Design and Analysis of a Reconfigurable Platform for Frequent Pattern Mining

Song Sun and Joseph Zambreno, *Member*, *IEEE*

Abstract—Frequent pattern mining algorithms are designed to find commonly occurring sets in databases. This class of algorithms is typically very memory intensive, leading to prohibitive runtimes on large databases. A class of reconfigurable architectures has been recently developed that have shown promise in accelerating some data mining applications. In this paper, we propose a new architecture for frequent pattern mining based on a systolic tree structure. The goal of this architecture is to mimic the internal memory layout of the original pattern mining software algorithm while achieving a higher throughput. We provide a detailed analysis of the area and performance requirements of our systolic tree-based architecture, and show that our reconfigurable platform is faster than the original software algorithm for mining long frequent patterns.

Index Terms—Frequent pattern mining, frequent item set mining, data mining, reconfigurable computing, systolic tree, FPGA.

1 Introduction

The goal of frequent pattern (or frequent item set) mining is to determine which items in a transactional database commonly appear together. Given the size of typical modern databases, an exhaustive search is usually not feasible, making this a challenging computational problem. The *FP-growth* algorithm [1] stores all transactions in the database as a tree using two scans. The FP-growth algorithm was originally designed from a software developer's perspective and uses recursion to traverse the tree and mine patterns. It is cumbersome (and sometimes impossible) to implement recursive processing directly in hardware, as dynamic memory allocation typically requires some software management. For this reason, the dynamic data structures (such as linked lists and trees) which are widely used in software implementations are very rarely used in a direct hardware implementation. Consequently, it would be very difficult to directly translate this FP-growth algorithm into a hardware implementation.

In [2], we have previously introduced a reconfigurable systolic tree architecture for frequent pattern mining, and describe a prototype using a Field Programmable Gate Array (FPGA) platform. The systolic tree is configured to store the support counts of the candidate patterns in a pipelined fashion as the database is scanned in, and is controlled by a simple software module that reads the support counts and makes pruning decisions. In this paper, we focus on improving the original scheme introduced in [2] by eliminating the counting nodes, and provide a new COUNT mode algorithm. A new database projection approach which is suitable for mining a systolic tree is proposed and implemented. Based on the reconfigurable platform presented, the area requirement of FPGAs and software/hardware mining time are studied. We performed experiments over several benchmarks. The experimental results show that our FPGA-based approach can be several times faster than a software implementation of the FP-growth algorithm.

The remainder of this paper is organized as follows: Section 2 describes related work in the area of data mining acceleration. The frequent pattern mining problem is formulated in Section 3. In this section, the definition and creation of the systolic tree are presented. The operations of pattern mining over systolic trees are discussed in Section 4. In Section 5, we discuss a scaling technique for handling large transactional databases and its adaptation to our scheme. A high performance projection algorithm based on FP-growth is also proposed in this section. A detailed area and performance study on the systolic tree is performed in Section 6, with a comparative analysis of our approach and the original FP-growth software algorithm. Finally, we conclude the paper in Section 7.

2 Related Work

One advantage that FPGAs have over traditional computing platforms is the ability to parallelize algorithms at the operand-level granularity, as opposed to the module-level or higher. Consequently, there have been several recent efforts into hardware-based accelerators for data mining algorithms. In [3], the authors described a hardware architecture for a Decision Tree Classification (DTC) algorithm, and showed that optimizing the Gini score computation significantly increases the overall performance. A parallel implementation of the Apriori algorithm on FPGAs was first done in [4]. 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. The HAPPI architecture proposed in [5] applies the pipeline methodology to resolve the bottleneck of Apriori-based hardware schemes. HAPPI

• *The authors are with the Department of Electrical and Computer Engineering, Iowa State University, 2215 Coover Hall, Ames, IA 50011. E-mail: {sunsong, zambreno}@iastate.edu.*

Manuscript received 3 Sept. 2009; revised 5 Apr. 2010; accepted 12 Oct. 2010; published online 18 Jan. 2011.
Recommended for acceptance by C. Aykanat.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-09-0402.
Digital Object Identifier no. 10.1109/TPDS.2011.34.

1045-9219/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

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		

Fig. 1. A simple transactional database.

outperforms the architecture in [4] when the number of different items and the minimum support values are increased [5]. Conclusions from several independent evaluations indicate that FP-growth is one of the best-performing association rules mining algorithms [6], [1].[1] While software solutions exist, we are unaware of any existing hardware implementations of FP-growth algorithms.

3 Systolic Tree: Design and Creation

3.1 Frequent Pattern Mining

Given a transactional database D, where each row represents a transaction as shown in Fig. 1, let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of all items in the database D and $D = \{t_1, t_2, \ldots, t_d\}$ be the set of all transactions. For any transaction t_i, $t_i \subseteq I$. Let $\Gamma(P)$ be the support count of a pattern (item set) P, where P is a set of items. $\Gamma(P) = |\{t_i | P \subseteq t_i, t_i \in D\}|$ is equal to the number of transactions containing P. A pattern (item set) P is frequent if $\Gamma(P)$ is no less than a predefined minimum support threshold ξ. The objective of frequent pattern mining is to find the set of patterns in D which satisfy $\Pi(D) = \{P | \Gamma(P) \geq \xi\}$. In Fig. 1, $I = \{A, B, C, D\}$ and let $\xi = 4$. $\Gamma(\{A, C, D\}) = 2 < 4$, hence $\{A, C, D\}$ is not frequent; $\Gamma(\{B, C\}) = 4 \geq 4$, hence $\{B, C\}$ is frequent.

3.2 Systolic Tree

In VLSI terminology, a *systolic tree* is an arrangement of pipelined processing elements (PEs) in a multidimensional tree pattern [10]. The goal of our architecture is to mimic the internal memory layout of the FP-growth algorithm while achieving a much higher throughput. The role of the systolic tree as mapped in FPGA hardware is then similar to the FP-tree as used in software. The formal definition of FP-tree can be found in supplementary Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34. Given a transactional database, the relative positions of the elements in the systolic tree should be the same as in the FP-tree. To achieve this objective, the systolic tree in this work can be designed based on the following observations:

1. A control PE which corresponds to the root node in the FP-tree acts as the input and output interfaces to the systolic tree.
2. The systolic tree construction algorithm starts from the root node upon receiving a new transaction. For two nodes which store any two items in a transaction, one node must be an ancestor of the other. If two transactions share a common prefix, the shared parts are merged using one prefix path. The degree of a node is equal to the number of transactions which share the common prefix. In the worst case, the degree of a node is equal to the number of frequent items N. Barring the use of dynamic partial reconfiguration, the PEs in hardware cannot be created and deleted dynamically as in software. If each PE is connected to N child PEs, the hardware structure and operation in each PE will be very complex. To avoid this hindrance, each PE in the systolic tree connects to its leftmost child. The other children connect to their leftmost siblings. To reach the rightmost child, the signal from the parent PE must travel through all the children on the left.
3. In the FP-growth algorithm, the first item in each transaction is stored in a child of the root while the other items are stored in its descendants. However, all PEs in the systolic tree operate in parallel and there is no "pointer" concept in the hardware implementation. Due to the intrinsic characteristics of the tree, the operation of a transaction usually starts from the root. Thus, the control PE acts as the input/output interface of the systolic tree. Each clock cycle, an item is transferred to the control PE which may forward the item to its children.

Definition 1 (Systolic Tree). *A systolic tree is a tree structure which consists of the following PEs:*

1. ***Control PE**. The root PE of the systolic tree does not contain any item. Any input/output data of the systolic tree must go through it first. One of its interfaces connects to its leftmost child.*
2. ***General PE**. All other PEs are general PEs. Each general PE has one bidirectional interface connecting to its parent. The general PE which has children has one interface connecting to its leftmost child. Those general PEs which have siblings may have an interface connecting to its leftmost sibling. The general PE may contain an item and the support count of the stored item.*

*Each PE has a **level** associated with it. The control PE is at level 0. The level of a general PE is equal to its distance to the control PE. The children of a PE has the same level.*

Property 1. *Each general PE has only one parent which connects to its leftmost child directly. The other children connect to their parents indirectly through their left siblings.*

Property 2. *A systolic tree which has W levels with K children for each PE has $\sum_{i=0}^{W} K^i$ PEs.*

A PE has three modes of operation: *write* mode, *scan* mode, and *count* mode which will be discussed in the following sections. The systolic tree is built in write mode (Algorithm 1). Input items are streamed from the root node in the direction set by the defined write mode algorithm. The support count of a candidate item set is extracted in both scan mode and count mode. We refer to this process as candidate item set *matching*.

1. According to the extensive benchmarks of the FIMI Workshop as part of ICDM'04 [7], both LCMv2 [8] and kDCI [9] are very competitive algorithms for common mining tasks when compared with FP-growth. For this reason, analyzing and accelerating LCMv2 and kDCI is a promising avenue for future work.

```
Input: Input item i_t
  match← 0; InPath← 1
  if PE is empty then //Step (1)
    store the item i_t; count ← 1; match ← 1; stop forwarding
  else if (i_t is in PE) and (InPath=1) then //Step (2)
    match ← 1; count++; stop forwarding
  else if match=0 then //Step (3)
    Inpath ← 0; forward i_t to the sibling
  else //Step (4)
    forward i_t to the children
  end if
```

Algorithm 1. WRITE Mode Algorithm in Each PE

3.3 Systolic Tree Creation

The design principle of the WRITE mode algorithm is that the built-up systolic tree should have a similar layout with the FP-tree given the same transactional database. The ith item in a transaction is mapped to the ith level in the systolic tree. For any two PEs in the path of the same transaction, the PE in the ith level is the ancestor of the PE in the $i+1$th or higher level. A PE is never in the same path of a transaction with its siblings. Suppose the transactional database has N frequent items. The number of PEs in the first level is at most N. The depth of the systolic tree is at most N. Suppose all items in Fig. 1 are frequent. The first items in all transactions only include items A and B. Therefore, we only need two PEs in the first level of the systolic tree. However, counting the number of items in each level may impair the overall pattern mining performance. Thus, the values of K and W are usually set to be equal to N. The last item of a transaction may not be put into the leaf of the systolic tree if the number of items in the transaction is less than N. If two transactions share the same prefix, they will share a path in the systolic tree. In summary, the design intuition behind the WRITE mode algorithm is that the path an item travels through the FP-tree is the same as the path it travels in the systolic tree.

The WRITE mode algorithm is presented in Algorithm 1. The same algorithm runs in each PE of the systolic tree in every clock cycle. That is, the inner hardware structure of each PE is the same. Initially all PEs are empty. An item is loaded into the control PE each clock cycle which in turn transfers each item into the general PEs. After all items in a transaction 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 broadcast to all PEs which reinitialize themselves for the next transaction. The initialization includes resetting $match$ and $Inpath$ flags in the first line of Algorithm 1. The input of the algorithm is an item i_t. The $match$ flag is set when the item in the PE matches i_t. The $Inpath$ flag is not set when the PE does not contain any item from the current transaction.

Upon receiving a new item i_t, the three if statements in Algorithm 1 are evaluated sequentially. The two switches $match$ and $InPath$ are used to make sure that the latter items in a transaction will follow the path of the former items in the same transaction. The first if statement allocates a PE for the incoming item i_t if it appears in the current path for the first time. In this case, the transaction represented by the items contained in the PEs from the root to the current one is a new transaction which has never been put into the systolic tree before. The second if statement is executed if the current item matches the item in the current PE. In this case, the transaction represented by the items contained in the PEs from the root to the current one is a transaction which has been put into the systolic tree before.

Matching happens in the first and second if statements; thus the items will not be forwarded further. If a PE is in the path of the input transaction, $match$ and $InPath$ must be set. As discussed above, a PE is never in the same transaction with its siblings. If the incoming item does not match the current PE, the third if statement is triggered. There are two scenarios when the mismatch happens. One is that the current PE is in the path of the input transaction. The incoming item should be forwarded to its children. The match will happen after several transfers. The other is that the current PE is not in the path of the input transaction. It should transfer the incoming item to its rightmost sibling. An example of systolic tree creation can be found in supplementary Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34, with a correctness proof provided in Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34.

4 Pattern Mining Using Systolic Trees

To mine frequent patterns in the systolic tree, a collaborating hardware/software platform is required. The software sends a candidate pattern to the systolic tree. After some clock cycles, the systolic tree sends the support count of the candidate pattern back to the software. The software compares the support count with the support threshold and decides whether the candidate pattern is frequent or not. After all candidate patterns are checked with the support threshold in software, the pattern mining is done. The approach to get the support count of a candidate pattern is called candidate item set (pattern) matching. The platform architecture of the software and hardware co-design will be introduced in Section 6. The following section presents the formulated design principle of candidate item set matching; a more detailed discussion can be found in supplementary Appendix D, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34.

4.1 Candidate Item Set Matching

The main principle of matching is that any path containing the queried candidate item set will be reported to the control PE. Note that such a path may contain more items than the queried item set. Before introducing item set matching, we examine some useful properties of the systolic tree which will facilitate frequent pattern mining. As mentioned in previous sections, each frequent item is assigned a sequence number. The items in each transaction enter the systolic tree in an increasing order in both the WRITE mode algorithm and the SCAN mode algorithm.

Lemma 1. *The support count stored in a PE is no less than the support count stored in the children PE; the item stored in a PE is smaller than the items stored in the children PE.*

Lemma 2. *For any nonempty PE_i, the support count of the item set represented by the path from the root to PE_i is equal to the support count stored in PE_i; the support count of any item set including $item_i$ in the path is equal to C_i.*

The first step to get the support count of a candidate item set is to locate those PEs which contain the last item. Those PEs, called *reporting PEs*, are responsible for reporting the stored count. In the second step, the support counts stored in reporting PEs are reported to the software. Since each PE has independent hardware components and stored data, a flag $IsLeaf$ is set in the reporting PE. In the second step, each PE checks its own $IsLeaf$ flag. If it is set, the PE will report its count. The matching algorithm should run after the systolic tree is built. A signal which indicates the SCAN mode is first broadcast from the control PE to all PEs. Similar to the creation of the systolic tree, such a control signal enters the control PE before the first item of each candidate item set enters the systolic tree. The signal initializes the data in each PE.

The design principle of the SCAN mode algorithm is similar to that of the WRITE mode algorithm. The items arriving later in the candidate item set follow the path of the previous items. However, the goal of the WRITE mode algorithm is to find the path which shares the same prefix with the new transaction. In contrast, the goal of the SCAN mode algorithm is to locate all the paths which contain the dictated item set. In the SCAN mode algorithm, the PEs where the last item resides are the reporting PEs. Since there may be multiple reporting PEs, some items in the candidate item set must be duplicated when forwarded in PEs. As discussed in previous sections, each PE has two output interfaces. One is connected to its right sibling; the other is connected to its leftmost child. In each clock cycle, the new incoming item is only forwarded to the paths which contain the candidate item set. Thus, we assume there are two doors which are corresponding to the two interfaces in each PE. The bottom door is connected to the leftmost child. The right door is connected to the sibling. The door is locked when the incoming item should not be sent to the sibling or children. The design of the SCAN algorithm is based on the following propositions:

Lemma 3. *The bottom door in a PE should be locked if the input item is smaller than the stored item. The PE with a locked bottom door will never be in a path which includes the candidate item set.*

The SCAN mode algorithm is shown in Algorithm 2. In step three even though $item_t$ is smaller than the stored item $item_c$, $item_t$ should be sent to the siblings of PE_c. This is because the items in the siblings may be smaller than $item_c$ or equal to $item_t$. If the input item $item_t$ is larger than the stored item $item_c$, the stored items in the siblings or children of PE_c may be equal to $item_t$. Thus, $item_t$ should be forwarded to all open doors as shown in step four. According to Lemma 3, a matching happens if $item_t$ is equal to $item_c$ and the bottom door is open as shown in step two. The $item_t$ still needs to be sent to the siblings of PE_c in case some descendant of the siblings has a stored item matching $item_t$. In sum, $item_t$ should be sent to the sibling of PE_c regardless of its relationship with $item_c$. That is, the right door of PE_c should always be open in the SCAN mode algorithm. The $IsLeaf$ flag is set if PE matches the last item in the queried candidate item set. The PE with $IsLeaf$ set is responsible for reporting the number of the candidate item set to the parent. Since the item in the candidate item set is sent sequentially, the flag $IsLeaf$ is cleared if another item in the candidate item set which is larger than the stored item passes through the PE. Whenever a PE receives the control signal which indicates a new item set matching, the bottom door is opened and the $IsLeaf$ flag is cleared.

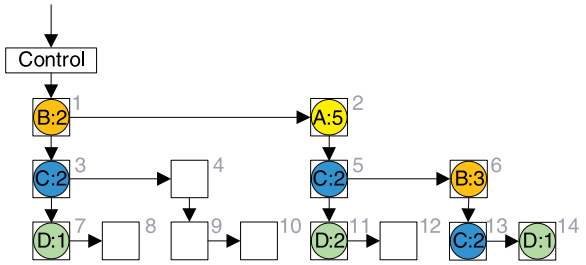


Fig. 2. Equivalent systolic tree-based hardware architecture for acceleration.

Input: Input item i_t
Open the bottom door; $IsLeaf \leftarrow 0$
if PE is empty **then** //Step (1)
stop forwarding
else if (i_t is in PE) and (bottom door is open) **then** //Step (2)
$Isleaf \leftarrow 1$; forward i_t to the sibling
else if $i_t <$ the item in PE **then** //Step (3)
$IsLeaf \leftarrow 0$; close the bottom door; forward i_t to the sibling
else if $i_t >$ the item in PE **then** //Step (4)
$IsLeaf \leftarrow 0$;forward i_t to the sibling;
forward i_t to the child if the bottom door is open
end if

Algorithm 2. SCAN Mode Algorithm in Each PE

4.2 Candidate Item Set Count Computation

According to Property 1 of the systolic tree, there is only one path tracing back from any PE to the control PE since each PE has a unique parent. Once all items in a candidate item set are sent to the systolic tree, a control signal signifying the COUNT mode is broadcast to the whole systolic tree. 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 sibling in WRITE and SCAN mode. After a candidate frequent item set is delivered into the systolic tree, the PEs report the support count of the candidate item set to its unique parent. The PE which is not directly connected to its parent sends its count to the left sibling. The parent PE collects the support counts reported by the children PEs and sends them to its own parent direction. The COUNT mode algorithm in each PE is given in Algorithm 3, where the support counts are transferred to each PE's parent in a pipelined fashion. The inputs of the algorithm are two count values sent by a PE's sibling and child, respectively. The N_{Myself} variable is the number of the locally stored item. The $Sent$ flag is set when the local number has been reported to the parent. Only the PEs with the $IsLeaf$ flag set can report its local N_{Myself} value, while other PEs deliver the count values sent by the child and

sibling to their parents. The control PE adds up all count values and sends it as an output signal.

Input: N_{Right}, N_{Bottom}
$Sent \leftarrow 0; N_{Child} \leftarrow N_{Right} + N_{Bottom}$
if (Sent=0) and (IsLeaf=1) **then**
$Sent \leftarrow 1$;
forward $(N_{Myself} + N_{Child})$ to its parent direction
else
forward N_{Child} to its parent direction
end if

Algorithm 3. COUNT Mode Algorithm in Each PE

5 Tree Scaling by Database Projection

It is unreasonable to assume that a tree-based representation can fit in the available hardware resources (either memory or logic) for any arbitrary database. In the case that memory or logic is not large enough to hold the whole tree, the database must be divided into multiple smaller databases with fewer frequent items. Without the technique of *database projection*, the brute-force candidate item set matching will take an intolerable amount of time when the number of frequent items is large.

5.1 Introduction to Database Projection

To use the systolic tree to mine frequent item sets, the original database is projected into subdatabases. Each of the projected database has no more than $N = min(K, W)$ frequent items and is guaranteed to fit into the FPGAs. Let's illustrate the database projection with an example. Suppose the FPGA logic can at most hold a systolic tree with two frequent items. The database in Fig. 1 has four frequent items and should be projected into subdatabases each of which has at most two frequent items. The frequent items are usually sorted in frequency-decreasing order, which introduces a dense tree structure. For illustrative purpose, we arrange the frequent items in an alphabetic order, i.e., A, B, C, D. Starting at frequent item A, the set of transactions that contain A are collected as an A-projected database. Since there are three frequent items B, C, D in the A-projected database, it should be further projected into two subdatabases. One is an AB-projected database. The transactions in the AB-projected database are obtained by removing B from those transactions containing B in the A-projected database. The other is an $A\phi$-projected database which is composed of those transactions in the A-projected database after removing B. Since the transactions in both of them only contain C and D, the projection process terminates. The projected databases which will be put onto an FPGA are shown as dotted leaves in Fig. 3. Notice the difference between the A-projected database and the $A\phi$-projected database. The former database contains all the transactions starting with A while the latter contains only those transactions starting with A but not including item B. The formal definition and property of database projection are shown below:

Definition 2 (Projected Database [1]). *Given a transactional database, DB,*

1. *Let a_i be a frequent item. The a_i-projected database for a_i is derived from DB by collecting all the transactions containing a_i and removing from them a. infrequent items, b. all frequent items before a_i in the list of frequent items, and c. a_i itself.*
2. *Let a_i be a frequent item in α-projected database. Then the αa_j-projected database is derived from the α-projected database by collecting all entries containing a_j and removing from them a. infrequent items, b. all frequent items before a_j in the list of frequent items, c. a_j itself.*

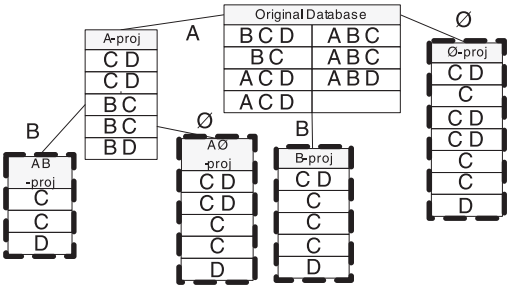


Fig. 3. Projected transactional database.

Lemma 4. *Given a database with n frequent items, the number of projected subdatabases whose transaction only contain the last $N = min(K, W)$ frequent items is at most 2^{n-N} where $n \geq N$.*

The proof of the lemma can be found in supplementary Appendix E, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34.

5.2 Algorithm for Database Projection

Observing Lemma 4, we know that the process of projection encounters the combinatorial problem of the candidate generation which is resolved in the FP-growth algorithm. Inspired by this observation, we will explore how to perform database projection on the fly during the running of FP-growth algorithm. A new field called *subroot* is created in each node. The *subroot* is a list of pointers, each of which points to a subset of the FP-tree. The algorithm to create *subroot* for each node is illustrated in Algorithm 4, where γ is a set of N frequent items which will be placed into the projected databases. The selection of the N items from the n frequent items is critical to the performance of the system. Recall that our goal is to take advantage of the hardware accelerated systolic tree and use it to do the cumbersome mining work. Hence, the first N most frequent items should be put into γ. Therefore, the sequence of the frequent items in our header table is different from the original FP-growth algorithm. The first N frequent items are moved to the end of the header table before projecting the database as shown in Fig. 4. In this way, the size of the FP-tree without the first N frequent items is much smaller than the original tree and the number of projected databases is greatly reduced.

The FP-trees pointed by *subroot* hold those transactions which contain only the last N frequent items in the new header table. Suppose each transaction in the projected database of Fig. 1 only contains the last two frequent items, i.e., $N = 2, n = 4$. The dashed lines in Fig. 4 are the *subroot*

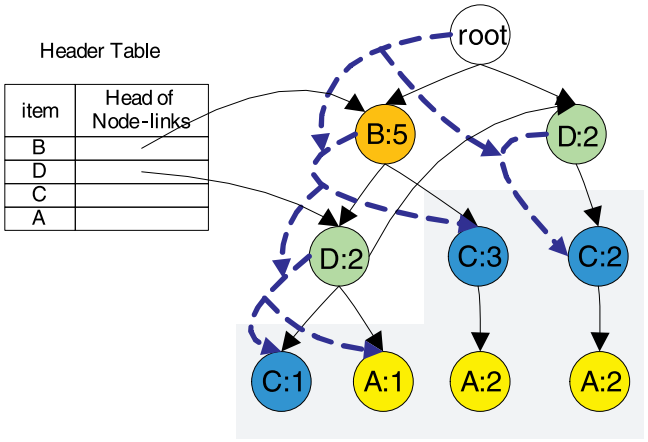


Fig. 4. Subroots.

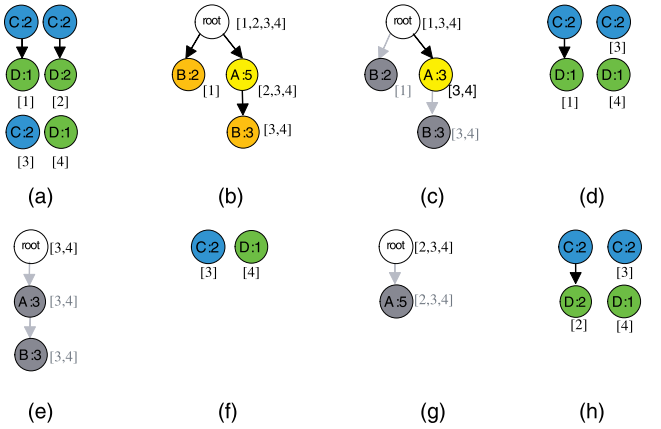


Fig. 5. Database projection based on FP-growth. (a) Projected trees. (b) Global FP tree. (c) Conditional FP tree of B. (d) B-projected trees. (e) Conditional FP tree of AB. (f) AB-projected trees. (g) Conditional FP tree of A. (h) A$\emptyset$-projected trees.

pointers in each node. One can see that the *subroot* of the parent node is a superset of the *subroot* of its children. The *subroot* algorithm starts from the nodelink of D. The first node (D:2) on the left branch has two children (C:1) and (A:1). Hence, its *subroot* is composed of two pointers pointing to the two children. The second node (D:2) on the right branch which has only one child has only one pointer in its *subroot*. Following the nodelink of B in the header table, the node (B:5) has two children. One is (D:2) which does not contain an item in γ. According to Algorithm 4, the *subroot* of (D:2) is merged into the *subroot* of (B:5). The other child (C:3) contains an item C which is in γ. Thus, the pointer which points to node (C:3) is merged into the *subroot* of (B:5). Similarly, the *subroot* of the root node is a collection of pointers pointing to all the projected transactions which contain only the last N frequent items in the header table.

Input: The FP-tree of DB
Output: The subroot in each node
Set *subroot* to *null* for each node
for $i = n - N$ to 1 **do**
  for Each node n^p following the nodelink of the i^{th} item **do**
    for Each Child Pointer T_j^p of n^p **do**
      if the node n_j^p pointed by T_j^p contains an item $\in \gamma$ **then**
        $n^p.subroot \leftarrow n^p.subroot \cup T_j^p$
      else
        $n^p.subroot \leftarrow n^p.subroot \cup n_j^p.subroot$
      end if
    end for
  end for
end for
the *subroot* of the root is the *subroot* unions of its children

Algorithm 4. Subroot Creation Algorithm

The FP-growth algorithm from [1] is shown in Algorithm 5. Those lines marked with stars are not in the original FP-growth algorithm and will be discussed later. Following this algorithm, the conditional trees generated from Fig. 4 are shown in Fig. 5. For illustration purpose, the items are arranged in alphabetic order instead of the order shown in Fig. 4. To use the FP-growth algorithm to generate projected databases, the mining process starts from the $(n - N)$th item in the header table. In Fig. 5a, each subtree which will be put into the projected databases has a label in square bracket. The *subroots* generated for the global FP-tree are shown in Fig. 5b. The *subroot* creation algorithm for conditional trees is a little different from Algorithm 4. Instead of generating *subroot* for each node in the conditional tree, only the *subroot* in the root and leaves are computed for each conditional tree. This will greatly reduce the runtime of the *subroot* creation algorithm. Those trees pointed by *subroots* are called *projected trees*. The projected trees of each conditional tree are shown separately in Figs. 5d, 5f, and 5h. The database projection algorithm is shown in Algorithm 5 with those lines marked with one star added to the original FP-growth algorithm. One can find out that the projected databases in Fig. 5 are the same those in Fig. 3. Note that the ϕ-proj database in Fig. 3 is corresponding to the projected tree of the global FP-tree in Fig. 5a.

Input: An FP-tree of a database DB; A support threshold ξ
Output: The complete set of frequent patterns
Call FP-growth(FP-tree, null)
*Call subroot creation algorithm
*generate projected database $Proj_\phi$ for global FP-tree
**generate *freq_pattern_set*($Proj_\phi$))

Procedure *FP-growth*($Tree, \alpha$)
if $Tree$ contains a single prefix path **then**
  let P be the single prefix path part of $Tree$
  let Q be the multipath part with the top branching node replaced by a null root
  for each combination β of the nodes in the path P **do**
    generate pattern $\beta \cup \alpha$ into *freq_pattern_set(P)*
    **generate patterns $(\beta \cup \alpha) \times$*freq_pattern_set*($Proj_\alpha$)
    **into *freq_pattern_set(R)*
  end for
else
  let Q be $Tree$
end if
for each item a_i in Q **do**
  generate patten $\beta = a_i \cup \alpha$ with support=a_i.support
  construct β's conditional pattern-base
  construct β's conditional FP-tree $Tree_\beta$
  *generate *subroots* for $Tree_\beta$
  *generate β's projected database $Proj_\beta$
  **generate patterns $\beta \times$*freq_pattern_set*($Proj_\beta$)
  if $Tree_\beta \neq \phi$ **then**
    call *FP-growth*($Tree_\beta, \beta$)
  end if
  let *freq_pattern_set(Q)* be the set of patterns so generated
end for
return *freq_pattern_set(R)*$\cup$*freq_pattern_set(P)*$\cup$ *freq_pattern_set(Q)*$\cup$*freq_pattern_set(P)*$\times$*freq_pattern_set(Q)*

Algorithm 5. FP-Growth Database Projection Method

5.3 Frequent Item Set Generation with Database Projection

For each frequent item set β, the frequent item sets generated from the projected database of β (β-proj, or $Proj_\beta$) are combined with β. This process is denoted as $\beta\times$ *frequent_pattern_set* ($Proj_\beta$) in Algorithm 5. The optimization of the single prefix-path FP-tree is the same as the approach proposed in [1]. However, the database projection needs not be performed for the single prefix path P since it has been generated into *frequent_pattern_set*($Proj_\alpha$). The frequent item set generated from the projected database of the tree containing a single prefix path should be combined with each pattern in that single prefix path separately (denoted as *freq_pattern_set(R)* in Algorithm 5). This is because the top branching node of Q is replaced by a null root. Each frequent pattern β generated in tree Q is combined with the frequent patterns in $Proj_\beta$. *freq_pattern_set(Q)* is combined with *freq_pattern_set(P)* where the occurrence frequency of each combined frequent item set is the minimum support among those items. Those lines marked with double stars deal with frequent item set generation from the projected database. Algorithm 5 provides a way to mine the frequent item sets in parallel. In Section 6, we will demonstrate how to use the newly proposed algorithm to fully utilize the power of reconfigurable platform where software and hardware mine the frequent item sets concurrently.

6 Experimental Evaluation

6.1 Area Requirements

In our VHDL implementation, different sizes of systolic trees are generated automatically by specifying the depth and degree parameters. We placed and routed various configurations of this implementation using Xilinx ISE 9.1.03i. The targeted device is a Xilinx Virtex-5 XC5VLX330 FPGA with package ff1760 and -2 speed grade. From the place and route report, each node requires 50 LUTs. Approximately 25 percent of the available slices are used when both K and W are four. When both K and W are five, the slice usage is more than 100 percent. A brief analysis of FPGA clock frequency can be found in supplementary Appendix F, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34.

6.2 Performance Comparison

For performing a fair comparison, we selected as our target environment the XtremeData XD2000i, which is a complete platform to explore the benefits of FPGA coprocessing with traditional microprocessor computing systems [11]. The XD2000i development system comprises of a Dual Xeon motherboard with one Intel Xeon processor and two Stratix III EP3SE260 FPGAs in the other socket, running the CentOS Linux operating system. More details regarding the XD2000i architecture and programming environment are provided in supplementary Appendix F, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34.

In our simulated environment, the systolic tree of size four was integrated with other hardware IP components provided by XtremeData in one of the FPGAs. The Intel CPU executes Algorithm 5, which was implemented in C++. For each frequent item set β, a projected tree is generated as a set of pointers pointing to the paths in the original FP-tree. The algorithm reads the transactions pointed by those pointers and sends them to the systolic tree in FPGA. The sending operation returns immediately. The FPGA hardware receives the transactions and creates the systolic tree using Algorithm 1. Thereafter, the candidate item sets are generated and matched by hardware without software intervention in a pipelined style. Algorithms 2 and 3 are executed in each node of the systolic tree. After all frequent item sets are collected, they are sent back to the software. After receiving the frequent item sets from the hardware, the software combines them with β.

TABLE 1
Benchmark Information

Benchmark	Items	Num. Trans.	Max Trans.	Min Trans.	Total Items
chess	75	3196	37	37	118252
BMS-WebView-2	3340	77512	161	1	358278
connect	129	67557	43	43	2904951
BMS-POS	1657	515597	164	1	3367020
pumsb	7117	49064	74	74	3629404
kosarak	41270	990002	2498	1	8019015

During this process, the original database is projected sequentially which is similar to *partition projection* mentioned in [1]. The experimental results presented below are based on this sequential mode. Reconfigurable development systems with multiple FPGAs integrated in the same board would not benefit from this model since the running time of FPGAs in this case is smaller than the running time to generate a projected database. *Parallel projection*, which scans each transaction in the original database and projects it into multiple projected databases in parallel can fully utilize the multi-FPGA system. Designing an efficient parallel projection software algorithm and applying it in a parallel FPGA implementation is a planned avenue of future work. We can expect that the parallel FPGA implementation will be much faster than the sequential model used in this paper.

The experiments are performed on real data sets described in [7] and from the KDD Cup 2000 data [12]. Table 1 characterizes the data sets in terms of the number of distinct items and transactions, the maximum and minimum size of the transactions. The performance measure was the execution time of the systems on the benchmarks with different support thresholds. The execution time only includes the time for disk reading and memory I/O, but excludes the disk writing time. The original software FP-growth algorithm which is implemented in C++ is from [13]. In order to have a fair comparison, the software FP-growth algorithm also runs on our target XtremeData XD2000i platform without using the FPGA hardware.

The performance results for each benchmark are listed in Fig. 6. The systolic tree algorithm is almost two times faster than FP-growth for the chess and BMS-WebView-2 data sets. For the kosarak data set, the mining time of the systolic tree is larger than FP-growth. The mining time for FP-growth grows larger than the systolic tree algorithm with the decrease of support threshold in BMS-POS. The

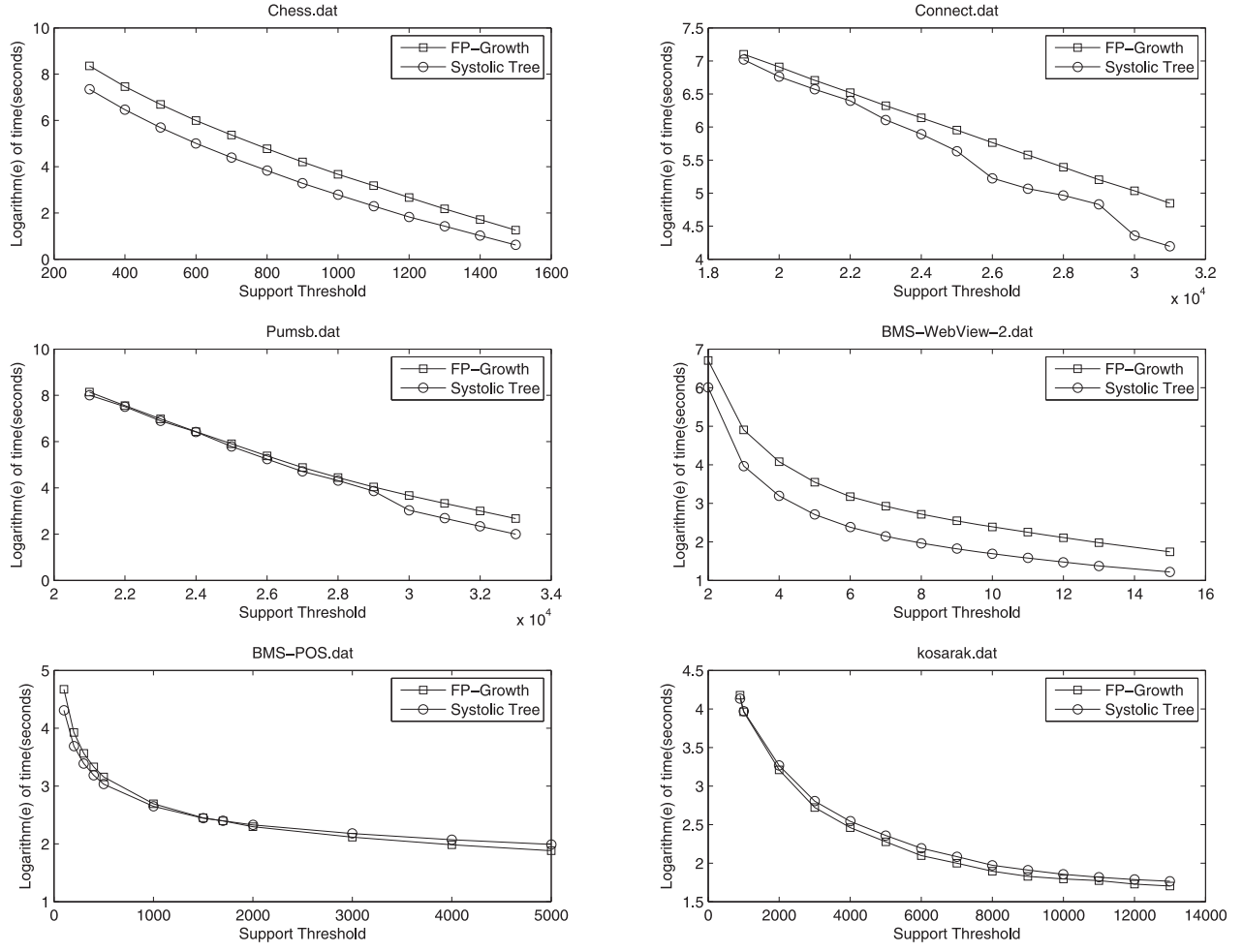


Fig. 6. Performance Comparison with FP-growth.

advantage of the systolic tree over FP-growth becomes dramatic with long frequent patterns, which is challenging to the algorithms that mine the complete set of frequent patterns. In retrospect, the systolic tree algorithm removes the most frequent items and mines them in the projected databases. Compared with the original FP-growth algorithm, the systolic tree algorithm reduces the runtime by mining the dense part of the FP-tree in hardware and the sparse part in software simultaneously. However, it introduces the overhead of database projection. If the overhead is not amortized by the runtime reduction, the systolic tree algorithm is slower than the original FP-growth algorithm. This is illustrated in the kosarak benchmark. One can expect that the mining time will decrease with more frequent patterns mined in hardware when the size of the systolic tree is fixed. A brief illustration of this scenario can be found in supplementary Appendix F, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34.

7 Conclusion

A new systolic tree-based approach to mine frequent item sets is proposed and evaluated. Due to the limited size of the systolic tree, a transactional database must be projected into smaller ones each of which can be mined in hardware efficiently. A high performance projection algorithm which fully utilizes the advantage of FP-growth is proposed and implemented. It reduces the mining time by partitioning the tree into dense and sparse parts and sending the dense tree to the hardware. The algorithm was implemented on an XtremeData XD2000i high performance reconfigurable platform. Based on the experimental results on several benchmarks, the mining speed of the systolic tree was several times faster than the FP-tree for long frequent patterns. Improving the mining efficiency on sparse patterns will be included in our future work.

References

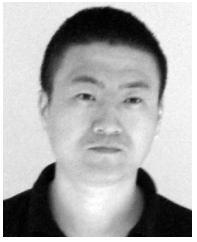
[1] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53-87, Jan. 2004.

[2] S. Sun and J. Zambreno, "Mining Association Rules with Systolic Trees," *Proc. Int'l Conf. Field-Programmable Logic and Applications (FPL '08)*, Sept. 2008.

[3] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, "An FPGA Implementation of Decision Tree Classification," *Proc. Conf. Design, Automation, and Test in Europe (DATE)*, pp. 189-194, Apr. 2007.

[4] Z. Baker and V. Prasanna, "Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs," *Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*, pp. 3-12, Apr. 2005.

[5] Y.-H. Wen, J.-W. Huang, and M.-S. Chen, "Hardware-Enhanced Association Rule Mining with Hashing and Pipelining," *IEEE Trans. Knowledge and Data Eng.*, vol. 20, no. 6, pp. 784-795, June 2008.
[6] 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," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 577-588, 2005.
[7] R. Bayardo, B. Goethals, and M. Zaki, eds., *Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMI)*, Nov. 2004.
[8] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets," *Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMI)*, 2004.
[9] C. Lucchese, S. Orlando, and R. Perego, "kDCI: On Using Direct Count Up to the Third Iteration," *Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations (FIMI)*, 2004.
[10] J. Bentley and H. Kung, "A Tree Machine for Searching Problems," *Proc. Int'l Conf. Parallel Processing (ICPP)*, pp. 265-266, 1979.
[11] Xtremedata. Product documents, www.xtremedata.com, 2011.
[12] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng, "KDD-Cup 2000 Organizers' Report: Peeling the Onion," *SIGKDD Explorations*, vol. 2, no. 2, pp. 86-98, 2000.
[13] B. Goethals FP-Growth Implementation, adrem.ua.ac.be/~goethals/software, 2011.



Song Sun received the BS degree in computer science and engineering from Beijing Information Technology Institute in 1998, and the MS degree from the University of Science and Technology of China in 2001. Currently, he is a research assistant working toward the PhD degree in the Reconfigurable Computing Lab at Iowa State University, Ames, IA. His research interests include high performance reconfigurable computing and embedded systems.

Joseph Zambreno (M'02) received the BS degree (Hons.) in computer engineering in 2001, the MS degree in electrical and computer engineering in 2002, and the PhD degree in electrical and computer engineering from Northwestern University, Evanston, IL, in 2006. Currently, he is an assistant professor in the Department of Electrical and Computer Engineering at Iowa State University, Ames, where he has been since 2006. He was a recipient of a National Science Foundation Graduate Research Fellowship, a Northwestern University Graduate School Fellowship, a Walter P. Murphy Fellowship, and the Electrical Engineering and Computer Science Department Best Dissertation Award for his PhD dissertation "Compiler and Architectural Approaches to Software Protection and Security." His research interests include computer architecture, compilers, embedded systems, and hardware/software co-design, with a focus on runtime reconfigurable architectures and compiler techniques for software protection. He is a member of the IEEE and the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

APPENDIX A
FP-TREE

The Frequent-Pattern (FP) tree [1] is a compact data structure to store frequent itemsets.

Definition 3 (FP Tree). *An FP-tree is a tree structure which is defined below:*

1) *It consists of one root labeled as "null", a set of item-prefix subtrees as the children of the root, and a frequent-item-header table.*
2) *Each node in the item-prefix subtree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.*
3) *Each entry in the frequent-item-header table consists of two fields, (1) item-name and (2) head of node-link (a pointer pointing to the first node in the FP-tree carrying the item-name).*

Based on the definition, the FP-tree construction algorithm from [1] is as follows. There are two scans of the database in constructing an FP-tree. The frequent items are determined in the first scan while the infrequent items are discarded. The frequent items are then sorted in decreasing order of support counts so that fewer nodes in the FP-tree are used to represent the transactions. The sorting according to the support count is not required in the systolic tree due to the fixed structure of the systolic tree. However, all items must be shown in a consistent order. For example, item A must be placed before item B in a transaction as shown in Fig. 1. Similarly, item B must be placed before item C, and so on.

The FP-tree is constructed in the second scan. The FP-tree is constructed by reading the transactions one at a time and mapping each transaction onto a path in the FP-tree. In Fig. 7, each node contains an item along with an integer which is the count of that item in that path. Initially the FP-tree contains only the root node. After reading a new transaction, the algorithm starts from the root node to look for a node whose label matches the first item in the transaction. If such a node is not found, a new node is created with the label as the first item. Otherwise, the algorithm starting from the current node looks for a child whose label matches the second item of the transaction. A node's counter is increased by one in the case of a match. This process continues until every transaction is mapped to a path in the FP-tree. For Fig. 1, suppose the minimum support threshold is 4 ($\xi = 4$) and all items are frequent. The FP-tree constructed from Fig. 1 is shown in Fig. 7. For illustration purposes, the items are not sorted in the support-decreasing order but alphabetic order.

APPENDIX B
SYSTOLIC TREE CREATION:AN EXAMPLE

Let's illustrate the creation of a systolic tree with the example shown in Fig. 2. In order to clearly differentiate PEs, a number in light scale is placed in the top-right corner.

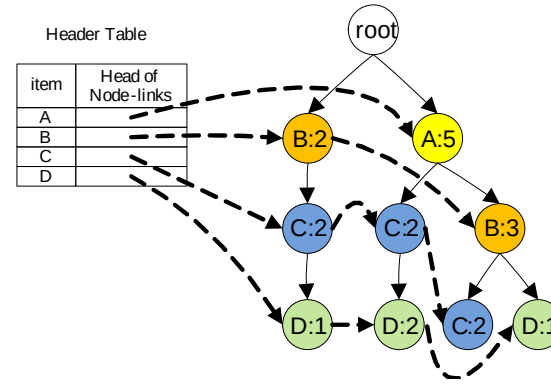


Fig. 7. FP-Tree (Software Data Structure Representation)

1) Initially all PEs are empty and do not contain any item.
2) The first transaction $\{B, C, D\}$ in Fig. 1 is scanned firstly. In the first clock cycle, item B enters the control PE. In the second clock cycle, item C enters the control PE and item B enters PE1. The first if statement in Algorithm 1 is executed in PE1. Item B is stored in PE1 and not forwarded further. The count corresponding to B is 1, and the $match$ flag is set to 1. In the third clock cycle, item D enters the control PE and item C enters PE1. Step (4) is triggered in PE1 and item C is sent to PE3.
3) In the fourth clock cycle, a control signal stating the termination of the current transaction and the start of a new transaction is sent to the control PE. D enters PE1 and C enters PE3. Step (4) is executed in PE1 and D is sent to PE3. In PE3 the first if statement is executed and item C is stored and not forwarded.
4) In the fifth clock cycle, the first item B of the second transaction in Fig. 1 enters the control PE. The control signal enters PE1. The $match$ and $InPath$ are initialized in PE1. Item D in the first transaction enters PE3. Step (4) is executed in PE3 and D is sent to PE7.
5) In the sixth clock cycle, the second item C of the second transaction in Fig. 1 enters the control PE. Item D enters PE7 and is not forwarded further. Item B in the second transaction enters PE1 and the step (2) in Algorithm 1 is triggered. The count in PE1 is increased by 1 and $match$ is set. The control signal enters PE2 and PE3 and initializes the flags.
6) In the seventh clock cycle, a control signal stating the termination of the current transaction and the start of the new transaction is sent to the control PE. The item C enters PE1 and step (4) is triggered.
7) In the eighth clock cycle, the first item A of the third transaction enters the control PE. The control signal enters PE1. The item C enters PE3 and step (2) is executed. The count in PE3 is increased.
8) The items in the transactional database enter the systolic tree sequentially. After all transactions are stored in PEs, the state of the systolic tree is shown in Fig. 2.

APPENDIX C
COMPLETENESS OF THE SYSTOLIC TREE

We can now provide some formal assurances that Algorithm 1 builds the systolic tree correctly.

Lemma 5. *During the creation of the systolic tree, the items arriving later follow the path of the previous items in the same transaction.*

Proof A control signal initializes $match$ and $InPath$ in all PEs in the systolic tree in the beginning of a transaction. According to Algorithm 1, the first item is either stored in a PE or forwarded to the sibling of the current PE. In the PE of the former case, $match = 1$ and $InPath = 1$; in the PEs of the latter case, $match = 0$ and $InPath = 0$. An empty PE or a PE containing the item will be found after some clock cycles in the latter case. Suppose the PE in the first level where the first item resides is $PE^i (i \geq 1)$. The $(i-1)$ PEs on the left have $match = InPath = 0$ and are not empty. In PE^i, $match = 1$ and $Inpath = 1$. The other PEs have $match = 0$ and $InPath = 1$. The second item of the transaction is sent to the $(i-1)$ PEs on the left firstly. Since these PEs are not empty and have $match = 0$, $InPath = 0$, step (3) is triggered. The second item is forwarded to the right PEs cycle by cycle until the PE^i. Since each item in a transaction is unique, the second item is different from the first item. According to the write mode algorithm, step (4) is triggered. The second item will be sent to a child of PE^i. A PE in the second level will be located for the second item. The third item will follow the path of the second item in a similar way. The other items can be analyzed similarly. □

Lemma 6. *A new transaction not sharing a prefix with previous transactions will always be stored correctly in the systolic tree.*

Proof The first item the new transaction is stored in the first level of the systolic tree. Since the new transaction does not share any prefix with stored transactions, an empty PE^i is located for the first item. As stated in Lemma 5, the latter items follow the path of the former items. Consequently, all children of PE^i must be empty. The second item will follow the path of the first item and be stored in the first child of PE^i. The other items in the new transactions will be stored in the systolic tree similarly. The $count$ is equal to 1 in all PEs storing the new transaction. □

Lemma 7. *A new transaction sharing a prefix with previous transactions will always be stored correctly in the systolic tree.*

Proof There may be multiple stored transactions sharing a prefix with the new transaction t_u. Suppose t_v is one of the stored transactions which share the longest prefix with t_u. Let $t_u = \{item_1^u, item_2^u, \cdots, item_{|t_u|}^u\}$ and $t_v = \{item_1^v, item_2^v, \cdots, item_{|t_v|}^v\}$, $item_i^u = item_i^v$, for $i = 1, 2, \cdots, s$, where s is the length of the prefix. t_v is mapped to a path $p_v = \{PE_1^v, PE_2^v, \cdots, PE_{|t_v|}^v\}$ in the systolic tree. PE_i^v is on the i^{th} level of the systolic tree.

According to Algorithm 1, the PEs which are the left siblings of $PE_i^v (i \leq s)$ execute step (3). Step (2) is executed in $PE_i^v (i \leq s)$ upon receiving the i^{th} item of the new transaction t_u. Step (4) is executed in $PE_i^v (i \leq s)$ upon receiving the $j^{th} (j > i)$ item in t_u. When PE_s^v receives the $s+1^{th}$ item, step (4) is triggered and the $s+1^{th}$ item is sent to the children of PE_s^v. Since t_v shares the longest prefix with t_u, an empty PE in the $s+1^{th}$ level is allocated for the $s+1^{th}$ item. The rest of the items in t_u are stored in a similar way, as discussed in Lemma 6. □

Theorem 1. *Given a transactional database, the systolic tree will be built up correctly using the WRITE mode algorithm.*

Proof Initially the systolic tree is empty. After the systolic tree is built up, a path can be found from the control PE to any non-empty PE. For an arbitrary path $(PE_0, PE_1, PE_2, \cdots, PE_k)$ in the systolic tree where PE_0 is the control PE, let C_{PE_k} be the count at the PE_k and C'_{PE_k} be the sum of counts of children PEs of PE_k. According to the Algorithm 1, the path registers $C_{PE_k} - C'_{PE_k}$ transactions.

Based on Lemma 6 and Lemma 7, each transaction in the database is stored to one path in the systolic tree. Therefore, the systolic tree registers the complete set of transactions. □

APPENDIX D
CANDIDATE ITEMSET MATCHING

A. Design Principle

Lemma 1. *The support count stored in a PE is no less than the support count stored in the children PE; the item stored in a PE is smaller than the items stored in the children PE.*

Proof According to the construction process of the systolic tree, the i^{th} item of a new transaction with i items is stored in the i^{th} level of the tree. The support count in each PE on the path is increased by 1. Those PEs which are on $i+1^{th}$ or higher levels have the support count unchanged. Therefore, the support count in a parent PE is no less than the descendant PEs on the same path. Besides, the items in each transaction enter the systolic tree in an increasing order. The item stored in a PE must be smaller than the items stored in its children. □

Lemma 2. *For any non-empty PE_i, the support count of the itemset represented by the path from the root to PE_i is equal to the support count stored in PE_i; the support count of any itemset including $item_i$ in the path is equal to C_i.*

Proof Suppose the itemset represented by the path from the root to PE_i is $\{item_1{:}\, C_1, item_2{:}\, C_2, \cdots, item_i{:}\, C_i\}$. From Lemma 1, we know $C_1 \geq C_2 \geq \cdots \geq C_i$. The support count of itemset $\{item_1, \cdots, item_j\} (1 \leq j \leq i)$ is equal to C_j. Therefore, the support count of itemset $\{item_1, item_2, \cdots, item_i\}$ is C_i.

Since the support count of $item_i$ is C_i, the support count of any itemset including $item_i$ is no more than C_i. Meanwhile, the support count of any itemset including $item_i$ is no more than C_i since C_i is the smallest among $C_1, \cdots, C_i$. Therefore, the support count of any itemset in the path including $item_i$ is equal to C_i. □

To facilitate the understanding of other properties of a systolic tree related to candidate itemset matching, we first go through an example which performs matching on the constructed systolic tree in Fig. 2. Suppose the candidate itemset to be dictated is $\{C, D\}$. The paths containing the itemset in Fig. 2

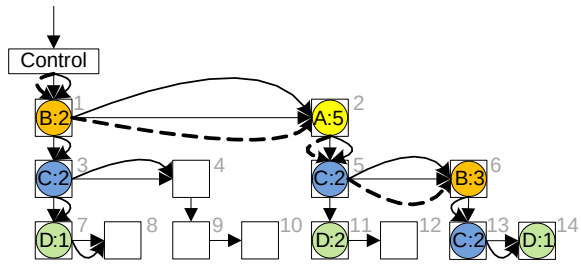


Fig. 8. SCAN Mode Example

are $\{PE1, PE3, PE7\}$ and $\{PE2, PE5, PE11\}$. The path $\{PE1, PE3, PE7\}$ can also be expressed as $\{B{:}\,2, C{:}\,2, D{:}\,1\}$ where the number after ':' indicates the support count. The first path indicates that itemset $\{B, C, D\}$ appears once in the database. Notice the path indicates that itemset $\{B, C\}$ appears twice. And itemset $\{B, C\}$ appears once together with item D. According to Lemma 2, the count stored in PE7 is the support count of C, D. Therefore, PE7 should report its count C_7 to the software. In path $\{PE2, PE5, PE11\}$, $\{A, C, D\}$ appears twice and itemset $\{A\}$ appears five times in the database. However, $\{A\}$ only appears twice with items C and D. According to Lemma 2, PE11 should report its count C_{11} to the software. The total support count of the itemset $\{C, D\}$ is equal to $C_7 + C_{11} = 1 + 2 = 3$.

Lemma 3. *The bottom door in a PE should be locked if the input item is smaller than the stored item. The PE with a locked bottom door will never be in a path which includes the candidate itemset.*

Proof Suppose the current PE is PE_c and the stored item is $item_c$. The set of the paths which include PE_c is P^c. According to Lemma 1, the item stored in a PE is smaller than the items stored in the children PE. If the incoming $item_t$ is smaller than $item_c$, $item_t$ must be smaller than the stored items in the descendants of PE_i. The algorithm will not find a PE in P^c whose stored item is equal to $item_t$. Therefore, the dictated itemset will never be contained in P^c. It is meaningless to forward $item_t$ to the children of PE_c. Thus, the bottom door should be locked. □

B. An Example

Let's illustrate the whole matching process with the example shown in Fig. 8. Suppose the threshold support count is four ($\xi = 4$). Now we want to check whether the candidate itemset {B,D} is frequent.

1) In the first clock cycle, a control signal which indicates the SCAN mode is first broadcast from the control PE.
2) In the second clock cycle, the item B is sent to the control PE. The control signal is sent to PE1 and then broadcast to other PEs in subsequent clock cycles.
3) In the third clock cycle, PE1 receives B. Step (2) in Algorithm 2 is triggered and B is forwarded to PE2. The $IsLeaf$ flag is set in PE1. The item D enters the control PE.
4) In the fourth clock cycle, PE2 receives the item B. Step (4) is triggered in PE2. The item B is sent to PE5. PE1 receives the item D. Step (4) is executed in PE1. The $IsLeaf$ flag is cleared in PE1.
5) In the fifth clock cycle, the item D enters PE2 and PE3. The item B enters PE5. Step (4) is executed in PE2 and PE3. Step (3) is executed in PE5. The bottom door of PE5 is locked. The item B is sent to PE6.
6) In the sixth clock cycle, the item D enters PE4, PE7 and PE5. The item B enters PE6. Step (1) is executed in PE4. Step (2) is executed in PE7. The $IsLeaf$ flag is set in PE7. Step (4) is triggered in PE5. Step (2) is executed in PE6.
7) In the seventh clock cycle, the item D enters PE8 and PE6. Step (4) is executed in PE6. The $IsLeaf$ flag is cleared.
8) In the ninth clock cycle, the item D enters PE14 and step (2) is executed. The $IsLeaf$ flag is set in PE14.

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 parent. The count in both PE7 and PE14 is one. Thus the total count of the candidate itemset $\{B, D\}$ is two which is less than four. Therefore, this candidate itemset is not frequent. In Fig. 8, the dashed lines show the track of the item B; the bold 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.

C. Completeness of the Itemset Matching

We can provide some formal assurances that the SCAN mode algorithm can dictate the candidate itemset correctly.

Lemma 8. *During the matching, the items in the candidate itemset arriving later follow the path of the previous items in the same transaction.*

Proof A control signal initializes $IsLeaf$ and opens the bottom door before dictating a candidate itemset. In the horizontal direction, the items are always sent to the right sibling as discussed above. Therefore, if we can prove the items in the candidate itemset arriving later follow the path of the previous items in the vertical direction, the assertion is also true in the horizontal direction.

Next we show that the assertion is true in the vertical direction. Suppose the candidate itemset is $\{item_1, item_2, \cdots\}$. For an arbitrary path $\{PE_s, PE_{s+1}, \cdots, PE_t\}$ in the systolic tree, PE_{s+i} is the parent of PE_{s+i+1} and $item_{s+i}$ is the item stored in PE_{s+i}. If a PE is empty, no item is forwarded. We assume all PEs are non-empty. Consider the clock cycle t at which $item_1$ enters PE_s in Algorithm 2.

1) If step (2) is triggered in PE_s, $item_1$ is equal to $item_s$ and $item_1$ is not forwarded in the vertical direction any more. Since $item_2 > item_1$ and $item_1 = item_s$, $item_2 > item_s$. Thus step (4) is triggered in PE_s in clock cycle $t + 1$. Consequently, $item_2$ is sent to the direction of PE_{s+1}.
2) If step (3) is triggered in PE_s, the bottom door is locked and $item_1$ is forwarded to the sibling. In clock cycle $t + 1$, either step (3) or step (4) is triggered. In both steps, $item_2$ is sent to the sibling. In step (4), $item_2$ is

not forwarded to PE_{s+1} since the bottom door of PE_s is locked.

3) If step (4) is triggered in PE_s, $item_1$ is sent to PE_{s+1} and the sibling. Since $item_2 > item_1$ and $item_1 > item_s$, then $item_2 > item_s$. Thus step (4) is triggered in PE_s in clock cycle $t+1$.

In all cases discussed above, $item_2$ follows the path of $item_1$. Similarly, $item_2$ follows the path of $item_1$ in other PEs. In a similar way we can prove $item_{i+1}$ follows the path of $item_i$ in the candidate itemset. □

Lemma 9. *The first item of a candidate itemset is correctly located in the systolic tree.*

Proof Suppose the first item of the candidate itemset is $item_1$. Since $item_1$ is the first item in the candidate itemset, the bottom doors of all PEs are open. Consider the leftmost path $\{PE_1, PE_2, \cdots, PE_w\}$ which consists of the leftmost PEs of each level. $PE_i(1 \leq i \leq w)$ is the parent of PE_{i+1} and $item_i(1 \leq i \leq w)$ is the item stored in PE_i.

1) If the leftmost path contains $item_1$, there is a stored item (say $item_r$) equal to $item_1$. According to the SCAN mode algorithm and Lemma 1, step (4) is triggered in $PE_i(1 \leq i < r)$. Step (2) is triggered in PE_r. In both cases, $item_1$ is sent to the siblings and paths which potentially contain it. $PE_i(r < i \leq w)$ will not receive $item_1$. Thus any PE which is a descendant or siblings of $PE_i(r < i \leq w)$ will not contain $item_1$. This is because all of them are descendants of PE_r. According to the construction process of the systolic tree and Lemma 1, the items stored in them are larger than $item_1$.
2) If the leftmost path does not contain $item_1$, either step (3) or step (4) is triggered in $PE_i(1 \leq i \leq w)$. In step (4), $item_1$ is sent to all connected PEs. $item_1$ is not sent to the descendants in step (3). This is justified in Lemma 3.

A similar analysis can be applied to other paths. Therefore, $item_1$ will be sent to any PE which is likely to contain it. Therefore, the first item of a candidate itemset is correctly located in the systolic tree. □

Theorem 2. *A candidate itemset is dictated correctly using the SCAN mode algorithm.*

Proof Suppose the candidate itemset is $item_1, item_2, \cdots$. Based on Lemma 8 and Lemma 9, $item_1$ is correctly located in the systolic tree and other items follow the path of $item_1$. For an arbitrary path of which $item_1$ is located in PE_{s1}, suppose $item_1$ enters PE_{s1} in clock cycle t. In clock cycle $t+1$, $item_2$ enters PE_{s1}. Step (4) is triggered and $item_2$ is forwarded the leftmost child of PE_{s1}. Using a similar proof as in Lemma 9, $item_2$ is correctly located in the sub-tree rooted in PE_{s1}. Similarly, other items in the candidate itemset are correctly located in the systolic tree. Therefore, the candidate itemset is located correctly using the SCAN mode algorithm.

Also notice that the $IsLeaf$ flag is set when $item_1$ enters PE_{s1}. Since step (4) is triggered while $item_2$ entering PE_{s1}, the $IsLeaf$ flag is cleared. After all items in the candidate itemset are processed, only PEs whose stored items match the final item set $IsLeaf$ and serve as the reporting PEs. Therefore, the candidate itemset is dictated correctly using the SCAN mode algorithm. □

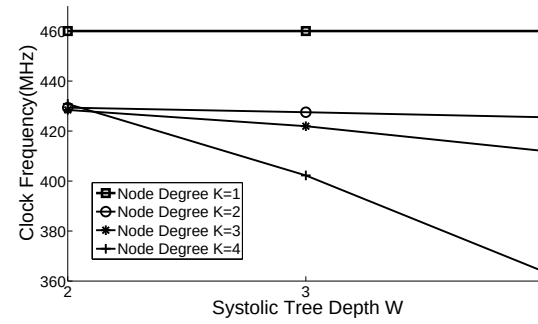


Fig. 9. Systolic Tree Clock Frequency

D. Systolic Tree Clock Frequency

APPENDIX E
DATABASE PROJECTION

Lemma 4. *Given a database with n frequent items, the number of projected sub-databases whose transaction only contain the last $N = min(K, W)$ frequent items is at most 2^{n-N} where $n \geq N$.*

Proof Suppose the n frequent items are ordered in arbitrary order as $i_1, i_2, i_3, \cdots, i_n$. The items in each transaction are also reordered according to the sequence of $i_1, i_2, i_3, \cdots, i_n$. Those transactions begin with item $\{i_{n-N+1}\}, \{i_{n-N+2}\}$, ..., or $\{i_n\}$ can be put into the ϕ-projected database. The ϕ-projected database starting with the last N frequent items contains no more than N frequent items in each transaction. The number of i_{n-N}-projected databases is 1. The items in each transaction of it are included in $\{i_{n-N}, i_{n-N+1}, ..., i_N\}$. The number of sub-databases whose transactions begin with i_{n-N-1} is 2. One is the $i_{n-N-1}i_{n-N}$-projected database; the other is the $i_{n-N-1}\phi$-projected database. The number of sub-databases starting with $i_j(1 \leq j \leq n-N)$ is 2^{n-N-j}. Therefore, the total number of sub-databases is $\sum_{j=1}^{j=n-N} 2^{n-N-j} = 2^{n-N}$. □

Theorem 3. *For each frequent itemset α, the α-projected database can be derived from the projected trees of α conditional FP-tree.*

Proof Let T_i^α be an arbitrary transaction in the *projected trees* of the α's conditional FP-tree. Based on the *subroot* and FP-tree construction algorithms, there must be a transaction in the original database containing both T_i^α and α. Based on the completeness of FP-tree [1], any transaction containing both T_i^α and α can be found in α's conditional FP-tree and its projected trees. □

APPENDIX F
EXPERIMENTAL ANALYSIS

The clock frequency of a post place-and-route FPGA design or different combination values of K and W is shown in Fig. 9. As can be seen in this figure, the clock frequency drops gradually when the node degree or the tree depth increases due

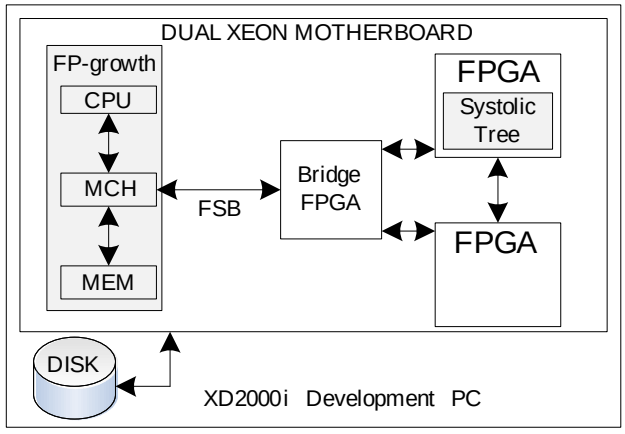


Fig. 10. Simplified XtremeData XD2000i Architecture

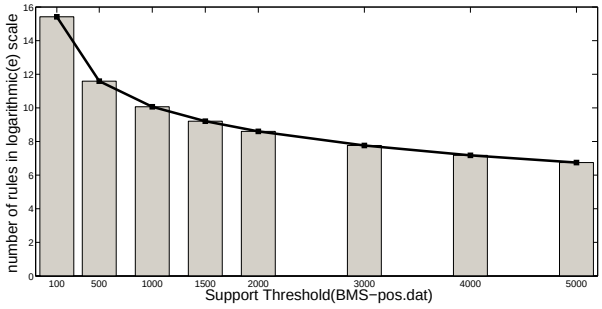


Fig. 11. Number of Rules Mined in Hardware

to the increasing number of interfaces and interconnections between PEs and the increasing routing delay for a larger circuit.

A. XtremeData XD2000i Platform

As previously mentioned, we chose the XtremeData XD2000i high performance reconfigurable computing platform to perform our performance experiments. The XD2000i development system comprises of a Dual Xeon motherboard with one Intel Xeon processor in the first socket, and two Altera Stratix III EP3SE260 FPGAs in the second socket, as shown in Fig. 10. The Xeon processor has four cores running at 1.6GHz and communicates with the FPGAs and the 4GB of system memory through an Intel Memory Controller Hub (MCH). The CPU prepares a data buffer of a known size in a system memory area used for sending and receiving data from the application FPGAs. Only one send request may be active at a time. The receive portion must complete before the next send is submitted. However, the sender does not have to wait to receive an acknowledgement of receipt and may return immediately so that additional CPU work may be performed before submitting the receiving request.

B. Performance Analysis of BMS-pos.dat

In Fig. 11, the value of the y axis is the number of frequent patterns mined in hardware which includes the patterns generated from the hardware directly and the patterns combined with those generated from the software. The number grows exponentially with the decrease of the support threshold. Even though the projection overhead also increases with the decrease of the support threshold, the number of patterns in hardware goes up sharply when the support threshold is less than 1000. This explains why the systolic tree is faster than FP-growth for BMS-POS.dat when the support threshold is less than 1000.