Frequent Itemsets Mining in Data Streams Using Reconfigurable Hardware

Lázaro Bustio[1,2(✉)], René Cumplido[2], Raudel Hernández[1], José M. Bande[1], and Claudia Feregrino[2]

[1] Advanced Technologies Application Center, 7ª ♯ 21812 e/218 y 222, Rpto. Siboney, Playa, CP 12200 Havana, Cuba
{lbustio,rhernandez,jbande}@cenatav.co.cu
[2] National Institute for Astrophysics, Optics and Electronic, Luis Enrique Erro No 1, Sta. Ma. Tonantzintla, 72840 Puebla, Mexico
{lbustio,rcumplido,cferegrino}@ccc.inaoep.mx

Abstract. Data streams are unbounded and infinite flows of data arriving at high rates which cannot be stored for offline processing. Because of this, classical approaches for Data Mining cannot be used straightforwardly in data stream scenario. This paper introduces a single-pass hardware-based algorithm for frequent itemsets mining on data streams that uses the top-k frequent 1-itemsets. Experimental results of the hardware implementation of the proposed algorithm are also presented and discussed.

Keywords: Data mining · Frequent itemsets mining · Data streams · Reconfigurable hardware · Parallel algorithms

1 Introduction

Data Mining is a research area that investigates the tools and techniques needed to efficiently extract information from large volumes of data. One particularly important technique in Data Mining is frequent itemsets mining aimed at discovering those sets of items that can be found together more than a given number of occurrences in a data set (named *frequent itemsets*) [1].

One scenario that is gaining attention is data streams mining. Frequent itemsets mining on data streams is incipient and the majority of the developed algorithms for this task cannot deal with data streams in an exhaustive fashion because of high incoming rates of data, short processing times needed, and the impossibility of storing the incoming stream. Data streams mining can be found in video and audio streams, network traffic and commercial transactions among others [13]. Such applications need to operate at high speed, so some hardware-based approaches have been proposed to achieve that goal. In such approaches, custom hardware architectures were used as processing platforms due to their capacity to exploit parallelism.

In this paper, a parallel algorithm for frequent itemsets mining on data streams (which was designed to be implemented and executed in hardware) that

M. Ceci et al. (Eds.): NFMCP 2015, LNAI 9607, pp. 32–45, 2016.
DOI: 10.1007/978-3-319-39315-5_3

uses a Landmark Window Model [9] is proposed. This algorithm is based on a systolic tree which is well suited for data streams handling. Also a pre-processing stage that allows to obtain the most important itemsets and to ensure an optimal use of the hardware resources available on the selected hardware device is introduced. In this pre-processing stage, the top-k frequent items are detected and used as starting point for the mining process. To prove the feasibility of the proposed method, several experiments were conducted showing that it outperforms several well known software-based implementations reported in the state-of-the-art which were selected as baseline.

This paper is structured as follows: in Sect. 2 the theoretical basis that support this research is presented. A review of state-of-the-art is addressed in Sect. 3 while Sect. 4 introduces the proposed algorithm. Results are shown in Sect. 5 and finally, conclusions are given in Sect. 6.

2 Theoretical Basis

Let $I = \{i_1, i_2, .., i_n\}$ be a set of n items and T be a transactional dataset[1]:

Definition 1 (Itemset). *An itemset X is a set of items over I such that $X \subseteq I$.*

Definition 2 (Transaction). *A transaction $t \in T$ over I is a couple $t = (tid, X)$ where tid is the transaction identifier, and $X \subseteq I$.*

Definition 3 (Support). *The support of an itemset X is the fraction of transactions in T containing X.*

An itemset is called *frequent* if its support is greater than a given minimal support threshold *minsup*.

Definition 4 (Data stream). *A data stream is a continuous, unbounded and not necessarily ordered, real-time sequence of data items.*

In data stream three main characteristics are presented [2,13,27]:

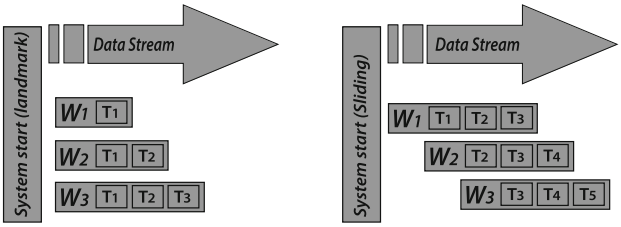
- *Continuity.* Items in stream arrive continuously at a high rate.
- *Expiration.* Items can be accessed and processed just once.
- *Infinity.* The total number of data is unbounded and potentially infinite.

Definition 5 (Window). *A window in a data stream is an excerpt of transactions.*

Windows can be constructed using one of following approaches [9,17]:

- *Landmark window model.* This model employs some point (called landmark) to start recording where a window begins. The support count of an itemset i is the number of transactions containing i between the landmark and the current time (see Fig. 1a).

[1] Dataset is referred to databases, unstructured data file, relational databases or any other data source. In this paper, dataset is used to refer data streams.



(a) Landmark Window Model. (b) Sliding Window Model.

Fig. 1. Different windows model. In both figures, three landmarks are presented and in each of them, the processing window is described using the number of transactions that were included.

– *Sliding window model*. This model uses the latest W transactions in the mining process. As newest transactions arrive, the oldest transactions in the sliding windows are excluded. This model can be compared with a FIFO queue. The use of this model imposes a restriction: as some transactions will be included in the mining process, methods for finding expired transactions and discounting the frequency counting of the itemsets involved are required (see Fig. 1b).

In this paper, the Landmark Window Model was selected and used because it can be seen as a general case (and starting point) of others complex windows model such as Sliding, and Landmark is more classical incremental learning instead of a Sliding Window which better deals with evolving characteristics of data streams [9,13,29]. In future works, the Sliding Window Model will be adopted for the mining process.

2.1 Reconfigurable Computing

Reconfigurable Computing is referred to the use of hardware devices in which the functionality of the logic gates is customizable at run-time, and FPGAs are the main exponent of this approach. The architecture of FPGAs is based on a large number of logic blocks which perform basic logic functions. Because of this, FPGAs can implement from a simple logical gate to a complex processing system. FPGAs can be reprogrammed; that is, the circuit can be "erased" and then, a new architecture that implements a new algorithm can be placed. This capability of the FPGAs allows the creation of fully customized architectures, reducing cost and technological risks that are present in traditional circuits designs [10].

3 Related Works

Analyzing the reviewed literature, it can be noticed that there are three preferred approaches: algorithms that use Apriori [1] as the starting point, algorithms that use FP-Growth [14] and those that use Eclat [37]. Algorithms that mimic Apriori [3,4,33,36] implemented in hardware require loading the candidates itemsets

and the dataset into the hardware. This strategy is limited by the capacity of the chosen hardware platform. If the number of transactions to manage is larger than the hardware capacity, transactions must be loaded separately in many consecutive times degrading overall performance (the acceleration obtained by the use of FPGA is lost in I/O operations). These issues are forbidden in data streams mining because they do not fulfill the Continuity and Expiration restrictions. One valid option for frequent itemsets mining on data streams is to develop tree-based approaches where transactions flow inside the tree.

Similarly to Apriori-based algorithms, the FP-Growth-based algorithms [22,24,30–32] require to copy the mining dataset to the processing platform. They also require to go through the datset in two times, except Mesa et al. [24], however it still needs to download the entire dataset to the hardware device. This is impractical in data streams mining scenario due to the Expiration restriction. Like other reviewed algorithms, authors focused their attention in better data structures that allow the efficient counting of frequent itemsets. However, algorithms based on FP-Growth use the FP-Tree data structure [14] that is based on prefix-trees which are well suited for data streams mining applications.

Eclat-based algorithms [28,39] use the vertical dataset representation in order to save memory and processing time. In [28,39], authors use the intersection of items to compute the support. They show that it is more efficient than hash-trees. All the Eclat-based implementations propose an hybrid approach, where the most time and/or memory consuming functions were downloaded to custom hardware while the software controls the execution flow and data structures. Although the vertical dataset representation allows to save memory and processing time, it is not compatible with the Expiration restriction of data streams.

4 A Method for Frequent Itemsets Mining Using Reconfigurable Hardware

The basic idea of the proposed method is to develop a tree structure of processing units where transactions from data streams flow from the root node to the leafs nodes. The tree structure can be seen as a *systolic tree* where each of its nodes has one bottom node (child) and one right node.

Figure 2 shows the systolic tree where vertically-arranged nodes represent a prefix path and parent nodes contain the prefix itemset (that represent transactions of data streams) for their children. Taking a random node r, the sub-tree who had the node r as root is formed by all possible combinations of items with the itemset stored in r as their prefix, this leads to recursive mining strategies. The systolic tree data structure implements a distributed control scheme where processing and control logic are distributed among the nodes. A schematic of the systolic tree is shown in Fig. 3.

Data streams are data sources that usually evolve in time. Because of this, it is unrealistic and meaningless in several applications to determine all frequent itemsets that have been transmitted. Also in such cases, changes of patterns and their trends are more relevant than the patterns themselves. Detecting changes

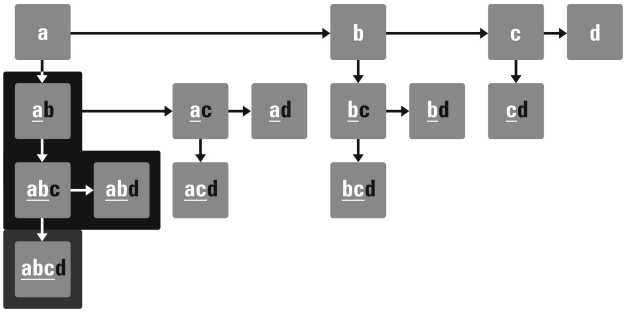


Fig. 2. Systolic tree data structure used in data stream mining. Remarked section shows a sub-tree with nodes that have the itemset *ab* of a transaction *t* as root.

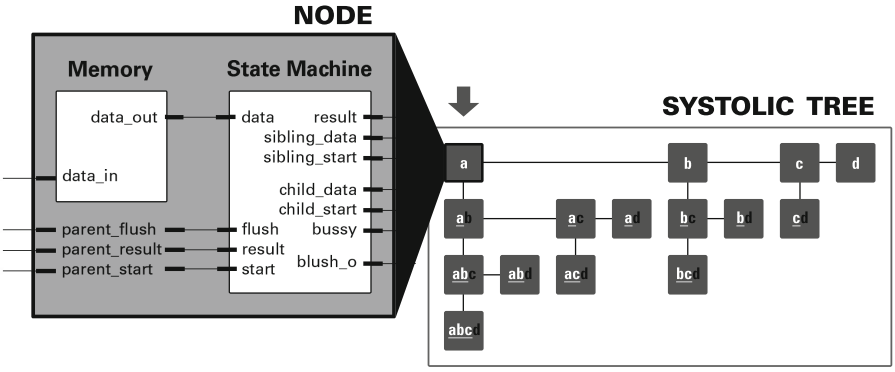


Fig. 3. Schematic of the processing node (gray and magnified at left side of the figure) and structure of the systolic tree (at right side of the figure).

on transmitted patterns in data streams is studied in several ways, and one of these ways is to find the most occurring 1-itemsets (also named *heavy hitters*) [11,20,23,25,26,38]. This problem have been also attacked from the hardware perspective [18,34].

4.1 Frequent 1-Itemsets Detection

In data streams mining the detection of top-frequent 1-itemsets can be seen as a pre-processing stage where the most representative itemsets transmitted in the stream are discovered. Using the top-frequent 1-itemsets as starting point to reduce the search space also allows to reduce the computational resources needed and this ensures to obtain the most valuable frequent itemsets. Also, frequent 1-itemsets detection is useful to verify a concept drift in the knowledge [15,16,35] which is transmitted in the stream. The use of top-frequent 1-itemsets in data stream mining tasks was used in [5,19] and the validity of using top-frequent 1-itemset detection is supported on the "Downward Closure" property [1].

In the reviewed literature, it can be noticed that reported algorithms use the k-first different items received from the data stream (where k is the maximum number of single items that can be assimilated by hardware platforms). These k-first items are taken in arrival order and may not be the most representative of the stream. It may occur that the processing systolic tree deals with items that are not interesting and do not generate any frequent itemsets but nevertheless consume hardware resources. In order to alleviate this situation, in this paper a strategy to ensure the received itemsets will be the most representatives is adopted: this is achieved by obtaining top-k frequent 1-itemsets in each window. This process is performed in the window creation phase and a trade-off must be adopted: to use some transactions (e.g. 1/10 of the window length) to obtain top-k frequent 1-itemsets in order to obtain the most valuable frequent itemsets.

In the proposed processing scheme the top-k frequent 1-itemsets are used, where k is directly related to available hardware resources. Those top-k frequent 1-itemsets are obtained in a pre-processing stage implemented in software. Later, those top-k frequent 1-itemsets are transmitted to the mining process (which is implemented in hardware).

4.2 Proposed Method

Figure 4 shows the proposed processing scheme. It shows the pre-processing stage (software) and the mining process (hardware). Algorithm 1 is executed simultaneously in each node of the systolic tree. For each transaction t in Landmark window when a new itemset X arrives to a node occupied by an item $i_j \in I$, one of following decisions must be taken:

1. If $\{i_j\} \subseteq X$ then the frequency counter of the node is incremented and the itemset $X - \{i_j\}$ is flowed to child and right node.
2. If $\{i_j\} \not\subseteq X$ then X is flowed to right node.

Algorithm 1 simultaneously uses Depth First Search (DFS) and Breadth First Search (BFS) traversals (from lines 20 to 25) allowing a high level of parallelism: two dimensional searches are performed concurrently. These DFS and

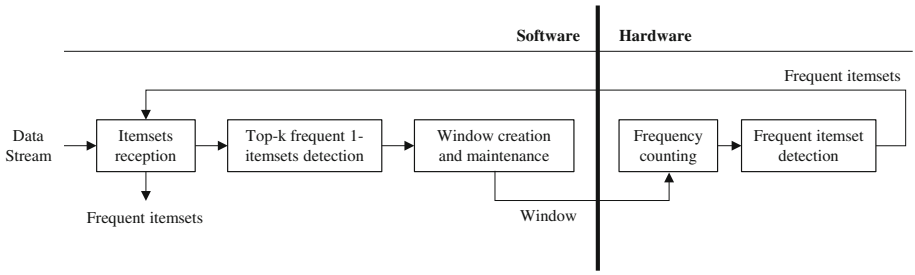


Fig. 4. Processing scheme proposed which is composed by a software-side which executes a frequent 1-itemsets detection and a hardware-side which executes the mining process in the systolic tree.

Algorithm 1. Parallel algorithm for finding the frequency counting of flushed transactions in a Landmark Window.

Input: Landmark Window *landmark_window*
Output: Systolic tree with the counting frequency of each itemset.

```
1  n_i ← systolyc_tree.RootNode;
2  foreach transaction t in landmark_window do
3  |  Traverse(t, n_i);
4  Procedure Traverse(t, n_i);
5  if n_i.IsOccupied == false then
6  |  n_i.IsOccupied = true;
7  |  n_i.Item = t.ItemAt(0);
8  |  FlushInParallel(t, n_i);
9  else
10 |  if t.Contain(n_i.Item) == true then
11 |  |  FlushInParallel(t, n_i);
12 |  else
13 |  |  ñ_i ← n_i.RightNode;
14 |  |  Traverse(t, ñ_i);
15 EndProcedure;
16 Procedure FlushInParallel(t, n_i);
17 n_i.Counter + +;
18 t̃ = t.Exclude(n_i.Item);
19 if t̃.IsEmpty == false then
20 |  StartParallalelBlock::
21 |  ñ_i ⟵ n_i.ChildNode;
22 |  Traverse(t̃, ñ_i);
23 |  ñ_i ⟵ n_i.RightNode;
24 |  Traverse(t̃, ñ_i);
25 |  EndParallelBlock;;
26 EndProcedure;
27 return systolic_tree;
```

BFS traversal strategies are also employed to determine which itemsets can be regarded as frequent once the frequency counting of each itemset was computed and stored in nodes of the systolic tree.

After the frequency of each itemset is computed, a backtracking strategy using the "Downward Closure" [1] is employed to obtain the frequent itemsets from the systolic tree. This strategy is implemented in Algorithm 2. Once again, a simultaneous BFS and DFS strategy is implemented lines 7 to 10.

To obtain frequent items, a recursive strategy that uses the "Downward Closure" property [1] stated before is used. In this strategy, if a node is declared as *frequent*, then its child and right nodes must be processed recursively to determine whether they are frequent or not. On the contrary, if a node is regarded

Algorithm 2. Parallel algorithm for finding frequent itemsets.

Input: Flushing flag $flush$, Minimum support value min_sup.
Output: Frequent itemsets and their frequency counting $< itemset, frequency >$.

```
1  n_i ← systolyc_tree.RootNode;
2  if (flush == true) then
3      FlushMethod(n_i, min_sup);
4  Procedure FlushMethod(n_i, min_sup)
5  if (n_i.counter ≥ min_sup) then
6      if (n_i.ChildNode! = null)and(n_i.RightNode! = null) then
7          StartParallalelBlock:
8          FlushMethod(n_i.ChildNode, min_sup);
9          FlushMethod(n_i.RightNode, min_sup);
10         EndParallelBlock;
11         n_i.GatewayMode = true;
12     else
13         FlushResult.Add(n_i, n_i.Counter);
14 else
15     FlushResult.Add(n_i, n_i.Counter);
16 return FlushResult;
```

as *infrequent*, its descendants will be infrequent and the process can be stopped (consequence of the "Downward Closure" property). In this way, the traverse strategy to obtain the frequent itemsets in the systolic tree is optimized.

5 Results

The proposed method was modeled in VHDL using Xilinx ISE Suite 14.2 and targeted for a Virtex 5 XC5VLX330T FPGA device. After the architecture was synthesized and implemented, the occupied area is 90.3 %, holding 1161216 processing nodes. This allows to handle a maximum of 20 different items, while the largest itemsets handled by FPGA-accelerated architecture reported is about 11 items. The maximum operation frequency obtained for this architecture is 238 Mhz. This means that the proposed architecture can process 5.07×10^6 transactions (containing at most 20 items) per second (worst case), this derives in a throughput of 1.19 Gbps. Besides, as far as we know, this is the first FPGA-accelerated architecture for frequent itemset mining over data streams reported.

To validate the performance of the proposed systolic tree architecture, some experiments were conducted simulating data streams with 4 different datasets. MSNBC and Chess datasets were taken from UCI repository [21] and the other two were created with the Almaden IBM Synthetic Dataset Generator [12]. Employed datasets are described in Table 1.

Table 1. Dataset specifications.

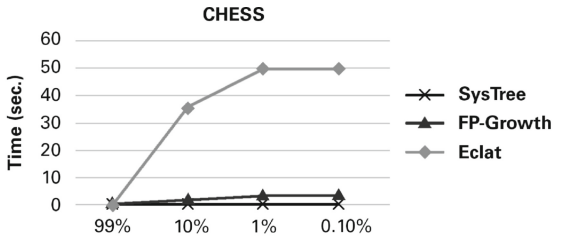
Dataset	Size (Mb)	#Trans	#Items	Ave.IT
MSNBC	4.219	989 818	17	2.82
Chess	0.340	3 196	75	37
T10I20D100K	1.468	100 000	20	10
T20I20D1M	27.123	1 000 000	20	20

Since there are no hardware architectures reported for frequent itemsets mining on data streams, the best software implementations (concerning of performance and resources consumption) of Apriori [7], FP-Growth [8] and Eclat [7] algorithms that were reported in the state-of-the-art, were used as baseline. These implementations were downloaded from Borgelt's website [6] and several experiments were conducted using MSNBC, Chess, T10I20D100K and T20I20D1M datasets on an Intel Core i3 at 1.8 GHz and 4 Gb of RAM. Experiments using all datasets were conducted in software using several *minsup* values and timing results were compared versus the same experiments conducted in hardware. Timing results are shown in Fig. 5. These results are shown to give an idea of the performance of the proposed method compared against some well established state-of-the-art algorithms. Window size was set to 100 000 transactions. This allows to handle in one window the MSNBC, Chess and T10I10D100K datasets, while T20I20D1M was separated in 10 windows.

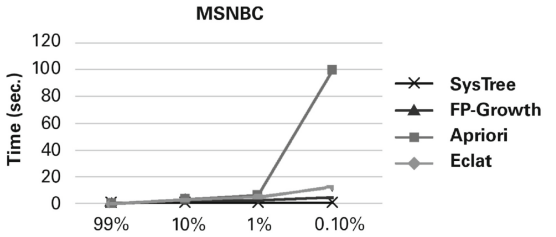
5.1 Discussion

As it is shown in Fig. 5, the proposed architecture outperforms all the baseline algorithms in all datasets. Note that, for the sake of clarity, processing times that exceeds 3 min are not shown. All datasets, except Chess, contain 20 or less single items, so they can be mapped directly into the hardware architecture. Chess dataset contains 75 items so the systolic tree only takes into account for the mining process the most frequent 20 items arrived. In this case, the mining process is approximate, discovering a subset of total frequent itemsets but this subset express more accurately the behavior of frequent itemsets if it were obtained using the first 20 items arrived. Itemsets regarded as frequent in this case have the same frequency counting as if they had been calculated with any baseline algorithm. For MSNBC, T10I20D100K and T20I20D1M, the mining process is exact and it was performed one order of magnitude times faster than baseline algorithms.

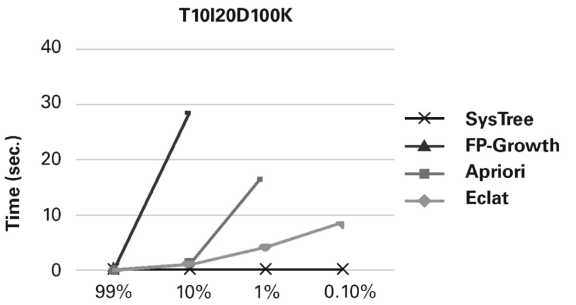
Experiments demonstrate that the proposed architecture is insensitive to variations in support threshold: this is explained because of all hardware structures needed for mining incoming data stream is available and it is the same whether the minimum support threshold is 1 % or 99 %. Restrictive support threshold conduce to less frequent itemsets while relaxed support threshold conduce to more frequent itemsets. Performance of software implementations of



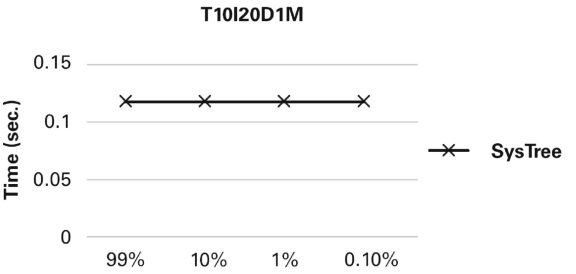
(a) Execution time for the proposed architecture with Chess dataset.



(b) Execution time for the proposed architecture with MSNBC dataset.



(c) Execution time for the proposed architecture with T20I20D100K synthetic dataset.



(d) Execution time for the proposed architecture with T10I20D1M synthetic dataset. Here, timing results that exceeds 180 s were dropped from chart.

Fig. 5. Performance evaluation of the proposed algorithm and architecture for different datasets. The X axis represents the variation of the support threshold expressed in %.

Apriori, Eclat and FP-Growth are quite sensitive in its performance to changes on support value.

Theoretically, the size (in number of nodes) of the systolic tree is determined by $|I|$, where I is a super-set of items in the incoming data stream (see Sect. 2). Since $|I|$ cannot be established a priori (due the Infinity property of data streams), the size of the systolic tree will be determined by the capacity of the development platform. Assuming that the development platform contains enough computational resources (ideal case), the size of the systolic tree (in number of nodes) that can hold any streams formed by items of I will be:

$$nodes = 2^{|I|} - 1 \tag{1}$$

In a real case, the available resources of a FPGAs are limited. Supposing 100 % of area occupation in selected hardware device, a number p will be the maximum number of processing nodes that can be supported by the selected device. Then the maximum number max_{items} of items in I in the incoming data stream that can be handled by the chosen device will be:

$$max_{items} = \log_2(p + 1) \tag{2}$$

For example, if the selected hardware device can hold 1024 nodes ($p = 1024$) in systolic tree, then the maximum number of items of I that can be handled will be 10 ($|I|$).

It is important to notice that for a certain number n of items in set I, if

$$2^n - 1 > p \tag{3}$$

the systolic tree cannot hold all possible combinations of itemsets and therefore some itemsets will not be taken into account during the mining process. In other words, the number of processing nodes needed to handle all possible combinations of itemsets generated from I exceeds the maximum number of processing nodes that can be mapped into the selected FPGA. Here, mining will be approximate with no false positives produced. In this case, the selection of top-k frequent 1-itemsets allows to obtain the most valuable items to produce frequent itemsets. Obtaining some information about the incoming data stream introduces some delay (which is negligible) in the mining process but it implies a higher quality of the produced itemsets. If the systolic tree is occupied by items in arrival order, it cannot be guaranteed that all of the received items in transaction will produce itemsets that could be frequent. In such case, the use of the systolic tree is not optimal. By the contrary, if the systolic tree is occupied by items that had been proved to be frequent, it can be ensured that all combination of those items will produce frequent itemsets. Using this strategy the mining process will be also approximate, but it is guaranteed that the produced itemsets will be the most frequent ones. From the point of view of users, it is more useful to obtain "some" frequent itemsets than to obtain "all" frequent itemsets [11,23,25]. In this paper, using the top-frequent 1-itemsets it is ensuring that frequent itemsets obtained describe better the data stream behavior than just using the k-first arrived 1-itemsets.

On the contrary, if:

$$2^n - 1 \leq k \tag{4}$$

the systolic tree can hold all the possible combinations of itemsets, therefore the mining process will be exact.

6 Conclusions

This paper introduces a new parallel algorithm for frequent itemsets mining in data streams which is designed to be implemented in a custom hardware architecture. The proposed algorithm is based on a tree data structure that allows to increase the mining performance. The corresponding hardware architecture is based on a systolic tree approach where the control logic is distributed among all processing nodes. Here, the use of top-k frequent 1-itemsets allows to optimize the use of the systolic tree and to obtain the most valuable frequent itemsets. Experimental results showed that the hardware architecture is able to extract correctly all itemsets from data streams with a significant speed up when compared against software-based implementations.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 1994, pp. 487–499. Morgan Kaufmann Publishers Inc., San Francisco (1994)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2002, pp. 1–16. ACM, New York (2002)
3. Baker, Z.K., Prasanna, V.K.: Efficient hardware data mining with the Apriori algorithm on FPGAs. In: Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2005, pp. 3–12. IEEE Computer Society, Washington (2005)
4. Baker, Z.K., Prasanna, V.K.: An architecture for efficient hardware data mining using reconfigurable computing systems. In: 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006, FCCM 2006, pp. 67–75, April 2006
5. Baralis, E., Cerquitelli, T., Chiusano, S., Grand, A., Grimaudo, L.: An efficient itemset mining approach for data streams. In: König, A., Dengel, A., Hinkelmann, K., Kise, K., Howlett, R.J., Jain, L.C. (eds.) KES 2011, Part II. LNCS, vol. 6882, pp. 515–523. Springer, Heidelberg (2011)
6. Borgelt, C.: Software for frequent pattern mining. http://www.borgelt.net/fpm.html. Accessed on 20 May 2015
7. Borgelt, C.: Efficient implementations of Apriori and Eclat. In: FIMI. CEUR Workshop Proc., vol. 90. CEUR-WS.org (2003)
8. Borgelt, C.: An implementation of the FP-growth algorithm. In: Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, OSDM 2005, pp. 1–5. ACM (2005)

9. Cheng, J., Ke, Y., Ng, W.: A survey on algorithms for mining frequent itemsets over data streams. Knowl. Inf. Syst. **16**(1), 1–27 (2008)
10. Compton, K., Hauck, S.: Reconfigurable computing: A survey of systems and software. ACM Comput. Surv. **34**(2), 171–210 (2002)
11. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Finding hierarchical heavy hitters in data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB 2003, vol. 29, pp. 464–475. VLDB Endowment (2003)
12. Corporation, I.B.M.: IBM quest market-basket synthetic data generator. http://www.cs.loyola.edu/cgiannel/assoc_gen.html. Accessed on 20 April 2015
13. Golab, L., Ozsu, M.: Data stream management issues - a survey. SIGMOD Rec. **32**(2), 5–14 (2003)
14. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD 2000, pp. 1–12. ACM, New York (2000)
15. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2001, pp. 97–106. ACM, New York (2001)
16. Jiang, N., Gruenwald, L.: Research issues in data stream association rule mining. SIGMOD Rec. **35**(1), 14–19 (2006)
17. Jin, R., Agrawal, G.: Frequent pattern mining in data streams. In: Aggarwal, C. (ed.) Data Streams. Advances in Database Systems, vol. 31, pp. 61–84. Springer, US (2007)
18. Lai, Y.K., Wang, N.C., Chou, T.Y., Lee, C.C., Wellem, T., Nugroho, H.T.: Implementing on-line sketch-based change detection on a NetFPGA platform. In: 1st Asia NetFPGA Developers Workshop (2010)
19. Lee, W., Stolfo, S.J., Mok, K.W.: Adaptive intrusion detection: A data mining approach. Artif. Intell. Rev. **14**(6), 533–567 (2000)
20. Thanh, L.H., Calders, T.: Mining top-k frequent items in a data stream with flexible sliding windows. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2010, pp. 283–292. ACM, New York (2010)
21. Lichman, M.: UCI machine learning repository (2013). Accessed on 20 April 2015
22. Malarvizhi, S.P., Sathiyabhama, B.: Enhanced reconfigurable weighted association rule mining for frequent patterns of web logs. Int. J. Comput. **13**(2), 97–105 (2014)
23. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB 2002, pp. 346–357. VLDB Endowment (2002)
24. Mesa, A., Feregrino-Uribe, C., Cumplido, R., Hernández-Palancar, J.: A highly parallel algorithm for frequent itemset mining. In: Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Kittler, J. (eds.) MCPR 2010. LNCS, vol. 6256, pp. 291–300. Springer, Heidelberg (2010)
25. Metwally, A., Agrawal, D.P., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 398–412. Springer, Heidelberg (2005)
26. Metwally, A., Agrawal, D., Abbadi, A.E.: An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Trans. Database Syst. **31**(3), 1095–1133 (2006)
27. Pramod, S., Vyas, O.P.: Data stream mining: A review on windowing approach. Global J. Comput. Sci. Technol. **12**(11-C) (2012)

28. Shi, S., Qi, Y., Wang, Q.: Accelerating intersection computation in frequent itemset mining with FPGA. In: 2013 IEEE 10th International Conference on High Performance Computing and Communications 2013, pp. 659–665, November 2013
29. Shie, B.E., Yu, P.S., Tseng, V.S.: Efficient algorithms for mining maximal high utility itemsets from data streams with different models. Expert Syst. Appl. **39**(17), 12947–12960 (2012)
30. Sun, S., Steffen, M., Zambreno, J.: A reconfigurable platform for frequent pattern mining. In: International Conference on Reconfigurable Computing and FPGAs, 2008, ReConFig 2008, pp. 55–60, December 2008
31. Sun, S., Zambreno, J.: Mining association rules with systolic trees. In: International Conference on Field Programmable Logic and Applications, 2008, FPL 2008, pp. 143–148, September 2008
32. Sun, S., Zambreno, J.: Design and analysis of a reconfigurable platform for frequent pattern mining. IEEE Trans. Parallel Distrib. Syst. **22**(9), 1497–1505 (2011)
33. Thoni, D.W., Strey, A.: Novel strategies for hardware acceleration of frequent itemset mining with the Apriori algorithm. In: International Conference on Field Programmable Logic and Applications, 2009, FPL 2009, pp. 489–492, August 2009
34. Tong, D., Prasanna, V.: Online heavy hitter detector on FPGA. In: 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1–6. IEEE (2013)
35. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2003, pp. 226–235. ACM, New York (2003)
36. Wen, Y.H., Huang, J.W., Chen, M.S.: Hardware-enhanced association rule mining with hashing and pipelining. IEEE Trans. Knowl. Data Eng. **20**(6), 784–795 (2008)
37. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**(3), 372–390 (2000)
38. Zhang, Y., Singh, S., Sen, S., Duffield, N., Lund, C.: Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications. In: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, IMC 2004, pp. 101–114. ACM, New York (2004)
39. Zhang, Y., Zhang, F., Jin, Z., Bakos, J.D.: An FPGA-based accelerator for frequent itemset mining. ACM Trans. Reconfigurable Technol. Syst. **6**(1), 2:1–2:17 (2013)