditional k-means, both data objects assignment and k centroids recalculation are off-loaded to the GPU in parallel. They have implemented this GPU-based k-means on the newest generation GPU with Compute Unified Device Architecture (CUDA). The numerical experiments demonstrated that the speed of GPU-based k-means could reach as high as 40 times of the CPU-based k-means.

Mario Zechner et al. [4] have proposed the algorithm which realized in a hybrid manner, parallelizing distance calculations on the GPU while sequentially updating cluster centroids on the CPU based on the results from the GPU calculations. 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 labelling 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 labelling. The results from the labelling 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. Optionally a threshold difference check of the overall movement of the centroids can be performed to avoid iterating infinitely for some special cluster configurations.

Ren Wu et al. [5] have proposed parallel k-means clustering using GPUs to accelerate clustering of very large data sets. They investigate if GPUs can be useful accelerators even with very large data sets that cannot fit into GPUs on-board memory. They have used MineBench as their baseline for performance comparison and used randomly generated data sets. They have introduced Multi-tasked Streaming for large datasets

Kai J. Kohlhoff et al. [6] have proposed how K-means can be fully implemented on massively parallel general purpose computing platforms without putting limits on the number of data points, clusters, or dimensionality of the data other than the available GPU main memory, while avoiding thread divergence and maintaining near-optimal GPU occupancy.

3. DETAILS OF PROPOSED SYSTEM

The focus of the proposed work is to show the advantages by performing K-Means algorithm on GPU for optimization. Clustering operations performed as fundamental tools in many applications hence to increase speed of application require reducing execution time of Clustering algorithm.

3.1 Basic Building Block of System

The Figure below shows the architectural view of proposed system and flow of execution.

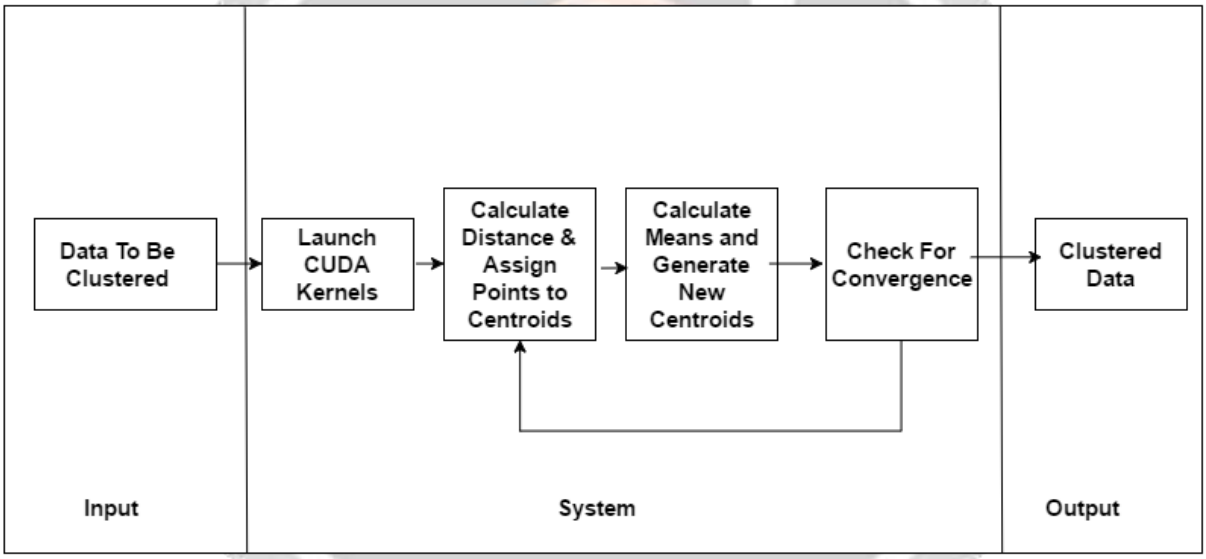

Fig -1: Architecture of Proposed System.

3.1.1 Calculate Distance

Distance calculation is the first step of K-Means clustering. In this step we calculate distance between each point and all the cluster centroids. So for example we have N points and K cluster centroids then, The distance from each of N points i.e ($n_1, n_2, n_3, \ldots, n_n$) to each of K centroids i.e ($k_1, k_2, \ldots, k_n$) will get calculated.

3.1.2 Assign Points to Centroids

Centroid assignment is Second step of K-Means clustering. After Calculation of distance step finishes the data is passed for centroid assignments. In these step N points gets assigned to K cluster centroid points on the basis of the distance calculated in previous step. This step will create new clusters after completion of point assignment.

3.1.3 Calculate Mean Value and Generate New Centroid

Once all the points get assigned to their respective closest centroid and new clusters get generated. The next step is to calculate the mean of cluster and make it as new centroid of that cluster.

3.1.4 Assign Points to Centroids

After new centroid calculation gets completed the last step is to check if the centroids from last iteration and this iteration differ. If there is difference in the centroids then we have to repeat all above steps by calling Calculate Distance step. If the centroids are same then we have to stop. And copy output data to disk.

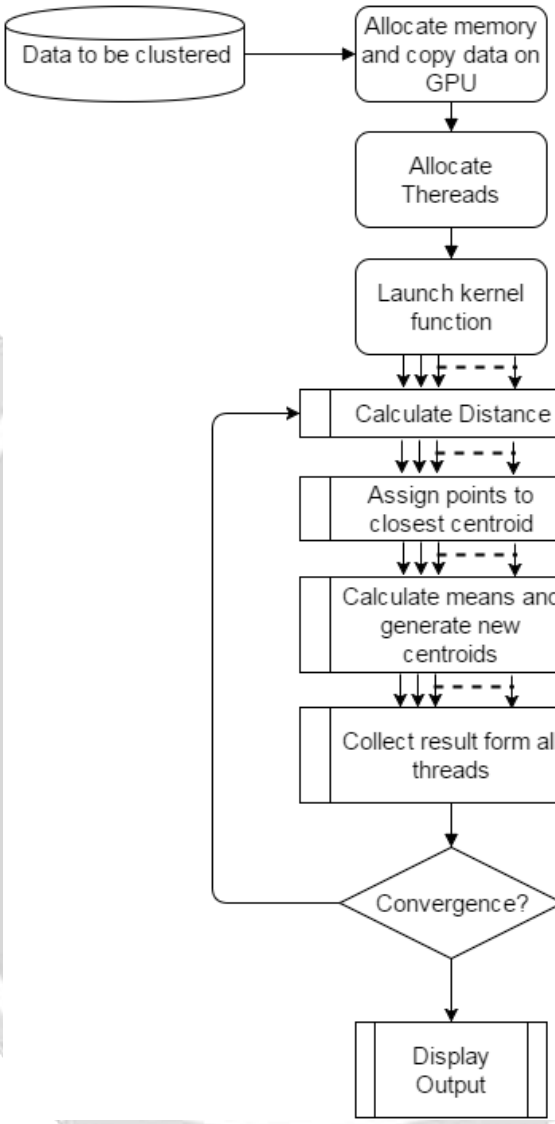

Fig -2: Flow diagram of Proposed System.

3.2 Optimization using Shared Memory

The shared memory is read-and-write memory that resides physically on the GPU. It is placed as opposed to off-chip DRAM, so it is much faster than global memory. The threads in only one block are allowed to access shared memory. Threads in one block do not have access to shared memory in different blocks. This type of memory provides an excellent speed-up because threads in one block communicate with each other and use share memory. In the system, K-means algorithm use shared memory for optimization. In this system processing data is stored on to shared memory. Shared memory has less latency than global memory. Data is stored in shared memory by using shared keyword. At the time of processing algorithm, accessing data from shared memory by simply call variable name.

3.3 Thread Control

Graphics processing unit is based on the thread level parallelism to maximum exploitation of its function units where each processor on GPU works as single instruction multiple data. For obtaining this, kernel function is call from host and execute on GPU device. Here computation is done by large number of threads, organized in thread blocks. To achieve high performance require GPU as active as possible [20].

4. Experimental Result

Experiments are performed On Intel Core i3 machine with Nvidia gpu is used with CUDA 6.0 and Ubuntu 12.04 linux distribution. The performance behavior is studied depending on change in number of clusters. The performance of K-Means is also recorded for parallel implementations. The results are shown in form of Tables and Charts as follows.

4.1 The results when number of cluster = 10

Table -1: Results with Number of Clusters = 10

Number of Points	Base System	Proposed System
100	3.867 ms	4.382 ms
1000	13.943 ms	12.687 ms
5000	116.315 ms	57.766 ms
10000	483.94 ms	107.078 ms
50000	4221.0 ms	210.993 ms
100000	13315 ms	359.776 ms

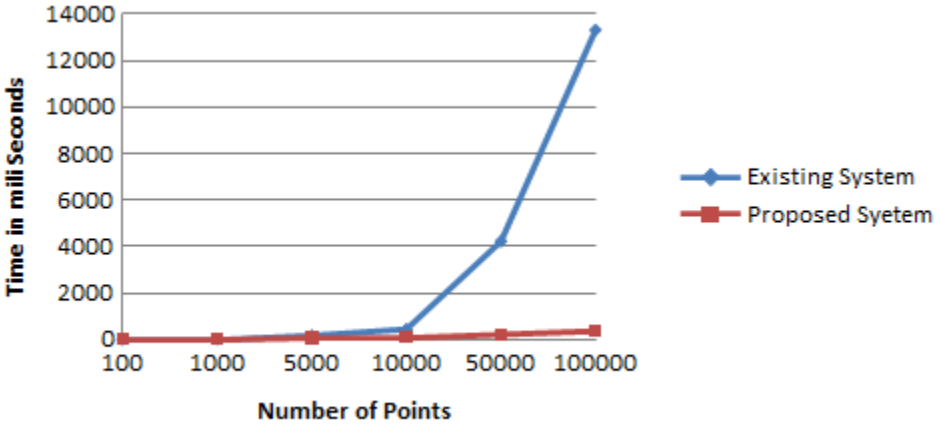

Fig -1: Line Chart Showing Results with Number of Clusters = 10

4.2 The results when number of cluster = 20

Table -2: Results with Number of Clusters = 20

Number of Points	Base System	Proposed System
100	15.67 ms	22.095 ms
1000	130.883 ms	37.412 ms
5000	924.227 ms	115.035 ms
10000	3454 ms	282.417 ms
50000	41012 ms	636.716 ms
100000	77712 ms	1220 ms

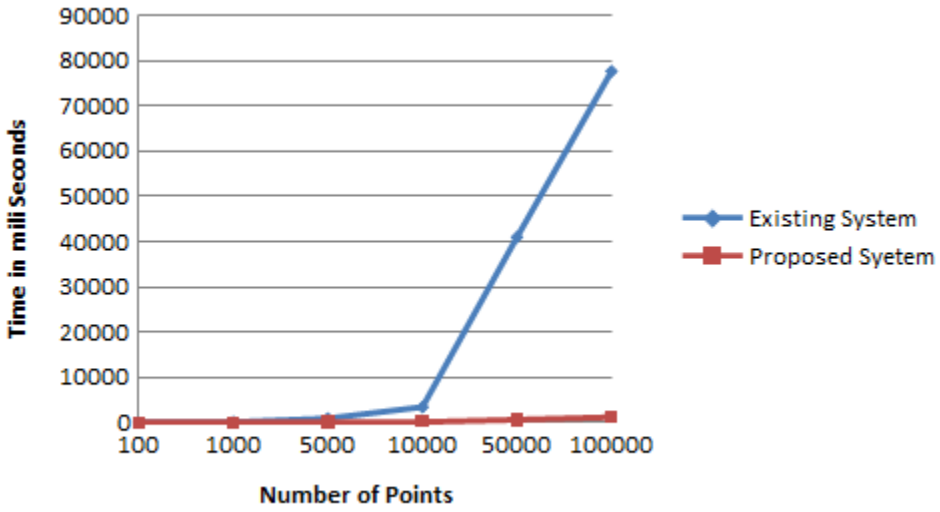

Fig -2: Line Chart Showing Results with Number of Clusters = 20

5. CONCLUSIONS

This system provides a parallel approach for clustering algorithm that is K – Means algorithm. In this system the fully parallel k-means clustering algorithm is proposed. As the number of cluster and data points increase it gives better performance. This implementation wok effectively for energy efficient and low cost system enabled with GPU card for high speed up. The efficient use of all the key feature of CUDA to gain significant performance is done. Here, comparative study between proposed system and existing system is done. From the above results we conclude that, proposed implementation of K-Means algorithm is comparatively fast in execution speed.

Every time a new GPU is introduced with improved Computational features, the horizon further advances. As future work, this application can be ported to multiple GPU devices that will run in parallel. As the number of GPU cards used increases, a proportional speed up of the application is expected. Also we can implement parallelism in other popular algorithms in different areas.

6. ACKNOWLEDGEMENT

The author wish to thank Matoshri college of Engineering and Research Centre Nasik, HOD of Computer Engineering Department, Guide and Parents for supporting and motivating for this work because without their blessing this was not possible.

7. REFERENCES

[1]. Shalom, S. A., Dash, M., and Tue, M. (2008). Efficient k-means clustering using accelerated graphics processors. In Data Warehousing and Knowledge Discovery (pp. 166-175). Springer Berlin Heidelberg.
[2]. Farivar, R., Rebolledo, D., Chan, E., and Campbell, R. H. (2008, July). A Parallel Implementation of K-Means Clustering on GPUs. In PDPTA (pp. 340-345).
[3]. Hong-Tao, B., Li-li, H., Dan-tong, O., Zhan-shan, L., and He, L. (2009, March). K-means on commodity gpus with cuda. In Computer Science and Information Engineering, 2009 WRI World Congress on (Vol. 3, pp. 651-655). IEEE.
[4]. Zechner, M., and Granitzer, M. (2009, April). Accelerating k-means on the graphics processor via cuda. In Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on (pp. 7-15). IEEE.
[5]. Wu, R., Zhang, B., and Hsu, M. (2009, May). Clustering billions of data points using GPUs. In Proceedings of the combined workshops on Unconventional high performance computing workshop plus memory access workshop (pp. 1-6). ACM.
[6]. Kohlhoff, K.J.; Pande, V.S.; Altman, R.B., "K-Means for Parallel Architectures Using All-Prefix-Sum Sorting and Updating Steps," Parallel and Distributed Systems, IEEE Transactions on , vol.24, no.8, pp.1602,1612, Aug. 2013 ,doi: 10.1109/TPDS.2012.234.
[7]. K.J. Kohlhoff, M.H. Sosnick, W.T. Hsu, V.S. Pande, and R.B. Altman, "CAMPAIGN: an Open-Source Library of GPU-Accelerated Data Clustering Algorithms," Bioinformatics, vol. 27, no. 16, pp. 2322-2323, doi:10.1093/bioinformatics/btr386, 2011.
[8]. S. Che, J. Meng, J.W. Sheaffer, and K. Skadron, "A Performance Study of General Purpose Applications on Graphics Processors," Proc. First Workshop General Purpose Processing on Graphics Processing Units, 2007.
[9]. R. Wu, B. Zhang, M. Hsu, and Clustering, "Billions of Data Points Using GPUs," Proc. Combined Workshops Unconventional High Performance Computing Workshop Plus Memory Access Workshop (UCHPC-MAW '09), 2009, doi: 10.1145/1531666.1531668.
[10]. T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp.103-114, 1996, doi:10.1145/235968.233324.
[11]. R.T. Ng and J. Han, "CLARANS: A Method for Clustering Objects for Spatial Data Mining," IEEE Trans. Knowledge Data Eng., vol. 14, no. 5, pp. 1003-1016, Sept./Oct. 2002.
[12]. M. Ester, H-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," Proc. Second Int'l Conf. Knowledge Discovery Data Mining (KDD '96), pp. 226-231, 1996.
[13]. J.M. Engreitz, B.J. Daigle Jr., J.J. Marshall, and R.B. Altman, "Independent Component Analysis: Mining Microarray Data for Fundamental Human Gene Expression Modules," J. Biomedical Informatics, vol. 43, pp. 923-944, 2010, doi: 10.1016/j.jbi.2010.07.001.
[14]. M. Harris, "Optimizing Parallel Reduction in CUDA" Nvidia CUDA SDK Whitepaper, http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf, 2007.
[15]. W.D. Hillis and G.L. Steele Jr., "Data Parallel Algorithms," Comm.ACM, vol. 29, no. 12, pp. 1170-1183, Dec. 1986, doi:10.1145/7902.7903.
[16]. M. Harris, S. Sengupta, and J.D. Owens, "Parallel Prefix Sum (Scan) with CUDA," GPU Gems 3, H. Nguyen, ed., Pearson Education, 2007.
[17]. M.P. Liang, D.R. Banatao, T.E. Klein, D.L. Brutlag, and R.B. Altman, "WebFEATURE: An Interactive Web Tool for Identifying and Visualizing Functional Sites on Macromolecular Structures," Nucleic Acids Research, vol. 31, no. 13, pp. 3324-3327, 2003.
[18]. S.P. Lloyd, "Least Squares Quantization in PCM," IEEE Trans. Information Theory, vol. IT-28, no. 2, pp. 128-137, Mar. 1982.
[19]. SimTk, "The Simulation Toolkit, Part of the Simbios Project," https://simtk.org, 2010.
[20]. N Wilt. The CUDA Handbook: A Comprehensive Guide to GPU Programming. Addison-Wesley Professional.
[21]. Wikipedia-CUDA http://en.wikipedia.org/wiki/CUDA
[22]. CUDA Zone. Official webpage of the nvidia cuda api. Website, http://www.nvidia.com/object/cuda home.html