Volume 3, Issue 10, October 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Parallel Optimized Algorithm for Apriori Association Rule Mining on Graphics Processing Unit with Compute Unified Device Architecture (CUDA)

Abhaya Kumar Sahoo[1], Amardeep Das[2], Mayank Tiwary[3]
Department of Information Technology
C.V.Raman College of Engineering
Bhubaneswar, India

Abstract— Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently .Now GPU(Graphics Processor Unit) has taken a major role in high performance computing for general purpose applications. Compute Unified Device Architecture (CUDA) programming model provides the programmers adequate C language like APIs to better exploit the parallel power of the GPU. Data mining has significant applications in various domains. Currently, data mining toolkits cannot meet the requirement of applications with large scale database in terms of computation and speed. In this paper, we propose two parallel association rule mining algorithms i.e. parallel Apriori algorithm and parallel Apriori_Cluster algorithm on GPU with CUDA platform. These algorithms are implemented in CUDA with C language, which focuses on computation time compared with execution time of serial program in CPU.This parallel implementation out forms significantly on Tesla C2070 GPU and an Intel core i7 @ 3.40 GHz CPU. Our clustering of nodes is based on fiber optic communication. Our results have shown that GPU + CUDA parallel architecture is feasible for performance measurement in terms of execution time and promising for data mining applications.

Keywords— Parallel computing, GPU, CUDA, Cluster, Data mining, Parallel Apriori Association rule mining

1. INTRODUCTION

Data mining is to discover interesting, meaningful, and understandable patterns hidden in massive data sets [1]. It has become a hot research domain in recent years [2–10]. Important application areas are business intelligence, customer relationship management, WWW, scientific simulation, artificial intelligent, machine learning, e-commerce, bioinformatics, and many more. The size of various data sets has increased tremendously in recent years as speedups in processing and communication have greatly improved the capability for data generation and collection in all areas. It is quite common to see databases on the order of gigabytes or terabytes. A sequential data mining algorithm handling these large data sets would potentially be unable to run in-core or would take a tremendous amount of time. Therefore, users have to turn to rely on parallel and distributed computing techniques to accelerate the computation. The use of multiple processors enables the use of more memory so that a larger dataset can be handled in the main memory attached to the processors. Hence, parallel data mining are in strong demand in various domains. Many parallel data mining algorithms have been proposed on distributed-memory parallel machines and shared-memory parallel machines [6, 7, 9] in the past decades. Low cost and low power consumption requirements are the motivating powers of recent development of parallel architectures. One recent solution is the multi-core system. Although the multi-core system is low in cost and power consumption compared with traditional supercomputers, its scalability is poor since multiple cores have to be integrated onto a single chip. Another promising solution is Graphics Processing Unit (GPU). A graphics processing unit (GPU) 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 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.

Currently, high level languages have emerged to support easy programming on GPUs. CUDA is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU). GPUs with CUDA provide tremendous memory bandwidth and computing power. For example, NVIDIA's Tesla C2070 can achieve a sustained memory bandwidth of 144 GB/s and a single-precision peak performance of 1.03 TFLOPS/s; NVIDIA's Tesla C1060 can achieve a bandwidth of 102 GB/s and a single-precision peak performance of 933 GFLOPS/s [10].Low cost is another factor of GPU. Low cost provides the medium-sized business and individuals great opportunities to afford supercomputing facilities. Therefore, recently, there has been a trend to accelerate computational intensive applications on a GPU + CPU

heterogeneous system where the GPU acts as the computation accelerator, including scientific, communication, military, business, medical and other domains. Due to the great advantages of GPUs, we would like to explore the performance of parallel data mining algorithms on CUDA-enabled GPU.We propose a parallel-Apriori association rule mining algorithm using CUDA based technique. Experiments are on an HP xw8600 workstation with a NVIDIA Tesla C2070 card and an Intel C*ore i7 @ 3.40* GHz CPU. By comparing the execution time of *Parallel-Apriori* with an efficient serial Apriori program on the *Core i7 @ 3.40* GHz Intel Core i7 CPU, it shows up to 6 times speedup on a real-world data set and 13.5 times on synthetic data sets.

The rest of this paper is organized as follows. Section 2 presents the background knowledge of GPU architecture and CUDA programming model. Section 3 overviews the related work. Section 4 describes the typical algorithm briefly. In Section 5, we present our CUDA-based technique for data mining parallelization on GPU with CUDA architecture. Then CUDA implementation of Parallel-*Apriori* proposed technique is presented in Section. 6. Experimental results are presented in Section 7 and we summarize our work in Section 8.

2. NVIDIA's CUDA-enabled parallel Computing

2.1 GPU architecture

Nowadays GPUs has evolved into a highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth.

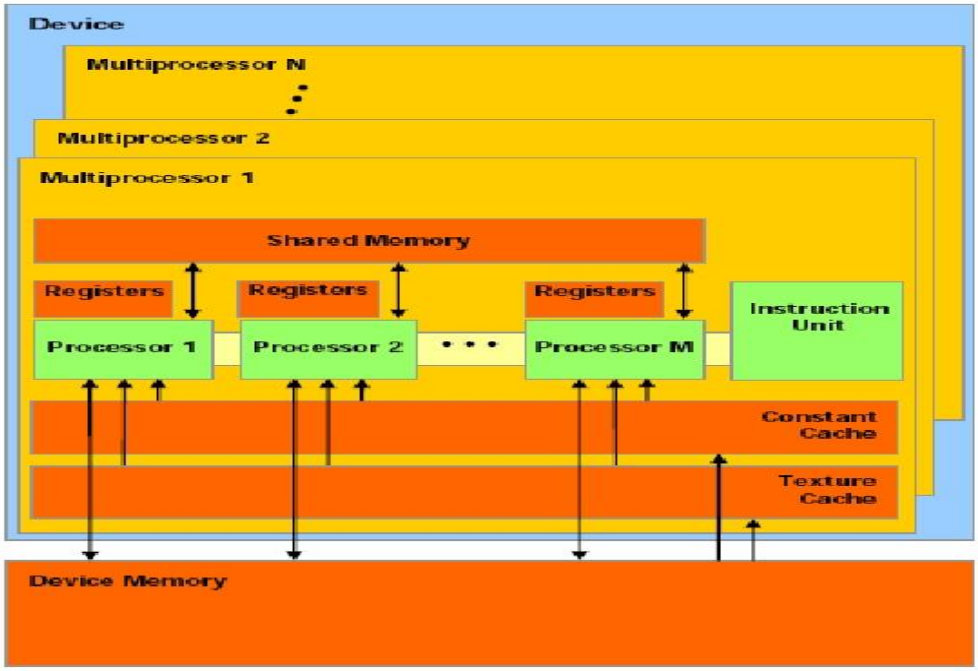

Fig1. A set of SIMD stream multiprocessors with memory hierarchy

In addition to rendering process, they are also suitable to general compute-intensive, highly parallel computation. NVIDIA's GPU with the CUDA programming model provides an adequate API for non-graphics applications. CPU sees a CUDA device as a many-core co-processor. At the hardware level, CUDA-enabled GPU is a set of SIMD stream multiprocessors (SMs) with 14 stream processors (SPs) each. Tesla C2070 has 448 parallel cores. Each SM contains a fast *shared memory*, which is shared by all of its SPs as shown in Fig. 1. It also has a read-only *constant cache* and *texture cache* which is shared by all the SPs on the GPU. A set of local 32-bit *registers* is available for each SP. The SMs communicate through the *global/device* memory. The global memory can be read or written by the host, and is persistent across kernel launches by the same application. Shared memory is managed explicitly by the programmers. Compared to the CPU, more transistors on the GPU are devoted to computing, so the peak floating-point capability of the GPU is an order of magnitude higher than that of the CPU, as well as the memory bandwidth due to NVIDIA's efforts on optimization.

2.2 CUDA programming model

At the software level, the CUDA model is a collection of *threads* running in parallel. The unit of work issued by the host computer to the GPU is called a *kernel*. CUDA program is running in a thread-parallel fashion. Computation is organized as a *grid* of *thread blocks* which consists of a set of threads as shown in Fig. 2. At instruction level, 32 consecutive threads in a *thread block* make up of a minimum unit of execution, which is called a *thread warp*. Each SM executes one or more thread blocks concurrently. A block is a batch of SIMD-parallel threads that runs on the same SM at a given moment. For a given thread, its index determines the portion of data to be processed. Threads in a single block communicate through the shared memory. CUDA consists of a set of C language extensions and a runtime library that provides APIs to control the GPU. Thus, CUDA programming model allows the programmers to better exploit the parallel power of the GPU for general-purpose computing.

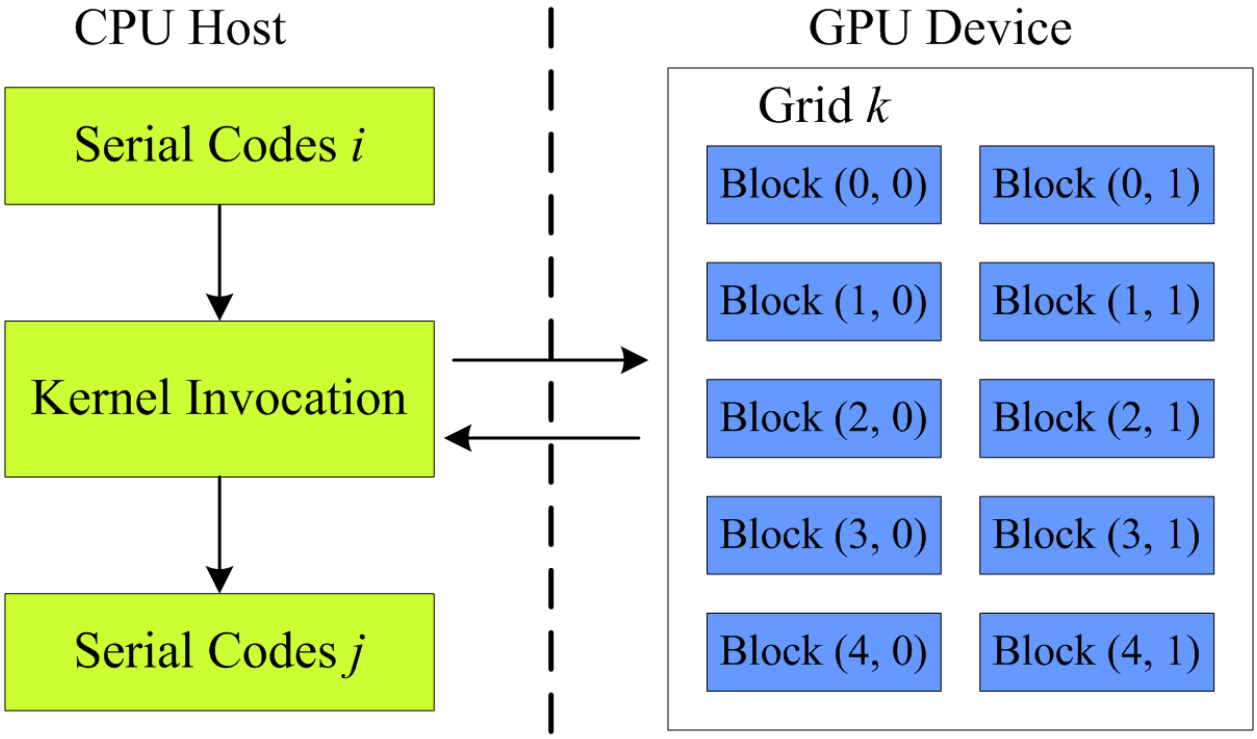

Fig2.Serial execution on the host and parallel execution on the device

3. Related work

3.1 Data mining

Data mining is the computational process of discovering patterns in large amount of data stored in databases, data warehouses, or other information repositories. Data mining techniques can be classified into several categories based on the nature of the problems being solved, such as association rules mining (ARM), classification, clustering, sequence mining etc.

3.2 Parallel data mining on GPUs

Parallel associations rules mining has been developed on distributed-memory parallel machines and shared-memory parallel machines. This apriori association rule mining algorithm is parallelized on symmetric multi-processing computer (SMPs). Our *parallel-Apriori* uses transactional database as the input data, avoiding the cost of pre-processing. The performance of our *parallel-Apriori* outperforms that of the Apriori algorithm in GPU architecture.

4. Typical data mining algorithm

In this section, we give a brief description of widely used data mining algorithm i.e. Apriori algorithm which has been well parallelized on shared-memory CPU architecture, and now attract the interests of GPU researchers.

4.1 Apriori

Association rules mining (ARM) [1] is one of the most widely used techniques in data mining and has tremendous applications in business, science, and other domains. The objective of ARM is to identify frequently co-occurring item sets in a database. It first finds all the item sets whose occurrences are beyond a minimum support threshold, and then generates rules from the frequent item sets based on a minimum confidence threshold.

Apriori is the most influential ARM algorithm due to its easy implementation and has been included in all the existing commercial and non-commercial data mining.

Pseudo-code of main function Input: Database, *D*, minimum support threshold, *min_sup*

Output: Frequent item sets, *L*

(1) *L*1 = find_frequent_1-itemsets(*D*);
(2) for (*k* = 2; L*k*−1 _= ∅; *k*++){
(3) *Ck*= apriori_gen(*Lk*−1);
(4) for each transaction *t* ∈*D*{ //support counting
(5) *Ct* = subset(*Ck*, *t*);
(6) for each candidate *c* ∈*Ct*
(7) *c*.count++;
(8) }
(9) *Lk*= {*c* ∈*Ckc*.count≥ *min_sup*};
(10) }

(11) return$L$ = U$k$$Lk$;
Pseudo-code ofsub-functions of AprioriProcedure apriori_gen(Lk−1) //candidate generation
(1) for each itemset$l1$ Lk−1
(2) for each itemset$l2$ ∈Lk−1
(3) if*((*$l1$[1] = $l2$[1])∧· · ·∧*(*$l1$[k −1] < $l2$[k −1])*)* then{
(4) c= $l1$ _ _$l2$;
(5) ifhas_infrequent_subset*(*c,Lk−1*)* then
(6) delete$c$;
(7) else add c to Ck;
(8) }
(9) return Ck;
Procedure has_infrequent_subset(c; Lk−1)
(1) for each (k −1)-subset s of c
(2) if$s$ /∈Lk−1 then
(3) return TURE;
(4) return FALSE;
Let I = {$i1$, $i2$, . . . ,im} be a set of m distinct attributes, or called *items*. Each transaction T in the database D has a unique identifier, and contains a set of items, in the form of _TID, $i1$, $i2$, . . . ,ip. An itemset with k items is called a *k-itemset*. The support of a *k-itemset*X is the fraction of the transactions in D containing X. X is a frequent itemset if X's support is greater than the user-specified minimum support threshold ε. The goal of frequent itemset mining is to find the complete set of frequent itemsets in a database. The *downward closure property* (any subset of a frequent itemset must also be frequent) is successfully introduced to cut down the number of candidates generated in each iteration. The detail of Apriori is presented in Tables 1a, 1b [1]. It performs a level-wise search. In each iteration (line 2 to 10), firstly, a set of k-candidates is generated by joining two frequent $(k-1)$-itemsets if they share a common $(k-2)$-prefix. A pruning procedure is invoked to eliminate any candidate which contains an infrequent subset; secondly, the support of every candidate itemset is counted by scanning the database. The loop terminates when no more frequent itemsets are discovered. The generation of candidate itemsets is computational intensive. The support counting to filter out frequent item sets is memory bound. Both of them are suitable for parallelization since the operations on different item sets are independent. The most challenging part is that the number of candidates or frequent item sets is unpredictable in any iteration.

5. Parallelization technique on CUDA-enabled platform

In order to make the best use of the architecture of the GPU and CUDA platform, we propose a scheme for data mining parallelization on CUDA-based platform. A whole task is assigned to CPU and inside GPU; the no of blocks and threads per block is decided before kernel starts as it depends on data size. We propose an algorithm to generate no of blocks and threads which give us maximum optimization with current Tesla graphics card. In CUDA platform, the proper combination of threads and blocks gives us much better result than other.

6. Algorithm implementation

In this section, we present the detailed CUDA implementation of typical data mining algorithm, *parallel-Apriori;* especially, we present how our proposed scheme is applied to this algorithm.

Parallel-Apriori Algorithm(1)

1. Generate item set in random manner using rand ( ) modulo (0 to n).
2. Initialize memory for itemset,compareset and result set using cudaMalloc().
3. For i from 1 to N(no of rows) Do
4. For j from 1 to C(no of columns)Do
5. itemset[i*c+j]=rand()%(0…n);
6. End For
7. End For
8. Copy the item set from CPU to GPU using cudaMemcpy().
9. Initialize result set to zero using cudaMemset().
10. Calling of apriori_GPU kernel with appropriate kernel launch parameters.
11. Apriori_GPU<<<No. of Blocks, No.of Threads per Block>>>(itemset,compareset,resultset);
12. Inside apriori_GPU kernel, division of item set is to be copied to shared memory.
13. Creating a shared variable using the keyword __shared__intshared_result_set[No. of threads per block]
14. Shared_result_set[] is used to store an intermediate result of every block.
15. Binding the data to be compared with the texture (read-only) memory usingcudaBindTexture().
16. Compare data present in read-only memory with divided item set of each block.
17. If compare is SUCCESS,
18. then shared_result_set[threadIdx.x]+=1;
19. If garbage value is encountered.
20. then quit the threads ;

21. Combining the shared_result_ set (shared memory) of each block for final result set inside global memory of GPU(resultset).
22. Return of GPU kernel to CPU.
23. Copy the result set from global memory to CPU result set.
24. Creating of association rules from CPU result set based on minimum support.
25. End

In this algorithm, we have introduced 3-D threading in such a way that it reduces maximum bank conflicts for the read-only memory.

Parallel-Apriori_Cluster Algorithm for GPUs in cluster (2)

1. Repeat step 1 to 7 of above algorithm (1) for generating itemset.
2. Divide and send itemset from CPU (Root Node) to other CPUs (Child Nodes) in cluster.
3. Considering any child node
4. Repeat all steps of above algorithm(1)
5. Repeat step 4 for every child nodes in cluster.
6. Sending of result set of CPU of each child node in cluster to root node.
7. Combine all resultset sent from child nodes and create final result set by root node.
8. Creating of association rules from CPU result set based on minimum support.
9. End

7. Experimental Result

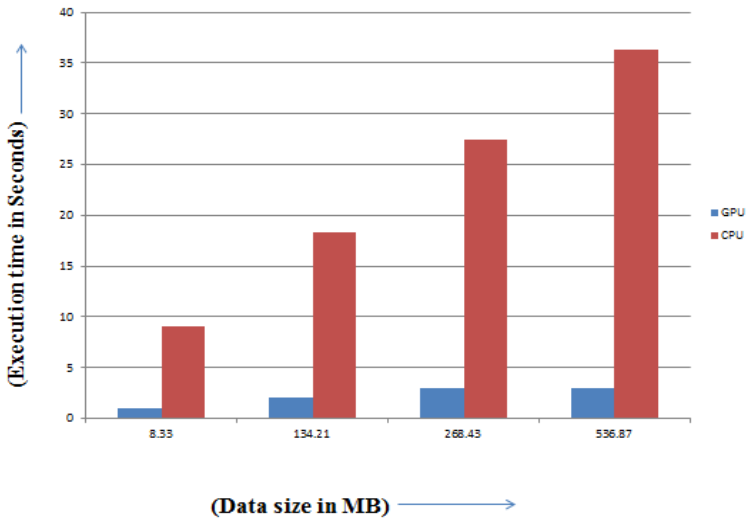

Fig 3. Comparison of Execution time for CPU and GPU

According to the above graph for large data set, CPU serial execution time is very large .For same date set, our optimized GPU parallel algorithm (1) executes in very less time as compared to CPU.

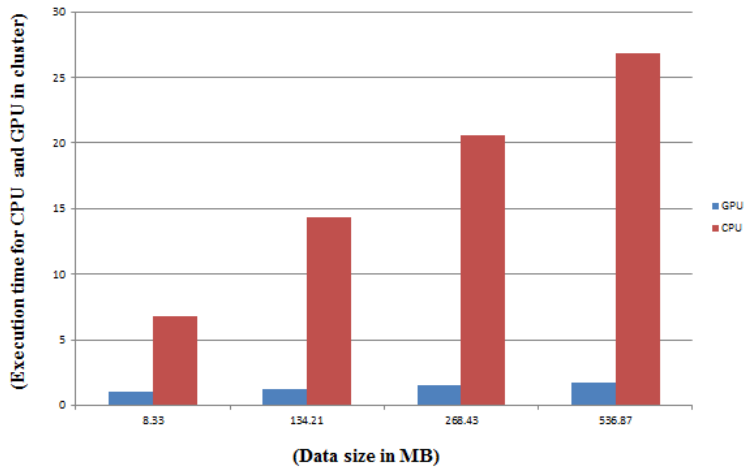

Fig 4. Comparison of Execution time for CPU and GPU in cluster

According to the above graph for large data set, CPU serial execution time is very large .For same date set, our optimized GPU parallel algorithm (2) executes in very less time as compared to CPU in cluster.

8 . Conclusion

In this paper, we have focused on parallelization of data mining applications on the GPU with CUDA architecture. To exploit the new parallel platform for data mining, We proposed optimized CUDA-based parallel techniques i.e. parallel apriori algorithms. Our experiment showed that *parallel –Apriori algorithm(1)*shows up to maximum 20 times speedup over an efficient serial Apriori program on a real-world data and *parallel –Apriori_Cluster algorithm(2) for* GPUs in cluster shows up to maximum 25 times speedup over an efficient serial Apriori program on a real-world data. As our size of dataset increases, our speedup also increases. From the above exploration, we believe that the GPU with CUDA parallel computing architecture will provide compelling benefits for data mining applications. In addition, its superior floating-point computation capability and low cost will definitely appeal to medium-sized business and individuals. Applications that used to rely on a cluster or a supercomputer to process will be solved on a desktop. In this way, We will get better performance in terms of execution time by optimizing our parallel algorithm based on release of new CUDA graphics cards.

References

[1]. Kamber M, Han J (2005) Data mining: concepts and techniques, 2nd edn. Morgan Kaufmann, SanMateo
[2]. Peng Y, Kou G, Shi Y, Chen ZX (2008) A descriptive framework for the field of data mining and knowledge discovery. Int J InfTechnolDecisMak 7(4):639–682
[3]. Olson D, Shi Y (2007) Introduction to business data mining. McGraw-Hill/Irwin, New York
[4]. Zhou L, Lai KK, Yen J (2009) Credit scoring models with AUC maximization based on the weighted SVM. Int J InfTechnolDecis Mak 8(4):677–696
[5]. Zhang Q, Segal RS (2008) Web mining: a survey of current research, techniques, and software. Int J InfTechnolDecis Mak 7(4):683–720
[6]. Zaki MJ (1999) Parallel and distributed association mining: a survey. IEEE Concurr7(4):4–25, Special issue on Parallel Mechanisms for Data Mining
[7]. Srivastava A, Han E, Kumar V, Singh V (1999) Parallel formulation of decision-tree classification algorithms. Data Mining Knowledge Discovery(3):237–261
[8]. Gaber MM, Yu PS (2006) Detection and classification of changes in evolving data streams. Int J InfTechnolDecis Mak 5(4):659–670
[9]. Liu Y, Pisharath J, Liao WK, Memik G, Choudhary A, Dubey P (2004) Performance evaluation and characterization of scalable data mining algorithms. In: 16th IASTED international conference on parallel and distributed computing and systems (PDCS). MIT, Cambridge, pp 620–625
[10]. NVIDIA (2008) CUDA programming guide 2.1. http://www.nvidia.com/object/cuda_develop.html

