Review for K-Means On Graphics Processing Units (GPU)

Piyush P. Baramkar,
M.Tech. Information Technology,
Walchand College of Engineering,
Sangli, India.

Dr. D. B. Kulkarni,
H.O.D. Information Technology,
Walchand College of Engineering,
Sangli, India.

***Abstract*—K-Means is the most popular algorithm used in data mining for clustering. The size and dimensions of data sets has increased tremendously due to daily transactions and internet. Recently various cost effective parallel architectures are emerged such as GPU with CUDA (Compute Unified Device Architecture). A significant speedup is achieved when general purpose applications are implemented on GPU using CUDA. Several implementations are available for K-Means on GPU for small dimensions. All features of CUDA can significantly use for improving speedup like coalesced memory access and thread divergence.**

Keywords—Clustering, K-Means, GPU, CUDA, Data Mining, Hybrid programming/architecture.

I. INTRODUCTION

Clustering is unsupervised learning method that partitions a data set of data objects into clusters, such a way that intra-cluster shows maximum similarity while inter-cluster shows minimum similarity. K-Means is one of the most popular data mining clustering algorithms used now a days and is widely used in variety of fields such as pattern recognition, statistical data analysis, bioinformatics and image analysis. It has been chosen as one of the Top 10 data mining algorithm. The huge data set contains large number of data objects with large number of dimensions takes more running time with K-Means algorithm for clustering. Hence clustering for large scale data sets is usually a time-consuming task [1].

Recently, Graphics Processing Units (GPUs) develop continuously as a general-purpose high performance parallel hardware and provide promising platform for parallelizing K-Means. GPUs are provided as a dedicated hardware for manipulating computer graphics. GPUs have evolved into highly parallel many-core processors, due to the demand of huge computing power for real-time and high-definition 3D graphics. The latest development of computing power and memory bandwidth in GPUs has driven the use of general-purpose computing on GPUs (GPGPU) [6].

The application of clustering is wildly spread among various different fields such as text mining, computer vision or computational biology. The popularity of k-means is explained by its low implementation complexity and well described mathematical properties. K-Means will only find non-optimal local-minima, depending on the initialization of centroids. This is known as seeding problem and was addressed in many researches. The run-time performance of K-Means is increases as data is growing rapidly, as finding the correct parameter of k also critical as it change during every run and can only be done by performing several runs with initial seeding [2].

Recently hybrid CPU-GPU architecture has successfully emerged as a high performance computing platform. Simply this architecture consists of many-core GPUs with multi-core CPUs. GPUs are responsible for parallel data computation, and CPUs play a role of data distribution and task synchronization. GPU has higher density of computational cores on chip and require lower energy per instruction than CPU. By adapting the hybrid CPU/GPU architecture instead of standalone CPU or GPU the energy consumption and space requirement reduced dramatically as compared to building a supercomputing site. At present, there are various supercomputing sites who adapt this novel computing architecture in Top500. If the programming complexity of hybrid CPU-GPU architecture reduced successfully, the hybrid CPU-GPU architecture will become more and more popular in future.

This paper provides a brief literature review of all versions of K-Means implementation on GPUs using CUDA that have published till date. Section II brief describes K-Means with its parallel implementation details on GPU. Section III explains literature review of various K-Means implementations on GPUs. Finally, Section IV discusses the findings of this review work and future scope for improvement.

II. RELATED WORK

A. K-Means

K-Means is a well-known partitioned clustering algorithm widely used in both industrial practices and academic research. It shares the properties of a more large class of statistical algorithms. The number of clusters K is the input to the K-Means algorithm. K-Means iteratively finds the K centroids of the data clusters. Each iteration consists of two steps:

- Step 1: Partition the data set into K subsets by assigning each data point to the subset whose centroid is the closest centroid to the data point.
- Step 2. Recalculate the K cluster centroids as the geometric centers of the subsets.

The algorithm repeats these two steps until convergence criteria fulfilled i.e. no data point moves from one cluster to another. It has been shown that K-Means always converges to a local optimum and stops after finite number of iterations. There is still active research on the K-Means algorithm itself [3].

B. *Parallelization of K-Means*

When parallelizing an application, it is important to understand for a designer that of the underlying characteristics of the architecture. For example, if the target architecture is multi-core or an 8-core general purpose computer, it is best to distribute the computation task to number of threads created depending on the number of independent processing elements in that system. A suitable work distribution can be achieved by dividing the work evenly among each of the created threads. This can be performed using traditional OS multi-threading support such as Java threads or PThreads in POSIX supported by operating systems like (Linux, UNIX). In such architectures, it is best to assign a thread to process element based on its availability, due to context switching overhead between thread as there may be millions of elements ready to be processed caused dominating bottleneck [2].

GPU hardware is many-core architecture which uses CUDA model to schedule and manage automatically the workload that consists of many individual tasks (threads). The creating of these threads has very negligible overhead, as it is hardware generated threads and takes only one clock cycle to start a thread. This suggests that the programmer should break the task into many logical units without regard to the underlying architecture of the parallel system, this encouraging the use of a huge number of threads to increase the chances that scheduler will find an optimal schedule for large number of tasks to execute on parallel SIMD architecture. Such a programming model is best utilized with the applications having large number of independent tasks and threads so called as "embarrassingly parallel" applications [2].

As the time-dominant phase of the K-Means algorithm is to assignment of data points in each cluster, which takes O(N*K) time. In this phase, the algorithm computes the Euclidean distance of each data point to the chosen set of centroids and tries to reassign each data point to the nearest cluster. To implement K-Means algorithm on CUDA, we can assign the distance calculation part of each data point to a single thread. Therefore each thread will loop over for all the initial cluster centroids, calculating its assigned data point's distance by finding the minimum distance from its data point to a cluster centroid, and become a member of the nearest cluster centroid [2].

When all threads are finished this task, the membership of each data point is derived, and thus the first phase of the algorithm is also finished. If the number of data points were equal to the number of processing elements, this pass could finish in one step. However, if we handle large amount of data the number of processing elements is limited. Therefore, with P cores we can accelerate the first phase to O((N*K)/P). If the data points are distributed in a 1-dimensional array, the distance calculation can be computed with a single scalar subtraction operation. If the data points resided in a 2-dimensional domain, it would increase computational requirements per element without adding any pressure on memory bandwidth. We will see later that in many cases, this parallel implementation of algorithm is limited by memory bandwidth rather than processing power [9].

Therefore, K-Means algorithm with multidimensional applications is likely to see larger speedup factors, as they are heavily floating point computational operations. With large dimensions data sets makes the application ideal for GPU architectures, which are also better suited for computational heavy classes of applications.

III. LITERATURE SURVEY

In academia, various parallel data mining algorithms have been proposed on distributed as well as shared memory parallel architecture models in the past decades. However, current available successful commercial mining systems does not applies these parallel data mining techniques, including, IBM Intelligent Miner, SPSS Clementine and SAS Enterprise Miner. Actually neither of these commercial systems achieved real time analysis. One of the reasons that affect the development of parallel data mining algorithms is the high cost of various parallel computing systems. Such as clusters, which are not affordable by small or medium businesses [11].

Low power consumption and low cost are the motivating powers of recent development in parallel architectures. One probable recent solution is the multi-core system. Examples are Intel core-duo or core-quad products. Although the multi-core systems are consumed low power and low in cost compared with traditional supercomputers. As multiple cores integrated onto a single chip, its scalability is poor. Another popular solution is Graphics Processing Unit (GPU). GPU is originally a highly specialized many-core architecture designed for graphics rendering for the computer gaming industry [7].

Recently high level languages like OpenCL, CUDA (Compute Unified Device Architecture) have developed to support easy programming on GPUs. NVIDIA's GPU with CUDA environment provides standard extensions to C-like languages to manipulate the GPUs. GPUs with CUDA provide tremendous computing power and memory bandwidth for applications. There are computational intensive applications which run on a GPU + CPU heterogeneous system architecture where the GPU acts as the computation accelerator, including scientific, medical, military, business, communication, and other domains. Successful examples are Computational Fluid Dynamics (CFD), Neural Network, Support Vector Machine (SVM), Magnetic Resonance Imaging (MRI), Finite Difference Time Domain (FDTD), intrusion detection, etc [1].

A wide variety of applications have been achieved huge speedups with GPGPU implementations. Kruger et al. presented framework for solving linear algebra problems on graphics processor units. Harris et al. present a cloud dynamics simulation with the help of partial differential equations and various other N-body and molecular dynamics simulations have also shown huge speedups [9]. Some database operations have been implemented on the GPU by pixel engine, and variety of other applications, such as AES encryption and sequence alignment have been successfully implemented on GPUs [8].

There are several implementations of the popular parallel K-means data clustering algorithm on GPUs exist [12]. One common restriction among this parallel implementation is that dimensionality of test data is limited to small values of 60 or below. An exception to this is the work of Zechner and Granitzer that specifically design K-Means for arbitrary large number of dimensions, but only implements the labeling stage of the algorithm on GPU, thus a substantial portion of the work is done with the CPU which encounters frequent memory transfers [4].

Che et al. report eight fold speedup for the cluster assignment stage of K-means on an Nvidia 8800GT for nearly 1.6 million data objects with 42 dimensions each [10]. Farivar et al. claim nearly 130-fold speedup with an Nvidia 8600GT using data set of 581,012 data objects with 54 dimensions. Wu et al. report up to 11-fold speedup with one billion data objects of eight or less dimensions when comparing running time of a Nvidia GeForce GTX280 to multicore processors with highly optimized parallel CPU code [2]. Finally, Zechner and Granitzer show speedups on an Nvidia GeForce 9600GT for data set consist of 500,000 data objects and 200 dimensions up to 42-fold and when compared to sequential code compiled with Intel C++ and Visual C++ compilers gives up to 13-fold speedup [4].

A non-optimal solution to the NP-hard problem of partitioned clustering was proposed by Lloyd whose most well-known variant is the K-Means algorithm. Shuai Che et al. and Li Zhan et al. published nearly similar work where Shuai Che et al. used CUDA for parallelizing partial steps of k-means on GPU, while Li Zhan et al. parallelize new centroids recalculation step also on GPU and thus algorithm performance become better [8][5].

Recently, Fang et al. proposed GPUMiner, a system consisting of three components:

- CPU-based storage and buffer manager to handle I/O and data transfer between CPU and GPU.
- GPU-CPU co-processing parallel mining module.
- GPU based mining visualization module.

GPUMiner used bitmap as the data structure because bitmap can enhance the efficiency of SIMD execution [13]. Apriori in GPUMiner shows significant speedups but it has the following weaknesses:

- GPUMiner uses Bitmap as a specialized data format which required tedious preprocessing work to convert transactional database to bitmap.
- The size of the database would be huge, since each item takes one bit in bitmap, no which exceed global memory on the GPU.

A fast CUDA-based K-means is proposed dedicatedly for very large data sets which cannot be fit into the global memory of the GPU [3].

IV. RESULTS AND COMPARISONS

You Li and Kaiyong Zhao implemented speeded parallel k-Means algorithm using CUDA 2.3 on a PC having NVIDIA GTX280 GPU and Intel(R) Core(TM) i5 CPU. They also provide the comparative results between speeded K-Means and other parallel popular benchmarking K-Means implementations such as HP_K-Means, UV_K-Means and GPUMiner. In below tables N is number of data objects in data sets, D is number of dimensions for every data object, K is number of clusters and M is number of maximum iterations [13].

Table 1: Comparison of various K-Means Implementations

N	K	D	Speeded K-Means	HP K-Means	UV K-Means	GPU Miner
2 Millions	100	2	0.22	1.45	2.84	61.39
	400	2	0.79	2.16	5.96	63.46
	100	8	0.35	2.48	6.07	192.05
	400	8	1.23	4.53	16.32	226.79

R. Wu, B. Zhang and M. Hsu provide a significant result for large data sets. The experiments are performed on HP XW8400 workstation with dual quad core Intel Xeon 5345 equipped with an Nvidia GeForce GTX 280. It calculates speedup on GPU over CPU on large data sets. Table II shows speedup compared to increased number of clusters which gives average speedup up to 10.2x. Similarly Table III shows speedup with constant clusters and increased dimensions up to 7.2x [3].

Table 2: Speedup for increase number of clusters

Data Set				Time(Sec)		Speedups
N	D	K	M	CPU(8C)	GPU	
1,000,00,000	2	200	50	4139	508	8.2
1,000,00,000	2	400	50	7470	744	10.0
1,000,00,000	2	600	50	10847	1012	10.7
1,000,00,000	2	800	50	14176	1248	11.0
1,000,00,000	2	1000	50	17515	1558	11.2
						10.2

Table 3: Speedup for increase number of dimensions

Data Set				Time(Sec)		Speedups
N	D	K	M	CPU(8C)	GPU	
1,000,00,000	2	2000	50	3415	299	11.4
1,000,00,000	4	2000	50	5969	446	13.4
1,000,00,000	6	2000	50	8343	2528	3.3
1,000,00,000	8	2000	50	10711	3354	3.2
						7.8

V. DISCUSSION

According to literature survey we found that implementations of algorithms on Nvidia's GPU using CUDA enabled many-core architectures have two major obstacles:

- First is limited size of low-latency high bandwidth shared memory.
- Second, thread divergence causing lost clock cycles through idle threads.

The first obstacle persists along two dimensions of data sets first is the number of data points and other is their dimensionality. It's relatively easy to split data sets up along one of those. Splitting data set efficiently along both requires specific method that allows processing large high dimensional data within the limitation of a small amount of shared memory and by using collaborative power of many parallel threads. The need for algorithms that can handle large data sets becomes also apparent when looking towards number of excellent serial K-Means algorithms that have been developed specifically for handling large data [12].

The apparent limitation of previous implementations of performing high-occupancy code to low dimensions is too restrictive for applications and makes these implementations less useful. Thread divergence, is not just specific to CUDA, but can be also found in other architectures, such as vector processors with multiple threads share a common instruction pointer. Without properly preprocessing the input data sets, branching of the program flow is common. None of the previously mentioned implementations for CUDA on GPU include data preprocessing as an essential tool for achieving additional speedup.

Analyzing the various papers on K-Means on GPU we can deduce:

- Nearly all complex and time-cost computation operations of K-Means can be speedup substantially by offloading work to GPU.
- Exploiting the GPU for the labelling stage of K-Means proved to be beneficial especially for large data sets and high cluster number.
- Parallelize an elementary data processing operation used by many applications on a highly parallel GPU architecture.
- The GPU architecture using CUDA parallel computing will provide compelling benefits for data mining applications.

VI. CONCLUSION

On the basis of literature survey and referenced results we conclude that in K-Means speedup is depend on number of clusters compared to number of dimensions or data objects. We found that there are many efficient parallel K-Means algorithm emerges as the size of data increases with good speedup on GPU. But we can utilize many more features of CUDA to get significant and more increased speedup compared to all previous parallel implementations.

ACKNOWLEDGMENT

We express our sincere thanks to the all authors, whose papers in the area of parallel data mining, K-Means algorithm and GPGPU published in various conference proceedings and journals.

REFERENCES

[1] Liheng Jian, ChengWang, Ying Liu, Shenshen Liang,Weidong Yi, Yong Shi, "Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture(CUDA) ", The Journal of Supercomputing Springer , Volume 64, Issue 3, June 2013.

[2] R. Farivar, D. Rebolledo, E. Chan, and R. Campbell, "A Parallel Implementation of K-Means Clustering on GPUs," Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications, 2008.

[3] R. Wu, B. Zhang and M. Hsu, "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.

[4] M. Zechner and M. Granitzer, "Accelerating K-Means on the Graphics Processor via CUDA," Proc. First Int'l Conf. Intensive Applications and Services (INTENSIVE '09), pp. 7-15 , 2009, doi: 10.1109/INTENSIVE.2009.19.

[5] H. Bai, L. He, D. Ouyang, Z. Li, and H. Li, "K-Means On Commodity GPUs With CUDA," Proc. WRI World Congress Computer Science and Information Eng., vol. 3, pp. 651-655, 2009, doi:10.1109/CSIE.2009.491.

[6] S.A.A. Shalom, M. Dash, and M. Tue, "Efficient K-Means Clustering Using Accelerated Graphics Processors," Proc. 10th Int'l Conf. Data Warehousing and Knowledge Discovery I. Song, J. Eder, and T. Nguyen, eds., pp. 166 175, 2008, doi: 10.1007/978-3-540-85836-2_16.

[7] 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.

[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] J. Kruger and R. Westermann, "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.

[10] Che S, Boyer M and Meng JY et al, "A performance study of general purpose applications on graphics processors using CUDA," J Parallel Distrib Compu 68(10), pp.1370-1380, 2008.

[11] Wu R, Zhang B and Hsu MC,"Cl 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.

[12] Kai J. Kohlhoff, Vijay S. Pande, and Russ B. Altman, "K-Means for Parallel Architectures Using All-Prefix-Sum Sorting and Updating Steps", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 24, NO. 8, AUGUST 2013.

[13] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu,"Speeding up K-Means Algorithm by GPUs".