Accelerating BIRCH for Clustering Large Scale Streaming Data Using CUDA Dynamic Parallelism

Jianqiang Dong, Fei Wang and Bo Yuan

Intelligent Computing Lab, Division of Informatics
Graduate School at Shenzhen, Tsinghua University
Shenzhen 518055, P.R. China
513712287@qq.com,wangfeifast@gmail.com,yuanb@sz.tsinghua.edu.cn

Abstract. In this big data era, the capability of mining and analyzing large scale datasets is imperative. As data are becoming more abundant than ever before, data driven methods are playing a critical role in areas such as decision support and business intelligence. In this paper, we demonstrate how state-of-the-art GPUs and the Dynamic Parallelism feature of the latest CUDA platform can bring significant benefits to BIRCH, one of the most well-known clustering techniques for streaming data. Experiment results show that, on a number of benchmark problems, the GPU accelerated BIRCH can be made up to 154 times faster than the CPU version with good scalability and high accuracy. Our work suggests that massively parallel GPU computing is a promising and effective solution to the challenges of big data.

Keywords: GPU, CUDA, Dynamic Parallelism, BIRCH, Big Data, Clustering

1 Introduction

In the era of big data, modern organizations in almost all industries are facing increasingly growing amount of heterogeneous data at unprecedented speed. Each day, around 2.5 quintillion bytes of data[1] are created such as sensor data, posts in social networks, digital images/videos, web search results, telecommunication records and financial transactions. The scale of the data available creates significant challenges for traditional techniques to effectively store, transfer, visualize and analyze the data within a reasonable amount of time.

Despite of the large number of existing data mining algorithms for tasks such as classification, clustering and frequent pattern analysis, there are two major issues that must be carefully addressed before they can be properly applied in the scenario of big data. Firstly, many algorithms assume that all data are stored in the main memory, which can be readily accessed. However, for real-world problems, the size of the data can easily exceed the memory capacity and, when multiple access to the dataset is required, the I/O cost may severely compromise the effi-

[1] http://www-01.ibm.com/software/data/bigdata/

ciency of the algorithm. To solve this issue, various data stream mining techniques have been proposed, which only require reading the data once. They are particularly suitable for situations where the entire dataset is too large to fit into the main memory or the data come in a continuous manner.

Secondly, most data mining algorithms are designed without explicitly taking parallel computing into account, although they may have inherent potential for parallelism. It is often assumed that each computing step is to be executed sequentially and no special efforts are devoted to harnessing the power of advanced multi-core and many-core computing devices that are becoming increasingly popular in the past decade. As a result, even with seemingly decent computational complexity in theory, the real running time of these algorithms on non-trivial datasets can be prohibitively intolerable. In fact, this issue is creating a large gap between academic research in data mining and industrial applications.

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [1, 2] is one of the most well-known hierarchical clustering algorithms for large scale data, which can incrementally cluster incoming data and requires only a single scan of the dataset in most cases. To make BIRCH more applicable on real-world problems, in this paper, we will investigate how to effectively accelerate BIRCH using parallel computing techniques.

In addition to CPU-based parallel computing architecture such as MPI[2] and OpenMP[3], in recent years, GPU (Graphics Processing Unit) computing is quickly becoming a new powerhouse for providing high performance computing capability at dramatically reduced cost and many GPU-based data mining algorithms have been proposed [3, 4]. Modern GPUs feature thousands of cores and support tens of thousands concurrent threads (many-core computing), making them specifically suitable for massively data parallel computing tasks. In many application areas such as fluid dynamics, financial engineering, life science and signal processing, researchers can often obtain 10~100× speedups on computing intensive problems by using standard workstations equipped with advanced GPU computing cards. In nowadays, the peak performance of high-end GPUs is over one TFLOPS (double-precision floating point) and the average cost to achieve one GFLOPS is already less than one dollar with GPU computing[4].

In the rest part of this paper, Section 2 gives a brief review on clustering algorithms for large scale data, especially the BIRCH algorithm. Section 3 introduces CUDA[5] (Compute Unified Device Architecture) and the Dynamic Parallelism technique, which brings incredible convenience to GPU computing with measurable performance improvement. The parallel implementation of BIRCH is detailed in Section 4 along with the experiment specification. The main experiment results are presented in Section 5 and this paper is concluded in Section 6 with some discussions and directions for future work.

---

[2] http://www.mcs.anl.gov/research/projects/mpi/

[3] http://www.openmp.org/

[4] http://en.wikipedia.org/wiki/FLOPS#Cost_of_computing/

[5] https://developer.nvidia.com/cuda-toolkit/

2 Clustering Big Data

Clustering is one of the most important unsupervised learning methods in pattern recognition and data mining [5]. There are mainly two categories of clustering methods: hierarchical clustering such as agglomerative clustering and partition-based clustering such as K-means. For data stream mining [6-8], CluStream [9] is an important hierarchical clustering algorithm, which uses many micro clusters to form a better macro cluster. In the meantime, the extended K-Means algorithm and STREAM [10] are two good examples of partition-based methods.

In BIRCH, a key concept is called clustering feature (CF), which is a triple holding the necessary information (e.g., linear sum and square sum) of all data points belonging to a certain cluster. The main procedure of BIRCH involves constructing a height balanced tree called CF-Tree. A parent node in the CF-Tree stores the summary of a large cluster while each child node maintains the information of its own small cluster. There are two major parameters in BIRCH named *B* and *T*. *B* is the branching factor of the CF-Tree, indicating the maximum number of children of the parent node. *T* is the threshold used to determine whether a new coming data point can be absorbed by an existing cluster.

In recent years, a few GPU implementations of hierarchical clustering algorithms have been proposed. For example, a speedup of more than 100 times was achieved on the agglomerative clustering algorithm [11, 12]. In the meantime, MPI and the Thrust library[6] have been used to accelerate BIRCH on various tasks such as text clustering problems [13, 14]. However, as Thrust can only parallelize certain components in BIRCH and it takes extra time to transfer data between host (CPU) and device (GPU), the overall speedup was only around 6 times.

3 GPU Computing

CUDA is a GPU programming environment developed by NVIDIA in 2007 and is the most widely used platform for GPU programming. The latest CUDA version is 5.5, which is the best match to GPUs with compute capability 3.5 such as Tesla K20 and GeForce GTX TITIAN. Please refer to CUDA C Computing Guide[7] and CUDA C Best Practices Guide[8] for a comprehensive review.

With the increasing popularity of CUDA, more and more computing libraries have been developed, such as Thrust, cuRAND[9] and so on. Thrust is a CUDA STL programming model, which can handle the data on GPU in a C++ STL way, and consists of many commonly used algorithms such as sorting and reduction. cuRAND is a CUDA random number library, which can generate various kinds of random number in a very efficient way.

---

[6] https://developer.nvidia.com/thrust/
[7] http://docs.nvidia.com/cuda/cuda-c-programming-guide/
[8] http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
[9] https://developer.nvidia.com/curand

CUDA Dynamic Parallelism[10] and Kepler Compute Architecture[11] together create the state-of-the-art GPU computing environment, making GPU computing much more efficient and easier to code compared to the last generation GPU architecture. Dynamic Parallelism offers exciting new capabilities by providing a mechanism for calling kernel functions from another kernel function so that GPU programs can be made more flexible and easier to synchronize. Fig. 1 shows an illustration of kernel launching in GPU.

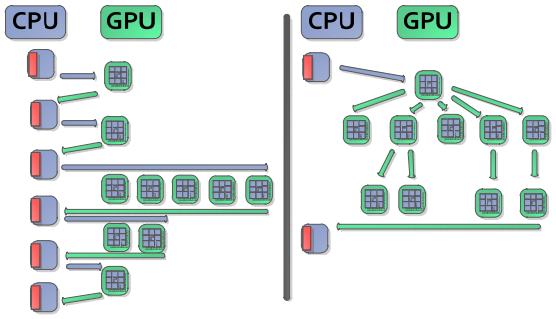

Fig. 1. Kernel launching with (right) and without dynamic parallelism (left)

4 Methodology

4.1 Standard BIRCH

The main operations in BIRCH include the construction of the CF Tree and assigning data points to a nearby CF value. In Table 1, each data point has to go through step 3 and step 4. Most data points are allocated into an existing cluster (step 5.1) while some data points create a new CF (step 5.2), and only very few data points result in the splitting of the CF tree and creating a new CF (step 5.3).

Table 1. The framework of BIRCH

Step	*BIRCH*
1	Initialize the data source.
2	For each data sample.
3	While (not leaf)
3.1	Find the nearest CF.
3.2	Go down to the nearest CF sub-tree.
4	Find the nearest CF in the leaf node.
5	Compute the distance *t* between the data point and the nearest CF
5.1	If *t*<=T then absorb.
5.2	If *t*>T and Leaf node is not full, then add new CF.
5.3	Else split the leaf node to allocate the new nodes.

[10] http://docs.nvidia.com/cuda/cuda-dynamic-parallelism/index.html
[11] http://www.nvidia.com/object/nvidia-kepler.html

4.2 GPU Accelerated BIRCH

In the proposed GPU based BIRCH (GBIRCH), a master kernel is launched first. Next, several slave kernels are launched using CUDA Dynamic Parallelism, with each slave kernel dealing with a subset of the data samples in the GPU memory. The slave kernel is terminated after its associated data samples have all been processed. Tables 2-4 present the pseudo code of the master kernel, the slave kernel, and the refinery kernel.

Table 2. Master kernel

Step	*GBIRCH_Master*
1	Set up the data source and partition the data.
2	Build up the CF Tree.
3	For each data block
3.1	Call *GBIRCH_Slave*.
4	Call *GBIRCH_Refinary*.
5	Output the data point with its cluster information.
6	Repeat steps 3~5, until all the data are processed.

Slave kernels are responsible for most of the computing tasks in BIRCH. Each slave kernel fetches some data records from the master kernel and computes the distances and finds the nearest CF Leaf. If the data record can be absorbed by an existing cluster, the slave kernel does the job individually. If a new CF is to be added to the CF tree, or the CF tree needs to be split, the data will be transferred back to the master kernel. In this way, the parallel slave kernels can always keep working on the same and well organized CF tree.

Table 3. Slave kernel

Step	*GBIRCH_Slave*	*Memory Allocation*
1	Get data from *GBIRCH_Master*.	Global to Shared
2	For each data sample	
3	While (not leaf)	Global to Shared
3.1	Find the nearest CF.	Shared and Register
3.2	Go down to the nearest CF sub-tree.	Global to Shared
4	Find the nearest CF in the leaf node.	Shared and Register
5	Compute t.	On a single thread
5.1	If t<=T, absorb the data sample.	
5.2	If t>T and Leaf is not full, add a new CF.	
5.3	Else save data point and submit to *GBIRCH_Master*.	

The refinery kernel keeps the CF tree accurate and well organized. Step 1 happens more frequently than other steps and is parallelized. Steps 2~4 are used to split the CF tree and no two threads can work on the CF tree at the same time (locking is required to ensure mutual exclusion).

Table 4. Refinery kernel

Step	*GBIRCH_Refinery*
1	Test the new CF Leaf from *GBIRCH_Slave*. If the CF parent is full, then add to split candidate.
2	Each thread begins to split with a candidate, and lock the CF tree when it is used in this process.
3	Roll up to split the higher level nodes, until reaching Root.
4	Repeat steps 2~3, until all the candidates are split.

In the CF tree, each node uses an array to store the CF values of its children and keeps *parentID*, *childID* and the number of children. In our work, the branching value was between 16~128. A smaller value may severely reduce the potential for parallelism while a larger value may result in a CF tree that is not deep enough to produce fine clustering granularity.

4.3 Benchmark Datasets

In the experiments, we used both synthetic and real datasets and all six datasets are widely used in data mining research. The first three datasets (DS1, DS2, DS3) were used in the original work on BIRCH [1, 2]. Each dataset consists of many centroids (from 100 to 1 million), and each centroid has 100 to 1 million data points. KDD CUP is a well-recognized annual international knowledge discovery and data mining competition. We used KDD CUP 98 data[12] (191,781 samples, 481 attributes), KDD CUP 99 dataset[13] (4,898,431 samples, 42 attributes) and KDD CUP 2012 data[14] (149,639,105 samples, 12 attributes) in our experiments.

5 Experiment Results

The software environment was: CUDA 5.0, Visual Studio 2010 and Windows 7. The hardware configuration was: Intel Core i5-2320 (CPU), 8GB RAM and NVIDIA Tesla K20 (Kepler Architecture GPU).

On the six datasets, the maximum speedup value of GBIRCH was 154.29 on KDD CUP 2012 data set (Table 5). In general, the results were better on larger data sets as the CF tree construction process is difficult to be executed in parallel on GPU (as the dataset gets larger, more time is spent on the absorbing process rather than the construction of the CF tree).

We also tested the scalability of GBIRCH using DS3. In Table 6, it is clear that as the number of records increases (the dimension was fixed to 2), the advantage of GBIRCH over BIRCH became more evident and as the dimension increased, the relative performance of GBIRCH was kept reasonably stable. Finally, the

[12] http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html
[13] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[14] http://www.kddcup2012.org/c/kddcup2012-track2/data

accuracy of GBIRCH was evaluated by measuring the percentage of data points that were assigned to the same cluster as in the CPU version. Table 7 shows that the impact of parallelization on the accuracy of BIRCH is trivial.

Table 5. Comparison of running time (ms)

Dataset	*CPU*	*GPU*	*Speedup*
DS1	498691	3983	125.20
DS2	502145	4486	111.93
DS3	501168	4296	116.66
KDD CUP 98	21318	2796	7.62
KDD CUP 99	632264	8512	74.28
KDD CUP 2012	3268765	21186	154.29

Table 6. Scalability test with regard to the number of records and dimensionality (ms)

# of Data	*CPU*	*GPU*	*Speedup*	*Dim*	*CPU*	*GPU*	*Speedup*
1,000,000	12401	1602	7.74	2	12401	1602	7.74
4,000,000	49820	1692	19.44	5	19185	2448	7.84
12,000,000	150543	2088	72.10	10	28129	3918	7.18
40,000,000	501168	4296	116.66	25	55224	8202	6.73

Table 7. Accuracy of GBIRCH

Dataset	*Total Number*	*Correct Number*	*Accuracy*
KDD CUP 98	191,781	191,781	100%
KDD CUP 99	4,898,431	4,897,012	99.9%
KDD CUP 2012	149,639,105	148,527,097	99.3%

6 Conclusions

The major motivation of this paper was to investigate an important question in the era of big data: how to effectively handle the challenges from clustering large scale data. In our work, we chose BIRCH, a well-known clustering technique for streaming data, as an example to show how advanced GPUs and CUDA platform with the latest Dynamic Parallelism capability can significantly reduce the time required for clustering large datasets. Experiment results demonstrated that, with a careful parallel implementation of the major procedures in BIRCH and the help of Dynamic Parallelism as well as the smart use of GPU memory, GBIRCH achieved encouraging speedups from 7 to 154 times over the original BIRCH on six benchmark datasets. In the meantime, GBIRCH also featured satisfactory scalability with regard to the size and dimensionality of the dataset.

There are a number of possible directions for future work. For example, we can implement the GPU versions of other popular data stream clustering algorithms such as CluStream or CURE [15]. The performance of GPU accelerated cluster-

ing algorithms on other data types such as text and XML or on high dimensional datasets is also worth investigation.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (No. 60905030) and NVIDIA CUDA Teaching Center Program.

References

1. Zhang, T., Raghu, R., Miron, L.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. ACM SIGMOD Record, vol. 25(2), 103-114 (1996)
2. Zhang, T., Raghu, R., Miron, L.: BIRCH: A New Data Clustering Algorithm and Its Applications. Data Mining and Knowledge Discovery, vol. 1(2), 141-182 (1997)
3. Fang, W., Lau, K., Lu, M. et al.: Parallel Data Mining on Graphics Processors. Technical Report HKUST-CS08-07 (2008)
4. Bai, H., He, L., Ouyang, D., Li, Z., Li, H.: K-Means on Commodity GPUs with CUDA. In: 2009 WRI World Congress on Computer Science and Information Engineering, pp. 651-655 (2009)
5. Jain, A. K., Murty, M. N., Flynn, P. J.: Data Clustering: A Review. ACM Computing Surveys, vol. 31(3), 264-323 (1999)
6. Mahdiraji, A. R.: Clustering Data Stream: A Survey of Algorithms. International Journal of Knowledge-based and Intelligent Engineering Systems, vol. 13(2): 39-44 (2009)
7. Berkhin, P.: A Survey of Clustering Data Mining Techniques. In: Kogan, J. et al. (eds.), Grouping Multidimensional Data, Springer, 25-71 (2006).
8. Barbará, D.: Requirements for Clustering Data Streams. ACM SIGKDD Explorations Newsletter, vol. 3(2), 23-27 (2002)
9. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: A Framework for Clustering Evolving Data Streams. In: 29th International Conference on Very Large Data Bases, pp. 81-92 (2003)
10. O'Callaghan, L., Meyerson, A., Motwani, R., Mishra, N., Guha, S.: Streaming-Data Algorithms for High-Quality Clustering. In: 18th International Conference on Data Engineering, pp. 685-694 (2002)
11. Shalom, S. A., Dash, M.: Efficient Partitioning Based Hierarchical Agglomerative Clustering Using Graphics Accelerations with CUDA. International Journal of Artificial Intelligence & Applications, vol. 4 (2), 13-33 (2013)
12. Shalom, S. A., Dash, M., Tue, M., Wilson, N.: Hierarchical Agglomerative Clustering Using Graphics Processor with Compute Unified Device Architecture. In: 2009 International Conference on Signal Processing Systems, pp. 556-561 (2009)
13. Garg, A., Mangla, A., Gupta, N., Bhatnagar, V.: PBIRCH: A Scalable Parallel Clustering Algorithm for Incremental Data. In: 10th IEEE International Database Engineering and Applications Symposium, pp. 315-316 (2006)
14. Bagga, A., Toshniwal, D.: Parallelization of Hierarchical Text Clustering on Multi-core CUDA Architecture. International Journal of Computer Science and Electrical Engineering, vol. 1, 72-76 (2012)
15. Guha, S., Rastogi, R., Shim, K.: CURE: An Efficient Clustering Algorithm for Large Databases. In: 1998 ACM International Conference on Management of Data, pp. 73-84 (1998)