2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

An Efficient Parallel ISODATA Algorithm Based on Kepler GPUs

Shiquan Yang, Jianqiang Dong and Bo Yuan
Intelligent Computing Lab, Division of Informatics
Graduate School at Shenzhen, Tsinghua University
Shenzhen 518055, P.R. China
yang244102596@gmail.com, 513712287@qq.com, yuanb@sz.tsinghua.edu.cn

***Abstract*—ISODATA is a well-known clustering algorithm used in various areas. It employs a heuristic strategy allowing the clusters based on the nearest neighbor rule to split and merge as appropriate. However, since the volume of the data to be clustered in real world is growing continuously, the efficiency of serial ISODATA has become a serious practical issue. The GPU (Graphics Processing Unit) is an emerging high performance computing platform due to its highly parallel multithreaded architecture. In this paper, we propose an efficient parallel ISODATA algorithm based on the latest Kepler GPUs and the dynamic parallelism feature in CUDA (Compute Unified Device Architecture). Performance study shows that our parallel ISODATA can achieve promising speedup ratios and features favorable scalability compared to the original algorithm.**

Keywords—ISODATA; GPU; CUDA; Dynamic Parallelism; Clustering

I. INTRODUCTION

GPUs (Graphics Processing Units) have been widely used in personal computers and workstations as the coprocessors of CPUs on image rendering and video processing tasks due to their tremendous computing power. Since the procedure involving vertex and fragment processing is computationally intensive and a data-parallel task, GPUs are inherently designed as Single Instruction Multiple Data (SIMD) processors with many cores, capable of processing a large number of threads simultaneously. The creation and management of the massively parallel threads are handled by the hardware and developers can focus on the parallelization of the problem of interest, rather than the details of scheduling and managing threads. Also, GPU programs often have good scalability and portability on different GPUs.

Due to the high availability of commodity GPUs and their parallel processing power, GPUs have evolved into powerful general purpose processing units (GPGPU). The computational power and memory bandwidth of mainstream GPUs are often an order of magnitude higher than CPUs of the same time. For example, the theoretical computing speeds (single precision) are over 300 GFLOPS for NVIDIA GeForce 8800 GTX and 4 TFLOPS for NVIDIA Tesla K20X (Kepler Architecture, Compute Capability 3.5), with peak memory bandwidth 86.4 GB/s and 249.6 GB/s respectively. Furthermore, the CUDA (Compute Unified Device Architecture) programming model introduced by NVIDIA in 2007 can make developers familiar with C programming language easily take advantage of GPUs, without the need to understand complex graphic APIs and the detailed knowledge of graphics hardware [15, 16].

In recent years, due to their enormous computational horsepower, very high memory bandwidth and low cost, researchers from various domains have been using GPUs as a new high performance parallel computing platform and achieved many encouraging results in areas such as computational chemistry, computational fluid dynamics, computational finance and numerical analysis [17, 18].

Clustering analysis is an unsupervised machine learning method used to group similar objects [11]. ISODATA (Iterative Self-Organizing Data Analysis Techniques) [7, 12] is a well-known clustering algorithm, which has been widely applied to remote sensing images segmentation [20], SAR imagery splitting [19] and palmprint recognition [14]. ISODATA is based on the distance similarity among objects and, compared to other traditional clustering techniques such as K-Means, it employs a heuristic strategy to enable the clusters to merge and split, depending on factors such as the distance between two clusters and the variance within a cluster [8, 10].

However, in the era of big data, the volume of data is becoming considerably large. For example, due to the higher resolution of remote sensing images, the size of a single image can reach up to several GBs with tens of millions of pixels. How to deal with such large amounts of data has become a really challenging issue that must be properly handled before ISODATA and other clustering techniques can be effectively applied in the real world. In fact, in the past decade, researchers have started to look for solutions using high performance computing techniques, including distributed systems and parallel computing [2, 3, 9, 13].

For example, the message-passing interface (MPI) environment was used to implement a distributed version of ISODATA [4]. Two MapReduce based parallel ISODATA algorithms have been proposed recently [1, 5]. They both used a map function to cluster the objects and a reduce function to update the centers. The difference is that the latter one considers the parallelization of the splitting procedure.

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

To the best of our knowledge, there is only one GPU-based parallel ISODATA algorithm in which the distance calculation and labeling steps were executed on the GPU [6]. However, the splitting procedure was ignored in their implementation, which is one of the key features of ISODATA.

In this paper, we propose an efficient parallel ISODATA algorithm based on the latest Kepler GPUs. Section II presents the general descriptions of the original ISODATA, the Kepler computing architecture and the CUDA programming model. Detailed algorithm analysis is given in Section III to show how to make the most of the parallel computing power and programmable capability of CUDA enabled GPUs using dynamic parallelism. The extensive performance study is conducted in Section IV and this paper is concluded in Section V.

II. PRELIMINARY

A. Serial ISODATA Algorithm

ISODATA is a widely used clustering algorithm in pattern recognition and data mining. It features an iterative process similar to the popular K-Means algorithm using Euclidean distance as the similarity measure to partition the data into different clusters. The unique feature of ISODATA is that it allows the merging and splitting of clusters dynamically. If the distance between two clusters is too small, they will be merged to create a single cluster. If the standard deviation within a cluster is over the user defined threshold, it will be split into two clusters. Consequently, ISODATA is more adaptive and flexible compared to the K-Means algorithm and is expected to perform better on different clustering problems.

Suppose that there is a set of samples and each sample represents a point in the d dimensional space. The goal is to assign the samples into different groups properly. Initially, c random points are selected as the centroids of each cluster. For each sample, its Euclidean distances to all centroids are calculated. The Euclidean distance D_{ij} between point x_i and centroid m_j is defined as

$$D_{ij} = \sqrt{\sum_{k=1}^{d} (x_{ik} - m_{jk})^2} \quad (1)$$

The samples are then partitioned into corresponding clusters Γ_i (i=1, 2, …, c) according to the nearest neighbor rule. If the number of samples within cluster Γ_i is less than the threshold, this cluster will be eliminated and its samples are assigned to the nearest clusters.

After that, the centroid of each cluster is recalculated. To determine whether certain clusters need to be merged or split, we need to further compute the average distances within clusters and the average distances between data points and centroids.

For the splitting procedure, the standard deviation vector of each cluster is calculated to check whether a splitting operation is required. The i^{th} component of the standard deviation vector of the j^{th} cluster is defined as

$$\sigma_{ij} = \sqrt{\frac{1}{N_j} \sum_{x_k \in \Gamma_j} (x_{ki} - m_{ji})^2} \quad (2)$$

where x_{ki} represents the i^{th} component of the k^{th} sample and m_{ji} represents the i^{th} component of the j^{th} centroid while N_j is the number of samples in the j^{th} cluster. If the maximum component value of the standard deviation vector is over the user-defined threshold, a splitting operation is activated.

As to the merging procedure, the distances between each pair of centroids are computed. Cluster pairs with distances less than the user-defined threshold are identified and sorted in an ascending order. Usually two clusters with the smallest distance are merged while it is possible to merge more cluster pairs in the same iteration.

The centroids are updated after the merging and splitting operations. Before the next iteration, the thresholds (inter-cluster distance and intra-cluster distance) can be modified as necessary by taking into account factors such as the expected number of clusters, the minimum number of samples in a cluster and the maximum iteration number.

B. CUDA Programming Model

CUDA is a parallel computing platform that enables NVIDIA GPUs to execute programs written in standard C with some extensions, dramatically reducing the challenges for developers to exploit the computing power of GPUs. A CUDA program invokes parallel functions called kernels that execute across many parallel threads. These threads are organized into thread blocks and each thread within a block executes an instance of the kernel, with a unique thread ID and block ID delivered by the CUDA runtime. A kernel function corresponds to a grid of thread blocks when launched by the host. A GPU executes one or more kernel grids while a streaming multiprocessor executes several thread blocks in groups of 32 threads called warps.

C. Kepler Compute Architecture

Although the last generation GPUs with Fermi architecture has already provided extraordinary parallel computing power, the latest generation of GPUs with the Kepler architecture has raised the performance bar again considerably. For example, GK110, the current flagship model in the Kepler family, is not only faster but also more energy efficient. It features over 4 TFLOPS (single precision) theoretical peak performance, equivalent to more than 15 GFLOPS per watt, three times as efficient as Tesla C2070 (Fermi architecture).

A full Kepler GK110 implementation includes 15 SMX units and six 64-bit memory controllers whereas different products may vary in the specific numbers. Each of the SMX unit contains 192 single precision CUDA cores, with each one containing fully pipelined floating point and integer arithmetic logic units. The total number of CUDA cores within GK110 is over two thousand, more than four times greater than that in GPUs with Fermi architecture. The SMX schedules threads in groups of 32 called warps and each SMX features four warp schedulers and eight instruction dispatch units, allowing four warps to be issued and executed concurrently. Kepler's quad warp scheduler

selects four warps each time and two independent instructions per warp can be dispatched in each cycle. Furthermore, unlike Fermi, Kepler allows double precision instructions to be paired with other types of instructions.

Dynamic Parallelism is a ground breaking feature currently supported by Kepler GPUs only. It allows a GPU to generate new parallel tasks by itself and synchronize on the results and schedule the work via dedicated hardware path without the intervention of the CPU. In previous GPUs, kernel functions can only be called from the host (CPU). When the parallel task running on the GPU is completed, the results are returned to the CPU and the GPU needs to wait for the host to launch another kernel to execute. However, with Kepler architecture, it is now possible to initiate a kernel launching from another kernel executing on the GPU. The parent kernel will immediately return back to the main thread after launching the child kernels and continue to execute until meeting an explicit device synchronization function. The major advantage is that it is easier to create and optimize recursive and data-dependent execution patterns. Moreover, since additional workloads can be executed simultaneously, GPU resources can be exploited more thoroughly.

III. PARALLEL ISODATA ALGORITHM

A. Design Considerations

The serial ISODATA can become very time-consuming when the volume of data to be clustered grows exceedingly large. There are three major procedures in ISODATA, including sample assigning, centroid updating and merging/splitting of existing clusters. During the sample assigning procedure, the Euclidean distances between clusters and data points need to be computed with a time complexity of O (NKD), where N, K and D are the number of data points, the number of clusters and the dimensionality respectively. The centroid updating operation calculates the average data coordinates within a cluster. The splitting procedure requires the standard deviation vectors and the time complexity is O (ND).

It is clear that sample assigning and standard deviation calculation are the two most computationally intensive parts in the algorithm, especially when the number of data points is large. Fortunately, they are both data independent operations, making them particularly suitable for parallel implementations.

B. Samples Assigning Based on GPU

The two operations in the sample assigning procedure are the distance calculation and identifying the nearest cluster to each data point. The data points to be clustered and the centroids are transferred from the host (CPU) memory to the global memory of the device (GPU). To take advantage of the coalesced memory accessing feature of device memory and locate elements with ease, data are stored in a 1D format. The Euclidean distances are computed by launching a kernel function specified by the *__global__* modifier. The kernel function creates blocks of threads in the GPU, with each thread computing only the distance between a single data point and a single centroid.

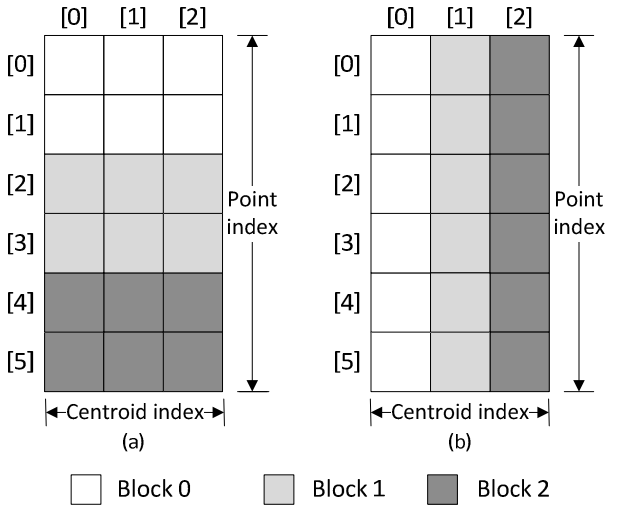

Fig.1. Point-wise thread blocks construction (a) and centroid-wise thread blocks construction (b)

In our method, the resultant distances are organized into an N-by-K array, where K is the number of centroids and N is the number of data points. In this way, each element of the distance array can be conveniently calculated by a single thread and the GPU threads can be organized into a 2D structure. The CUDA built-in variables *threadIdx* and *blockIdx* are used to determine the unique ID of each thread and the index of element in the distance matrix to be processed by the thread. These threads can be organized into blocks either in a centroid-wise manner or a point-wise manner (Fig. 1). In a point-wise manner, each thread block consists of threads calculating the distances between one or more points and all the centroids, corresponding to the rows of the distance matrix shown in Fig. 1(a). Similarly, the centroid-wise blocks compose of threads calculating the distances between the entire set of data points and one or more centroids, corresponding to the columns of the distance matrix shown in Fig. 1(b).

Note that in a point-wise block, the distances between a data point and all centroids are obtained after the execution of the block. Consequently, there is sufficient information about the nearest centroids of these data points immediately after the finishing of a thread block, without the need to wait for the whole kernel to finish. In order to combine the distance calculation and sample assigning steps together, the point-wise block design is preferred.

Due to the bottleneck of memory bandwidth, the overhead of data transfer between host and device is rather expensive, especially when the amount of data is huge. To minimize the transfer cost between CPU and GPU, the cluster labels of data are identified in GPU too. Since we employ a point-wise manner to construct thread blocks, each block of threads has already calculated the distances of certain data points to all centroids.

As a result, *Label Kernel* can be launched to identify the labels of these data points immediately as long as a thread block finishes the execution of *Distance Kernel*, as shown in Fig. 2. This functionality can be implemented by exploiting one of the novel features in CUDA 5.0 called *Dynamic Parallelism* (Compute Capability: 3.5). In this way, distance calculation and cluster labeling can be parallelized while

increasing the utilization of GPU resources and reducing the data transfer cost. After the cluster labels are determined, they are transferred back to host memory followed by the centroids updating operations.

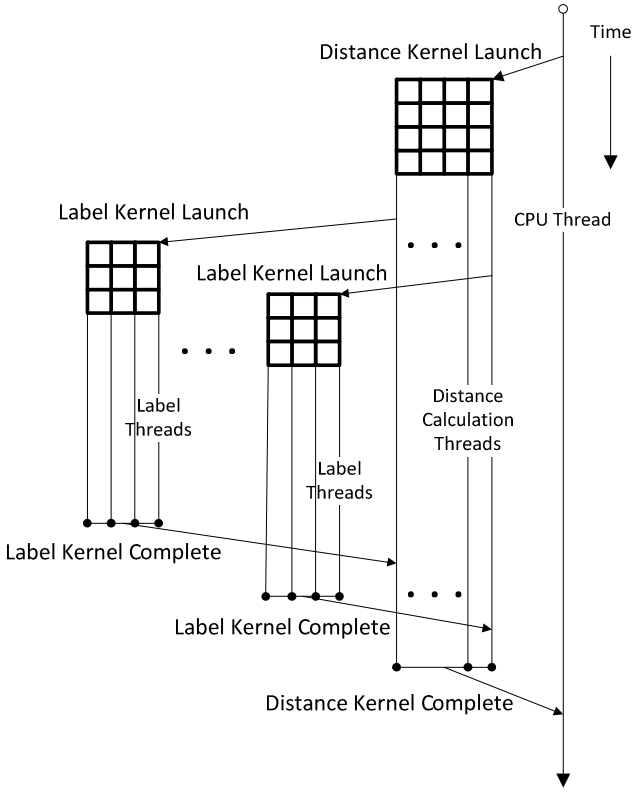

Fig.2. Parallel execution of distance kernel and label kernel

C. Splitting Based on GPU

The most costly operation in the splitting procedure is the calculation of standard deviation vector of each cluster. Similar to the distance calculation, a *K*-by-*D* array is used to store the variance values of each cluster, where *K* and *D* stand for the number of centroids and the dimensionality of data points respectively. The kernel launches a grid of threads of the same size as the array, with each thread responsible for the calculation of a single element of the variance matrix. Since we need to find the maximum element of the variance vector for each cluster, the centroid-wise block design is adopted in order to parallelize variance calculation and identifying the maximum variance element using *Dynamic Parallelism*.

D. Memory Optimizations

For distance calculation, every thread within a block needs to access the information of all centroids, which is stored in the global memory. As a result, the first K threads (K is the number of centroids) of the block are used to load the required centroid information from global memory to shared memory, which is much faster and directly accessible by all threads in the same block. Instead of frequently accessing the global memory with high latency, each thread can get the centroids stored in the shared memory and the I/O cost between GPU and the global memory can be reduced significantly.

IV. Experiments and Results

In this section, we thoroughly examined the performance of the proposed GPU-based parallel ISODATA algorithm in comparison to the serial version. Both the volume of data and the number of clusters were varied to investigate the scalability of the parallel ISODATA.

A. Experiment Setup

We implemented the parallel ISODATA in C using Visual Studio 2012 with CUDA 5.5. A serial version of ISODATA was also implemented in C. All experiments were performed on two desktop computers: i) Intel Core i5 3.2 GHz CPU and NVIDIA GeForce GT 640 GPU (GK208); ii) Intel Core i5 2.8 GHz CPU and NVIDIA Tesla K20 GPU. The GT 640 GPU has 384 CUDA cores and 1024 MB GDDR5 memory with memory bandwidth of 40 GB/s. The K20 GPU features 2496 CUDA cores and 5GB GDDR5 memory with memory bandwidth of 208 GB/s.

The focus of experimental studies was on the execution time of algorithms and the speedup in order to measure the performance of the parallel ISODATA compared to the serial version. The speedup was defined as the ratio between the execution time of the serial program and the execution time of the parallel version.

Since we were mainly interested in the efficiency instead of the effectiveness of the clustering algorithms, synthetic datasets were generated with randomly distributed data points. All computation was performed using single precision floating point operations.

B. Performance Study

In the first experiment, we compared the clustering results of both algorithms to validate the correctness of GPU-based ISODATA. There were totally 100K 10-dimensional data points, clustered into 5 clusters. For each cluster produced by the GPU, a corresponding cluster produced by the CPU was identified. The two sets of data points in the two clusters were compared and the number of common data points was counted. It is clear that the clustering results were extremely similar to each other, as shown in Table I.

TABLE I.
Comparison of Clustering Results by CPU and GPU

Cluster	CPU	GPU	Common Data Points
1	20554	20553	20553
2	20214	20221	20214
3	19732	19723	19723
4	20255	20257	20255
5	19245	19246	19245

In the second experiment, the number of clusters was 50 and the dimensionality was 10 while the number of data points was varied from 2K to 1M. Fig. 3 shows that the execution times per iteration were significantly reduced using GPUs. Furthermore, GT 640 and K20 were 30 to 40 times faster than their CPU counterparts with 30K data points and the speedup ratio kept increasing as the number of data points became larger (Fig. 4).

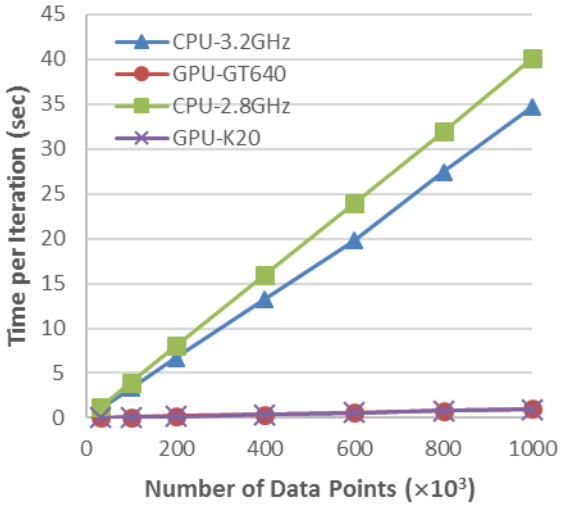

Fig. 3. Execution time per iteration (number of data points)

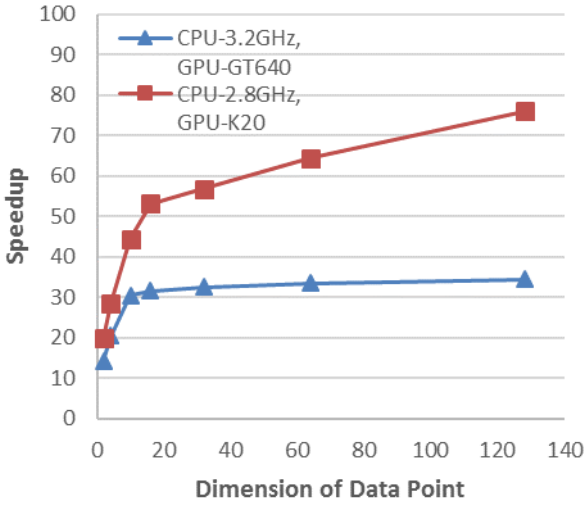

Fig.6. Speedup ratio (dimensionality of data point)

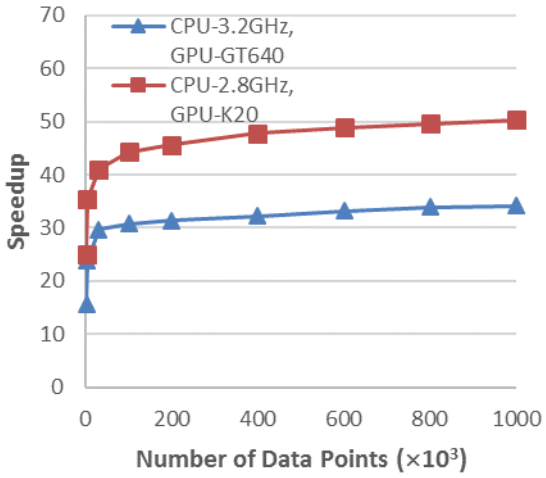

Fig. 4. Speedup ratio (number of data points)

Fig. 7. Execution time per iteration (number of clusters)

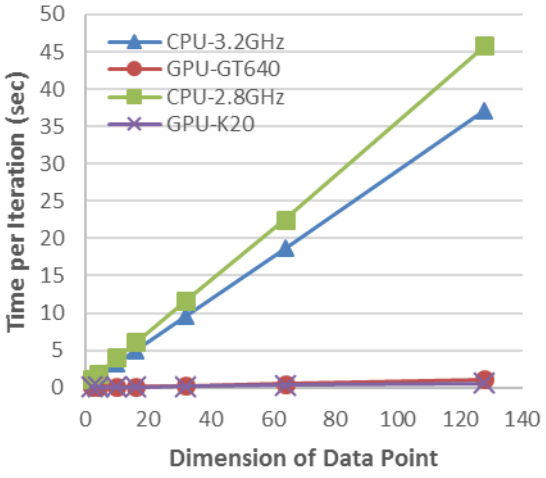

Fig. 5. Execution time per iteration (dimensionality of data point)

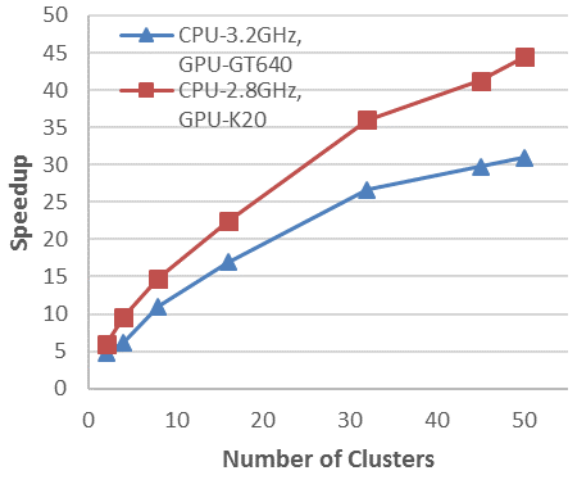

Fig. 8. Speedup ratio (number of clusters)

In the third experiment, the dimensionality was changed from 2 to 128 with 100K data points and 50 clusters. Fig. 5 shows that the GPU version scaled very well compared to the CPU version. When the dimensionality was over 60, the speedup ratios of K20 and GT 640 were more than 60 times and 30 times respectively (Fig. 6).

Finally, various numbers of clusters were used as shown in Fig. 7 and Fig. 8. The number of clusters ranged from 2 to 50 with 100K 10-dimensional data points. When the number of clusters was less than 10, the speedup ratio was around 5 to 15 times for K20 and 4 to 10 times for GT 640. When the number of clusters was over 30, the ratio increased to more than 35 and 25 times for K20 and GT 640 respectively. Note that when there were 50 clusters, K20 was nearly 45 times faster than its CPU counterpart.

Note that due to various random factors, the execution time of the algorithm may vary slightly in each trial. As a result, the speedup ratios may not be fully consistent even with the same experiment setting. In the above experiments, all results were averaged over 5 independent trials.

V. CONCLUSION

In this paper, we presented a highly efficient parallel ISODATA algorithm based on advanced Kepler GPUs. The sample assigning and variance calculating procedures were executed on the GPU, which are the most computationally intensive components of ISODATA. Two thread block designs were proposed to better parallelize the operations on the GPU and *Dynamic Parallelism*, which is the latest CUDA feature supported by Kepler GPUs was exploited to make the most of available GPU resources. Furthermore, shared memory was employed to reduce the global memory access traffic and I/O bottleneck.

Experimental results confirmed that the GPU accelerated ISODATA and the original ISODATA produced extremely similar clustering results and the difference was trivial. More importantly, the parallel ISODATA with two NVIDIA Kepler GPUs: GT 640 (consumer-level, less than $100) and K20 (high-end) achieved competitive performance in terms of speedup and scalability with regard to the number of data points, dimensionality and the number of clusters.

ACKNOWLEDGMENT

This work was supported by the NVIDIA CUDA Teaching Center awarded to Tsinghua University.

REFERENCES

[1] C. Wang, C. R. Wang, and X. Song, "A MapReduce based ISODATA algorithm," in *Proceedings of 2012 3rd IEEE International Conference on Intelligent Control and Information Processing,* July 2012, pp. 765-768.

[2] N. Memarsadeghi, D. M. Mout, N. S. Netanyahu, and J. L. Moigne, "A fast implementation of the ISODATA clustering algorithm," *International Journal of Computational Geometry & Applications*, vol. 17, no. 1, pp. 71-103, February 2007.

[3] G. A. Riccardi, and P. H. Schow, "Adaptation of the ISODATA clustering algorithm for vector supercomputer execution," in *Proceedings of Supercomputing*, vol. 2, pp. 141-150, 1988.

[4] M. K. Dhodhi, J. A. Saghri, I. Ahamad, and R. UI-Mustafa, "D-ISODATA: a distributed algorithm for unsupervised classification of remotely sensed data on network of workstations," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 280-301, 1999.

[5] B. Li, H. Zhao, and Z. H. Lv, "Parallel ISODATA clustering of remote sensing images based on MapReduce," in *Proceedings of 2010 IEEE International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery,* Oct. 2010, pp. 380-383.

[6] F. Ye, and X. Shi, "Parallelizing ISODATA algorithm for unsupervised image classification on GPU," in *Modern Accelerator Technologies for Geographic Information Science*, Springer, 2013, pp. 145-156.

[7] G. H. Ball, and D. J. Hall, "*ISODATA, a novel method of data analysis and pattern classification*," Stanford Research Institute, Menlo Park, California, April 1965.

[8] S. Phillips, "Reducing the computation time of the ISODATA and K-means unsupervised classification algorithms," in *Proceedings of 2002 IEEE International Conference on Geoscience and Remote Sensing Symposium,* 2002, pp. 1627-1629.

[9] N. B. Venkateswarlu, and P. Raju, "Fast ISODATA clustering algorithms," *Pattern Recognition*, vol. 25, no. 3, pp. 335-342, 1992.

[10] M. Merzougui, M. Nasri, and B. Bouali, "ISODATA classification with parameters estimated by evolutionary approach," in *Proceedings of 2013 8th IEEE International Conference on Intelligient Systems: Theories and Applications,* May 2013, pp. 1-7.

[11] A. K. Jain, and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1988.

[12] G. H. Ball, and D. J. Hall, "A clustering technique for summarizing multivariate data," *Behavioral Science*, vol. 12, no. 2, pp. 153-155, March 1967.

[13] H. T. Bai, L. L. He and Z. S. Li, "K-means on commodity GPUs with CUDA," in *Proceedings of 2009 IEEE World Congress on Computer Science and Information Engineering,* March 2009, pp. 651-655.

[14] F. Liu, C. X. Lin, P. Y. Cui, and T. Dong, "Palmprint recognition based on ISODATA clustering algorithm," in *Proceedings of 2007 IEEE International Conference on Wavelet Analysis and Pattern Recognition,* Nov. 2007, pp. 1129-1133.

[15] D. B. Kirk, and W. M. Hwu, *Programming massively parallel processors: a hands-on approach*. Morgan Kaufmann, 2010.

[16] CUDA. Available: https://developer.nvidia.com/cuda-toolkit

[17] NVIDIA Tesla K20-K20X GPU Accelerators Benchmarks. Available: http://www.nvidia.com/docs/IO/122874/K20-and-K20X-applicationperformance-technical-brief.pdf

[18] CUDA Research and Applications. Available: https://developer.nvidia.com/cuda-action-research-apps

[19] Y. F. Cao, C. X. Su, and J. J. Liang, "Efficient big-size light region splitting scheme for high-resolution SAR imagery," in *Proceedings of the International Conference on Information Engineering and Applications*, vol. 2, Part VII, pp. 515-520, 2013.

[20] M. M. Awad, "Improving satellite image segmentation using evolutionary computation," American Journal of Remote Sensing, vol. 1, no. 2, pp. 13-20, 2013.