Accelerating Parallel Frequent Itemset Mining on Graphics Processors with Sorting

Yuan-Shao Huang, Kun-Ming Yu, Li-Wei Zhou, Ching-Hsien Hsu, Sheng-Hui Liu

Accelerating Parallel Frequent Itemset Mining on Graphics Processors with Sorting

Yuan-Shao Huang[1], Kun-Ming Yu[1], Li-Wei Zhou[2], Ching-Hsien Hsu[1], Sheng-Hui Liu[2]
[1]Department of Computer Science and Information Engineering, Chung Hua University
Hsinchu, Taiwan
[2]School of Software, Harbin University of Science and Technology
Heilongjiang, China
m10002044@chu.edu.tw, yu@chu.edu.tw, 172851711@qq.com, chh@chu.edu.tw, hrbust.lsh@126.com

Abstract. Frequent Itemset Mining (FIM) is one of the most investigated fields of data mining. The goal of Frequent Itemset Mining (FIM) is to find the most frequently-occurring subsets from the transactions within a database. Many methods have been proposed to solve this problem, and the Apriori algorithm is one of the best known methods for frequent Itemset mining (FIM) in a transactional database. In this paper, a parallel Frequent Itemset Mining Algorithm, called Accelerating Parallel Frequent Itemset Mining on Graphic Processors with Sorting (APFMS), is presented. This algorithm utilizes new-generation graphic processing units (GPUs) to accelerate the mining process. In it, massive processing units of GPU were used to speed up the frequent item verification procedure on the OpenCL platform. The experimental results demonstrated that the proposed algorithm had dramatically reduced computation time compared with previous methods.

Keywords: Parallel Data Mining, Apriori, Graphic Processing Unit (GPU).

Acknowledgment

This paper is partial supported by the National Science Council of Taiwan, under grant number NSC 100-2632-E-216-001-MY3.

1 Introduction

With the development of information technology, all sectors of society have to handle massive explosions in their digital databases. The size of datasets has been increased exponentially in recent years in all fields as speed ups in processing and communication have greatly improved the capability for data generation and collection. Therefore the extraction of interesting and meaningful information has become a highly popular field of study. Data mining, known as Knowledge Discovery in Databases (KDD), is the process of automatically extracting useful hidden information from very large databases.

Frequent Itemset Mining (FIM) is one of the main tasks in data mining field which aims at finding interesting patterns from databases. The data in the database contains a set of items that are called transactions, each of which is labeled by a unique ID. The goal of FIM algorithms is to generate all possible itemsets and find the most frequently-occurring subsets that are bought together in not less than a given, user-specified threshold. The number of itemsets occurrences is called support, and the threshold is minimum support.

In recent years, parallel data mining algorithms has been attracted more and more attention. Modern Graphics Processing Units (GPU) have evolved into powerful processors that not only support typical computer graphics tasks but are also flexible enough to perform general purpose computations [6] [7] [10] [13]. Recently, there has been a trend to accelerate computational data mining algorithm on a GPU + CPU heterogeneous system which the GPU acts as the computation accelerator. Nowadays, high level languages have emerged to support easy programming on GPUs. OpenCL [11] seems to be emerging as an open and cross-vendor standard for exploiting computational power of both CPUs and GPUs. However, many classical algorithms have been proposed for single CPU architectures [4] [5]. If CPU-GPU hybrid architectures are used to speed up the mining purpose, it will improve performance.

In order to best utilize the power computing resources offered by GPUs and extend traditional, CPU-based data mining algorithms for mapping to CPU-GPU hybrid architecture, scalable GPU-based parallel evaluation model for speeding up the computing process was implemented in this study. A solution is proposed that would have all frequent itemsets sorted after constructing the TID table which will then greatly reduce the candidate itemsets when using CPU architecture. Suitable GPU threads were allocated after sorting the itemset in decreasing order. Therefore, the times of the checking process were reduced, and support counting was time efficient. The compared results showed that efficiency had been significantly improved.

The remainder of this paper is organized as follows. Section 2 provides an overview of data mining, describes the Apriori algorithm [2] [12], the Multi-core Apriori Transaction Identifiers algorithm (MATI) [14] and the Candidate Slicing Frequent Pattern Mining (CSFPM) [9] algorithm. The proposed algorithm Accelerating Parallel Frequent Itemset Mining on Graphics Processors with Sorting (APFMS) is introduced in Section 3. Section 4 presents the experimental results. In section 5, the conclusion of the paper is given.

2 Related Works

Data mining is a technology used to determine special relationships hidden in large amounts of data, and efficiency is especially crucial for an algorithm finding frequent item sets from a large database. Many methods have been proposed to solve this problem. Among them, parallel computing has become a popular trend, such as grid, cloud, multi-core or GPU computing platforms.

In this section, the most relevant studies, including Apriori algorithms, the Multi-core Apriori Transaction Identifiers (MATI) algorithm and the Candidate Slicing Frequent Pattern Mining (CSFPM) algorithm, are briefly reviewed.

2.1 Apriori algorithm

The Apriori Algorithm was proposed by R. Agrawal and R. Srikant in 1994 [2]. It's a classic algorithm for frequent itemset mining and association rule learning over transactional databases. It uses a level-wise behavior, which involves a number of dataset scans equal to the size of the largest frequent itemset. Apriori iteratively generates K+1 frequent itemsets by joining frequent K-itemsets. This step is candidate generation. First, the set of frequent 1-item sets is found by scanning the database to assess the count for each item, and then collecting the items that satisfy minimum support, denoted L_1. L_1 is used to find L_2, and L_2 to find L_3, and this continues, until all frequent itemsets are found. After generating each new set of candidates, the algorithm scans the database to count the number of occurrences of each itemsets. This step is called support counting. The Apriori Algorithm stops when all the frequent item sets have been generated. However, the algorithm scans datasets many times and may generate redundant candidate itemsets. When there are many frequent 1-item sets and the frequent patterns are very long, the number of generated candidate itemsets increases significantly. Therefore, the efficiency of the algorithm deteriorates significantly.

2.2 Multi-core Apriori Transaction Identifiers

Lately, novel algorithms on frequent pattern mining have been proposed. Yu et al. propose the MATI algorithm [14] to speed up the computation time of data mining by enhancing the efficiency of Apriori on multi-core architecture. The algorithm utilizes the AprioriTID algorithm [1] [8] at the first pass to shorten the database scanning process by creating the Transaction Identification (TID) tables. In MATI algorithm, two strategies are proposed, Item set Block and Task Dispatches. In the process of generating candidate in MATI, frequent itemsets are divided into multiple blocks, all frequent itemsets with the same prefix are put into the same block, and candidates are generated in the same block only. The frequent itemsets in the same itemset block will be generated on the same core avoiding data distributed on different cores.

2.3 Candidate Slicing Frequent Pattern Mining

Candidate Slicing Frequent Pattern Mining (CSFPM) [9] is proposed by Lin et al. This algorithm uses the Transaction Identification (TID) table to store the itemsets which shorten the database scanning process, as shown in Table 1. Corresponding to the TID table, two elements, the TID value table and the TID index table are created with GPU-FPM [15]. In Fig.1, TID value table stores the itemsets associated with their transaction numbers in GPU threads, TID index table stores the location numbers in GPU threads corresponded to its itemset. As numbers in the table starts from 0, the first itemset A contains 1, 2 in the TID value table, then in the TID index table the number is 0 to 1, and the itemset B contains 1, 2, 3, so the index number range 2 to 4, this process continues until all the itemsets have been dispatched.

The CSFPM algorithm divides candidate into smaller units with parallel computing on each GPU thread. Each GPU thread is only responsible checking for its own one candidate itemset in the TID value table. The GPU thread only checks and compares the numbers whether are equal or not. If the values are equal, then returns result 1, else result 0 is returned instead. The checking process is shown in Fig. 2. Item A and Item B has the common transaction value 1 and 2, then the output returns double 1. After the first computation finished, the result was returned with an array of 1110100, and then the number of 1 was calculated; all the numbers of 1 were summed to compare with the minimum support checking whether the candidate itemset was frequent or not, as shown in Fig.2 and Table 2.

The CSFPM algorithm is an implementation on CPU-GPU architecture based on the Apriori algorithm, and reducing the counting time of the GPU support to speed up the total computing time. In order to achieve better load balancing performance, the algorithm parallelizes the candidate itemsets and divides them into the GPU threads, assign one thread only checking to its own one transaction in a candidate item. This strategy can reduce the processor waiting time since the load between processing units is more balanced.

Table 1. TID table

Items	TID Value		
A	1	2	
B	1	2	3
C	1	4	

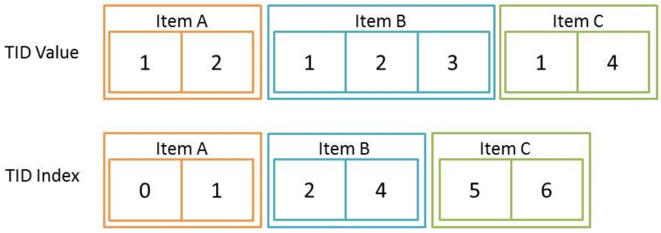

Fig. 1. TID value and TID index tables

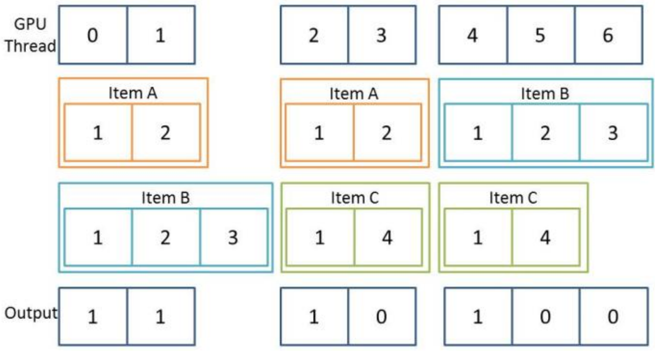

Fig. 2. Candidate items of GPU computing

Table 2. Computing the candidate number of repetitions per item

Item	Time
AB	2
AC	1
BC	1

3 Proposed Algorithm

Since the information in a data stream is large, the implementing of an efficient algorithm based on the Apriori algorithms has been the focus of many researchers. Due to the great advantages of GPUs, they have evolved become highly parallel, multithreaded with tremendous computational horsepower and very high memory bandwidth. In this paper, an Accelerating Parallel Frequent Itemset Mining on Graphics Processors with Sorting (APFMS) algorithm is presented. It is based on the advantages of CSFPM and MATI utilizing the sorting of the 1-frequent item sets from the dataset after constructing the TID table in order to cut down on computing time for better performance.

The APFMS algorithm was optimized by using the dividing method of CSFPM and the merging method of MATI. The MATI algorithm uses the follow technique. Unlike the original Apriori algorithm, the (k+1)-itemset is generated only by the k-itemset with the same k-1 prefix itemset. Fig. 3 illustrates the procedure for MATI, in the example, k=2 and itemset AB and AC are the 2-itemsets which were frequent, when the 3-itemset was be generated, the 2-itemset merged with each other to satisfy the (2-1=1) prefix, as AB and AC had the same prefix. A, thus AB and AC merged with each other to generate candidate 3-itemset, ABC. However, the AC and BC did not merge with each other as they did not have the same prefix. Therefore, it was not necessary to generate all possible itemsets as in the original Apriori algorithm; the MATI algorithm filtered the redundant itemsets which were not frequent and only merge the useful frequent itemsets.

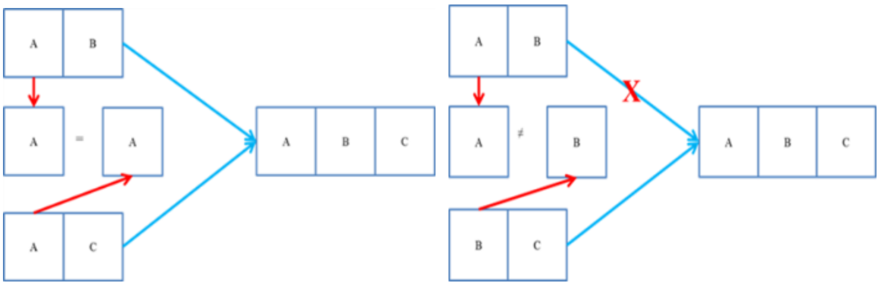

Fig. 3. Generating itemsets in MATI

GPU threads were allocated by the number of TID values in the Candidate Slicing Frequent Pattern Mining (CSFPM) algorithm. Therefore, the higher the TID value, the more GPU threads were allocated. Fig. 4 shows the processing of CSFMP in comparing the TID value. Owing to the varying TID values, the GPU threads did not compute efficiently. As a result, the APFMS algorithm proposed a strategy sorting all 1-frequent itemsets after constructing the TID table with the support of 1-frequent itemset each and in the decreasing order, and then the GPU threads were allocated according the new TID value after sorting. Therefore, the time complexity was reduced when the GPU checked the itemsets; whether they had the same prefix in order to merge so that the next rank candidate itemsets could be generated. The processing of the APFMS algorithm when comparing the TID values is presented in fig. 5.

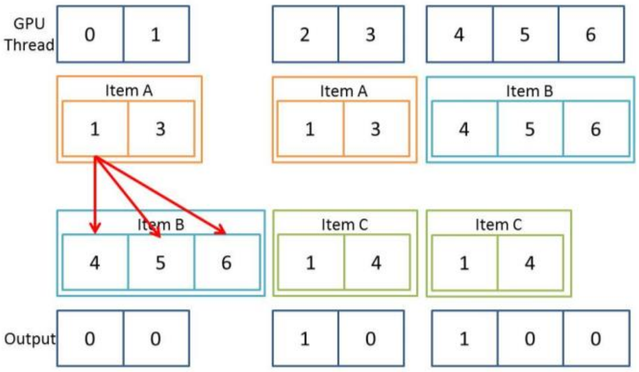

Fig. 4. CSFMP method

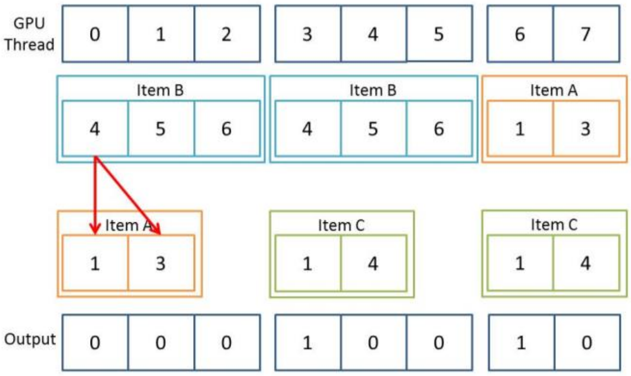

Fig. 5. APFMS method

As in fig. 5, when compared with the CSFMP algorithm in fig. 4, the APFMS algorithm sorted all frequent itemsets in descending order after constructing the TID table in CPU cores, and the results were transferred to the TID value tables in the GPU threads in the same order; Then, the larger itemsets were mapped in the front in the TID value tables. Due to this, the numbers of checking process support counting steps were reduced. Therefore, the support counting time will reduce as well. Following, are several basic steps for applying the APFMS algorithm.

Procedure

1. First, Scan the database, transform the transaction items to a TID table and build the corresponded TID value table and TID index table.
2. Calculate the Tidset and count the support number of each 1-candidate itemset, prune the non-valid itemsets and generate the 1-frequent itemset.
3. Sort the 1-frequent itemset in decreasing order.
4. Use the MATI merge function in CPU cores, that all K+1 itemsets are generated from K itemsets with the same K-1 prefix
5. Let the GPU cores calculate the support counting step, all datasets are transfer from CPU cores to GPU cores, dynamic GPU threads are allocated to calculate the item in the TID value table and its corresponding TID index table.
6. Return the 0-1 arrays from GPU cores to CPU cores, use CPU cores to calculate the number of 1 and compare with the threshold.
7. Generate the next rank candidate itemsets, repeat the step above until all the frequent itemsets are found.

The pseudo code of APFMS is shown in fig. 6.

CPU: Main Function (C++)
Database is *D,* merge round is *N*, candidate itemsets in *N* round is *Cn,* Frequent pattern is *Fn.* *Cn*is candidate patterns and more than 1. *Fn* is frequent pattern and more than 1.

```
GPU thread is GT.

Input a threshold
Compile the .CL and build GPU device
Scan and structure a TID table

if ( the number of orders in any item <Threshold ){
Delete the item
}

QuickSort(frequent itemsets, left, right)    //from largest to smallest

Transform the TID table into a TID Value and TID Index
Allocate memory space in GPU for TID Value and TID Index
Stores arrays TID Value and TID Index into GPU memory

For ( K = 2 ; ; K++ ){
Using MATI to generate Cn

Do {
If ( size of Cn>GT )
      Portion the Cn= PCn
Allocate memory space in GPU for PCn
Store PCn in GPU
Allocate memory space in GPU to save the results
Wait until GPU finishes its program execution.

Calculate the number of nonzero entries of each PCn comparison.
If ( this number ≧ the threshold )
      The pattern is frequent and save into Fn
} while(size of Cn ≠ 0)
If(candidate cannot be combined )
      Break;
Else
      Fn Combine the candidate to next level (N+1).
}

GPU: Kernel Function ( .CL )

Receive the candidate
Intidx = get_glonal_id(0)    // idx = GPU Thread id number
Intgpu_thread_value = identify the corresponding TID Value
if (gpu_thread_value == TID Value of Other candidate pattern )
      result = 1;
Else if (gpu_thread_value>TID Value of Other candidate pattern )
      result = 0;
```

Fig. 6. The pseudo code of APFMS

4 Experimental Results

In this section, the experiments are conducted to verify the performance of the APFMS method and the comparison with CSFPM on GPU are presented. In the experiments, the same hardware and software configurations were used and the Input data was from the IBM data generator [3] shown in Table 3.

Table 3. Hardware and software configurations

Items	Description
CPU	Intel Core i7-3960X 3.3GHz
Memory	16G DDR3 memory
GPU	NVidia GTX 580 1536MB GDDR5
OS	Microsoft Windows 7
Compiler	Microsoft Visual C++ 2010
SDK	OpenCL 1.1

In the experiment, the computation time shown in Fig. 7 indicates that with the dataset (T10I4D100KN100K) and the threshold 0.1%, the APFMS algorithm needed less time than the CSFPM algorithm, and hence resulted in higher efficiency with 300% speedup when the GPU was set at 65536 threads.

This experiment compared the computation time of APFMS with that of CSFPM with the same threshold of 0.1%, but with different methods, number of transactions and number of threads (153837). As in Fig. 8, APFMS performed better than CSFPM on the same platform.

Fig. 9 shows the execution time of CSFPM and APFMS with different GPU threads when the dataset was T10I4D1000KN100K, and the threshold 0.1%. In this experiment, with the transaction numbers increasing, the execution time of the APFMS increased as well. However, with the number of GPU threads getting higher than 153837, the checking times and the time complex of CSFPM were increasing. The CSFPM did not even finish computing when the number of GPU threads was set as more than 153837.

Using the same dataset as the CSFPM algorithm, the APFMS algorithm had a better speedup performance. Further, with the number of GPU threads increasing, the computing time increased as well, causing the CPU cores having to finish computation with more space and bandwidth. However, when the computing time delay exceeded the GPU thread limit time, the CSFPM algorithm stopped and jumped out of the GPU computing. By contrast, the APFMS went on go accelerating computation until finished. Therefore, the APFMS was proven to be more suitable with better performance.

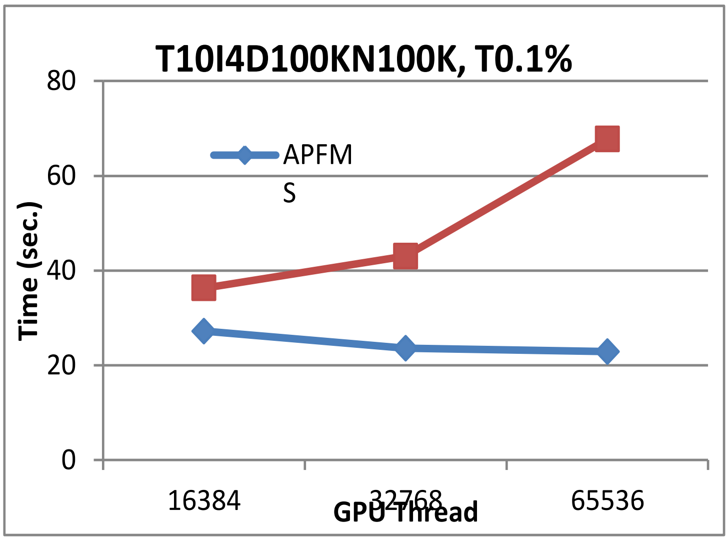

Fig. 7. Runtime with different number of GPU threads

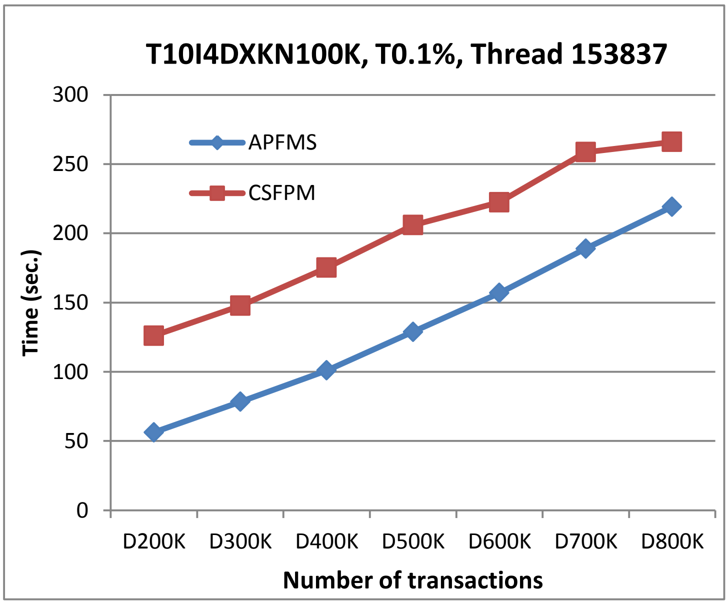

Fig. 8. Runtime with same number of GPU threads

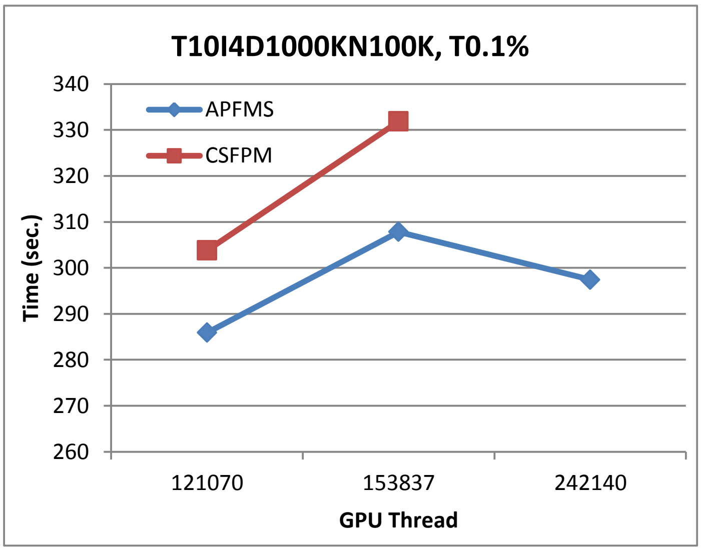

Fig. 9. Runtime with different number of GPU threads using APFMS and CSFPM with dataset T10I4D1000KN100K

5 Conclusions

Recently, with GPU providing extremely high parallelism and high bandwidth in memory transfer, its hybrid architectures are starting to be used for data mining. However, it is not easy to parallelize existing algorithms to achieve good performance on these hybrid architectures. Therefore, it is necessary to examine to what extent traditionally CPU-based data mining problems can be mapped to the GPU architecture.

In this paper, the Accelerating Parallel Frequent Itemset Mining on Graphics Processors with Sorting (APFMS) algorithm is proposed in order to improve the performance of a CSFPM.APFMS algorithm based on the advantages of CSFPM and MATI. The sorting of the 1-frequent item sets from the dataset after constructing the TID table is used in order to cut down computing time and the time complexity of GPU computing. The experiment results indicated that when the dataset was T10I4D100KN100K, with a threshold of 0.1%, the implementation had a 300% speed up compared the CSFPM, and a better load balancing performance was achieved with the increase of transaction numbers.

Future work on the research includes utilizing different types of GPU for better performance. As a result, in order to achieve heterogeneity in the GPU architecture, the different performance allocation will be considered with different candidate itemsets with different types of GPU.

References

[1] R.Agawal, Imilinski, T., and Swami, A. “Mining Association Rules between Sets of Items in Large Database,” Proceeding of the 1993 ACM SIGMOD International Conference on Management of Data, Vol. 22, Issue 2, pp. 207-216 (June 1993)

[2] R. Agrawal, and R. Srikant, “Fast algorithms for mining association rules,” in International Conference on Very Large Data Bases, pp. 487-499 (1994)

[3] R. Agrawal, and R. Srikant, “Quest Synthetic Data Generator. IBM Almaden Research Center, San Jose, California,” (2009)

[4] FerencBodon, “A trie-based APRIORI implementation for mining frequent item sequences,” OSDM '05 Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, 2005, pp.56-65.

[5] Christian Borgelt, “Frequent Item Set Mining,” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,” Vol. 2, Issue 6, pp.437-456 (1-2 Dec. 2012)

[6] Wenbin Fang, Mian Lu, Xiangye Xiao, Bingsheng He, QiongLuo, “Frequent itemset mining on graphics processors,” DaMoN '09 Proceedings of the Fifth International Workshop on Data Management on New Hardware, pp. 34-42 (2009)

[7] Ana Gainaru, Emil Slusanschi, Stefan Trausan-Matu, “Mapping data mining algorithms on a GPU architecture: a study,” ISMIS'11 Proceedings of the 19th international conference on Foundations of intelligent systems, pp. 102-112 (2011)

[8] Zhi-Chao Li, Pi-Lian He and Ming Lei “A high efficient AprioriTid algorithm for mining association rule,” Machine Learning and Cybernetics, vol. 3, no. 3, pp. 1812-1815 (2005)

[9] Che-Yu Lin, Kun-Ming Yu, Wen Ouyang, Jiayi Zhou, “An OpenCL Candidate Slicing Frequent Pattern Mining algorithm on graphic processing units,” In proceeding of: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp. 2344-2349 (2011)

[10] Wenjing Ma, GaganAgrawal, “A translation system for enabling data mining applications on GPUs” ICS '09 Proceedings of the 23rd international conference on Supercomputing, pp. 400-409 (2009)

[11] OpenCL.“OpenCL,”http://www.khronos.org/opencl/

[12] J. Park, M. Chen, and P. Yu, “An effective hash-based algorithm for mining association rules,” ACM SIGMOD Record, vol. 24, no. 2, pp. 175-186 (1995)

[13] Silvestri, Claudio, “gpuDCI: Exploiting GPUs in Frequent Itemset Mining,” 2012 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 15-17, pp. 416-425 (Feb. 2012)

[14] Kun-Ming Yu and Shu-Hao Wu. “An Efficient Load Balancing Multi-core Frequent Patterns Mining Algorithm,” 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp.1408-1412 (2011)

[15] J. Zhou, K.-M. Yu, and B.-C. Wu, “Parallel frequent patterns mining algorithm on GPU,” IEEE International Conference on Systems Man and Cybernetics, pp. 435-440 (2010)