Association Rule Mining with the Micron Automata Processor

Ke Wang[†], Yanjun Qi[†], Jeffrey J. Fox[‡], Mircea R. Stan[§] and Kevin Skadron[†]
[†]*Dept. of Comp. Sci.,* [‡]*Dept. of Mater. Sci.,* [§]*Dept. of Elec. & Comp. Eng.*
University of Virginia
Charlottesville, VA, 22904 USA
{*kewang, yanjun, jjf5x, mircea, skadron*}*@virginia.edu*

***Abstract*—Association rule mining (ARM) is a widely used data mining technique for discovering sets of frequently associated items in large databases. As datasets grow in size and real-time analysis becomes important, the performance of ARM implementation can impede its applicability. We accelerate ARM by using Micron's Automata Processor (AP), a hardware implementation of non-deterministic finite automata (NFAs), with additional features that significantly expand the APs capabilities beyond those of traditional NFAs. The Apriori algorithm that ARM uses for discovering itemsets maps naturally to the massive parallelism of the AP. We implement the multipass pruning strategy used in the Apriori ARM through the APs symbol replacement capability, a form of lightweight reconfigurability. Up to 129X and 49X speedups are achieved by the AP-accelerated Apriori on seven synthetic and real-world datasets, when compared with the Apriori single-core CPU implementation and Eclat, a more efficient ARM algorithm, 6-core multicore CPU implementation, respectively. The AP-accelerated Apriori solution also outperforms GPU implementations of Eclat especially for large datasets. Technology scaling projections suggest even better speedups from future generations of AP.**

***Keywords*-Automata Processor; association rule mining; frequent set mining**

I. Introduction

Association Rule Mining (ARM), also referred to as Frequent Set Mining (FSM), is a data-mining technique that identifies strong and interesting relations between variables in databases using different measures of "interestingness". ARM has been the key module of many recommendation systems and has created many commercial opportunities for on-line retail stores. In the past 10 years, this technique has also been widely used in web usage mining, traffic accident analysis, intrusion detection, market basket analysis, bioinformatics, etc.

As modern databases continue to grow rapidly, the execution efficiency of ARM becomes a bottleneck for its application in new domains. Many previous studies have been devoted to improving the performance of sequential CPU-based ARM implementations. Different data structures were proposed, including horizontal representation, vertical representation, and matrix representation [1]. Multiple algorithms have been developed including *Apriori* [2], *Eclat* [3] and *FP-growth* [4]. A number of parallel acceleration based solutions have also been developed on multi-core CPU [5], GPU [6] and FPGA [7].

Recently, Micron proposed a novel and powerful non-von Neumann architecture, the Automata Processor (AP). The AP architecture demonstrates a massively parallel computing ability through a large number of state elements. It also achieves fine-grained communication ability through its configurable routing mechanism. These advantages make the AP suitable for pattern-matching centered tasks like ARM. Very recently, the AP has been successfully used to accelerate the tasks of regular expression matching [8] and DNA motif searching [9].

In this paper, we propose an AP-based acceleration solution for ARM. A Non-deterministic Finite Automata (NFA) is designed to recognize the sets of frequent items. Counter elements of the AP are used to count the frequencies of itemsets. We also introduce a number of optimization strategies to improve the performance of AP-based ARM. On multiple synthetic and real-world datasets, we compare the performance of the proposed AP-accelerated *Apriori* versus the *Apriori* single-core CPU implementation, as well as multicore and GPU implementations of the *Eclat* algorithm. The proposed solution achieves up to 129X speedups when compared with the *Apriori* single-core CPU implementation and up to 49X speedups over multicore implementation of *Eclat*. It also outperforms GPU implementations of *Eclat* in some cases, especially on large datasets.

Overall, this paper makes three principal contributions:

1) We develop a CPU-AP computing infrastructure to improve the *Apriori* algorithm based ARM.
2) We design a novel automaton structure for the matching and counting operations in ARM. This structure avoids routing reconfiguration of the AP during the mining process.
3) Our AP ARM solution shows performance improvement and broader capability over multi-core and GPU implementations of *Eclat* ARM on large datasets.

II. Association Rule Mining

Association rule mining (ARM) among sets of items was first described by Agrawal and Srikant [10]. The ARM problem was initially studied to find regularities in the

shopping behavior of customers of supermarkets and has since been applied to very broad application domains.

In the ARM problem, we define $I = i_1, i_2, ..., i_m$ as a set of interesting items. Let $T = t_1, t_2, ..., t_n$ be a database of transactions, each transaction t_j is a subset of I. Define $x_i = \{i_{s1}, i_{s2}, ..., i_{sl}\}$ be a set of items in I, called an itemset. The itemset with k items is called k-itemset. A transaction t_p is said to cover the itemset x_q *iff* $x_q \subseteq t_p$. The support of x_q, $Sup(x_q)$, is the number of transactions that cover it. An itemset is known as frequent *iff* its support is greater than a given threshold value called minimum support, *minsup*. The goal of association rule mining is to find out all itemsets which supports are greater than *minsup*.

III. Automata Processor

Micron's Automata Processor (AP) is a massively parallel non-von Neumann accelerator designed for high-throughput pattern mining.

A. Function elements

The AP chip has three types of functional elements - the state transition element (STE), the counters, and the Boolean elements [8]. The state transition element is the central feature of the AP chip and is the element with the highest population density. Counters and Boolean elements are designed to work with STEs to increase the space efficiency of automata implementations and to extend computational capabilities beyond NFAs.

B. Speed and capacity

Micron's current generation AP - D480 chip is built on 45nm technology running at an input symbol (8-bit) rate of 133 MHz. The D480 chip has two half-cores and each half-core has 96 blocks. Each block has 256 STEs, 4 counters and 12 Boolean elements. In total, one D480 chip has 49,152 processing state elements, 2,304 programmable Boolean elements, and 768 counter elements [8]. Each AP board can have up to 48 AP chips that can perform matching in parallel [11]. Each AP chip has a worst case power consumption of 4W [8]. The power consumption of a 48-core AP board is similar to a high-end GPU card.

Each STE can be configured to match a set of any 8-bit symbols. The counter element counts the occurrence of a pattern described by the NFA connected to it and activates other elements or reports when a given threshold is reached. One counter can count up to 2^{12} which may not be enough for ARM counting in some cases. In such a scenario, two counters can be combined to handle a larger threshold. Counter elements are a scarce resource of the AP current-generation chip and therefore are a main limiting factor of the capacity of the ARM automaton proposed in this work.

C. Input and output

The AP takes input streams of 8-bit symbols. Each AP chip is capable of processing up to 6 separate data streams concurrently, although we do not use this feature for this work. The data processing and data transfer are implicitly overlapped by using the input double-buffer of the AP chip. Any STE can be configured to accept the first symbol in the stream (called start-of-data mode, small "1" in the left-upper corner of STE in the following automaton illustrations), to accept every symbol in the input stream (called all-input mode, small "∞" in the left-upper corner of STE in the following automaton illustrations) or to accept a symbol only upon activation. The all-input mode will consume one extra STE.

Any type of element on the AP chip can be configured as a reporting element; one reporting element generates a one-bit signal when the element matches the input symbol. One AP chip has up to 6144 reporting elements. If any reporting element reports at a cycle, the chip will generate an output vector which contains signals of "1" corresponding to the elements that report at that cycle and "0"s for reporting elements that do not report. If too many output vectors are generated, the output buffer can fill up and stall the chip. Thus, minimizing output vectors is an important consideration for performance optimization.

D. Programming and reconfiguration

Automata Network Markup Language (ANML) is an XML language for describing the composition of automata networks. ANML is the basic way to program automata on the AP chip. Besides ANML, Micron provides a graphical user interface tool called the AP Workbench for quick automaton designing and debugging. A "macro" is a container of automata for encapsulating a given functionality, similar to a function or subroutine in common programming languages. A macro can be defined with parameters of symbol sets of STEs and counter thresholds which can be instantiated with actual arguments. Micron's AP SDK also provides C and Python interfaces to build automata, create input streams, parse output and manage computational tasks on the AP board.

Placing automata onto the AP fabric involves three steps: compilation optimization, routing configuration and STE symbol set configuration. The initial compilation of automata onto the AP involves all these three steps, while the pre-compiled automata only requires the last two steps. The compilation optimization usually takes tens of seconds. The routing configuration of the whole board needs about 5 milliseconds. The symbol set configuration takes approximately 45 milliseconds for an entire board.

IV. Related Work

A. Sequential algorithms

After describing the association rule mining problem [10], Agrawal and Srikant proposed the *Apriori* algorithm. The *Apriori* algorithm is a well known and widely used algorithm. It prunes the search space of itemset candidates in a breadth-first-search scheme the using *downward-closure* property.

The Equivalent Class Clustering *Eclat* algorithm was developed by Zaki [3]. The typical *Eclat* implementation adopts a vertical bitset representation of transactions and depth-first-search. The low level operation, e.g. the bit-level intersection of two itemsets, exposes more instruction-level parallelism, which enables *Eclat* to outperform *Apriori* on conventional architectures.

Han *et al.* [4] introduced another popular ARM algorithm, *FP-growth*. By utilizing a Frequent-Pattern tree data structure to avoid multi-pass database scanning, *FP-growth* has very good performance in many cases. However, the poor memory-size scaling of the Frequent-Pattern tree prevents the use of *FP-growth* for very large databases.

B. Multi-thread & multi-process

Zaki *et al.* [12] developed a parallel version of the *Apriori* algorithm for a shared memory (SM) multi-core platform. This implementation achieved 8X speedup on a 12-processor SM platform for synthetic datasets. Liu *et al.* [13] proposed a parallel version of *FP-growth* on a multi-core processor. This work achieved 6X speedup on an 8-core processor.

Pramudiono and Kitsuregawa [14] proposed a parallel algorithm of *FP-growth* achieving 22.6X speedup on a 32-node cluster. Ansari *et al.* [15] developed an MPI version of the *Apriori* algorithm and achieved 6X speedup on an 8-node cluster.

C. Accelerators

An FPGA-based solution was proposed to accelerate the *Eclat* algorithm [7] by Zhang *et al.* This solution achieved a speedup of 68X on a four-FPGA board with respect to the CPU sequential implementation of *Eclat*.

Fang *et al.* [16] designed a GPU-accelerated implementation of *Apriori*. 2X-10X speedup is achieved with NVIDIA GeForce GTX 280 GPU when compared with CPU sequential implementation. Zhang *et al.* [6] proposed another GPU-accelerated *Eclat* implementation and achieved 6X-30X speedup relative to the state-of-the-art sequential *Eclat* and *FP-Growth* implementations. Zhang also proposed the *Frontier Expansion algorithm*, which hybridizes breadth-first-search and depth-first-search to expose more parallelism in this *Eclat* implementation. This implementation also generalizes the parallel paradigm by a producer-consumer model that makes the implementation applicable to multi-core CPU and multiple GPUs.

According to the previous cross-algorithm comparison, there is no clear winner among the different sequential algorithms and implementations [17]. However, to our knowledge, Zhang's *Eclat* [6] is the fastest parallel ARM implementation. Thus we compare our AP-accelerated *Apriori* implementation with Zhang's parallel *Eclat* implementation on both multi-core CPU and GPU platforms. However, as more parallelism is exposed, the vertical representation of many itemsets has to be kept in the memory (main memory or GPU global memory) simultaneously. The trade-off between memory and performance (parallelism) still exists, particularly for large datasets on the GPU. In contrast, our AP-accelerated *Apriori* solution does not rely on local memory and therefore is less sensitive to the data size.

Further exploration is needed regarding how different algorithms scale with diverse multicore CPUs and specialized accelerators, and the role of optimization techniques such as blocking and data-layout optimization. This is an interesting area for future work.

V. Mapping ARM problem onto the AP

A. Apriori *Algorithm*

The *Apriori* algorithm framework is adopted for the AP to reduce the search space as itemset size increases. The *Apriori* algorithm is based on *downward-closure* property: all the subsets of a frequent itemset are also frequent and thus for an infrequent itemset, all its supersets must also be infrequent. In the *Apriori* framework, candidates of $(k+1)$-itemsets are generated from known frequent k-itemsets by adding one more possible frequent item. The mining begins at 1-itemset and the size of candidate itemsets increases by one at each level. In each level the *Apriori* algorithm has two major operations:

1) Generating candidates of frequent $(k+1)$-itemsets from known frequent k-itemsets
2) Counting support numbers of candidate itemsets and comparing these support numbers with *minsup*

The support counting step is the performance bottleneck of the *Apriori* algorithm, particularly for the large databases. The hardware features of the AP are well suited for matching and support-counting many itemsets in parallel. Therefore, we propose to use the AP to accelerate the support-counting step in each level.

B. Program infrastructure

Figure 1 shows the complete workflow of the AP-accelerated ARM proposed in this paper. The data preprocessing stage creates a data stream from the input transactional dataset and makes the data stream compatible with the AP interface. Preprocessing consists of the following steps:

1) Filter out infrequent items from transactions
2) Recode items into 8-bit or 16-bit symbols
3) Recode transactions

4) Sort items in transactions
5) Connect transactions by a special separator symbol to form the input data stream for the AP

Step 1 is a common step in almost all existing ARM implementations that helps to avoid unnecessary computing on infrequent items and reduces the number of items and transaction sizes. Depending on the population of frequent items, the items can be encoded by 8-bit ($freq\_item\# < 255$) or 16-bit symbols ($254 < freq\_item\# < 64516$) in step 2. Different encoding schemes lead to different automaton designs. Step 3 deletes infrequent items from the transactions, applies the codes of items to all transactions, encodes transaction boundary markers, and removes very short transactions (less than two items). Step 4 sorts items in each transaction (in any given order) to avoid needing to consider all permutations of a given itemset, and therefore saves STE resources on the AP. We adopt descending sorting according to item frequency (proposed by Borgelt [18]). The data pre-processing is only executed once in the whole workflow.

Each iteration of the loop shown in Figure 1 explores all frequent k-itemsets from the candidates generated from $(k-1)$-itemsets. The candidates are generated from the CPU and are compiled onto the AP by using the automaton structure designed in this paper. The input data formulated in pre-processing is then streamed into the AP for counting.

C. Automaton for matching and counting

Figure 2 shows the initial automaton design for ARM. The items are coded as digital numbers in the range from 0 to 254, with the number 255 reserved as the separator of transactions. Each automaton for ARM has two components: matching and counting. The matching component is implemented by an NFA, the groups of STEs in Figure 2a and 2b, to recognize a given itemset. Note that unlike string matching, the itemset matching in ARM needs to consider the cases of discontinuous patterns of items.

For example, consider the itemset of {6,11}; in transactions such as [1,6,11] or [3,6,11,15], item "11" is next to item "6", while, in other cases such as [2,6,8,11] or [6,7,8,9,11], there are an unknown number of items between "6" and "11". The NFA we designed can capture all possible continuous and discontinuous variants of a given itemset. The only requirement is the order of items appearing in the transactions, which is already guaranteed by sorting in data pre-processing.

As shown in Figure 2, the NFA for itemset matching can be divided into multiple levels. Each level except "Level 0" has two STEs: the top STE holds the activation in this level and the bottom STE triggers the next level if one item in a given transaction matches it. For each automaton corresponding to a given itemset, activation begins at "Level 0" and will move forward (to the right) to "Level 1" when the transaction separator is seen in the input. Each level will

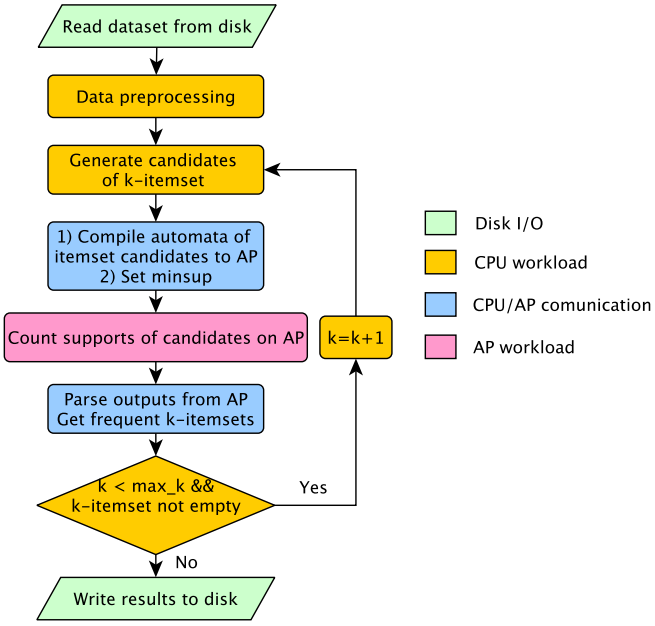
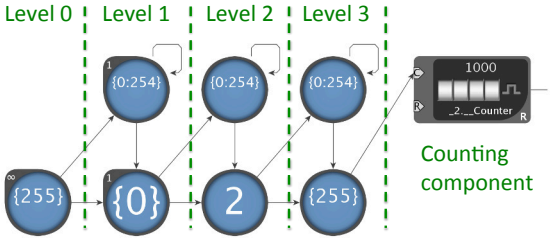
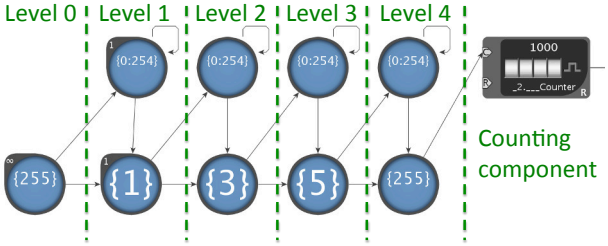


Figure 1: The workflow of AP-accelerated ARM



(a) Automaton for itemset {0, 2}



(b) Automaton for itemset {1, 3, 5}

Figure 2: Initial design of automata for ARM itemset matching and support-counting. Blue circles and black boxes are STEs and counters, respectively. The numbers on an STE represent the symbol set that STE can match. "0:254" means any item ID in the range of 0-254. Symbol "255" is reserved as the transaction separator.

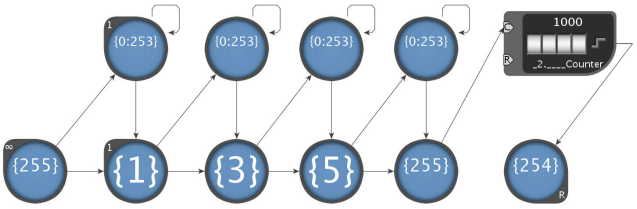


Figure 3: Optimization for minimizing the output. The node with 254 is the "reporter".

trigger the next level if the item represented by this level (bottom STE) is seen in the input. If the item of the current level is not seen, the activation of the current level will be held by the top symbol, until the end of this transaction when separator symbol is seen. The itemset matching is restarted in the beginning of each transaction by the "Level 0" STE.

The counting component uses an on-chip counter element to calculate the frequency of a given itemset. If the last level has been triggered, the matching component waits for the separator symbol to indicate the end of a transaction. The separator symbol then activates the counter, incrementing it by one. If the threshold, which is set to *minsup*, is reached in the counter, this automaton produces a report signal at this cycle. After processing the whole dataset on the AP, the output vectors are retrieved. Each itemset with a frequency above the minimum support will appear in the output. Although the automata shown in Figure 2 already implement the basic functions of matching and counting for ARM, there is still much room for performance optimization. We will talk about the performance optimization in the next subsection. We only show the automata for 8-bit encoding scheme in this paper. The automata for 16-bit encoding scheme are designed in a similar way but use two connecting STEs to match an item.

D. Performance optimization

In this paper we propose three optimization strategies to maximize the computation performance of the AP.

1) Output optimization: The first strategy is to minimize the output from the AP. In the initial automaton design shown in Section V-C, the AP chip creates a report vector at each cycle whenever there is at least one counter report. Each report vector carries the information about the cycle count for this report. Therefore, the AP chip creates many report vectors during the data processing. These report vectors may fill up the output buffers and cause stalls during processing. However, solving the ARM problem only requires identifying the frequent itemsets; the cycle at which a given itemset reaches the minimum support level is irrelevant. We therefore modify the design of the reporting element and postpone all reports to the last cycle (Figure 3). We utilize the "latch" property of the counter to keep activating another STE connected to this counter after the counter *minsup* is reached. We call this STE the "reporter". One symbol (i.e., 254) is reserved to indicate the end of a transaction stream and this end-of-stream symbol matches to the reporter STE and triggers the actual output. Consequently, the global set of items is 0-253, which ensures that the ending symbol 254 will not appear in the middle of the transaction stream. With this modification, only one output vector will be produced in the end of data stream.

Another benefit of this modification is that it eliminates the need to merge multiple report vectors as a post-processing step on the CPU. Instead, the counting results can be parsed from only one report vector.

2) Avoid routing reconfiguration: As shown in Figure 2, when the mining of k-itemsets finishes, the automata for $(k+1)$-itemset need to be compiled onto the AP to replace the automata for k-itemsets. The automata reconfiguration involves both routing reconfiguration and symbol replacement steps, because the NFAs that recognize itemsets of different sizes have different structures (compare Figure 2a and Figure 2b). On the other hand, the AP also provides a mechanism to only replace the symbol set for each STE while the connections between AP elements are not modified. The time of symbol replacement depends on how many AP chips are involved. The max symbol replacement time is 45ms if all STEs update their symbol sets.

To remove the routing reconfiguration step, we propose a general automaton structure supporting itemsets with different sizes. The idea is to add multiple entry paths to the NFA shown in Figure 2. To count the support of a given itemset, only one of the entry paths is enabled by matching to the transaction separator symbol, while the other entry paths are blocked by a reserved special symbol. This special symbol can be the same as the data stream ending symbol (i.e., "254") discussed in Section V-D1. This structure is called multiple-entry NFA for variable-size itemset (ME-NFA-VSI). 10% total reconfiguration time, 5ms, is saved by using the ME-NFA-VFI structure.

Figure 4 shows a small-scale example of an ME-NFA-VSI structure that can count an itemset of size 2 to 4. Figure 4a shows the ANML macro of this ME-NFA-VSI structure, leaving some parameters to be assigned for a specific itemset. %e01 - %e03 are symbols for three entries. An entry can be configured as either "255" or "254", to present "enabled" and "disable" status. Only one entry is enabled for a given itemset. %I represents the global set of items, I. %i01 - %i04 are individual symbols of items in the itemset. %SP is the transaction separator and %END is the ending symbol of the input stream.

To count a 2-itemset, the first two entries are blocked by "254" and the third entry is enabled by "255" (Figure 4b). Similarly, this structure can be configured to counting a 3-itemset and a 4-itemset by enabling a different entry point (Figure 4c and 4d).

Another optimization has been made to reduce STE usage

of ME-NFA-VSI structure by switching entry STEs from all-input mode to start-of-data mode with a bi-directional connection to "%I" STE (Figure 4a). The max number of the optimized ME-NFA-VSI structures that can fit on the AP chip is mainly limited by the number of counter elements. Therefore it is possible to compile large ME-NFA-VSI structures on the AP chip without sacrificing capacity. In the 8-bit symbol encoding scheme, one block of the AP chip can support two ME-NFA-VSI structures that match itemsets of size 2 to 40. For the 16-bit-symbol encoding scheme, we use an ME-NFA-VSI structure that matches itemset of size 2 to 24. 24 is a reasonable upper bound of itemset size we discovered in our test cases.

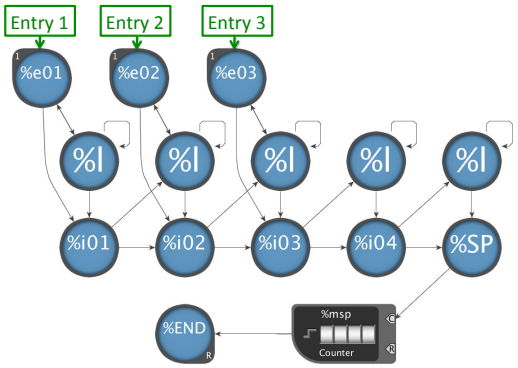
3) Concurrent mining k-itemset and $(k+1)$-itemset: At the very beginning (k is small) and the end (k is large) of mining, the number of candidates could be too small to make full use of the AP board. In these cases, we predict the number candidates of the $(k+1)$-itemset by assuming all k-itemset candidates are frequent. If the total number of k-itemset candidates and predicted $(k+1)$-itemset candidates can fit onto the AP board, we generate $(k+1)$-itemset candidates and concurrently mine frequent k-itemsets and $(k+1)$-itemsets in one round. This optimization takes advantage of unified ME-NFA-VSI structure and saves about 5%-10% AP processing time in general.

VI. Experimental Results

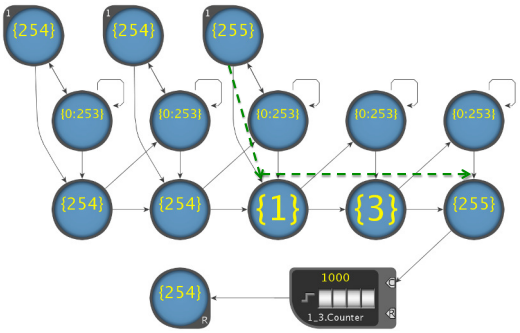
The performance of our AP implementation is evaluated using CPU timers (host codes) and an AP simulator in the AP SDK [11] (AP codes), assuming a 48-core D480 AP board.

A. Capacity and Overhead

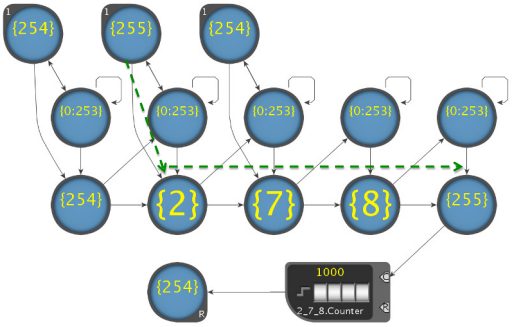
In our experiments, our AP-accelerated *Apriori* algorithm (Apriori-AP) switches between 8-bit and 16-bit encoding schemes automatically in the "data preprocessing" stage shown in the flowchart (Figure 1). In an 8-bit scheme, the items are coded with symbols from "0" to "253". If more than 254 frequent items are represented after filtering, two 8-bit symbols are used to represent one item (16-bit symbol scheme). In both encoding schemes, the symbol "255" is reserved for the transaction separator, and the symbol "254" is reserved for both the input ending symbol and the entry-blockers for the ME-NFA-VSI structure. By using the ME-NFA-VSI structure, one AP board can match and count 18,432 itemsets in parallel with sizes from 2 to 40 for 8-bit encoding and 2 to 24 for 16-bit encoding. In all our experiments, 24 is a reasonable upper bound of the sizes of the itemsets. If there are more than 18,432 candidate itemsets, multiple passes are required. Before each single pass, a symbol replacement process is applied to reconfigure all ME-NFA-VSI structures on the board, which takes 0.045 second.



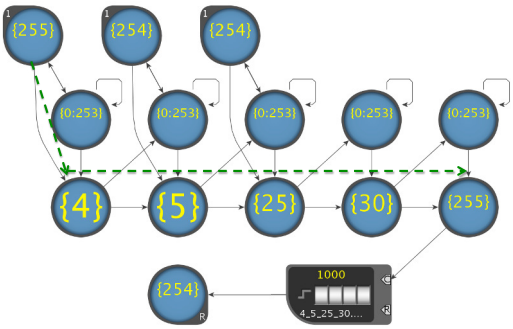
(a) AP macro of ME-NFA-VSI



(b) Automaton for itemset {1, 3}



(c) Automaton for itemset {2, 7, 8}



(d) Automaton for itemset {4, 5, 25, 30}

Figure 4: A small example of multiple-entry NFA for variable-size itemset support counter for 2-itemset, 3-itemset, and 4-itemset. (a) is the macro of this ME-NFA-VSI with parameters.

B. Comparison with other implementations

We use the computation times from Borgelt's *Apriori* CPU sequential implementation [18] (Apriori-CPU) as a baseline. Because the AP accelerates the counting operation at each *Apriori* iteration, we show the performance results of both the counting operation and the overall computation in this section. We also compare a state-of-the-art CPU serial implementation of *Eclat* (Eclat-1C), a multi-threading implementation of *Eclat* (Eclat-6C) [6] and a GPU-accelerated implementation of *Eclat* (Eclat-1G) [6]. All of the above implementations are tested using the following hardware:

- CPU: Intel(R) Xeon(R) CPU E5-1650(6 physical cores, 3.20GHz)
- Mem: 32GB, 1.333GHz
- GPU: Nvidia Kepler K20C, 706 MHz clock, 2496 CUDA cores, 4.8GB global memory

For each benchmark, we compare the performance of the above implementations over a range of minimum support values. A lower support number requires a larger search space and more memory usage, since fewer itemsets are filtered during mining. To have all our experiments finished in a reasonable time, we select minimum support numbers that produce computation times of the Apriori-CPU implementation that is in the range from 1 second to 5 hours for any dataset smaller than 1GB and from 1 second to 10 hours for larger datasets. The relative minimum support number, defined as the ratio of minimum support number to the total number of transactions, is used in the figures of this section.

C. Datasets

Three commonly-used real-world datasets from the *Frequent Itemset Mining Dataset Repository* [19], three synthetic datasets and one real-world dataset generated by ourselves (ENWiki) are tested. The details of these datasets are shown in Table I and II. T40D500K and T10020M are obtained from the IBM Market-Basket Synthetic Data Generator. Webdocs5X is generated by duplicating transactions of Webdocs 5 times.

The ENWiki is the English Wikipedia downloaded in December 2014. We have removed all paragraphs containing non-roman characters and all MediaWiki markups. The resulting dataset contains about 1,461,281 articles, 11,507,383 sentences (defined as transactions) with 6,322,092 unique words. We construct a dictionary by ranking the words using their frequencies. Capital letters are all converted into lower case and numbers are replace with the special "NUM" word. In natural language processing field the idea that some aspects of word semantic meaning can be induced from patterns of word co-occurrence is becoming increasingly popular. The association rule mining provides a suite of efficient tools for computing such co-occurred word clusters.

D. Apriori-*AP* *vs.* Apriori-*CPU*

Figure 5 shows the performance comparison between our Apriori-AP solution and the classic Apriori-CPU implementation on three real-world datasets. The computation time of Apriori-CPU grows exponentially as minimum support number decreases for three datasets, while Apriori-AP shows much less computation time and much slower growth of computation time as minimum support number decreases. As a result, the speedup of Apriori-AP over Apriori-CPU grows as support decreases and achieves up to 129X speedup. The drop in the speedup at the relative minimum support of 0.1 for Webdocs is caused by switching from 8-bit encoding to 16-bit encoding, which doubles the size of the input stream. The speedup increases again after this point. For small and dense datasets like Pumsb, data processing time is relatively low, while the symbol replacement takes up to 80% of the total computation time. Though the symbol replacement is a light-weight reconfiguration, frequent symbol replacement decreases the AP hardware utilization. Also the increasing CPU time of Apriori-AP on small and dense datasets leads to a smaller relative utilization of the AP when the minimum support decreases. In contrast, larger datasets like Accidents and Webdocs spend relatively more time on data processing, and the portion of data processing time goes up as the support decreases. This analysis indicates our Apriori-AP solution exhibits superior relative performance for large datasets and small minimum support values.

Figure 6 shows similar trends of Apriori-AP speedup over Apriori-CPU on three synthetic benchmarks. Up to 94X speedups are achieved for the T100D20M dataset. In all above the cases, the difference between the counting speedup and overall speedup is due to the computation on the host CPU. This difference will decrease as the total computation time increases for large datasets.

The symbol replacement latency can be quite important for small and dense datasets that require multiple passes in each *Apriori* iteration, but this latency may be significantly reduced in future generations of the AP. Figure 7 shows

Table I: **Real-World Datasets**

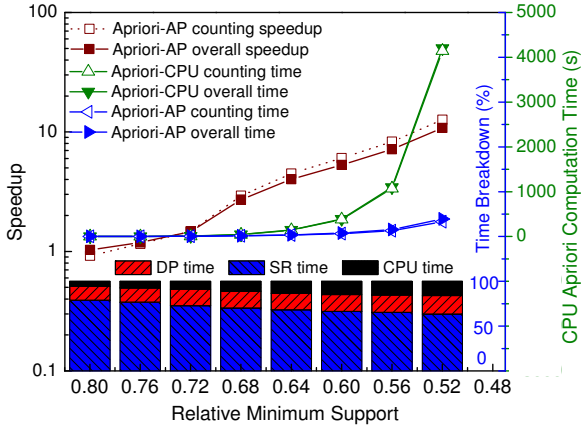
Name	Trans#	Aver. Len.	Item#	Size (MB)
Pumsb	49046	74	2113	16
Accidents	340183	33.8	468	34
Webdocs	1692082	177.2	5267656	1434
ENWiki	11507383	70.3	6322092	2997.5

Aver. Len. – Average number of items per transaction.

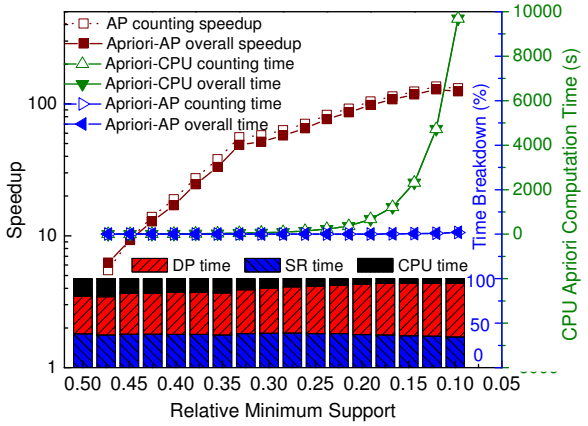
Table II: **Synthetic Datasets**

Name	Trans#	Aver. Len.	Item#	ALMP	Size (MB)
T40D500K	500K	40	100	15	49
T100D20M	20M	100	200	25	6348.8
Webdocs5X	8460410	177.2	5267656	N/A	7168

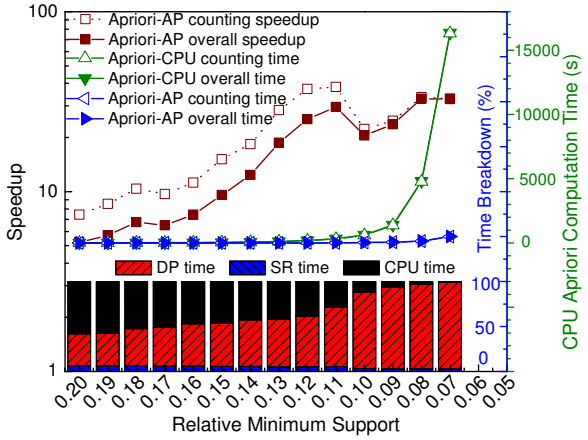
Aver. Len. – Average number of items per transaction.
ALMP – Average length of maximal pattern



(a) Pumsb



(b) Accidents



(c) Webdocs

Figure 5: The performance results of Apriori-AP on three real-world benchmarks. DP time, SR time and CPU time represent the data process time on AP, symbol replacement time on AP and CPU time respectively. Webdocs switches to 16-bit encoding when relative minimum support is less then 0.1. 8-bit encoding is applied in other cases.

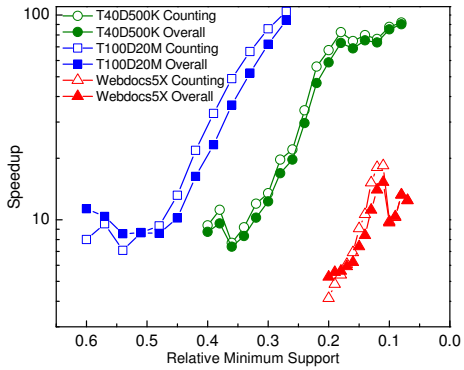


Figure 6: The speedup of Apriori-AP over Apriori-CPU on three synthetic benchmarks

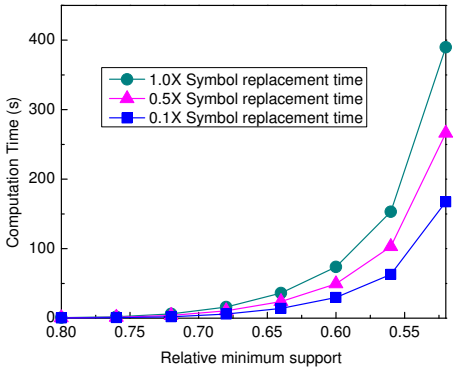


Figure 7: The impact of symbol replacement time on Apriori-AP performance for Pumsb

how symbol replacement time affects the total Apriori-AP computation time. A reduction of 90% in the symbol replacement time leads to 2.3X-3.4X speedups of the total computation time. The reduction of symbol replacement latency will not affect the performance behavior of Apriori-AP for large datasets, since data processing dominates the total computation time.

E. Apriori *vs.* Eclat

Equivalent Class Clustering (*Eclat*) is another algorithm based on Downward-closure. *Eclat* uses a vertical representation of transactions and depth-first-search strategy to minimize memory usage. Zhang *et al.* [6] proposed a hybrid depth-first/breadth-first search scheme to expose more parallelism for both multi-thread and GPU versions of *Eclat*. However, the trade-off between parallelism and memory usage still exists. For large datasets, the finite memory (main or GPU global memory) will become a limiting factor for performance, and for very large datasets, the algorithm fails. While there is a parameter which can tune the trade-off between parallelism and memory occupancy, we simply use the default setting of this parameter [6] for better performance.

Figure 8 shows the speedups that the *Eclat* sequential

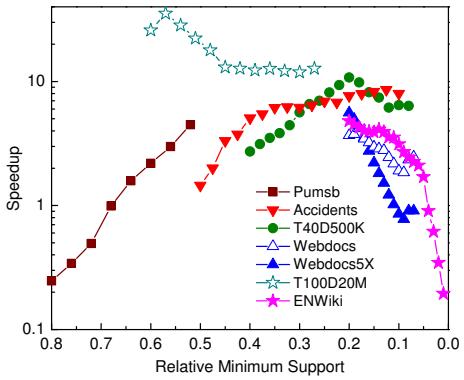


Figure 8: The performance comparison of CPU sequential *Apriori* and *Eclat*

algorithm achieved with respect to sequential Apriori-CPU. Though *Eclat* has 8X performance advantage in average cases, the vertical bitset representation become less efficient for sparse and large dataset (high #trans and #freq_item ratio). This situation becomes worse as the support number decreases. The Apriori-CPU implementation usually achieves worse performance than *Eclat*, though the performance boost of counting operation makes Apriori-AP a competitive solution to parallelized *Eclat*.

Three factors make *Eclat* a poor fit for the AP, though it has better performance on CPU:

1) *Eclat* requires bit-level operations, but the AP works on byte-level symbols
2) *Eclat* generates new vertical representations of transactions for each new itemset candidate, while dynamically changing the values in the input stream is not efficient using the AP
3) Even the hybrid search strategy cannot expose enough parallelism to make full use of the AP chips

Figure 9 and 10 show the performance comparison between Apriori-AP (45nm for current generation of AP), and sequential, multi-core, and GPU versions of *Eclat*. Generally, Apriori-AP shows better performance than sequential and multi-core versions of *Eclat*. The GPU version of *Eclat* shows better performance in Pumsb, Accidents and Webdocs when the minimum support number is small. However, because of the constraint of GPU global memory, Eclat-1G fails at small support numbers for three large datasets - ENWiki, T100D20M and Webdocs5X. ENWiki, as a typical sparse dataset, causes inefficient storage of bitset representation in Eclat and leads to early failure of Eclat-GPU and up to 49X speedup of Apriori-AP over Eclat-6C. In other benchmarks, Apriori-AP shows up to 7.5X speedup over Eclat-6C and 3.6X speedup over Eclat-1G. This figure also indicates that the performance advantage of Apriori-AP over *Eclat* GPU/multi-core increases as the size of the dataset grows.

F. Normalizing for technology

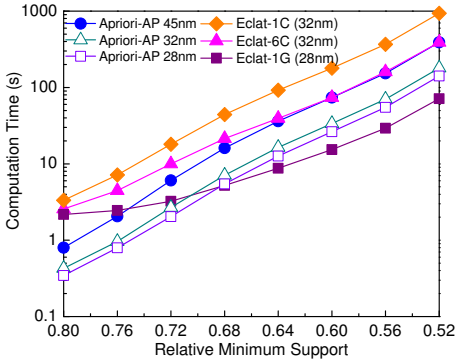
The AP D480 chip is based on 45nm technology, while the Intel CPU Xeon E5-1650 and Nvidia Kepler K20C on which we test *Eclat* are based on 32nm and 28nm technologies respectively. To compare the different architectures in the same semiconductor technology mode, we show the performance of technology projections on 32nm and 28nm technologies in Figure 9 and 10 assuming linear scaling for clock frequency and square scaling for capacity [20]. The technology normalized performance of Apriori-AP shows better performance than multi-core and GPU versions of Eclat in almost all of the ranges of support that we investigated for all datasets, with the exception of small support for Pumsb and T100D20M. Apriori-AP achieves up to 112X speedup over Eclat-6C and 6.3X speedup over Eclat-1G.

G. Data size

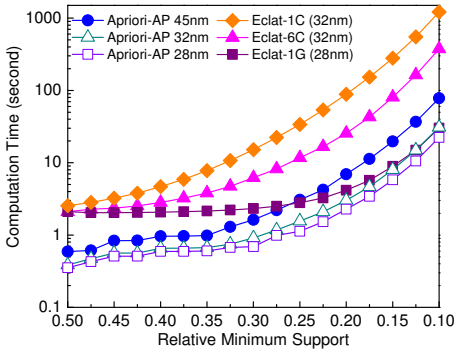
The above results indicate that the size of the dataset could be a limiting factor for the parallel *Eclat* algorithms. By varying the number of transactions but keeping other parameters fixed, we studied the behavior of Apriori-AP and Eclat as the size of the dataset increases (Figure 11). For T100, the datasets with different sizes are obtained by the IBM synthetic data generator. For Webdocs, the different data sizes are obtained by randomly sampling the transactions or by concatenating duplicates of the whole dataset. In the tested cases, the GPU version of *Eclat* fails in the range from 2GB to 4GB because of the finite GPU global memory. Comparing the results using different support numbers on the same dataset, it is apparent that the smaller support number causes Eclat-1G to fail at a smaller dataset. This failure is caused by the fact that the ARM with a smaller support will keep more items and transactions in the data preprocessing stage. While not shown in this figure, it is reasonable to predict that the multi-core *Eclat* implementation would fail when the available physical memory is exhausted. However, Apriori-AP will still work well on much larger datasets, assuming the data is streamed in from the hard drive (assuming the hard drive bandwidth is not a bottleneck).

VII. Conclusions and the Future Work

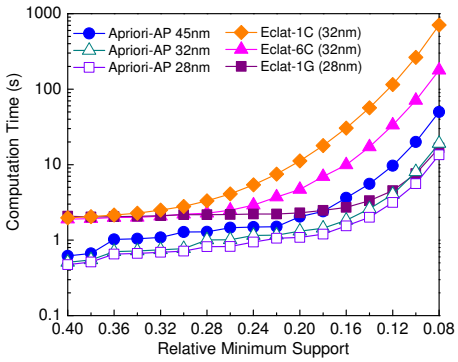
We present a hardware-accelerated ARM solution using Micron's new AP architecture. Our proposed solution includes a novel automaton design for matching and counting frequent itemsets for ARM. The multiple-entry NFA based design was proposed to handle variable-size itemsets (ME-NFA-VSI) and avoid routing reconfiguration. The whole design makes full usage of the massive parallelism of the AP and can match and count up to $18,432$ itemsets in parallel on an AP D480 48-core board. When compared with the *Apriori*-based single-core CPU implementation, the proposed solution shows up to 129X speedup in our experimental



(a) Pumsb (16MB)



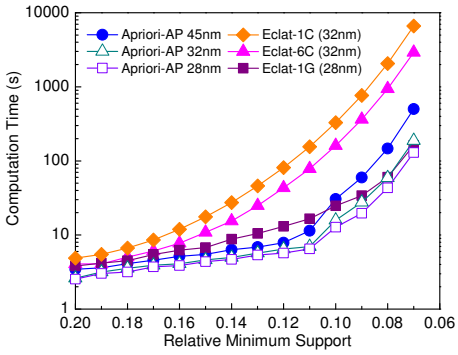
(b) Accidents (34MB)



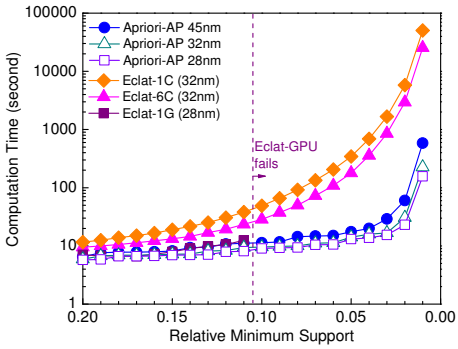
(c) T40D500K(49MB)

Figure 9: Performance comparison among Apriori-AP, Eclat-1C, Eclat-6C and Eclat-1G with technology normalization on three small datasets

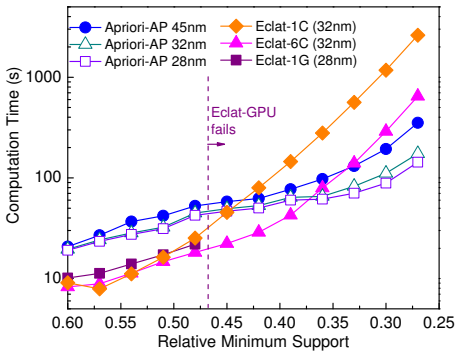
results on seven real-world and synthetic datasets. This AP-accelerated solution also outperforms the multicore-based and GPU-based implementations of *Eclat* ARM, a more efficient algorithm, with up to 49X speedups, especially on large datasets. When performing technology projections on future generations of the AP, our results suggest even better speedups relative to the equivalent-generation of CPUs and GPUs. Furthermore, by varying the size of the datasets from small to very large, our results demonstrate the memory constraint of parallel *Eclat* ARM, particularly for GPU



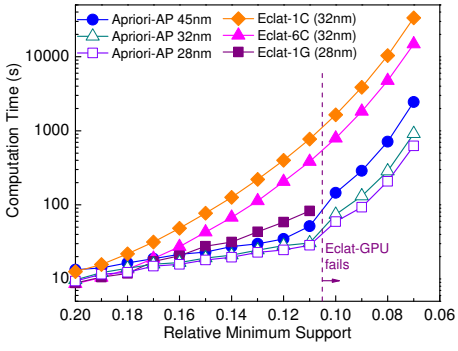
(a) Webdocs(1.4GB)



(b) ENWiki(3.0GB)

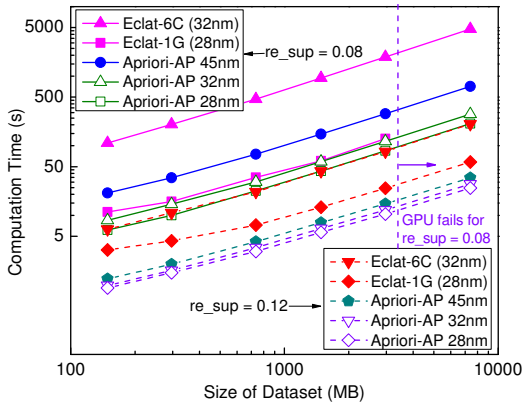


(c) T100D20M (6.3GB)

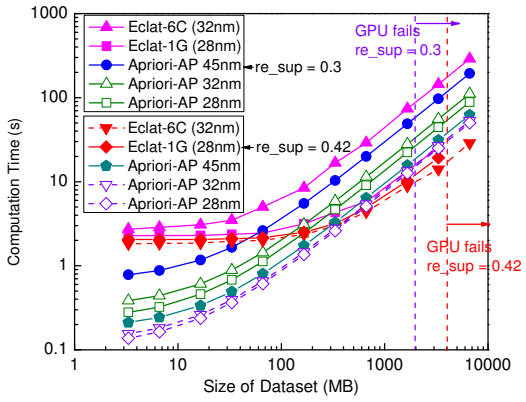


(d) Webdocs5X (7.1GB)

Figure 10: Performance comparison among Apriori-AP, Eclat-1C, Eclat-6C and Eclat-1G with technology normalization on four large datasets



(a) Webdocs



(b) T100

Figure 11: Performance prediction with technology normalization as a function of input size

implementation. In contrast, the capability of our AP ARM solution scales nicely with the data size, since the AP was designed for processing streaming data.

With the challenge of the "big data" era, a number of other complex pattern mining tasks such as frequent sequential pattern mining and frequent episode mining, have attracted great interests in both academia and industry. We plan to extend the proposed CPU-AP infrastructure and automaton designs to address more complex pattern-mining problems.

Acknowledgment

This work was supported in part by the Virginia CIT CRCF program under grant no. MF14S-021-IT; by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA; NSF grant EF-1124931; and a grant from Micron Technology.

References

[1] J. Han *et al.*, "Frequent pattern mining: Current status and future directions," *Data Min. Knowl. Discov.*, 2007.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. of VLDB '94*, 1994.

[3] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. on Knowl. and Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.

[4] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. of SIGMOD '00*, 2000.

[5] Y. Zhang, F. Zhang, and J. Bakos, "Frequent itemset mining on large-scale shared memory machines," in *Proc. of CLUSTER '11*, 2011.

[6] F. Zhang, Y. Zhang, and J. D. Bakos, "Accelerating frequent itemset mining on graphics processing units," *J. Supercomput.*, vol. 66, no. 1, pp. 94–117, 2013.

[7] Y. Zhang *et al.*, "An fpga-based accelerator for frequent itemset mining," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 6, no. 1, pp. 2:1–2:17, May 2013.

[8] P. Dlugosch *et al.*, "An efficient and scalable semiconductor architecture for parallel automata processing," *IEEE TPDS*, vol. 25, no. 12, 2014.

[9] I. Roy and S. Aluru, "Finding motifs in biological sequences using the micron automata processor," in *Proc. of IPDPS'14*, 2014.

[10] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. of SIGMOD '93*, 1993.

[11] H. Noyes, "Micron's automata processor architecture: Reconfigurable and massively parallel automata processing," in *Proc. of Fifth International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, June 2014, keynote presentation.

[12] M. J. Zaki *et al.*, "Parallel data mining for association rules on shared-memory multi-processors," in *Proc. of Supercomputing '96*, 1996.

[13] L. Liu *et al.*, "Optimization of frequent itemset mining on multiple-core processor," in *Proc. of VLDB '07*, 2007.

[14] I. Pramudiono and M. Kitsuregawa, "Parallel fp-growth on pc cluster," in *Proc. of PAKDD '03*, 2003.

[15] E. Ansari *et al.*, "Distributed frequent itemset mining using trie data structure," *IAENG Intl. J. Comp. Sci.*, vol. 35, no. 3, p. 377, 2008.

[16] W. Fang *et al.*, "Frequent itemset mining on graphics processors," in *Proc. of DaMoN '09*, 2009.

[17] B. Goethals, "Survey on frequent pattern mining," Univ. of Helsinki, Tech. Rep., 2003.

[18] C. Borgelt, "Efficient implementations of apriori and eclat," in *Proc. FIMI '03*, 2003, p. 90.

[19] "Frequent itemset mining dataset repository," http://fimi.ua.ac.be/data/.

[20] J. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits*, 2nd ed. Pearson Education, 2003.