# GPU-Accelerated Apriori Algorithm

Hao JIANG[a], Chen-Wei XU[b], Zhi-Yong LIU[c], and Li-Yan YU[d]

*School of Computer Science and Engineering, Southeast University, Nanjing, China*
*[a]hjiang@seu.edu.cn, [b]wei1517@126.com, [c]zyliu278@163.com, [d]756250742@qq.com*

**Abstract:** This paper propose a parallel Apriori algorithm based on GPU (GPUApriori) for frequent itemsets mining, and designs a storage structure using bit table (BIT) matrix to replace the traditional storage mode. In addition, parallel computing scheme on GPU is discussed. The experimental results show that GPUApriori algorithm can effectively improve the efficiency of frequent itemsets mining.

## 1 Introduction

The overhead of association rule mining mainly comes from the generation and processing of frequent itemsets. In order to get frequent itemsets, a large amount of computation and enough storage is needed. Therefore, the improvement of association rule mining algorithm should focuse on how to improve the processing speed and reduce the storage space. This paper combines Apriori algorithm with GPU Technology, and introduces the algorithm of GPUApriori. On the one hand, a compressed storage scheme is designed to simplify the connection operation and reducing the storage usage of the algorithm. On the other hand, in order to fit the parallel computing architecture of GPU [4], parallelization of the Apriori algorithm is proposed.

## 2 Improvement of Storage Structure

### 2.1 Apriori Algorithm

The Apriori are based on the property that subset of a frequent itemsets must be frequent, it uses an iterative procedure with the data subsets, and utilizes large frequent itemsets generated in the preceding stage to produce frequent itemsets in the next stage[7]. In Apriori algorithm, there are two parts: Firstly, generated all 1-item candidates, and counted their number of occurrences to achieve 1-item frequencies. Secondly, joined the k-item frequencies to generate (k+1)-item candidates, tested their amount and quality (fit the above property) to generate (k+1)-item frequencies. The algorithm iterates the process until the number of the (k+1)-item candidates is 0.

### 2.2 Maintaining the Integrity of the Specifications

Apriori algorithm mainly uses horizontal and vertical storage structure. This paper introduces another mode (the bit table (BIT) matrix). This way can be well applied to the multi-core concurrency of GPU, and has the ability to replace the complex joined operation of Apriori with the operation "&" between two matrices. Therefore, BIT matrix Storage Structure can improve algorithm efficiency. Figure 1 shows the difference between the BIT matrix storage structure and the traditional one.

Horizontal Storage Structure

| Trans IDs | Items IDs |
|---|---|
| 1 | ABCD |
| 2 | ABD |
| 3 | ACD |
| 4 | BCD |

Vertiacl Storage Structure

| Items IDs | Trans IDs |
|---|---|
| ABD | 1,2 |
| ACD | 1,3 |
| BCD | 1,4 |

BIT Matrix1 Storage Structure

| Item IDs / Trans IDs | A | B | C | D |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 |
| 3 | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 1 | 1 |

**Figure 1.** BIT Matrix and Traditional Storage Structure

The BIT matrix can be well applied to the multi-core concurrency of GPU. In these matrices, one column uses one bit to represent, and every column (item) information for each row (transaction) is stored by some "unsigned integer". The number of "unsigned integer" depends on the how many the column in the dataset. Figure 2 shows the state of the BIT matrix in memory.

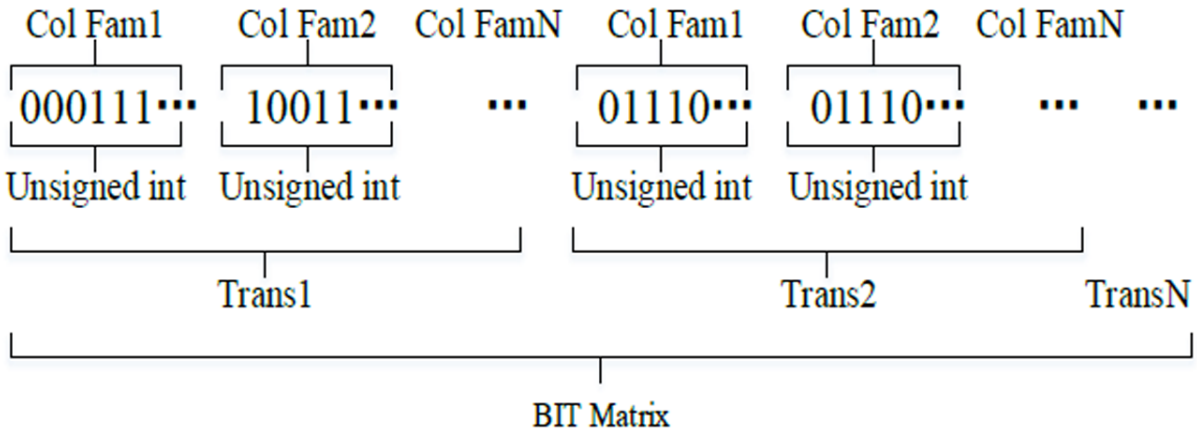


**Figure 2.** Storage Form of BIT Matrix in Memory

This structure is different from the general sense of the array, so that a new kind of read and write operation logic need to be fulfill. Read and Write operations are made using a tag variable ("unsigned integer"), which value is "0x80000000". Through the shift manipulation of the tag variable and the operation "&" between the shifted tag variable and the corresponding variables in the BIT matrix, the reading operation can be achieved. Using the relevant variables table in the matrix and shifted tag variable to finish the operations of "|=" and "-=", then the value of "0" or "1" can be write in the BIT matrix.

The Apriori algorithm based on the BIT matrix storage structure also needs to complete the operation of converting the traditional storage structure into the BIT form. Firstly, in the conversion process, the array of BIT matrix is initialized, and set the value of all elements to "0". Then, needed to traverse every transaction in the database, and wrote "1" to the corresponding positions in the BIT when reading the items in each transaction, the operation of updating the number of per items can be performed simultaneously. Finally, 1-item candidates can be built when this procedure is over.

## 3 Gpuapriori Algorithm Based on BIT Matrix

### 3.1 Candidate Itemsets Generation

The candidate itemsets generation based on BIT is equivalent to the join step in Apriori algorithm. It uses the operation "&" between the k-item frequent BIT (FBIT) and the 1-item FBIT to generate (k+1)-item candidate BIT (CBIT), and completes the support count of (k+1)-item CBIT. Figure 3 describes the process of candidate itemsets generation based on BIT matrix in every tread.

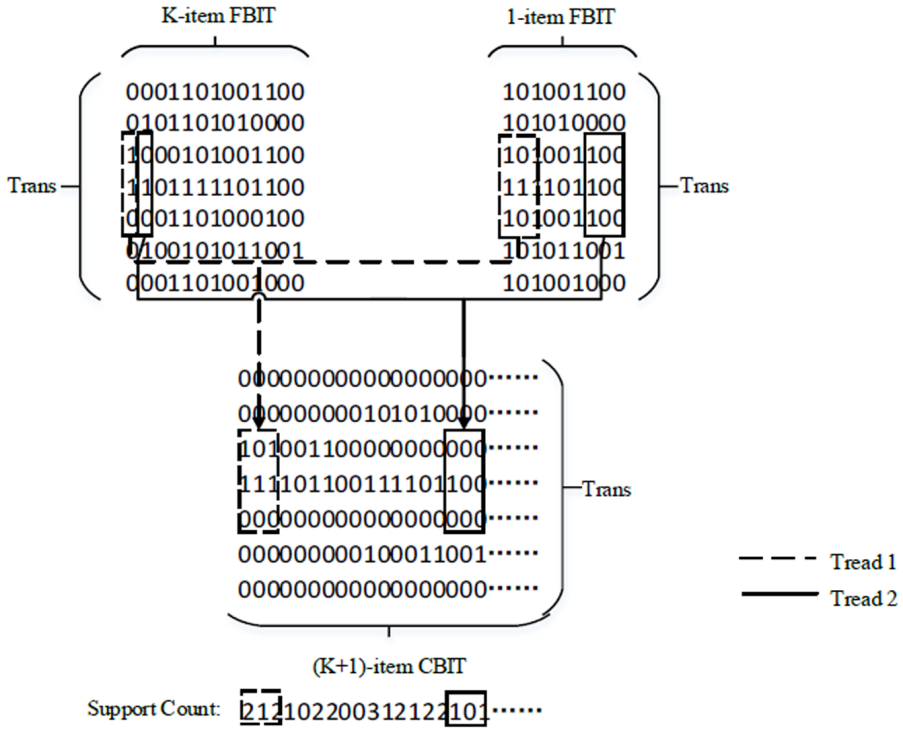

**Figure 3.** The Process of (k+1)-item CIT generation

BIT matrix is suitable for Gird-Block-Thread structure of GPU, multiple threads can be parallel processing of each part of the table. Algorithm 3-1 shows a single GPU thread executes the operation of candidate itemsets generation based on BIT matrix.

**Algorithm 1:** Join Step of GPUApriori

Input: k-item FBIT and 1-item FBIT
Output: (k+1)-item CBIT, Support Count of (K+1)-item CBIT: S1

1. 1-item FBIT is divided into several small F1 matrixes according to the size of 32*100, k-item FBIT divided into several FK columns.
2. Every GPU thread parallel processes a F1 matrix and a FK columns and achieves a CK+1 matrix (part of (k+1)-item CBIT) and its support count (refer to Figure 4).
3. GPU thread get the matrix CK+1 and set it to the corresponding position of (k+1)-item CBIT (Atomic operation).
4. Returned to 2 until all F1 matrix and FK column are processed completely.

### 3.2 Frequent Itemsets Generation

This section focuses on the pruning step of GPUApriori algorithm. The core of the pruning step is to sort out a Range table, which records all the indexes of the (K+1)-item FBIT, and then filters the (K+1)-item CBIT according to the index set.

Algorithm 3-2 describes an algorithm for filter the (K+1)-item CBIT to achieve the (K+1)-item FBIT.

**Algorithm 3-2:** Pruning Step of GPUApriori

Input: (K+1)-item CBIT, Support Count of (K+1)-item CBIT: S1, Minimum support: T
Output: (K+1)-item FBIT, Support Count of (K+1)-item FBIT: S2

1. Parallel set (K+1)-item CBIT's badsets to 0 in S1, Initialize the index table Range
   //badsets includes many items that do not accord with ascending order or contain duplicate, such as "121", "aba", "111".
2. Parallel sorting the items in S1 and Range according to the S1.
3. Searching for the first S1[i] greater than or equal to T. The index of the (K+1)-item FBIT can be achieved in the Table Range (Range[i~n]).
4. Parallel reset the (K+1)-item CBIT according to the Range[i~n], (K+1)-item FBIT and S2 can be obtained.

### 3.3 The Whole Process of Gpuapriori

The algorithm needs the cooperation between CPU and GPU. It includes four steps: reading the data and generate the 1-item CBIT, building the 1-item FBIT, joining operation between two FBIT and pruning steps of the CBIT, The flow chart of the GPUApriori is shown in the Figure 4.

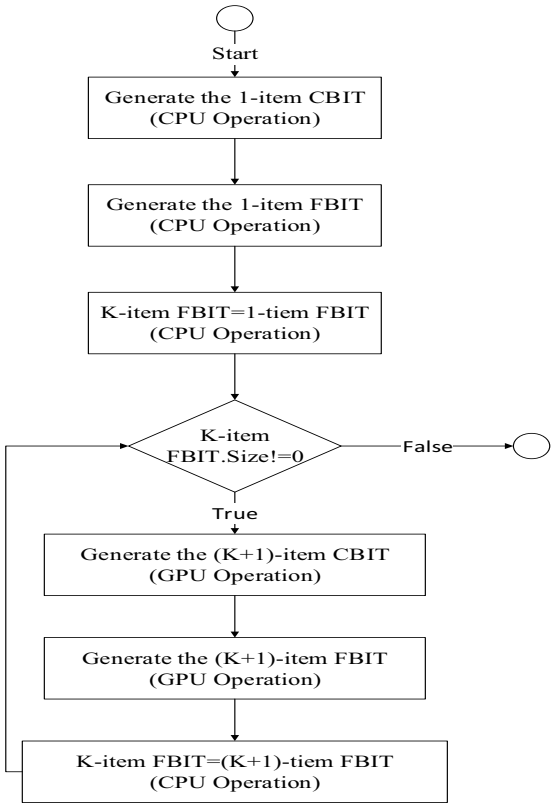


**Figure 4.** The Flow Chart of the GPUApriori

## 4 Experimental Result

### 4.1 Experimental Environment

The experiment of this paper is based on the GPU platform. The detailed information of this platform is listed in TABLE I.

**Table 1.** Table Type Styles

| | |
|---|---|
| **Hardware Platform** | CPU: Intel® CoreTM i7-4790K Processor<br>GPU: NVIDIA Quadro K600<br>Memory: 8GB(2*4GB Kingston 1600MHz)<br>Disk: ST1000DM003-1CH162-931.51GB |
| **Software Platform** | OS: Windows 7 Pro<br>CUDA: CUDA Toolkit 7.5<br>Visual Studio 2013 |

### 4.2 Experimental Method

This experiment tests the performance of two algorithms, 1) Traditional Apriori algorithm based on CPU, named CPUApriori; 2) Apriori algorithm based GPU and named GPUApriori. Experimental measure the performance of the algorithms with different support count from the same data set.

Our benchmark datasets are from the Frequent Itemset Mining Repository. Selected Retail, Mushroom, Chess, TI0I4D100K experimental datasets, and those datasets are presented in TABLE 2.

**Table 2.** Experimental Datasets

| Database | Type | #Items | #Trans |
|---|---|---|---|
| Retail | Sparse | 16,470 | 88,162 |
| T40I10D100K | Sparse | 1,000 | 100000 |
| Chess | Dense | 75 | 3196 |
| Mushroom | Dense | 119 | 8,124 |

### 4.3 Experimental Result

The experimental results are shown in Figure 5, Figure 6, Figure 7 and Figure 8.

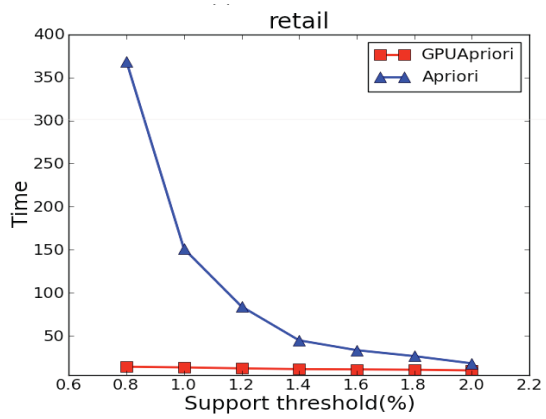


**Figure 5**

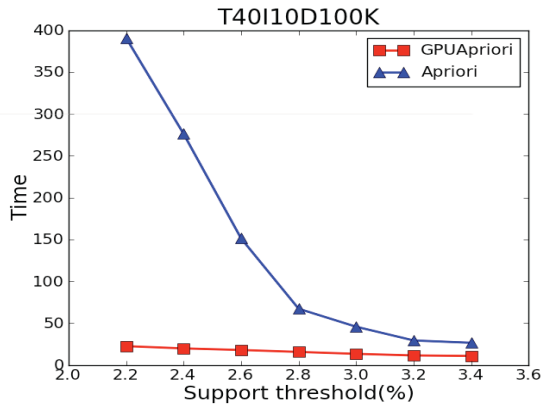


**Figure 6**

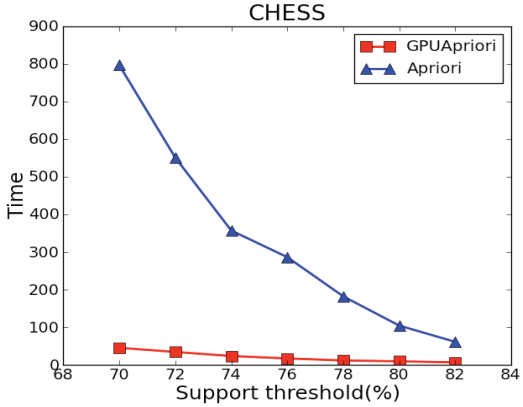


**Figure 7**

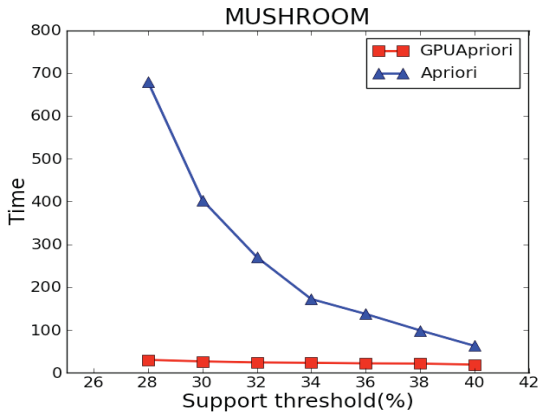


**Figure 8**

Figure 5, Figure 6 shows the running time of the two algorithms in the Retail and T40I10D100K experimental datasets with different support levels. Found from the comparison in contrast with the GPUAriori algorithm and the algorithm on the CPU, GPUAriori has a greater advantage. Moreover, when the minimum support is smaller, the speedup of GPApriori is higher than that of the another. Simultaneous. Figure 7 and Figure 8 show that the GPApriori algorithm has a much more efficient acceleration in a dense data set.

The Experimental shows that the GPUApriori algorithm performs well in performance testing, In addition, When the Apriori algorithm is calculating the larger and more dense dataset, GPU compared with the CPU algorithm will have a faster acceleration.

## 5 Discussion

In this paper, the association rules mining algorithm in data mining is studied, and presented the parallel processing of joining step and pruning step in GPUApriori based on BIT matrix. The parallel GPUApriori algorithm is implemented in the experiment, and the results show that the GPUApriori algorithm has better running efficiency, which reflects the parallel acceleration effect of GPU. In future, research should focus on the other algorithms of frequent itemsets mining, in order to develop other parallel algorithms based on GPU.

## References

1. JIAWEI HAN. Data mining concepts and techniques [M]. Morgan Kaufmann Publishers, 2011-07-25.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
2. Agrawal R, Strikant R. Fast algorithms for mining association rules. 20[J]. Proc. int. conf. very Large Database Vldb, 1994,23(3):21-30.K. Elissa, “Title of paper if known,” unpublished.
3. Han j, Kamber M, Pei J. Data mining, southeast Asia edition: Concepts and techniques [M]. Morgan kaufmann, 2006.

4. Owens J D, Houston M, Luebke D, et al. GPU computing.
5. Nvidia B. NVIDIA CUDA Programming Guide [J]. 2012.
6. Bell, Jason. Apache Spark [M] Machine Learning: Hands-On for Developers and Technical Professionals. John Wiley & Sons, Inc, 2015:273-314.
7. Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C] Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012:141-146.
8. J. Guo and Y.-g. Ren, Research on Improved A Priori Algorithm Based on Coding and MapReduce," in Web Infromation System and Application Conference (WISA), 2013, pp.294-299.
9. Sacasere A, Omiecinski E, Navathe S B. An Efficient Algorithm for Mining Association Rules in Large Databases [J]. Vidb Journal. 1995:432-444.
10. Park J S, Chen M S, Yu P S/ Efficient parallel data mining for association rules[C] International Conference on Information and Knowledge Management.ACM, 1995:31-36.
11. Qiu H, Gu R, Yuan C, et al. YAFIM: A Parallel Frequent Itemset Minging Algorithm with Spark[C] Parallel & Distributed Processing Symposium Workshops.IEEE, 2014:1664-1671.