The Fourteenth IEEE International Conference on e-Business Engineering

A Parallel FP-growth Algorithm Based on GPU

Hao JIANG
School of Computer Science and Engineering
Southeast University
Nanjing, China
E-mail: hjiang@seu.edu.cn

He MENG
School of Computer Science and Engineering
Southeast University
Nanjing, China
E-mail: menghe2012@sina.com

Abstract—**This paper proposes and implements a parallel scheme of FP-growth algorithm and implements this parallel algorithm (PFP-growth algorithm). Experimental results show that, compared with FP-growth algorithm, PFP-growth algorithm is more efficient, and the larger the data set is, the lower the support threshold is, the more remarkable the speedup is.**

Keywords-frequent itemset mining; FP-growth; GPU; parallel computing

I. INTRODUCTION

Association rule mining is an important approach in data mining. Association rule mining means mining potentially valuable links between data items from a large number of data items. In general, association rule mining can be divided into two sub-problems: frequent item sets generation and rule generation. Frequent itemsets mining is the core of association rules mining algorithm, its performance greatly determines the overall performance of association rules mining. At present, the frequent itemsets mining algorithms are divided into three major categories: Apriori-like algorithm; FP-growth-like algorithm and Eclat-like algorithm. Among them, the FP-growth algorithm is used widely.

FP-growth algorithm is an association rules mining algorithm proposed by Han Jiawei and his colleagues in 2000 [1]. Compared with the Apriori algorithm, FP-growth has an order of magnitude acceleration in the mining speed. However, the FP-growth algorithm is a recursive algorithm, which leads to the performance of the algorithm limited by the development of CPU, moreover, the recursion stack will occupy a large amount of memory space. In this paper, by use of the advantages of GPU in parallel computing, on the basis of FP-growth algorithm, we propose a parallel frequent itemset mining algorithm based on GPU, called PFP-growth algorithm.

The experimental results show the powerful parallel computing ability of GPU. There is no doubt that parallelization using GPU is a trend in the research of data mining algorithms.

II. BACKGROUND AND RELATED WORKS

A. The Introduction of GPU Computing and CUDA

The modern GPUs are designed for highly parallel tasks, so they are very efficient at manipulating computer graphics. The GPU can execute many floating point operations in finite time. Modern GPUs are massively parallel architectures with thousands of computing cores and are easily programmable. The main difference between CPU and GPU is CPU has less but more powerful cores and GPU has more but less powerful cores. CPU consumes more power than GPU does. CPU is good for task parallelism and whereas GPU is good for data parallelism.

GPU computing uses GPU and CPU together to accelerate executions of applications. In GPU computing inherent sequential part of the code run on the CPU while parallel portion of code is executed using thousands of parallel threads that runs on the GPU. We can explore more algorithms on GPU, which has been sidelined because availability of less computational power.

CUDA is a parallel programming platform designed by NVIDIA for NVDIA GPUs. It can be used to tap the power of the GPU. CUDA Provides a small set of extensions to standard programming languages. The CPU and GPU are treated as separate devices that have their own memory. In CUDA serial code executes itself in CPU and is called Host while parallel code executes on GPU and is called device. A CUDA kernel, is a code that is to be executed in parallel by all the GPU threads. Kernel is launched from the main program. When the kernel is launched, system creates many threads on GPU that executes simultaneously. Each thread has unique id and the threads are grouped into a warp, warps are group into a block, blocks are grouped into a grid. The CUDA platform is easy to learn and it generate optimized code on GPU and thus we are using it in this work.

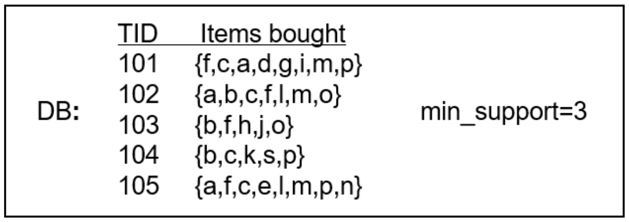
B. The Introduction of FP-growth Algorithm

FP-growth algorithm uses frequent pattern growth to mine frequent patterns without candidate generation [1, 2]. FP-growth algorithm is mainly divided into two steps: FP-tree construction and recursively mining FP-tree. In the process of mining frequent itemset, compared to the Apriori algorithm scanning the database repeatedly, the FP-growth algorithm only needs to scan the database twice. For the first scan, we can get frequent 1-itemsets, for the second scan, sort the items by the support count and delete the non-frequent items, then construct FP-tree by using the ordered frequent 1-itemsets. After FP-tree construction is completed, searching FP-tree by using FP-growth algorithm, find the conditional pattern bases of the corresponding itemsets, and then construct conditional FP-tree. Based on conditional FP-

978-1-5386-1412-9/17 $31.00 © 2017 IEEE
DOI 10.1109/ICEBE.2017.24

tree, the FP-growth algorithm is recursively implemented until all frequent itemsets is mined.

For example, about the transaction database shown in Figure 1, for the first scan, we can get the frequent 1-itemsets {f:4, c:4, a:3, b:3, m:3, p:3}.



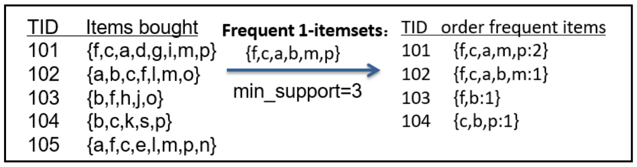
DB:

TID	Items bought
101	{f,c,a,d,g,i,m,p}
102	{a,b,c,f,l,m,o}
103	{b,f,h,j,o}
104	{b,c,k,s,p}
105	{a,f,c,e,l,m,p,n}

min_support=3

Figure 1. A transaction database as running example.

According to the frequent 1-itemsets, delete the non-frequent items contained in each transaction of the databases, and merge and sort them. The result is shown in figure 2.



TID	Items bought
101	{f,c,a,d,g,i,m,p}
102	{a,b,c,f,l,m,o}
103	{b,f,h,j,o}
104	{b,c,k,s,p}
105	{a,f,c,e,l,m,p,n}

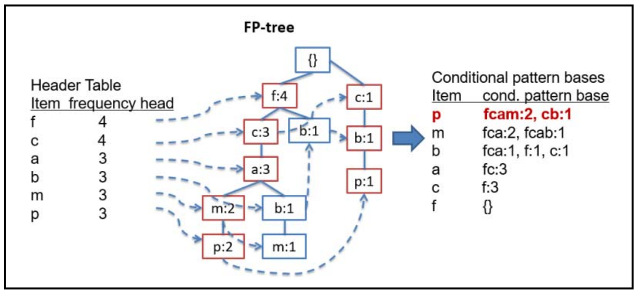
Frequent 1-itemsets: {f,c,a,b,m,p}

min_support=3

TID	order frequent items
101	{f,c,a,m,p:2}
102	{f,c,a,b,m:1}
103	{f,b:1}
104	{c,b,p:1}

Figure 2. The ordered frequent items

After the ordered frequent items is got, the FP-tree starts to be constructed. The process of construction is shown on the left of the figure 3. It is seen that the FP-tree consists of two parts: the header table and the prefix tree, in which the header table is the frequent 1-itemsets.



Header Table

Item	frequency	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

Conditional pattern bases

Item	cond. pattern base
p	fcam:2, cb:1
m	fca:2, fcab:1
b	fca:1, f:1, c:1
a	fc:3
c	f:3
f	{}

Figure 3. The construction of FP-growth and conditional frequent pattern bases

After FP-tree is constructed, mining frequent itemsets is started by using FP-growth algorithm. First, starting with the bottom item of the FP-tree, construct the conditional pattern base for each item. The construction process of the conditional pattern base is shown on the right side of figure 3. Then, accumulate the support count of each item in each conditional mode base, and filter the items below the threshold to construct the conditional FP-tree. Recursively mine each conditional FP-tree until the constructed new FP-tree is empty, or only one path is included, and then the FP-growth frequent itemset mining ends. When the constructed FP-tree is empty, the prefix is a frequent itemset. When only one path is included, all the itemsets in the path are frequent itemsets.

The performance of the FP-growth algorithm is improved by an order of magnitude over the Apriori algorithm. But the FP-growth algorithm using recursive method to mine frequent itemsets, occupy a large amount of memory space, and when the transaction contains a large number of items, it is easy to cause a stack overflow. At the same time, when FP-growth algorithm is implemented, FP-tree uses a linked storage structure, which makes it difficult to parallelize the algorithm on GPU.

III. The PFP-growth Algorithm

A. *The Data Structure of PFP-Growth Algorithm*

In the field of frequent itemsets mining, it is still difficult to implement the irregular data structure such as tree on GPU while parallelizing FP-growth algorithm on GPU. To solve the storage problem on GPU, the transaction data set FP-tree is implemented by an array on GPU, which is logically still a tree, denoted as FP-array [3].

FP-array is a dynamic two-dimensional array that records the relevant information about each node on the FP-tree, including the item's ID, the address of the parent node of the item in the FP-array and the support count of the item. The path of the current node to the root node can be checked by using the address of the parent node stored in the FP-array, and the support count of the current node is the support count of the path. The FP-array can replace FP-tree to mine frequent itemsets on GPU.

Before the construction of FP-array, we need to construct the FP-tree of the transaction data set firstly. The FP-tree constructed in the PFP-growth algorithm (named PFP-tree) is a little different from the FP-tree in the traditional FP-growth algorithm. Each node in PFP-tree has a pointer to its parent node, and each leaf node has a pointer to the next leaf node. The PFP-tree constructed from the database in figure 1 is shown in figure 4.

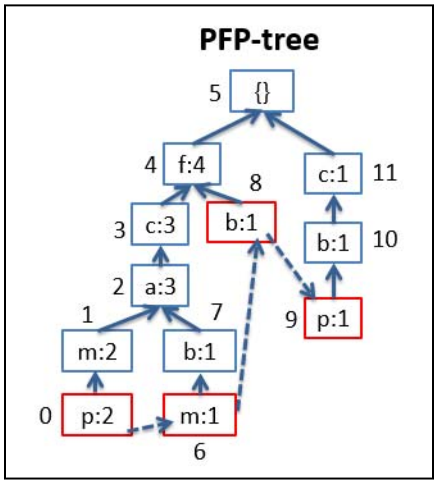
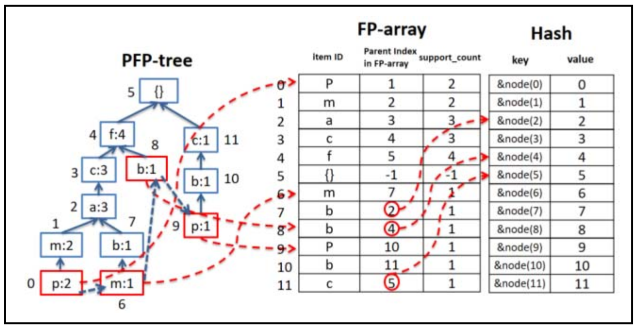


Figure 4. An example of PFP-tree

When the construction of PFP-tree completed, we start to construct the FP-array. Take the leftmost leaf node in FPNR-tree as the starting point, use the pointer pointing to the parent node to find the path of the leaf node to the root node, and put the information of passing nodes in the FP-array. In the searching process, we find that some internal nodes maybe appear on more than one path which leaf nodes to the root node, thus these internal nodes will repeat the record storage in FP-array.

To solve this problem, we introduce a Hash table to indicate the nodes already stored in the FP-array, which includes two properties, the address of the node and the index of the node in FP-array. Before put the information of one node in FP-array, we search the Hash table to determine whether the node has been recorded. If not, store the item's ID, the index in FP-array of its parent node and the support count in the FP-array, then put the node's address and index in the Hash table. If the Hash table has recorded the node, get the index, put it in FP-array with the item's ID and the support count, then stop the search of this path, and jump to the next leaf node to construct the FP-array. The figure 5 illustrates the process to construct FP-array. The algorithm to construct FP-array is shown in Algorithm 1.



FP-array

	item ID	Parent Index in FP-array	support_count
0	P	1	2
1	m	2	2
2	a	3	3
3	c	4	3
4	f	5	4
5	{}	-1	-1
6	m	7	1
7	b	2	1
8	b	4	1
9	P	10	1
10	b	11	1
11	c	5	1

Hash

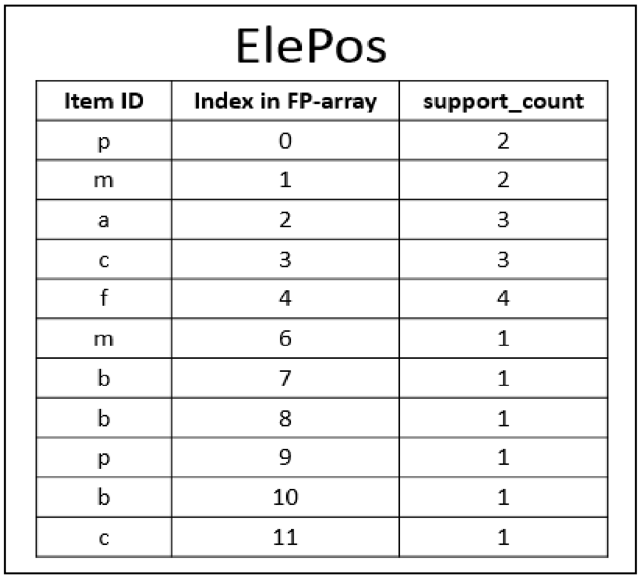
key	value
&node(0)	0
&node(1)	1
&node(2)	2
&node(3)	3
&node(4)	4
&node(5)	5
&node(6)	6
&node(7)	7
&node(8)	8
&node(9)	9
&node(10)	10
&node(11)	11

Figure 5. The contruction of FP-array

Algorithm 1 (constructing FP-array)

```
Input: PFP-tree, Hash table
Output: FP-array
1: for each node p in PFP-tree
2:    store the p's info. in FP-array and record it in Hash
3:    parentNode = the parent node of p
4:    while(parentNode !={})
5:      if parentNode of p in Hash then
6:         get the index of the node
7:            store the index with the item's ID and support
                 count of p in FP-array
8:      turn to step 1;
9:    else
10:        store the item's ID, index of parentNode and
               support count in FP-array
11:        record parentNode in Hash table
12:        p = parentNode
```

In order to mine frequent items in parallel on GPU, it needs to establish a new dynamic array, ElePos (Element Position), to store items which will be mined. ElePos includes the item's ID, the item's address in the FP-array and the support count of the item. The construction process of ElePos consists of two parts: Part one is initialized on CPU. By scanning the FP-array, the information of each of the frequent 1-items is written into ElePos. Part two is to update the ElePos on the GPU. After judging whether the candidate k+1 items are frequent on GPU, we can get the new frequent k+1-items. Then the ElePos is updated by the new frequent itemsets generated by all the threads. Figure 6 is an initialization ElePos obtained according to the FP-array in Figure 5.



ElePos

Item ID	Index in FP-array	support_count
p	0	2
m	1	2
a	2	3
c	3	3
f	4	4
m	6	1
b	7	1
b	8	1
p	9	1
b	10	1
c	11	1

Figure 6. An example of ElePos

B. *The Design of PFP-Growth Algorithm*

Aiming at some problems of FP-growth algorithm, this paper proposes a non-recursive parallel FP-growth algorithm based on GPU, or PFP-growth algorithm for short.

PFP-growth algorithm mainly includes two parts: Part one is to use the GPU to mine candidate k+1 items, such as algorithm 2; Part two is to judge whether the candidate k+1-items are frequent, and then generate the frequent k+1-items, and last, for the next mining, the ElePos is updated by the frequent k+1-items generated from all the threads, such as algorithm 3 [8, 9, 10, 11]. Both of these two algorithms are executed in parallel on GPU.

Algorithm 2 (mining candidate k+1-items)

```
Input: FP-array, item
Output: candidate k+1-items
1: t = the parent node of item
2: if t is not the root
3:     generate a new candidate k+1-item, put it into
       k1_ElePos
4:   update t = the parent node of t
```

```
5:    turn to 2
6:  end if
```

Algorithm 2 (Judging candidate k+1-items)

```
Input: k1_ElePoss, min_sup
Output: frequent k+1-items, ElePos
1:  transform k1_ElePoss into Map<key, value>, take ID
       as key, support count as value, if the key has exited,
       sum the value
2:  delete each element in Map which support count is
       lower than min_sup
3:  wait for other threads to complete
4:  update ElePos by frequent k+1-items generated from
       all the threads
```

The overall flow of the PFP-growth algorithm is shown in Figure 7. First, the algorithm constructs the FP-array from the data set, and then constructs ElePos according to the FP-array. During the process of mining, a dynamic array (k1_ElePos) like ElePos is needed to store the candidate k+1-items. Second, the FP-array and ElePos is transferred into the GPU memory, and the GPU allocates threads to execute the algorithm according to the maximum number of available threads and the number of elements in ElePos. If the number of elements in ElePos is less than the number of available threads in GPU, GPU directly allocates enough threads to parallel execute. If not, the ElePos is split in some segments for multiple processing. The candidate k+1-items are mined by each thread of GPU of algorithm 1, and they are stored into k1_ElePos. Third, the candidate k+1-items in k1_ElePos are judged to filter out items whose support counts are less than min_sup. Finally, the results of all threads are combined to generate the new ElePos for the next mining. When the mining is finished, the frequent itemsets are returned to CPU and outputted. Figure 8 is an example of the PFP-growth algorithm for mining frequent itemsets.

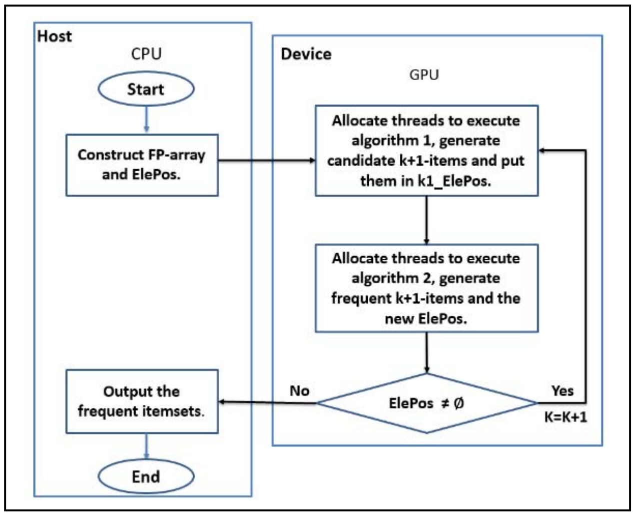
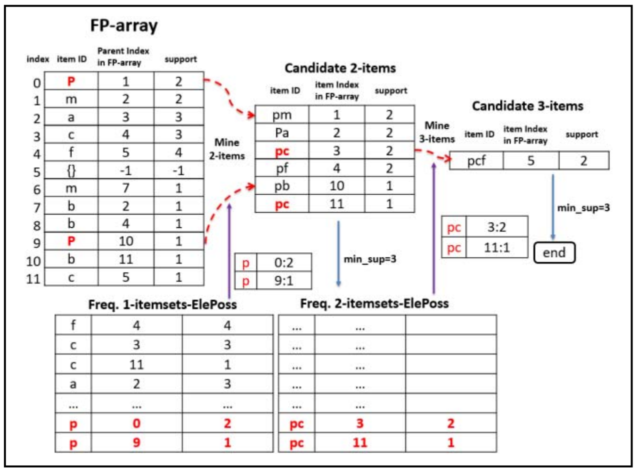

Host
CPU
Start
Construct FP-array and ElePos.
Output the frequent itemsets.
End
Device
GPU
Allocate threads to execute algorithm 1, generate candidate k+1-items and put them in k1_ElePos.
Allocate threads to execute algorithm 2, generate frequent k+1-items and the new ElePos.
ElePos ≠ Ø
No
Yes
K=K+1

Figure 7. he flow of PFP-growth algorithm

Compared with the traditional FP-growth algorithm, the PFP-growth algorithm stores the transaction data set with FP-array, which is read only, so threads can be efficiently executed in parallel to mine frequent itemsets. As a result, it speeds up remarkably the data processing and reduces dramatically the run time. Although there are still some shortcomings in the PFP-growth algorithm, such as when the transaction data are too large and dense, the candidate itemsets list is too large, resulting in GPU memory is insufficient, but overall, compared with the traditional FP-growth algorithm, PFP-growth algorithm greatly enhances the efficiency of mining frequent itemsets.



index	item ID	Parent Index in FP-array	support
0	P	1	2
1	m	2	2
2	a	3	3
3	c	4	3
4	f	5	4
5	{}	-1	-1
6	m	7	1
7	b	2	1
8	b	4	1
9	P	10	1
10	b	11	1
11	c	5	1

item ID	item Index in FP-array	support
pm	1	2
Pa	2	2
pc	3	2
pf	4	2
pb	10	1
pc	11	1

item ID	item Index in FP-array	support
pcf	5	2

p	0:2
p	9:1

pc	3:2
pc	11:1

f	4	4
c	3	3
c	11	1
a	2	3
...	...	...
p	0	2
p	9	1

...	...	
...	...	
...	...	
...	...	
...	...	
pc	3	2
pc	11	1

Figure 8. An example of PFP-growth for mining frequent items

IV. Experimental Results and Discussion

In this section, we implemented the PFP-growth algorithm and tested its performance comparison of PFP-growth with the traditional FP-growth algorithm. In the experiment, the CPU and GPU what we use are i7-4790K and GTX 980 respectively. The specific experimental environment is shown in Table Ⅰ. We report experimental results on four data sets, which are Retail, T10I4D100K, Chess and Mushroom. They all come from http://fimi.ua.ac.be/data/. Table Ⅱ presents some main information about them.

TABLE I. THE MAIN INFORMATION OF DATA SETS

Hardware Platform	CPU: Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz GPU: NVIDIA GeForce GTX 980 Ti Memory: 8GB(2*4GB Kingston DDR3L 1600MHZ) Disk: WDC10EZEX-60ZF5A0-1TB
Software Platform	OS: Windows 10 Professional Edition IDE: Visual Studio 2013

TABLE II. THE MAIN INFORMATION OF DATA SETS

Database	*Type*	*Items*	*Trans*
(a)retail	Sparse	16470	88162
(b)T40I10D100K	Sparse	1000	100000
(c)Chess	Dense	75	3196
(d)Mushroom	Dense	119	8124

The experimental results are shown in figure 8-11. They respectively present the comparison of the run times of PFP-growth algorithm and FP-growth algorithm on each data set with different support threshold.

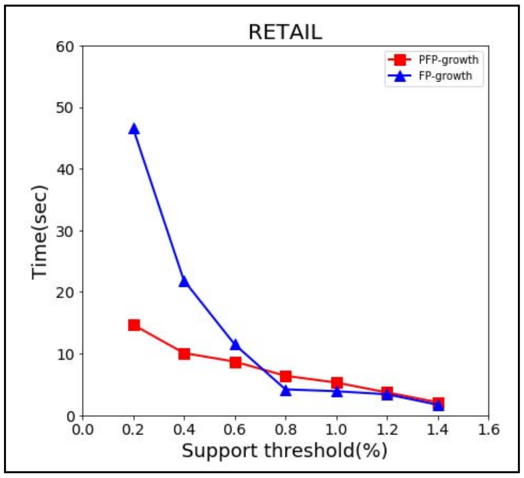


Figure 9. The result of RTAIL

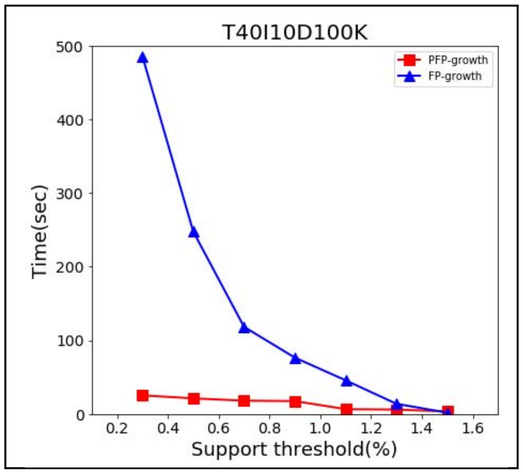


Figure 10. The result of T40I10D100K

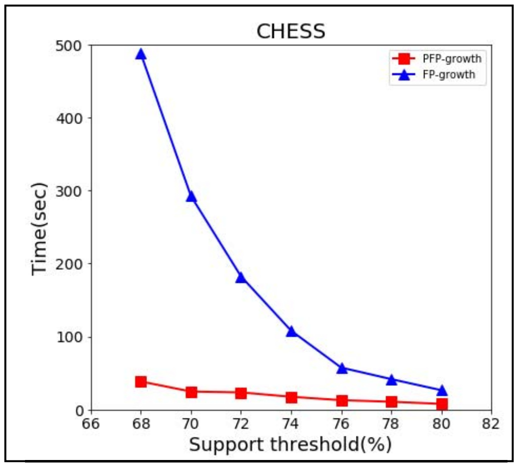


Figure 11. The result of CHESS

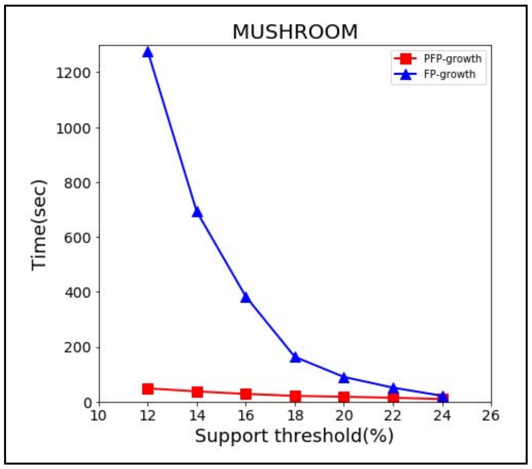


Figure 12. The result of MUSHROOM

From the experimental results, we can see that, no matter what kind of data set, with the support threshold decreasing, the difference in the run time between FP-growth algorithm and PFP-growth algorithm is getting larger and larger. In terms of time efficiency, the PFP-growth algorithm has obvious advantage. Especially in Figure 9, 10 and 11, it can be seen that in larger and denser data sets, PFP-growth algorithm has a much greater speedup. The experimental results show that, compared with FP-growth algorithm, the PFP-growth algorithm has dramatically improved the performance of mining frequent itemsets.

V. CONCLUTION

This paper firstly gives a brief introduction and analysis of the classical FP-growth algorithm. Because FP-growth algorithm uses recursive method to construct conditional pattern base and conditional pattern tree when mining frequent itemsets, it costs a lot of time and memory. When the data set is too large or the support threshold is too small, the efficiency of mining frequent itemsets is often unsatisfactory. To solve this problem, this paper proposes a PFP-growth algorithm based on GPU, which uses GPU's powerful parallel computing ability to mine frequent itemsets, greatly improves the mining efficiency. Experimental results show that, compared with the traditional FP-growth algorithm, PFP-growth algorithm has remarkable acceleration effect. This algorithm also reflects the tremendous advantages of GPU parallel computing, so it would be a trend to apply GPU parallel computing to other data mining algorithms.

ACKNOWLEDGMENT

This work was supported by Collaborative Innovation Center of Wireless Communications Technology and the National Natural Science Foundation of China (Grant No. 61672152).

REFERENCES

[1] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation. In Proceedings of ACM SIGMOD'00, pp 1-12, May 2000.

[2] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," DATA MINING AND KNOWLEDGE DISCOVERY, vol. 8, pp. 53-87, 2004.

[3] H. JIANG, Y. LIAO and S. NI, "A New Algorithm for Mining Frequent Itemset Using Efficient Data Structure," International Conference on Computer Science and Software Engineering, 2014.

[4] J. D. Owens, M. Houston and D. Luebke, "GPU computing," PROCEEDINGS OF THE IEEE, vol. 96, pp. 879-899, 2008.

[5] M. Garland, S. Le Grand and J. Nickolls, "Parallel computing experiences with CUDA," IEEE MICRO, vol. 28, pp. 13-27,2008.

[6] F. Zhang, Y. Zhang and J. D. Bakos, "Accelerating frequent itemset mining on graphics processing units," JOURNAL OF SUPERCOMPUTING, vol. 66, pp. 94-117,2013.

[7] ZH. Zhou, W. Wang and R. Kumar, "Parallel Frequent Pattern Mining without Candidate Generation on GPUs," 14th IEEE International Conference on Data Mining, PP. 1046-1052, 2014.

[8] F. Zhang, Y. Zhang, and J. Bakos, "Gpapriori: Gpu-accelerated Frequent Itemset Mining," Proc. of the 2011 IEEE International Conference on Cluster Computing, pp. 590–594, 2011.

[9] J. Zhou, K. M. Yu and B. C. Wu, "Parallel Frequent Patterns Mining Algorithm on GPU," Proc. of the 2010 IEEE International Conference on Systems Man and Cybernetics, 2010.

[10] S. Rathi and C. A. Dhote, "Parallel Implementation of FP Growth Algorithm on XML Data Using Multiple GPU," 2nd International Conference on Information Systems Design and Intelligent Applications, vol. 339, pp. 581-589, 2015.

[11] H. Li, Y. Wang and D. Zhang, "PFP: Parallel FP-Growth for Query Recommendation," ACM Conference on Recommender Systems, pp. 107-114, 2008.