MINING ASSOCIATION RULES WITH SYSTOLIC TREES

Song Sun and Joseph Zambreno

Dept. of Electrical and Computer Engineering
Iowa State University
Email: {sunsong, zambreno}@iastate.edu

ABSTRACT

Association Rules Mining (ARM) algorithms are designed to find sets of frequently occurring items in large databases. ARM applications have found their way into a variety of fields, including medicine, biotechnology, and marketing. This class of algorithm is typically very memory intensive, leading to prohibitive runtimes on large databases. Previous attempts at acceleration using custom or reconfigurable hardware have been limited, as many of the significant ARM algorithms were designed from a software developer's perspective and have features (e.g. dynamic linked lists, recursion) that do not translate well to hardware. In this paper we look at how we can accomplish the goal of association rules mining from a hardware perspective. We investigate a popular tree-based ARM algorithm (FP-growth), and make use of a systolic tree structure, which mimics the internal memory layout of the original software algorithm while achieving much higher throughput. Our experimental prototype demonstrates how we can trade memory resources on a software platform for computational resources on a reconfigurable hardware platform, in order to exploit a fine-grained parallelism that was not inherent in the original ARM algorithm.

1. INTRODUCTION

Much effort has been put into looking for highly efficient ARM algorithms [1, 2]. Apriori narrows the range of candidate frequent itemset by applying the principle that a superset of items must not be frequent unless any subset of it is frequent. Through repeated iterations Apriori determines the n-length frequent itemset in the n^{th} iteration. Two drawbacks prevent a more widespread adoption of Apriori. The first is that frequent itemsets consume large amounts of time and memory to compute. The other drawback is that repeated scans of the database put a lot of pressure on I/O devices and lead to intolerable performance overhead [3]. The *FP-growth* algorithm [2] solves the second problem of Apriori by reading the database only twice. By storing all transactions in a compact tree, frequent itemsets are explored by recursively traversing the tree and performing a brute-force enumeration. If the tree is predictably large, the entire database can be projected to produce smaller trees. In addition to searching for more efficient algorithms, researchers have looked into parallel computing and hardware implementations [4, 5, 6].

In this paper, we look at this problem from a hardware perspective and propose a new hardware architecture based on a systolic tree structure. The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 explores FP-growth algorithms in more depth. The systolic tree and the candidate itemset dictation are discussed in Section 4. Section 5 is the implementation evaluation. Section 6 is the conclusion.

2. RELATED WORK

A parallel Apriori algorithm version is proposed in [7]. In [8], the FP-growth algorithm is implemented on PC clusters. Though parallel computing platforms can improve the processing speed of software algorithms, measured speedup has not scaled with the number of processors. In [8], scaling efficiency with number of processors is only 13.4/16 = 0.84 and 22.6/32 = 0.71, respectively. In [7], only 2 times speedup was achieved when using 8 nodes. An advantage FPGAs have over parallel computing platforms is the ability to parallelize algorithms at the instruction-level granularity, as opposed to a higher level module-level granularity.

The poor scaling of software-based algorithms has sparked research into hardware-based methods for data mining. A parallel implementation of the Apriori algorithm on FPGAs was first done in [9]. Due to the processing time involved in reading the transactional database multiple times, the hardware implementation was only 4 times faster than the fastest software implementation. They further explored the parallelism and developed a bitmapped CAM architecture which provides 24 times performance gain over the software version [10]. Achieving a better speedup using Apriori will be difficult due to the nature of the algorithm which must read the entire database once for every item in the worst case. FP-growth algorithms are an alternative to Apriori-based solutions. Conclusions from several independent evaluations in-

Table 1. Transactional Database

ID	$Items$	ID	$Items$
1	B,C,D	5	A,B,C
2	B,C	6	A,B,C
3	A,C,D	7	A,B,D
4	A,C,D		

dicate that FP-growth is the current optimal association rules mining algorithm [11, 2]. While software solutions exist, we do not know of any existing hardware implementations of the FP-growth algorithms.

3. FP-GROWTH ALGORITHM

The FP-growth algorithm is composed of two phases. In the first phase the FP-tree is built up. The frequent itemset is generated in the second phase. All transactions in the database are represented compactly in the FP-tree. If one transaction is a prefix of another transaction, they share the same path in the FP-tree. As transactions share more paths, the tree is denser. In the worst case, the number of nodes in the tree is 2^n where n is the number of items in the database. If the FP-tree is too large to fit into memory, the projection method [2] can be used to divide the tree into smaller, more manageable portions. The smaller sub-databases can be mined in multiple processors or processed sequentially in one device. Table 1 shows an example of a small transactional database which is composed of four ordered items$\{$A,B,C,D$\}$. Each row in the table corresponds to a transaction. Figure 1 shows an FP-tree that contains all the information of the transactions in the database. The number in each node represents the times the prefix from the root item to it occur in the transaction database. The level of a node is the number of the nodes from it to the root. The higher the level of a node, the larger length of the prefix it represents. Because two transactions can only share a path when one is a prefix of another, the number of the higher level nodes is never greater than that of the lower level nodes.

The FP-growth algorithm extracts the frequent itemset in a recursive enumeration manner. Starting from the highest level of the FP-tree toward the root, the itemsets ending with item D are checked first. This problem is further divided into several subproblems of finding frequent itemset ending with $\{$C,D$\}$,$\{$B,D$\}$ and $\{$A,D$\}$. Each subproblem ending with defined itemsets is further divided into smaller problems by building its conditional FP-tree. [2] and [12] discuss the operation on conditional FP-tree in detail.

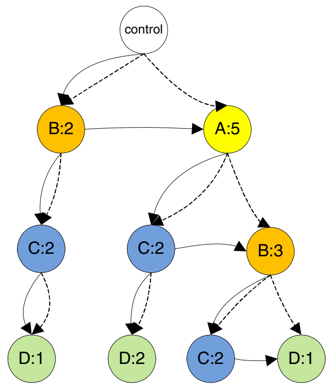

Fig. 1. The FP-tree Data Structure

4. OUR APPROACH

4.1. Systolic Tree

It is not always practical or efficient to directly translate a software algorithm into a hardware architecture. Our approach is to build the tree based on the maximum node degree estimation. When the actual node degree at some point in the tree exceeds the estimated node degree, some frequent itemset will not be found. Suppose the number of items in the database is n, the estimated maximum node degree estimation is K and the estimated depth of the systolic tree is W. Each node in the static tree structure has K children. The total number of nodes in the tree is $K^W + K^{W-1} + ... + K^1 = \frac{K(K^W-1)}{K-1}$. When K is large, the number of children for each node is large which in turn requires each node to have a large number of interfaces. This will make the inner structure of each node very complex. To simplify the complexity of the node, we assign two instead of K interfaces to each node. One of the two interfaces is dedicated to the connection with its first child, the other one is connected to its nearest sibling. To further illustrate the difference between FP-tree and systolic tree architecture, consider the FP-tree data structure shown in Fig. 1. This FP-tree is different from the one defined in [2] in that it has a control node. The input and output of the whole systolic tree passes through the control node. The input can be data or control signals and the output is usually defined by the designer. The letter inside each node represents the item, and the number beside the node is the number of times that the item emerges in the transaction database. The dashed lines indicate the connections in the original FP-tree. The solid lines show the actual connections of the nodes in the systolic tree.

Figure 2 shows the static systolic structure where K=2 and W=3. Each node in the systolic tree architecture is also referred to as a processing element (PE). Each PE has its local data structure and corresponding operations upon receiving signals from outside. There are three kinds of pro-

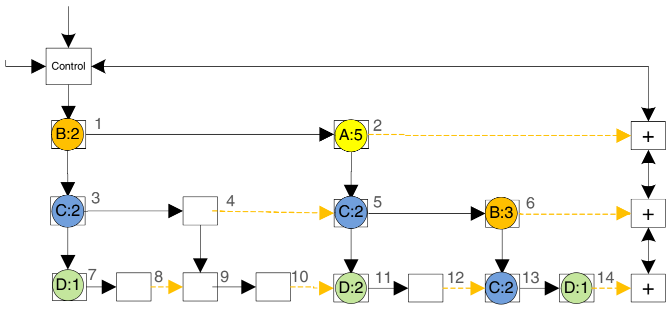

Fig. 2. The Systolic Tree Architecture

cessing elements in this figure. The root PE is the control node discussed above. The PEs in the rightmost column are the counting nodes which are specifically used for frequent itemset dictation, which we will talk about later. The third kind of processing elements are the general PEs. Each node in Fig. 1 has a counterpart in the general processing elements in Fig. 2, but the converse does not hold. Each general PE has one input from its parent and two outputs to its child and siblings respectively. Each PE only has a connection with its leftmost child. If it has to send the data to its rightmost child, the data will be passed to its leftmost child and then through its siblings, passing through all children on the way. The general processing elements which do not contain any item are empty. If the items in one transaction are transferred into the architecture in an ascending order, any PE must contain a smaller item than that of its children. However this does not guarantee that the node item in the systolic tree is always greater than its left-side siblings.

4.2. Systolic Tree Creation

Each PE in the systolic tree has three modes: WRITE mode, SCAN mode and COUNT mode. The last two modes will be discussed in later sections. When the tree is in building phase, PEs are in WRITE mode. An item is loaded into the control PE each cycle which in turn transfer each item into the general PEs. If the item is already contained in a PE, the corresponding count value will be increased. Otherwise, an appropriate empty PE will be located for it. The algorithm for WRITE mode in each PE is given in Fig. 3. The input of the algorithm is an item t. The $match$ flag is set when the item in PE matches t. The $Inpath$ flag is not set when the PE does not contain any item of the current transaction. For example, the PE under the control PE in Fig. 2 should not contain the item B in a new transaction {A,B,C}. After all items are sent to the systolic tree, a control signal that states the termination of an old transaction and the start of a new one is sent to the control PE. The signal will be broadcasted to all PEs which reinitialize themselves for the next transaction. The initialization includes resetting $match$ and

```
Algorithm: WRITE mode(item $t$)
$match := 0$; $InPath := 1$;
(1)if $PE\ is\ empty$ then
        $store\ the\ item\ t$;
        $count := 1$;
        $match := 1$;
        $stop\ forwarding$;
(2)if ($t\ is\ in\ PE$) and ($InPath = 1$) then
        $match := 1$;
        $count\ {+}{+}$;
        $stop\ forwarding$;
(3)if ($match = 0$) then
        $forward\ t\ to\ the\ sibling$;
        $InPath := 0$;
    else
        $forward\ t\ to\ the\ children$
```

Fig. 3. WRITE Mode Algorithm

$Inpath$ flags in the first line of Fig. 2.

Let's illustrate the creation of systolic tree with an example shown in Fig. 2. In order to clearly differentiate PEs a number in light scale is placed in the top-right corner. Suppose a new transaction A,B,D is to be added into the systolic tree. A control signal which indicates the WRITE mode is first broadcasted from the control PE. Then the transaction A,B,D is sent to the control PE sequentially. When PE1 receives A, the step (3) in Fig. 3 is triggered and A is forwarded to PE2. The $Inpath$ flag in PE1 is set to 0. Step (2) is triggered in PE2. The $count$ value in PE2 is increased by 1. The $match$ flag is set to 1. PE2 stops forwarding A to other neighbors. While item A is sent to PE2 by PE1, the second item B is sent to PE1 by the control node. Step (3) is triggered in PE1. Item B is sent to PE2. Since the $match$ flag is set to 1, the item B is sent to PE5. Next, step (3) is triggered in PE5. B is then sent to PE6 where it is not further forwarded. The $count$ value is increased by 1 in PE6. In a similar way, the item D is sent to PE14.

4.3. Candidate Itemset Dictation

The FP growth algorithm generates frequent itemsets by recursive enumeration. However, a recursive implementation is impractical in an FPGA implementation. The approach used in systolic tree architecture is what we call candidate itemset dictation. When we want to check whether a given itemset is frequent or not, it is sent to the systolic tree. The number of the itemset will be obtained in the output of the systolic tree after some clock cycles. The dictation must be performed after the systolic tree is built. When the tree is in itemset dictation phase, PEs are in SCAN mode.

In our systolic tree structure there is only one path trac-

```
**Algorithm**: SCAN mode(item $t$)
$open\ the\ bottom\ door$;
$match := 0$; $IsLeaf := 0$;
(1)**if** $PE\ is\ empty$ **then**
        $stop\ forwarding$;
(2)**if** $(t\ is\ in\ PE)$ **and** $(Bottom\ door\ is\ open)$ **then**
        $match := 1$;
        $IsLeaf := 1$;
        $forward\ t\ to\ the\ sibling$;
(3)**if** $t < the\ item\ in\ PE$ **then**
        $IsLeaf := 0$;
        $close\ the\ bottom\ door$;
        $forward\ t\ to\ the\ sibling$;
(4)**if** $t > the\ item\ in\ PE$ **then**
        $IsLeaf := 0$;
        $forward\ t\ to\ the\ sibling$;
        $forward\ t\ to\ the\ child\ if\ the\ bottom\ door\ is\ open$;
```

Fig. 4. SCAN Mode Algorithm

ing back from any PE to the control PE since each PE has a unique parent. The main principle of dictation is that any path containing the queried candidate itemset will be reported to the control node. Note that such path may contain more items than the queried itemset. To clarify the dictation algorithm, we deem there are two doors in each PE. The right door is always open. The bottom door is locked when no data should be sent to the children. The door policy is described in Fig. 4.

The $IsLeaf$ flag is set if PE matches the last item in the queried candidate itemset. The PE with $IsLeaf$ set is responsible for reporting the number of the candidate itemset to the counting PEs. Since the item is sent one by one, the flag $IsLeaf$ is cleared if another item in the candidate itemset which is larger than the stored item passes through the PE. If the input item is smaller than the stored item, the bottom door should be closed. The rationale behind this is that the item in the child can never be larger than that of its ancestor in a path. Whenever a bottom door in a PE is closed, the path passing through it will never contain the candidate itemset. However if the input item is larger than the stored item, it should be forwarded to all open doors. The rationale is that the path may contain items which are not in the candidate itemset and the candidate itemset is contained in the path.

Let's illustrate the whole dictation process with the example shown in Fig. 5. Suppose the threshold support count is four. Now we want to check whether the candidate itemset $\{$B,D$\}$ is frequent. A control signal which indicates the SCAN mode is first broadcasted from the control PE. Then the item B is sent to the control PE. When PE1 receives B, the step (2) in Fig. 4 is triggered and B is forwarded to PE2.

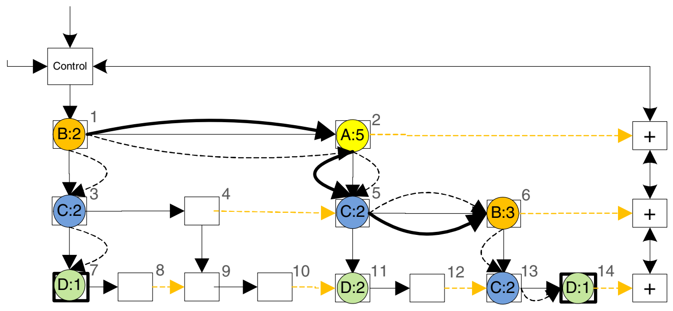

Fig. 5. SCAN Mode Example

Upon receiving the item B, the step (4) is triggered. The bold lines show the track of the item B. PE5 closes the bottom door and forwards the item to PE6. The bottom door in PE5 is locked. At this moment both PE1 and PE6 have the $IsLeaf$ flag set. Now consider the item D which is sent one clock cycle later after the item B is sent in a pipelined fashion. When PE1 receives the item D, the step (4) is triggered in Fig. 4. The $IsLeaf$ flag is cleared and D is forwarded to PE3 and PE2. The dashed lines show the path taken by item D. Notice that since the bottom door of PE5 is locked PE11 will not receive the item D. The $IsLeaf$ flag in PE6 is cleared upon receiving D. In the end the $IsLeaf$ flag is set in PE7 and PE14. As will be explained in the following section, these two PEs report their item count to the counting node. The item count value in both PE7 and PE14 is one. The total count of candidate itemset $\{$B,D$\}$ is two which is less than four. Therefore this candidate itemset is not frequent.

4.4. Candidate Itemset Count Computation

Once all items in a candidate itemset are sent to the systolic tree, a control signal signifying the COUNT mode is broadcasted to the whole systolic tree. The architecture of the systolic tree will change accordingly with response to the COUNT mode signal. Looking back at Fig. 2, we have shown that the first child's input interface is always connected to its parent while others accept input from the first child in WRITE and SCAN mode. In COUNT mode, all PEs receive input from its leftmost neighbor except the first PE in each row. The PEs with $IsLeaf$ flag set are ready to send their own item count and forward the count from their leftmost neighbor in a pipelined fashion when all PEs are in COUNT mode. The counting PEs on the rightmost column always enter the COUNT mode earlier than the general PEs. This is different from the operation used in WRITE mode or SCAN mode where the items are sent one by one in each clock cycle. The COUNT mode control signal sent to the counting PEs by the control PE should at least be delayed $(KW - W)$ cycles after it is sent to the general PEs. The

```
Algorithm: COUNT mode(int $number$)
$CountSent := 0$;
if $(CountSent = 0)$ and $(IsLeaf = 1)$ then
    $CountSent := 1$;
    $forward\ (count + number)\ to\ its\ neighbor$;
else
    $forward\ (number)\ to\ its\ neighbor$;
```

Fig. 6. COUNT Mode Algorithm

COUNT mode algorithm in each PE is listed in Fig. 6.

The input of the algorithm is an integer sent by its left neighbor. The *count* variable is the number of the locally stored item. The $CountSent$ flag is set when the local number has been reported to the counting PEs. Only the PEs with the $IsLeaf$ flag set can report its local *count* value while other PEs transfer the number sent by the left neighbor to the right neighbor. The counting PEs transfer the number sent by its left and bottom neighbors to its top neighbor. The control PE adds up all number sent by the counting PEs and sends it to the output signal.

4.5. Candidate Itemset Generation

The FP-growth algorithm uses a divide and conquer approach to generate frequent itemset by searching the tree in a bottom-up fashion. From the hardware perspective, the candidate generation method used in Apriori is more suitable for dictation. There are several candidate generation procedures. The $F_{k-1} \times F_{k-1}$ method is well suited for systolic tree [12]. A new candiate itemset with k items is dictated only if it is generated from two frequent itemsets with $k-1$ items whose first $k-2$ items are identical. This approach will decrease the number of candiate itemset dramatically.

5. IMPLEMENTATION EVALUATION

5.1. Simulated Result

The systolic tree architecture was simulated and synthesized in ISE 9.1.03i on target board Xilinx Virtex5 XC5VLX330. The performance and area requirement results in Table 2 are extracted from the post place and route report.

The total number of PEs includes control PE and counting PEs. The biggest circuit in the table is 25% of the device since the area usage is exponentially related to the value of W and K while the clock frequency decreases very slightly with the increase of W and K. The control signal and data width is eight bits in our system. The throughput is around 3Gbps. Remember that to build the systolic tree the items in transactional database are sent to the tree one by one in each clock cycle. The time required to build the systolic tree is $\frac{|B|}{throughput}$ seconds where $|B|$ is the size of the database in bits.

5.2. Mining Time Comparison

To find all frequent itemsets, we can dictate all candidate frequent itemsets. If a transaction database has n items, the total number of candidate frequent itemsets is 2^n. A database can be divided into multiple sub-databases with a smaller number of items [2]. These sub-databases can be mined in a parallel manner. The parallel manner is also called "Parallel Projection" where all sub-databases are mined simultaneously. The time for creating systolic tree and transfering data through I/O device is usually trivial compared with the time for mining.

To check the support count of a candidate itemset, the C items are sent to the systolic tree in a pipelined fashion. It takes C cycles in the WRITE mode. The time in SCAN mode is for an item propagated from the control node to the furthest node. Since each node has K degrees and the depth of the tree is W, it takes $(K-1) \times W$ cycles. In COUNT mode the support count is collected from those nodes where the last item in the candidate itemset resides. This includes the time for the general node propagating the support count to the counting node in the same row and the time for the counting node to propagate the support count to the control node. The runtime for the former case is determined by the characteristic of the database. Suppose the the database has n frequent items. Not every candiate itemset will be produced by the $F_{k-1} \times F_{k-1}$ method with equal probability. In general the probability of a candiate with k items is $\frac{C_n^k}{2^n}$. Each node in the systolic tree has the equal possibility to be the reporting nodes. The average time to dictate the 2^n candidates for the former case is $\frac{1}{2^n} \times \sum_{k=1}^{n} n^k C_n^k$.

The latter in the worst case is W cycles which is the time for the bottom counting node to propagate the item to the control node. The time in this part is negligible compared with that of the former case computed above. For an arbitrary transactional database with n frequent items, it can always be projected into multiple sub-databases with N frequent items by Parallel Projection. Since these sub-databases are mined in parallel, the time required for mining is solely determined by the size of the systolic tree. If the size of the tree N is equal to the number of frequent items n, i.e, $K = W = N = n$, the number of clock cycles for mining are $2^n \times \sum_{k=1}^{n} n^k C_n^k$.

Based on the simulated result, we compare the mining time of systolic tree with FP-growth algorithm in Fig. 7. The FP-growth algorithm is from [13]. The mining time of the software algorithm is collected from a PC with Pentium D 3GHz CPU, 2GB RAM. The benchmark is chess.dat from [14] which is prepared from the UCI datasets and PUMSB. This database has 75 items and 3196 transactions. In our experiments, we change the support count threshold to get

Table 2. Experimental Results

Design Parameters	K = 2		K = 3		K = 4	
	W = 3	W = 4	W = 3	W = 4	W = 3	W = 4
Total PEs	18	35	43	125	88	345
Total Slices	648(1.25%)	1267(2.44%)	1548(2.99%)	4616(8.90%)	3169(6.11%)	12674(24.25%)
Clock Freq (MHz)	427.533	425.532	421.941	412.031	402.253	363.769

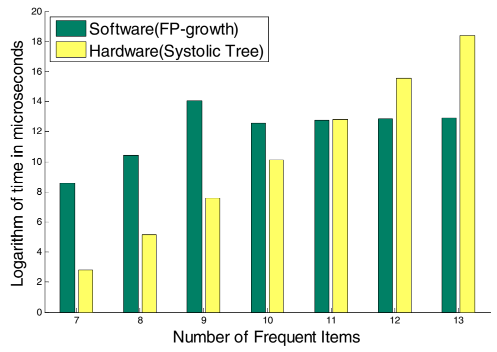

Fig. 7. Hardware and Software Mining Time Comparison

different numbers of frequent items. Note that the run time of the FP-growth algorithm is closely related to the size of the FP-tree while the run time of the systolic tree implementation is only determined by the number of frequent items. It can be observed that the threshold size of the systolic tree must be no more than 11 in order to be faster than FP-growth algorithm. When the size of the systolic tree is 10, the mining speed is 24 times faster than FP-growth. Our future work will shrink the size of the systolic tree to shorten the dictation time, thus increasing the threshold size of the tree.

6. CONCLUSION

In this paper we proposed a systolic tree hardware architecture for association rules mining. Similar to the FP-growth algorithm, our architecture only requires two database reads. Our preliminary experiments show that with the careful selection of the size of the systolic tree, the mining time can be greatly accelerated compared to current software approaches.

7. REFERENCES

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 1994 International Conference on Very Large Data Bases (VLDB)*, 1994.

[2] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, 2004.

[3] S. Kotsiantis and D. Kanellopoulos, "Association rules mining: A recent overview," in *GESTS International Transactions on Computer Science and Engineering Vol.32*, 2006.

[4] A. Choudhary, R. Narayanan, B. Ozisikyilmaz, G. Memik, J. Zambreno, and J. Pisharath, "Optimizing data mining workloads using hardware accelerators," in *Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2007.

[5] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "Minebench: A benchmark suite for data mining workloads," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2006.

[6] J. Zambreno, B. Ozisikyilmaz, G. Memik, and A. Choudary, "Performance characterization of data mining applications using minebench," in *Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2006.

[7] Y. Ye and C.-C. Chiang, "A parallel apriori algorithm for frequent itemsets mining," in *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, 2006.

[8] I. Pramudiono and M. Kitsuregawa, "Parallel FP-growth on PC cluster," in *Proceedings of the Seventh Pacific-Asia Conference of Knowledge Discovery and Data Mining(PAKDD03)*, 2003.

[9] Z. K.Baker and V. K.Prasanna, "Efficient hardware data mining with the Apriori Algorithm on FPGAs," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2005.

[10] ——, "An architecture for efficient hardware data mining using reconfigurable computing systems," in *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2006.

[11] A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y.-K. Chen, and P. Dubey, "Cache-conscious frequent pattern mining on a modern processor," in *Proceedings of the 31st International Conference on Very Large Databases*, 2005.

[12] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2005.

[13] F. Coenen, *The LUCS-KDD implementation of the FP-growth algorithm*, website, 2003, http://www.csc.liv.ac.uk/~frans/KDD/Software/FPgrowth/fpGrowth.html#downloading.

[14] B. Goethals, *Frequent Itemset Mining Dataset Repository*, website, http://fimi.cs.helsinki.fi/data/.