Hardware Architectures for Frequent Itemset Mining Based on Equivalence Classes Partitioning

Martin Letras
Computer Science Department
National Institute of Astrophysics, Optics and Electronics
Puebla, Mexico
Email: mletras@ccc.inaoep.mx

Raudel Hernández-León
Data Mining Research Team
Advanced Technologies Application Center
La Habana, Cuba
Email: rhernandez@cenatav.co.cu

Rene Cumplido
Computer Science Department
National Institute of Astrophysics, Optics and Electronics
Puebla, Mexico
Email: rcumplido@inaoep.mx

***Abstract*—Frequent itemset mining algorithms have proved their effectiveness to extract all the frequent itemsets in datasets, however in some cases they do not produce the expected results in an acceptable time according to the application requirements. For this reason, FPGA-based hardware architectures for frequent itemset mining have been proposed in the literature to accelerate this task. Most of the reported architectures are limited by the number of distinct items that could be processed and the available resources in the employed FPGA device. This study proposes a compact hardware architecture for frequent itemset mining capable of minimg all the frequent itemsets regardless of the number of distinct items and transactions in the dataset. The proposed architectural design implements a partition strategy based on equivalence classes. The partition on equivalence classes allows to divide the search space into disjoint sets that can be processed in parallel. Accordingly, a parallel architecture is proposed to exploit the benefits of the proposed search strategy.**

***Index Terms*—Frequen Itemset Mining, Hardware Architecture, FPGA.**

I. Introduction

Nowadays, information technology is present in every activity that we perform: in smartphones, personal computers, and even household appliances connected to the Internet that interchange and generate big amounts of data [11]. This amount of data and the diversity of the information exceeds the human capacity to process it and obtain rules that describe the relationship among the data [12].

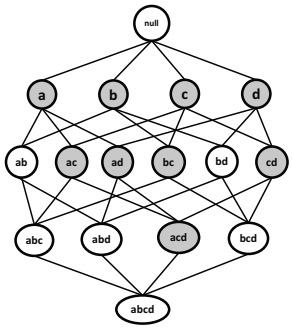
Data mining has emerged to solve this problem using automatic or semi-automatic processes to analyze datasets to find patterns and then perform classification or prediction tasks [18]. One of the most spread technique in Data Mining is the Association Rule Mining technique, which computes rules in form of implications among a set of items [2]. A crucial step in the association rules generation is to count the frequency of items and itemsets to know their relevance; this process is known as frequent itemset mining [2].

Nevertheless, looking for frequent itemsets may become an expensive task due to large amount of data, sparse datasets, and a low minimum support value (henceforth s_{min}). For these reasons, sometimes the implementations of these algorithms cannot return a solution in an acceptable time. One way to deal with this problem is to improve existing algorithms in order to reducing execution time and proposing new heuristics to explore the search space or using different data representations. In recent years, there is a trend to develop specialized hardware architectures to reduce the execution time of algorithms. In literature, there are several hardware architectures based on FPGA (Field Programmable Gate Arrays) and GPUs (Graphic Processor Units) that perform a full implementation of frequent itemset mining algorithms.

This paper describes a FPGA-based hardware architecture for frequent itemsets mining that takes advantage of inner parallelism in the algorithms. The main goal is to produce a compact hardware architecture in area resources that is able to find all the frequent itemsets regardless of the number of items and transactions in the datasets. The architecture also must be able to achieve a better speed up compared to optimized software implementations of frequent itemset mining algorithms. Most of the reported work in literature has been designed for a fixed problem size, in others words, they have a limit on the number of different items that are processed, restricted by the resources of the device employed or by memory constraints. Our proposed architectures address the previous limitations by partitioning the problem, then generating partial solutions for each partition, and finally, combining all the partial solutions to construct a global solution. The partition scheme is mainly based on equivalence classes. This approach divides the entire search space into disjoint sets of the original search space, in consequence, all the equivalence classes may be processed in an independent way. Accordingly, a parallel architecture also is proposed to exploit the benefits of the independent partitions into equivalence classes.

This paper is organized as follows; in section 2, frequent itemset mining and the most important theoretical concepts are exposed. In section 3 related work and previously proposed hardware architectures for frequent itemset mining are disclosed. Section 4 introduces the proposed search strategy based on equivalence classes, continuing in section 5 with the implementation of the proposed architectures. section 6 describes the experimental setup and the execution time comparison. Finally, in section 7 conclusions and possible future research lines are drawn.

978-1-5090-3682-0/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPSW.2016.98



ID	Items
1	a, d
2	b, c, d
3	a, c
4	a, c, d
5	a, b
6	a, c, d
7	b, c
8	a, c, d
9	b, c
10	a, d

Fig. 1. Transaction dataset and search space

II. Frequent Itemset Mining

Frequent itemset mining is a method for market basket analysis, and was introduced in [1] by Agrawal. Finding frequent itemsets and associations rules is essential for marketing applications, improving the arrangement of products on shelves and suggesting other products.

The first algorithm for frequent itemset mining was formalized by Agrawal in the 90's [1, 2], and it is used to find patterns in datasets. These datasets are represented by transactions; each transaction is labeled with a unique identifier. Frequent itemset mining can be defined as follows:

Formally, let $I = \{i_1, \ldots, i_n\}$ be a set of items. Let D be a set of transactions, where each transaction T is a set of items such as $T \subseteq I$. And let X be an itemset such as $X \subseteq I$; without loss of generality, we will assume that all items in each transaction are sorted in lexicographic order. The support value of the itemset X is the number of transactions over D containing X. An itemset is called frequent if its support is greater than or equal to a given threshold value(S_{min}).

A brute force approach that traverses all the possible itemsets calculating their support and removing infrequent itemsets is inefficient, in the worst case, a total of 2^n itemsets could be generated for n distinct items. For example in figure 1, we have four items; the search space contains 16 itemsets but only 9 are frequent (the gray ones) for $S_{min} = 3$. The number of itemsets and operations grows exponentially according to the number of different items, transactions, and the S_{min} value. Several algorithms have been proposed to efficiently find all frequent itemsets. There are those based on candidate generation like Apriori [1, 2] that explores the search space using breath-first search, and other ones based on pattern growth like FP-Growth [8] that creates a tree structure called FP-tree, and Eclat [19] that uses a depth-first search. These algorithms find all the frequent itemsets but in some cases, they do not return a response in an acceptable time according to the application requirements.

III. Related Work

In recent years hardware architectures have been explored to offer a solution to the acceleration of frequent itemset mining algorithms. The leading technologies are GPU and FPGA and the most employed algorithms are Apriori [2], FP-Growth [8], and recently Eclat [19].

In [3, 4, 16, 17] the proposed architectures are an Apriori implementation that use systolic arrays. The reason to implement a systolic array is to decrease the number of the connections among processors elements and to ease the control. The disadvantage of these approaches is that they are limited by the resources of the FPGA device employed and the number of processor elements implemented on the chip.

In [14, 15] a systolic tree structure is proposed to implement a hardware version of FP-Growth algorithm. A systolic tree is an array of pipelined processing elements in a multi-dimensional tree pattern. In [10], another hardware architecture to execute FP-Growth algorithm is proposed. Authors proposed an equivalence class segmentation based on Eclat algorithm, but once the segmentation is done, FP-growth algorithm takes the control to generate frequent itemsets. The results reported show that the proposed architecture achieves better performance than the hardware implementation reported in [14], specifically for the Chess dataset, the architecture achieves a speedup of one order of magnitude.

Recently hardware architectures based on the Eclat algorithm have been developed [13, 20]. In [20], the architecture performs a depth first search strategy. Authors proposed a binary representation for datasets and itemsets. The architecture is formed by an external memory, FIFOs, an intersection module, and a support counting module. They obtain good results for sparse datasets, and a maximum acceleration of 48x is achieved.

All the previous works in literature have proved their efficiency to accelerate frequent itemset mining algorithms. But all of them are limited by the hardware resources available according to the FPGA device employed. For this reason, our goal is to implement a hardware architecture that can obtain all the frequent itemset regardless of the number of different items, transactions, and hardware resources.

IV. Data representation and search strategy proposed

Our proposal consists in a variant of the search strategy proposed for Eclat algorithm, and its primary objective is to perform independient partitions of the search space. The data representation employed is the vertical binary vector because the intersection and support counting operation can be implemented as a combinatorial system. The transactions are coded in 32 bits integers using the compressed array representation reported in [9]. The word size is 32 bits because the external memory is 32 bits wide. For example in figure 2, a binary vector dataset of five items is shown. For items a and b, their support values are calculated counting the set bits in the correspondent vectors being $S_a = 8$ and $S_b = 5$. The intersection operation is performed using boolean AND operations. For example to get the itemset ab, an AND operation between the binary vector of item a and b is performed. The result is the binary vector ab shown in figure 2, and $S_{ab} = 3$.

Our search strategy is a combination of breadth and depth

	a	b	c	d	e
1	1	1	0	1	1
2	0	1	1	1	0
3	1	0	1	0	1
4	1	0	1	1	1
5	1	0	0	0	1
6	1	0	1	1	0
7	1	1	1	0	0
8	1	0	1	1	1
9	0	1	1	0	1
10	1	1	0	1	1

(a) Dataset using vertical binary vectors.

	a
1	1
2	0
3	1
4	1
5	1
6	1
7	1
8	1
9	0
10	1

$S_a = 8$

	b
1	1
2	1
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	1

$S_b = 5$

	ab
1	1
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0

$S_{ab} = 3$

(b) Intersection and support counting operations in binary vectors.

Fig. 2. Data Representation and operations used by our proposal.

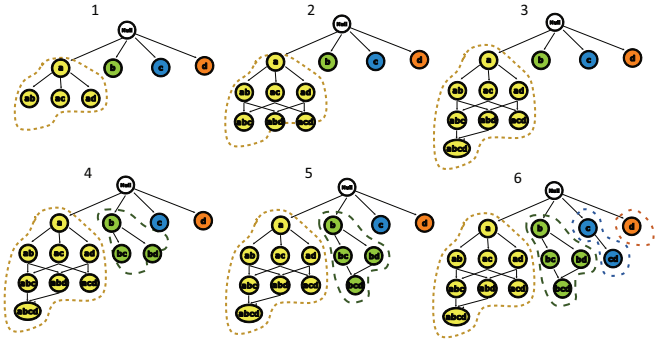


Fig. 3. Search strategy proposed for four items.

first search. This strategy has the advantage that the search space can be partitioned, and each partition of the search space can be processed in parallel. For example, figure 3 describes the behaviour of the proposed strategy. The first step consists in taking item a and generate all the 2-itemsets being ab, ac and ad frequent itemsets. The next step is to generate all the 3-itemsets. abc is generated intersecting ab and ac. abd is generated intersecting ab and ad. acd is generated intersecting ac and ad and so on, until no more itemsets could be generated.

In this way, this search strategy can generate the equivalence classes in an independent way.

V. Hardware Architectures Proposed

In order to exploit the benefits of the proposed search strategy; two hardware architectures have been proposed. The first one consists in a compact architecture that mines each equivalence class in a sequential manner. The second one is a parallel implementation that distributes the workload between two processor elements.

A. Compact Hardware Architecture

Our first proposal is the implementation of a full hardware implementation of the proposed search strategy, The behaviour of this architecture is divided into two parts; the first one consists in the generation of the frequent items and the second one consists in the frequent itemset mining using the proposed search strategy.

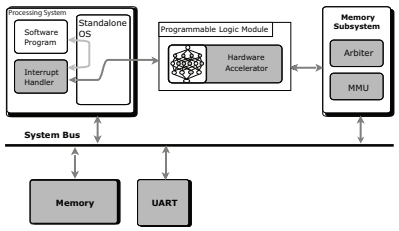


Fig. 4. Hardware architecture that performs the proposed search strategy.

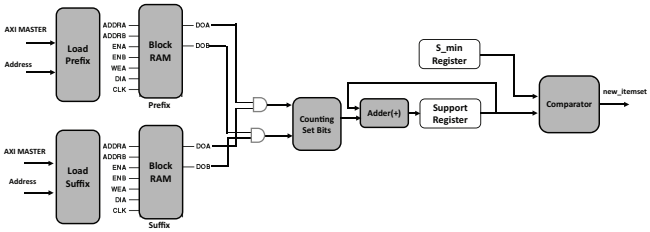
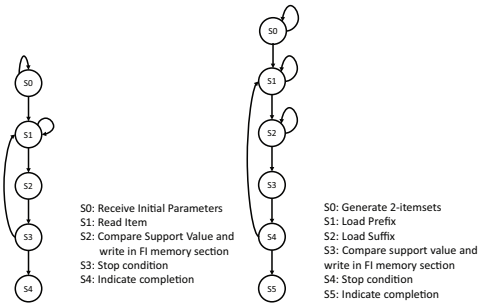


Fig. 5. Low level design of the proposed architectural design.

Figure 4 shows a high-level diagram of the proposed architecture. This architecture is composed of a general purpose processor, an UART module, an off-chip memory, a memory subsystem and the hardware accelerator.

Figure 5 describes a block diagram of the hardware accelerator. It consists of two dual block RAM memories called $prefix$ and $suffix$. The BRAMs have a storage capacity of 122 KiB, in consequence they can store one million of transactions but the architecture is not only limited to process one million of transactions because the Load Suffix and Load Prefix modules can iterate to cover more than one million of transactions. The outputs of each memory are connected to AND gates that perform the intersection using 32-bits words. The counting support module receives as inputs two 32-bit words that are the result of the AND gates. The output of the counting support module is accumulated in the support register until all the transactions have been covered. And finally, a comparator verifies the support register value with the S_{min} register value. If the current itemset is a frequent itemset, the prefix label is concatenated with the suffix label and then the concatenated label is stored in the off-chip memory with its corresponding binary vector. Figure 6 shows two finite states machines that describe the behavior of the proposed architecture. The first state machine corresponds to the frequent items generation task. In state $S0$, the architecture receives the initial direction where the binary vectors are stored, the number of transactions, the number of items, the label of the actual item, the direction where the frequent item labels will be stored and the S_{min} value. In state $S1$, the architecture reads the binary vector of the current item and stores it in the load prefix BRAM, and then the counting set bits module computes the support value. In state $S2$, if the item is frequent, its label is stored in the off-chip memory as a frequent itemset. State $S3$ verifies that all the items have



(a) Items mining state machine. (b) Itemset mining state machine.

Fig. 6. Finite state machines of the proposed search strategy.

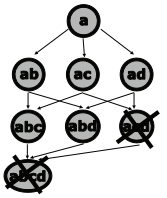


Fig. 7. Search space for item a.

been processed.

The second state machine describes the behaviour of the frequent itemset mining stage. In state $S0$, the 2-itemsets are mined. Table I describes the operations involved in state $S0$. The first step consists in receiving a set of initial items, for this example the initial items are $H = \{a, b, c, d\}$ being the $a - prefixed$ itemsets the equivalence class to process. The first item in the initial items list determines the equivalence class to process. The second step consists in performing the intersection and support counting of the 2-itemsets; Prefix and Suffix BRAMs are used in this task. The item a is stored in $prefix$ BRAM, and the next items will be stored in $suffix$ BRAM to perform the intersection and support counting operation. All the frequent 2-itemsets will be stored in the off-chip memory.

Once that the $2 - itemsets$ have been calculated, the next step corresponds to state $S1$ and it consists in the $k-itemsets$ mining. Table II describes the operations employed in this stage using the search space of figure 7. The $2 - itemsets = \{ab, ac, ad\}$, so in state $S1$ the binary vector of ab is stored in Prefix BRAM, and in state $S2$, then itemset ac is stored in Suffix BRAM. In state $S3$, the intersection operation and the support counting operation indicate that abc is a frequent itemset and in consequence, abc is written in the off-chip memory. The itemset ab is not flushed from the Prefix BRAM because there is an itemset that shares the same prefix. So, ab and ad are intersected to generate a new itemset. In this case, ab is flushed from memory because the next itemset in memory is abc and it is a 3-itemset and they do not share the same prefix. The intersection of two itemsets can only be performed, if both of them have the same cardinality and share the same prefix. For example, the prefix of itemset abc is ab and the prefix of itemset acd is ac, although they have the same cardinality they do not share the same prefix, and they cannot be intercepted to generate a new itemset. In contrast for itemsets abc and abd, they share the prefix ab and the same cardinality, in consequence they can generate the itemset $abcd$.

The previous steps are executed until no more itemsets can be generated.

TABLE I
2-ITEMSETS GENERATION.

Frequent itemsets in memory	Prefix BRAM	Suffix BRAM	Frequent
{ }	a	b	Yes
{ab}	a	c	Yes
{ab, ac}	a	d	Yes
{ab, ac, ad}	a	-	Yes

TABLE II
OPERATIONS PERFORMED BY THE ARCHITECTURE

Frequent itemsets in memory	Prefix BRAM	Suffix BRAM	Result in Suffix BRAM	Frequent	Output	Flush prefix BRAM
{ab, ac, ad}	ab	ac	abc	Yes	Yes	No
{ab, ac, ad, abc}	ab	ad	abd	Yes	Yes	Yes
{ab, ac, ad, abc, abd}	ac	ad	acd	No	Yes	Yes
{ab, ac, ad, abc, abd}	ad	-	-	No	No	Yes
{ab, ac, ad, abc, abd}	abc	abd	abcd	No	No	Yes

B. Dual Core Hardware Architecture

With the intention to get a speed up, a dual-core architecture is proposed. Figure 8 shows a high-level representation. In the logical programmable area, two hardware accelerators are implemented with the intention of distributing the workload between the two of them.

Previously it has mentioned that the proposed search strategy has the advantage of splitting the search space into disjoint sets or classes, in consequence, each core can process an equivalence class independently. Figure 8 describes the partition of the search space for four items. The first processor element receives the set of items $H = \{a, b, c, d\}$ and it processes the equivalence class a. Meanwhile, the second processor element receives the sets of items $H = \{b, c, d\}$ and it process the equivalence classes b, c and, d. The dual core architecture obtains a parallelism to process independent equivalence classes, and this impacts directly on the performance of the proposed search strategy.

VI. EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATION

A. Experimental setup

The hardware architectures have been evaluated using area and execution time metrics. The area evaluation is performed using the hardware report usage that provides Vivado HLS synthesizer. The execution of the architecture has been compared to the execution time of Apriori, Eclat and FP-Growth software implementations [5, 6]. These implementations can be found on the personal website of Christian Borgelt [7].

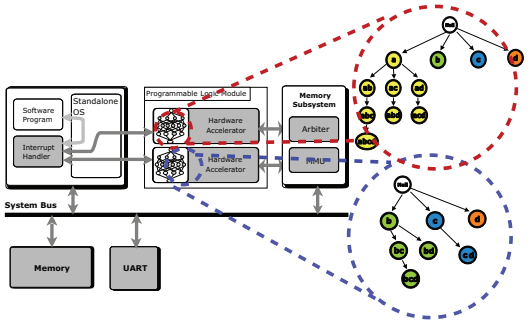


Fig. 8. Partition of the search space using 2 processor elements.

The software implementations have been tested on a PC with an Intel i3-3217U processor at 1.8 GHz and 8 GB DDR2 RAM memory with Windows 7 ultimate. The execution time considers the input and output operations and the CPU time for all the algorithms and the hardware architectures. The FPGA device employed is a Zynq 7020 of Xilinx.

B. Validation datasets

In the literature diverse datasets have been used to test the functionality of the software algorithms and hardware architectures. In [2], an algorithm is proposed to generate synthetic datasets that imitates the characteristics of transactions in the retailing environment.

TABLE III
DATASETS USED TO VALIDATE THE HARDWARE ARCHITECTURE.

Dataset	Size (MB)	Average Length Transaction	Number of Transactions	Number of Items
Chess	0.013	37	3196	75
T40I3N500k	11.9	40	500k	299
T40I3N1000k	24.1	40	1000k	300
T60I5N500k	18.9	60	500k	500

The characteristics employed to generate the datasets are: number of transactions $|D|$, average size of transactions $|T|$, average size of the maximal potentially large itemsets $|I|$, number of potentially large itemsets $|L|$ and, number of items $|N|$. All these characteristics are used to generate synthetic datasets. For this work, three values for $|T|$: 40, 60 and 90 have been chosen. The values for $|I|$ are 3, 5 and 10. Table III summarizes the dataset parameter settings, and also an estimated of the size in MB of the datasets.

C. Performance evaluation

In this section the performance results for the compact architecture and the dual core architecture are presented. From figure 9 to 12, the compact hardware architecture and the dual core hardware architecture are compared to Apriori, FP-Growth, and Eclat; the y axis represents the execution time of each experiment and the x axis represents the minimum support value. Figure 9 shows the performance of the three software implementations and the hardware architectures for the chess dataset. For chess dataset, FP-growth obtained the

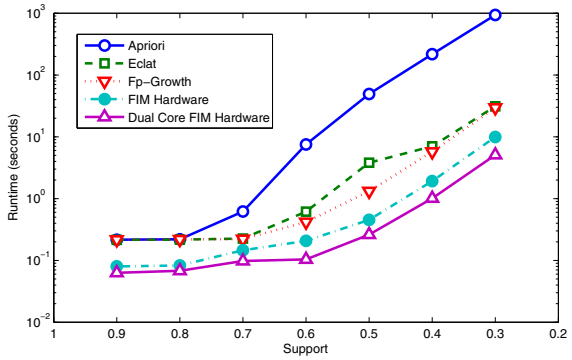


Fig. 9. Execution time comparison (Chess).

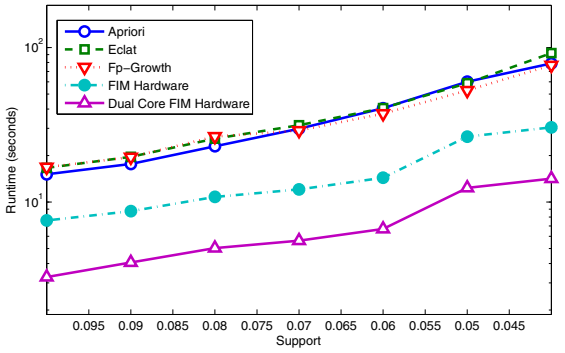


Fig. 10. Execution time comparison (T40I3N500k).

best results among the four software implementations. The maximum speedup obtained by the compact architecture is 2.9x and 5.8x for the dual core architecture compared to the Fp-growth algorithm.

The better performance reported for the hardware architectures is when they have to deal with sparse datasets (Figures 10, 11, 12). The experiments show that the proposed architectures have good performance and it obtains a speedup of 4x for the compact architecture and 12.7x for the dual core architecture compare with the best software implementations (it depends on the dataset and the support value). Table IV shows the area reports for both architectures. The operation frequency reported for both is 114 Mhz. The elements reported are DSP48E, Flip Flops and LUTs. For the compact architecture, the usage of Flip-Flops (3 %) and LUTs (9 %) is minimum because the architecture only needs a few registers to store S_{min} value and the control signals, this is the most compact design obtained.

For the dual core architecture, although there has been an increment in the resources employed, the architecture is still a compact one. Intuitively, an advantage of the compact design is that the number of cores that can be attached to the architecture can grow, and the workload can be divided among other processor elements to speed up the execution time.

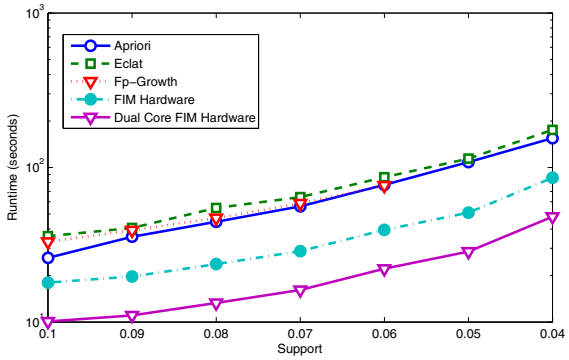


Fig. 11. Execution time comparison (T40I3N1000k).

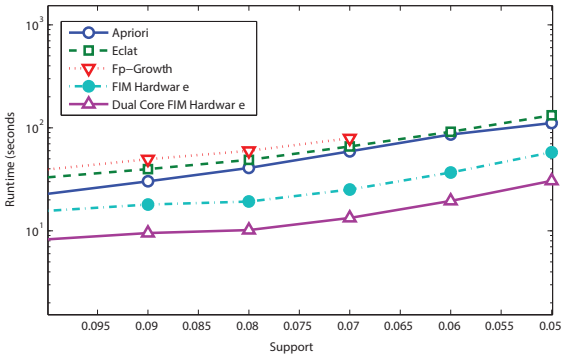


Fig. 12. Execution time comparison (T60I5N500k).

TABLE IV
HARDWARE RESOURCES USED BY PROPOSED HARDWARE ARCHITECTURE.

	Compact Architecture			Dual Core Architecture		
Name	**DSP48E**	**FF**	**LUT**	**DSP48E**	**FF**	**LUT**
Expression	-	0	2618	-	0	3806
Multiplexer	-	-	1943	-	-	2891
Registers	-	3475	-	-	5234	-
Shift Memory	-	0	164	-	0	279
Total	20	3475	4725	32	5234	6976
Utilization (%)	**9**	**3**	**9**	**14**	**4**	**13**

Compactness is the main advantage of the proposed hardware architecture. Although, it is a compact design, both versions accelerate the frequent itemset mining problem and speedup can be achieved.

VII. CONCLUSIONS

In this paper, a search strategy for frequent itemset mining that finds all the frequent itemsets regardless of the number of different items. The proposed search strategy fits well for hardware implementations because it splits the search space into separate equivalence classes making disjoint sets of the original dataset. In consequence, the amount of itemsets stored in the memory is reduced, this is an advantage for memory constrained scenarios like in the hardware architecture development. Another advantage of the partition into separate equivalence classes is that the equivalence classes can be distributed among a set of processor elements to parallelize and distribute the workload. The most remarkable feature of this architecture is that gets a 4x to 12.7x speedup despite its compactness.

Based on the results obtained in this research, it is possible to implement an array of processor elements, in other words, scale up the proposed dual-core architecture from 2 to n processor elements to get a better speedup.

ACKNOWLEDGMENT

Martin Letras is supported by the Mexican National Council for Science and Technology (CONACyT), scholarship number 298024.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] Zachary K Baker and Viktor K Prasanna. Efficient hardware data mining with the apriori algorithm on fpgas. In *Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on*, pages 3–12. IEEE, 2005.

[4] Zachary K Baker and Viktor K Prasanna. An architecture for efficient hardware data mining using reconfigurable computing systems. In *Field-Programmable Custom Computing Machines, 2006. FCCM'06. 14th Annual IEEE Symposium on*, pages 67–75. IEEE, 2006.

[5] Christian Borgelt. Efficient implementations of apriori and eclat. In *FIMI'03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations*, 2003.

[6] Christian Borgelt. An implementation of the fp-growth algorithm. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 1–5. ACM, 2005.

[7] Christian Borgelt. Christian borgelt's web pages. url-http://www.borgelt.net/fimgui.html, 2015.

[8] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[9] Raudel Hernández-León, J Hernández-Palancar, Jesús A Carrasco-Ochoa, and José Fco Martínez-Trinidad. Algorithms for mining frequent itemsets in static and dynamic datasets. *Intelligent Data Analysis*, 14(3):419–435, 2010.

[10] Alejandro Mesa, Claudia Feregrino-Uribe, René Cumplido, and José Hernández-Palancar. A highly parallel algorithm for frequent itemset mining. In *Advances in Pattern Recognition*, pages 291–300. Springer, 2010.

[11] Philip E Ross. Top 11 technologies of the decade. *IEEE Spectrum*, 48(1):27–63, 2011.

[12] Philip Russom et al. Big data analytics. *TDWI Best Practices Report, Fourth Quarter*, 2011.

[13] Shaobo Shi, Yue Qi, and Qin Wang. Accelerating intersection computation in frequent itemset mining with fpga. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 659–665. IEEE, 2013.

[14] Song Sun, Michael Steffen, and Joseph Zambreno. A reconfigurable platform for frequent pattern mining. In *Reconfigurable Computing and FPGAs, 2008. ReConFig'08. International Conference on*, pages 55–60. IEEE, 2008.

[15] Song Sun and Joseph Zambreno. Design and analysis of a reconfigurable platform for frequent pattern mining. *Parallel and Distributed Systems, IEEE Transactions on*, 22(9):1497–1505, 2011.

[16] DW Thoni and Alfred Strey. Novel strategies for hardware acceleration of frequent itemset mining with the apriori algorithm. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 489–492. IEEE, 2009.

[17] Ying-Hsiang Wen, Jen-Wei Huang, and Ming-Syan Chen. Hardware-enhanced association rule mining with hashing and pipelining. *Knowledge and Data Engineering, IEEE Transactions on*, 20(6):784–795, 2008.

[18] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[19] Mohammed Javeed Zaki. Scalable algorithms for association mining. *Knowledge and Data Engineering, IEEE Transactions on*, 12(3):372–390, 2000.

[20] Yan Zhang, Fan Zhang, Zheming Jin, and Jason D Bakos. An fpga-based accelerator for frequent itemset mining. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 6(1):2, 2013.