An Accelerator for Frequent Itemset Mining from Data Streams with Parallel Item Tree

Kasho Yamamoto, Tsunaki Sadahisa, Dahoo Kim, Eric S. Fukuda,
Tetsuya Asai, and Masato Motomura
Graduate School of IST, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan
(yamamoto@lalsie.ist.hokudai.ac.jp)

Abstract— Frequent itemset mining attempts to find frequent subsets in a transaction database. In this era of big data, demand for frequent itemset mining is increasing. Therefore, the combination of fast implementation and low memory consumption, especially for stream data, is needed. In response to this, we optimize an online algorithm, called Skip LC-SS algorithm [1], for hardware. In this paper, we present an efficient architecture based on this algorithm.

Keywords—data mining; frequent itemsets; stream processing; hardware accelerator

I. Introduction

Data stream mining for frequent itemsets (DSM-FI) is one of the most important and fundamental challenges of online data stream mining. DSM-FI cannot store an entire stream in memory and perform a single scan because the stream is treated as continuous. Counting the occurrences of all itemsets is also unrealistic due to the constraint of memory capacity. Therefore, substantial research has focused on the one pass approximation algorithm [3]. For example, Lossy counting [3] eliminates infrequent itemsets for each transaction. The drawback to this algorithm is that a sudden burst of stream data can produce memory overflow. On the other hand, the space saving [4] and Skip LC-SS algorithms address this memory overflow by fixing the number of stored itemsets. Unfortunately, this process requires substantial memory access and tends to produce memory access bottlenecking during the processing in CPU. We propose a coprocessor architecture that enables the high-speed processing of frequent itemset mining (FIM) using FPGA, which does not contain large memory stores but has substantial stores of small memory and can access them simultaneously.

II. Background and Preliminary Work

A. RELATED WORK

FIM implementation using FPGAs has already been studied. Baker et al. [5] proposed the first architecture for Apriori. In 2013, the architecture for Eclat was presented by Zhang et al. [6]. However, these architectures cannot process sudden bursts of stream data.

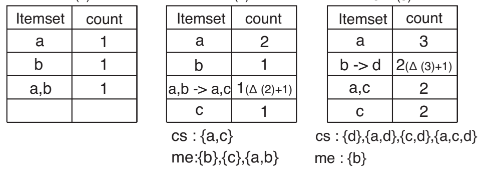
B. SKIP LC-SS ALGORITHM

The skip LC-SS algorithm stores a fixed number of itemsets. Therefore, only an amount of $\mathcal{O}(k)$ memory space is required. Here, constant k represents the number of stored itemsets. Itemset e is stored using a tuple, such as $\langle e, count(e)\rangle$, in entry table D, where $count(e)$ is the number of occurrences of e. In order to accelerate the implementation process, this algorithm skips a portion of the process under certain conditions. A brief outline of this baseline algorithm follows.

1. Consider itemset $E = \{e_1, e_2, ...e_{2^{|T_i|}-1}\}$ in the transaction Ti as follows:
   (a) if $\langle e_i, count(e_i)\rangle \in D$, $count(e_i) \mathrel{+}= 1$,
   (b) else if $|D| < k$, store new entry $\langle e_i, 1\rangle$ in D,
   (c) else register e_i as replacement candidate set (cs).
2. Replace me with cs, written $\langle c, \Delta(i) + 1\rangle$, where c is in cs and Δ is the error count $\Delta(1) = 0$.
3. Update $\Delta(i + 1)$ as follows:

$$\begin{aligned} \mathrm{A}: &\ |me| > |cs|, \Delta(i+1) = count(cs(i)),\\ \mathrm{B}: &\ |me| \le |cs|, \Delta(i+1) = \Delta(i) + 1. \end{aligned}$$

Example Fig.1 shows how to update the entry table using this algorithm. If $i = 3$, the replacement target can be freely selected from me. If $i = 4$, although cs is greater than ms, only one element of cs must be replaced.



i = 1 Δ(1) = 0

Itemset	count
a	1
b	1
a,b	1

i = 2 Δ(2) = 0

Itemset	count
a	2
b	1
a,b -> a,c	1(Δ (2)+1)
c	1

cs : {a,c}
me:{b},{c},{a,b}

i = 3 Δ(3) = 1

Itemset	count
a	3
b -> d	2(Δ (3)+1)
a,c	2
c	2

cs : {d},{a,d},{c,d},{a,c,d}
me : {b}

Fig. 1. Entry table update from stream S consists of three transactions: $\{a, b\}$, $\{a, c\}$, and $\{a, c, d\}$, with $k = 4$.

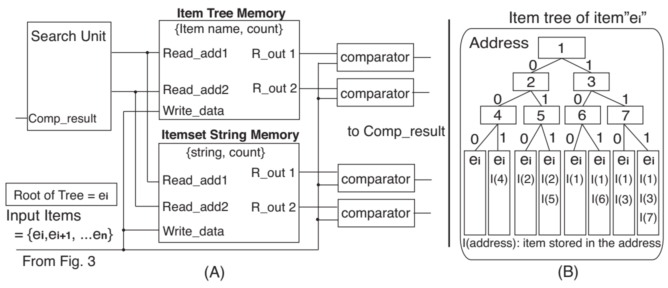


Fig. 2. (A) Item tree and search unit; (B) Address of item tree

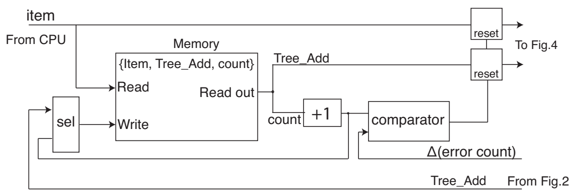


Fig. 3. Memory required to access BRAM tree and prune infrequent item

III. Architecture

An FPGA has parallel accessible memory (BRAM), but this memory is much smaller than that used in the software. Therefore, it is impossible to maintain the entry table as an array. We solve this problem by using a tree structure. In order to avoid an increase in the amount of data in pointer-based approach, we consider a binary-tree structure, which can be searched using the shift operation of the memory address. If the address of a parent node is n, its child node can be accessed using the left-shift operation. This means the address of a child node is $2n$, $2n+1$ [Fig. 2(B)]. Therefore, trees do not need to store node addresses. If a parent node has the same item as the input item, then side 1 should be searched. Otherwise, side 2 should be searched. In the former case, the search unit stores the opposite node's address in the stack and, once it reaches the leaf node, resumes the process from the address stored in the stack [Fig. 2(A)].

In addition, forming a tree plurality, in which each root represents an item stored in the small memory, allows us to simultaneously update each entry table. In order to access the BRAM that holds each item tree, the BRAM tree's addresses and each item must be stored. Memory stores the number of occurrences of each item, which enables the removal of infrequent items from the transaction by comparing Δ with the number of item occurrences given by the BRAM address. For more information, refer to the SS-ST algorithm [1] [Fig. 3].

Items e_i in transaction $T = \{e_1, e_2, ..., e_n\}$ are given to each item tree by the distributor, i.e., $\{e_1, e_2, ..., e_n\}, \{e_2, ..., e_n\}, \{e_n\}$ [Fig. 4]. This module determines whether the transaction length is greater than a certain length and terminates the transaction processing (t2-skip [1]).

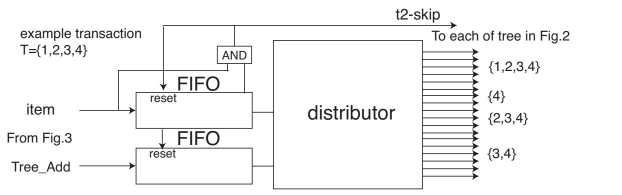


Fig. 4. Distribution of items to BRAM and performance of T_2 skip

The tree structure allows itemsets to be stored using less memory space than the method in which entry tables store itemsets in strings. Still, it cannot create entry tables as large as those created using the software. Furthermore, data that has already been replaced is, again, frequently immediately replaced. Therefore, we partially store itemsets in strings. Each string represents all the subsets of the string. This method reduces memory consumption from $\mathcal{O}(2^L * L)$ to $\mathcal{O}(L)$, where L is the transaction length. Also, in the best case scenario, this method processes replacement in $\mathcal{O}(2^L)$. If the itemset about to be replaced is stored in a string, we can reduce the cost of deleting and building trees. Morevoer, the FPGA simultaneously addresses both string and tree data. Therefore, this method has a high affinity with hardware.

IV. Conclusion

In this paper, we proposed an efficient architecture for frequent itemset mining from data streams. This architecture is an optimized skip LC-SS algorithm used for hardware. It accelerated implementation by addressing the CPU's memory access bottlenecking, e.g. replacement, using FPGA parallelism. In the future, we hope to make some improvements, especially as they pertain to the data structure.

References

[1] Y. Yamamoto, K. Iwanuma, S. Fukuda, "Resource-oriented Approximation for Frequent Itemset Mining from Bursty Data Streams," Proc. of SIGMOD '14.

[2] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," Proc. of Very Large Data Bases (VLDB '14).

[3] G. S. Manku, R. Motwani, "Approximate frequent counts over data streams, "Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB '02).

[4] A. Metwally, D. Agrawal, A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," Proc. Int. Conf. on Database Theory (ICDT '05).

[5] Z. Baker, V. Prasanna, "Efficient hardware data mining with the Apriori algorithm on FPGAs," Proc. of the Thirteenth Annual IEEE Symposium on Field Programmable Custom Computing Machines (FCCM '05).

[6] Y. Zhang, F. Zhang, Z. Jin, J. D. Bakos, "An FPGA-Based Accelerator for Frequent Itemset Mining," ACM Transactions on Reconfigurable Technology and Systems 6.1 (2013): 2.