Hardware Acceleration of Frequent Itemsets Mining on Data Streams

Lázaro Bustio-Martínez, René Cumplido, Raudel Hernández-León, Claudia Feregrino-Uribe

Reporte Técnico No. CCC-14-002
5 de Marzo de 2014

Hardware Acceleration of Frequent Itemsets Mining on Data Streams. PhD Proposal.

Lázaro Bustio-Martínez[a,b,*], René Cumplido-Parra[b], Raudel Hernández-León[a], Claudia Feregrino-Uribe[b]

[a] *Advanced Technologies Application Center.*
7ma A ♯21406 e/ 214 y 216, Rpto. Siboney, Playa. C.P. 12200. La Habana, Cuba.
[b] *National Institute for Astrophysics, Optics and Electronic.*
Luis Enrique Erro ♯1, Sta. Ma. Tonantzintla, 72840, Puebla, México.

Abstract

In recent times, processing of streams of data is gaining the attention of the scientific community due to its practical applications, i.e. network traffic analysis and surveillance, sensor networks control, financial transactions mining among others. Data stream is an unbounded and infinite flow of data arriving at high rates and, therefore, the classical approaches can not be used straightforward in this scenario. Frequent itemsets mining is a Data Mining technique that guarantees to obtain hidden patterns in data, and it have been used to extract useful knowledge from various data sources including data streams. Commercial solutions can not process in an exhaustive manner the produced data streams because of the high incoming rates and the huge number of transactions transmitted. In the most of cases, commercial solutions can not process exhaustively data stream at all. Because of this, finding alternatives to achieve better results in the discovering of frequent itemsets on data streams is an active research topic. One of such alternatives is to develop single-pass parallels algorithms that can be implemented in hardware to take advantage of the inner parallelism of such devices. The main contribution of this research is the design and develop of a new single-pass algorithm that can mine high incoming rates data streams. As preliminary results, the proposed methods can mine in exhaustive fashion the incoming data streams when the number of single items is low. When the number of single items in transactions is high, the proposed method obtains an approximate solution with no false positives itemset produced.

Keywords: Data mining, frequent items and itemsets mining, reconfigurable hardware, custom hardware architectures.

Contents

[*] Corresponding authors
Email address: `lbustio@cenatav.co.cu` (Lázaro Bustio-Martínez)

1. Introduction.

In recent years, there has been an explosion on the amount of data generated by all sort of human activities. In order for this data to be useful, it must be processed to obtain hidden knowledge. To perform this task, several approaches have been proposed and implemented mainly in software-based systems that offer limited performance when processing large amounts of data. Data generating rate is growing exponentially while Data Mining application performance has only increased by 10-15% [1]. In 2012, about 2.5 exabytes of data were created each day on the whole world, and that number is doubling every 40 months or so. For instance, Walmart supermarkets collects more than 2.5 petabytes (approximate) of data every hour from its customer transactions. Being able to mine such data set, will be useful to obtain invaluable new knowledge of buying patterns of customers. Modern-day algorithms cannot deal with such volumes of data, and it is evident that are needed to research and to propose new data acquisition paradigms [2].

Data Mining aims to pride the tools and techniques needed to face such immense data[1] volumes. In Data Mining scenario is extremely useful to record all the occurrences of certain patterns and that is what frequent items and itemsets mining performs. Frequent items are those data patterns that can be found more than a given number of occurrences in data, while frequent itemsets are those sets of data items that can be found always together more than a given number of occurrences in data. In other words, the goal of frequent items and itemsets mining is to determine which elements in a database (or any other data source) commonly appear together. The basic sketch in frequent itemsets mining is shown:

1. Get the basic patterns.
2. Count the number of occurrences of those basic patterns in data.
3. Remove those patterns that do not exceed a certain value (number of occurrence).
4. Perform all combination of length n (n = 2,3....) with the basic patterns obtaining n-frequents patterns.
5. Go to step 2 and repeat forthcoming steps until there is no possible combination or no combination exceeds the number of occurrences.

This sketch was enhanced by Agrawal [3] and then he proposed an algorithm named *Apriori*. Apriori was the first approach proposed for frequent itemsets mining and also, it is the simplest. Notice that it is in the first step where the frequent items mining is performed. The third step is also known as *Candidate Prune*, and the fourth step is known as *Candidate Generation* (See [3] for further references).

[1]Immense data volumes are referred to any data source that is in order of gigabytes of storage capacity or more.

Frequent items mining is an essential part of frequent itemsets mining and the second one fully include the first one. Due to the process to find frequent items is part of others Data Mining task (such as frequent itemsets mining, frequent sequence mining or association rules mining) it must be done efficiently. The algorithms that have been created for frequent items and itemsets mining tasks require large amounts of computational resources to solve the combinatorial explosion of itemsets that can be found in a dataset[2]. This is mainly due to the presence of thousands of different patterns or the use of a too low threshold of support (The support threshold is the minimum number of occurrences of an item or itemset to be considered as "frequent". See section 2 for further information).

To accelerate frequent items and itemsets mining process, some parallel algorithms have been proposed. Some of those algorithms use hardware devices (such as FPGAs and GPUs) to take advantage of the inner parallel hardware resources [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1, 14, 15, 16].

1.1. Proposal organization.

This proposal is organized as follows: the theoretical basis is presented in section two. Section three present the review of state-of-the-art of the frequent itemsets mining on both data streams and databases. Methodological foundations of this PhD Proposal are presented in section fourthwhile in section fifththe preliminary results are presented.

2. Theoretical basis.

In this section the main concepts and theoretical basis that will be used further in this PhD proposal are introduced.

2.1. The frequent itemsets problem.

Frequent itemsets mining was introduced by Agrawal et al. back in the early 90's[3], and it is used for finding common and potentially interesting patterns in databases. In this scope, data are represented by means of transactions, each of which is a set of items labeled by a unique ID. The purpose of frequent itemsets mining is to find the most frequently-occurring subsets from these transactions. The frequency of the subset is measured by *support ratio*, which is the number of transactions containing the subset divided by the total number of transactions in the database. Frequent itemsets are typically used to generate association rules and is also useful in sensor network mining[17], database management systems, information retrieval, bioinformatics, data streams analysis, and computer vision [18], among others.

Formalizing, let $I = \{i_1, i_2, .., i_n\}$ be a set of items :

[2] *dataset* is referred to database, unstructured file, data stream or any other data source.

Table 1: An example transaction database D.

tid	X
100	{beer, chips, wine}
200	{beer, chips}
300	{pizza, wine}
400	{chips, pizza}

Definition 1 (Itemset). A itemset X is a set of items over I such $X = \{i_i, ..., i_k\} \subseteq I$. If a set X contains k items, then the set X is called *k-itemset*. Normally is considered that the items in an itemset are lexicographically ordered.

Definition 2 (Transaction). A transaction T over I is a couple $T = (tid, I)$ where *tid* is the transaction identifier, and I is a $X \subseteq I$ itemset. A transaction $T = (tid, I)$ is said to support an itemset $X \subseteq I$, if $X \subseteq I$.

Definition 3 (Transaction Database). A transaction database D over I is a set of transactions over I such that each transaction has a unique identifier (duplicated transactions are not allowed).

Definition 4 (Support). The support of an itemset X in D is the number of transactions in D that contains X:

$$Support(X, D) = |\{tid|(tid, I) \in D, X \subseteq I\}| \tag{1}$$

Support can be expressed in terms of the number of occurrences of an itemset $(1, 2, 3...)$ or in term of percent of the total of transactions$(1\%, 2\%, 3\%...)$.

Definition 5 (Frequency). The frequency of an itemset X in D is the probability of X occurring in a transaction $T \subseteq D$:

$$Frequency(X, D) = P(X) = \frac{Support(X, D)}{|D|} \tag{2}$$

Note that $|D| = support(\{\}, D)$. An itemset is called $frequent$ if its support is no less than a given absolute minimal support threshold ϕ_{abs}, with $0 < \phi_{abs} \leq |D|$. The frequent itemsets discovered does not reflect the impact of any other factor except frequency of the presence or absence of an item.

Consider the example database shown in table 1 over the set of items I={beer, chips, pizza, wine}. Table 2 shows all frequent itemsets in D with respect to a minimal support threshold of 1.

It is important to notice two characteristics of itemsets to be considered:

Table 2: Itemsets and their support in D.

itemset	Cover	Support	Frequency
{}	{100, 200, 300, 400}	4	100%
{beer}	{100, 200}	2	50%
{chips}	{100, 200, 400}	3	75%
{pizza}	{300, 400}	2	50%
{wine}	{100, 300}	2	50%
{beer, chips}	{100, 200}	2	50%
{beer, wine}	{100}	1	25%
{chips, pizza}	{400}	1	25%
{chips, wine}	{100}	1	25%
{pizza, wine}	{400}	1	25%
{beer, chips, wine}	{100}	1	25%

- In an itemset, items cannot be repeated; that mean if an item is found more than once it will be considered as a single item (due to the *Set* concept).
- The order of items in itemsets is not important. If the order does matter, then we can say that we are in the presence of a *sequence*[19].

These characteristics facilitate the designing process of new frequent itemsets mining algorithms.

2.2. Data streams.

Now a days the world is becoming more and more interconnected and recent advances in hardware and software technology support this interconnection. Computing is heterogeneous and distributed and generates continuous streams of digital data. Data streams can be found in financial analysis, data centers and system health management, law enforcement, radio astronomy and physical sciences, telecommunications, health and life sciences, manufacturing process control, smarter traffic management and energy and utility management, among others[20]. Digital data that are transmitted in those streams are produced by diverse sources of information so the data itself may consist of various formats and structures. To handle appropriately such data, system must deal with high performance requirements of latency, error resilience or scaling.

There are a lot of definitions of *Data Streams.* Accordingly to [18] and [20] a Data Streams can be defined as:

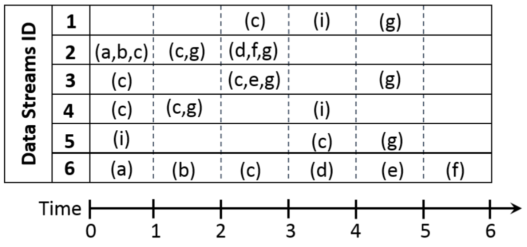


Figure 1: Visual representation of 6 data streams through the time.

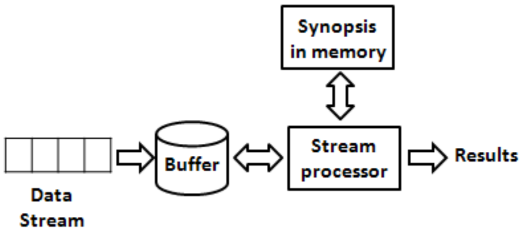


Figure 2: Computation model for data streams mining.

Definition 6 (Data streams). A data stream is a continuous, unbounded and not necessarily ordered (the order can be established implicitly by arrival time or explicitly by timestamp) real-time sequence of data items.

Based on definition 6, and according with [18] we can verify on data streams the following characteristics:

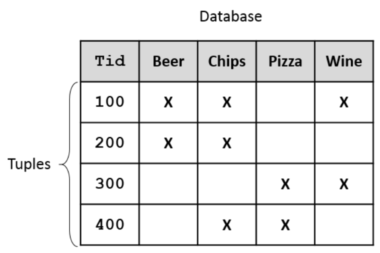
1. **Continuity:** Items in stream arrive continuously at a high rate.
2. **Expiration:** Items can be accessed and processed just once by the processing units in a data streams.
3. **Infinity:** The only assumption that we can make about bounds of streams is that the total number of data is unbounded and potentially infinite.

So, it is such a challenge to extend the Data Mining techniques, specifically frequent items and itemsets mining, to data streams scenario.

Fig. 1 shows the visual representation of 6 different data streams through time. The figure represents, for example, different products bought by six costumers in a supermarket, or bits that appear together in network broadcasting. It can be seen that in a data streams scenario is extremely difficult to know which item is next. Also is a challenge to process the items in some way due to the limited time to access them and the impossibility to store them for later processing. Fig. 2 shows the generalities of a data streams management system.

In the remainder, we assume without loss of generality that the items in an itemset are lexicographically ordered. Definition 2 can be used in streams scenario and here new concepts emerge:

Definition 7 (Transaction data stream). A transaction data stream is a sequence of incoming transactions in a data stream.


Database

Tuples

Tid	Beer	Chips	Pizza	Wine
100	X	X		X
200	X	X		
300			X	X
400		X	X	

Figure 3: In databases scenario, frequent itemsets are those sets who appear more than a specified number of times (named *support value*) in tuples of database. e.g., itemset {*beer, chips*} appear in 2 of the 4 transactions, so, for a minimum support value of 2, it can be regarded as frequent.

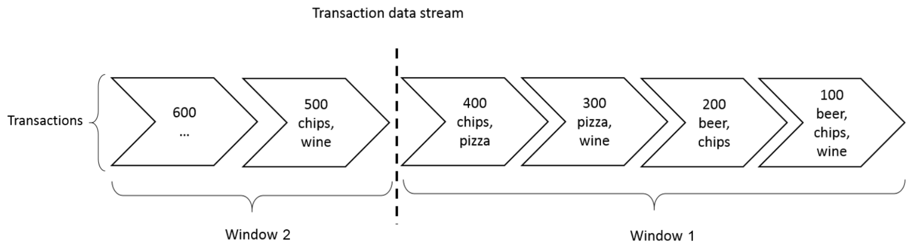


Figure 4: In data stream scenario, the frequent itemsets has meaning inside the *Window* concept and a window is equivalent to a table in databases. e.g., itemset {*beer, chips*} appear in 2 of the 4 transactions considered in window 1, so, for a minimum support value of 2, it can be regarded as frequent.

In databases scenario, frequent itemsets are those sets who appear more than a specified number of times in tuples of database. Fig. 3 shows the database presented in table 1. For a support value of 2, frequent itemsets are {*beer, chips*}, {*wine*}, {*pizza*}, {*beer*} and {*chips*}.

Fig. 4 represents, in a data stream fashion, the database showed in table 1 and in Fig. 3. In data stream scenario, the frequent itemsets has meaning inside the *Window* concept and a window is equivalent to a table in databases. For window 1 in Fig. 4, the frequent itemsets are {*beer, chips*}, {*wine*}, {*pizza*}, {*beer*} and {*chips*} for a support value of 2.

Following, window concept is defined.

Definition 8 (Window). A window in a data stream is an excerpt of items that pertain to the stream.

Definition 9 (Count-based Window). A window W is a count-based when it is composed of a sequence of batches, where each batch consists of an equal number of transactions.

Definition 10 (Time-based Window). A window W is a time-based if W consists of a sequence of fixed-length time units, where a variable number of transactions may arrive within each time unit.

In data streams mining is crucial to determine which model is the best suited to perform the mining process. There are three main models to use:

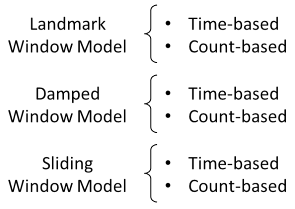


Figure 5: Streams mining models.

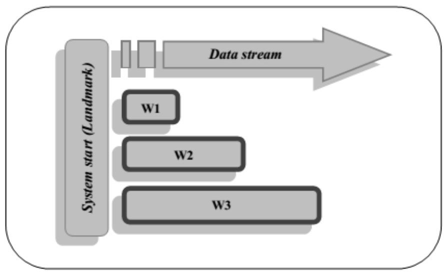


Figure 6: Landmark Window Model for data streams mining.

- Landmark window model.
- Damped window model.
- Sliding window model.

The window models presented can be represented as it is shown in Fig. 5.

2.2.1. Landmark Window Model.

The Landmark Window Model employs some points (called landmark) to start recording where a transaction begins and ends. The landmark usually refers to the time when the system starts. Moreover, the support count of an itemset in this model is the number of transactions containing it between the landmark and the current time. The Landmark Window Model is illustrated in Fig. 6. This window model cannot be aware of time, and therefore, cannot distinguish between new and old data.

2.2.2. Damped Window Model.

To distinguish between the oldest and new transactions a variation of the Landmark Window Model was proposed and named Damped Window Model. Damped Window Model assigns different weights to transactions where the recent ones have weight near to 1, and older ones have weight near to 0. As time passes, the weight of each transaction will be degraded. Fig. 7 shows this model.

The Landmark and the Damped Windows Model compute the support between the landmark and the current time. There are some applications where the interest is focused on data recently arrived within a fixed time

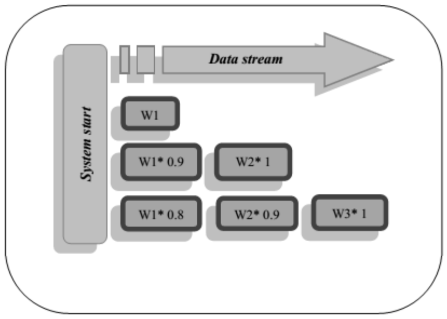


Figure 7: Damped Window Model is a variation of Landmark Window Model for data streams mining, where the recent transactions have more importance than the oldest ones.

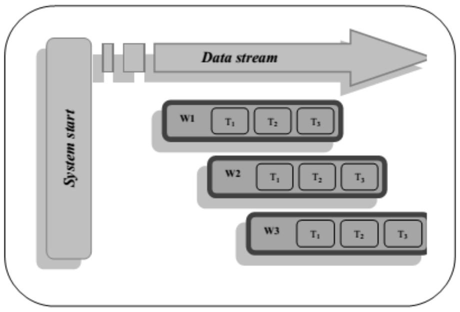


Figure 8: The Sliding Window Model only consider the latest transactions within a window of size W.

period, and these models can not be used in such applications.

2.2.3. Sliding Window Model.

In this model, given a window of size W only the latest W transactions are utilized in the mining process. As the new transactions arrives, the old ones in the sliding windows are excluded. The use of this model impose a restriction: as some transactions will be excluded of the mining process, methods for finding expired transactions and for discounting the support count of the itemsets involved are required. Fig. 8 shows this model. This model is based on the assumption that the number of frequent patterns is not particularly large and, therefore, it is possible to store the transactions in each sliding window in main memory.

Sliding window can be with overlapping or without overlapping. It is important to notice that the model to use depends on the application and/or the nature of the data streams.

2.3. Issues in Frequent Itemsets Mining.

The theoretical basis of frequent itemsets mining that has been presented can be applied to both databases and data streams scenarios. The major problem with frequent itemsets mining methods in general is the explosion of the number of results, so it is difficult to find the most attractive frequent itemsets. In both databases and data streams scenarios, usually the following features exist:

1. Elevate number of transactions.
2. Huge data volumes (order of gigabytes).
3. Many “data” and not enough “information”.
4. Limited computing resources.
5. Unpractical processing times.

Specifically in database scenarios, data can be read anytime while algorithms are executing. If data are modified, the algorithms can still access them as needed. The main issue in this scenario is concerned to the high number of items to handle (and therefore, memory and time consumption). Let be n the number of single items in a database, the number of candidates frequent itemsets is 2^n, or the same, this problem has computational complexity of $O(2^n)$. Handling this large amount of data is a challenging task, and strategies for efficient data access and data memory maintaining are needed [21].

As it was explained before, data streams are becoming more common, and they are more frequently used in many real life applications. Data streams (as it was previously stated in definition 6) can be defined as a continuous, ordered and potentially infinite sequence of items that occur in real time and also share the same limitations of frequent data streams on databases. New limitations are added to frequent itemsets mining problem if we consider the three characteristics of data streams (continuity, expiration and infinity):

1. It is impossible to store the stream for later processing.
2. Items in a data stream must be read and processed just once.
3. Items must be processed in an extremely short time interval.

Also, data streams are heterogeneous in format, content, rates, information and noise levels. These characteristics make the processing and analysis of data streams difficult. Data streams may also be composed by unstructured data types (audio, video, text) that cannot be easily handled using traditional approaches. Data streams involves an inherent temporal component; this is because the data evolves over time, and therefore, data streams mining exhibits temporal locality. Additionally there are multiple challenges associated with handling noisy, missing and uncertain or imperfect data present on data streams. These include techniques for data cleaning (with the mathematical background and processing models that are necessary to adopt), missing data handling algorithms and minimizing degradation of any derived result.

In other words, frequent itemsets mining on data streams can be seen as a particular case of frequent itemsets mining on databases that include extra challenges. Therefore, a straightforward translation of existing Data Mining algorithms may be inadequate for the stream mining processes. Frequent itemsets mining on data streams is an open research problem (see section 3.1 for further information).

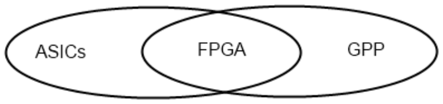


Figure 9: FPGAs combine the advantages of ASICs and GPP. It aims to fill the gap between hardware and software, achieving potentially much higher performance than software while maintaining a higher level of flexibility than hardware.

2.4. Platforms for algorithms implementation.

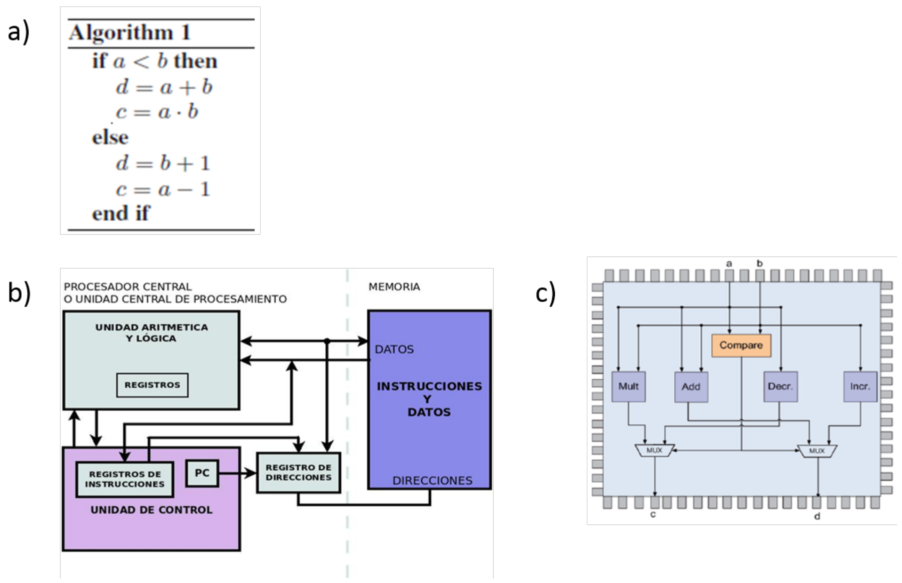
There are two main development platforms to implement algorithms: the first one are Application Specific Integrated Circuits (ASICs)[22]. ASICs are employed to perform a well-defined task, and; therefore, they are extremely fast and efficient. However, once ASICs have been built, they cannot be modified, being this sometimes a disadvantage. The second platform for the algorithms implementation isthe General Purpose Processors (GPP), where the software instructions are decomposed in a set of basic instructions that can be executed directly by the processor. In this approach, changing the software instructions implies a change in the algorithms behavior. GPPs provides high flexibility in algorithms execution, but the performance will be degraded. To execute certain functions, the GPP, first must read the instruction from memory and then decode their meaning into native GPP instructions to determine which actions must be done. The decoding process of the original instructions of an algorithm and the memory accesses introduce a delay into the execution of programs.

Just in the middle of both development platforms, Field-programmable gate arrays (FPGA) are the modern-day technology for building hardware prototypes. FPGAs combine the advantages of ASICs and GPP. FPGAs try to fill the gap between hardware and software, increasing the performance concerning of GPPs and maintaining a higher level of flexibility than hardware. Fig. 9 represents the position of FPGAs.

The implementation of algorithms in hardware can be divided into two branches:

- Reconfigurable Hardware Computing.
- GPU Computing.

1. Reconfigurable Hardware Computing is referred to the use of hardware devices in which the functionality of the logic gates is customizable at runtime, and FPGAs are the main exponent of this approach. The connections between the logic gates are also configurable. FPGAs appeared in 1984 as successors of the Complex Programmable Logic Devices (CPLDs). The architecture of a FPGAs is based on a large number of logic blocks which perform basic logic functions. Because of this, an FPGA can implement from a simple logical gate, to a complex mathematical function. FPGAs can be reprogrammed; that is, the circuit can be ”erased” and then, a new architecture that implements a brand new algorithm can be implemented. This capability of the FPGAs allows the creation of fully customized architectures, reducing cost and technological risks that are present in traditional circuits design. Fig. 10 shows a general diagram of a



```
Algorithm 1
  if $a < b$ then
    $d = a + b$
    $c = a \cdot b$
  else
    $d = b + 1$
    $c = a - 1$
  end if
```

Figure 10: Visual representation of the architecture of FPGAs and GPP, and the data flowing for some algorithm execution in software and hardware.

composition of a FPGA, and the comparison of how an algorithm is executed in hardware and software. The algorithm in Fig. 10a performs a comparison between two values a and b. If a is greater than b some processing is executed, and if b is equal or greater than a other processing is executed. Fig. 10b) shows the data flowing inside the software version of the algorithm, and it is evident that is more complex in terms of data access than executing in hardware. Fig. 10c) shows the parallel nature of FPGAs.

2. GPU Computing is based on the use of specialized processing platforms known as Graphics Processing Unit (GPU), which typically were originally designed to accelerate computer graphics processing, thus enhancing the performance of systems based on traditional GPPs. Recently, GPUs have gained the attention of the scientific community since they have been used as hardware accelerators for various non-graphics applications, such as scientific computation, matrices multiplications and distributed computing projects, among others. For additional information on the state-of-the-art GPU techniques, see [23].

Table 3 shows a comparison between different computer architectures reported in [24]. The data is reported per physical chip. The GPP is an Intel Core i7-965 Quad Extreme. The NVIDIA GPU is the Tesla C1060. The FPGA is the Virtex-6 SX475T.

From table 3 is derived that all architectures support parallel processing. Coding type separates those platforms into two groups: hardware-based and software-based coding techniques. The effectiveness of using some of those platforms is mainly determined by the programmer's aptitude. The device flexibility and errors corrections are serious issues. Except ASICs, all platforms are flexible, and all of them support error correction. Those facts explain why developers discard the ASICs as a viable candidate in algorithms implementation in

Table 3: Comparison between GPP, ASIC,FPGA and GPU. Symbol "-" means not measured

	GPP (or CPU)	ASIC	FPGA	GPU
Parallelism	Partially	Fully	Fully	Fully
Coding type	Software	Hardware	Hardware	Hardware
Flexibility	Yes	No	Yes	Yes
Error correction	Yes	No	Yes	Yes
Frequency (GHz)	3.2	< 0.55	< 0.55	1.3
Single Precision GFLOPS	102.4	-	550	936
Double Precision GFLOPS	51.2	-	137	78
Single Precision GFLOPS/Watt	0.8	-	13.7	5
Single Precision GFLOPS/USD	70	-	138	550

hardware. Considering the performance per dollar, the GPU outperforms GPP and FPGAs. FPGAs have the lowest frequency but the best performance per watt. With regard to the performance issue, applications typically exhibit vastly different performance characteristics depending on the platform. More about this can be found in Cullinan et al.[25].

Che et al. in [26] and Cullinan et al. in [25] compare GPUs and FPGAs. GPUs perform better when the dataflow exhibit no inter-dependencies and the required process can be done in parallel. The same performance is possible in FPGAs when the control structures has a lot detailed low level control structures which cannot be efficiently implemented in high level languages. If the memory accesses increase and parallelism is limited, the use of GPUs is not recommended. FPGAs are not recommended when they must deal with applications that require high complexity in logic and dataflow design. Most of the modules within a a GPP do not operatie on each cycle [27]. In comparison, usually most of the modules within a FPGAs are active in each clock cycle. FPGAs are more power efficient than GPP. Application systems on FPGAs are usually developed using VHDL or Verilog which usually require more design effort. As it is mentioned in [24], there is no one-size-fits-all architecture for scientific computing. GPP, GPU and FPGAs will be used in different scenarios depending on the characteristics of the algorithms.

3. Related work.

Hardware implementations of algorithms takes advantage of inner parallelism of hardware devices. In consequence hardware devices (such as GPUs and FPGAs) gain every day more attention to be used as development platforms. Frequent itemsets mining is formed by various algorithms (sequential and parallels) that can be

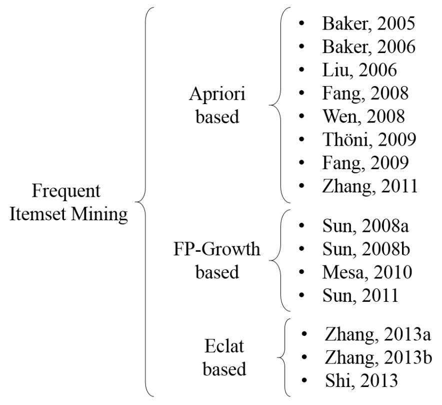


Figure 11: Frequent Pattern Mining taxonomy based on algorithm type that it is implemented.

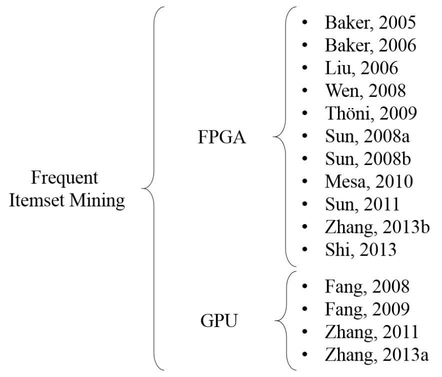


Figure 12: Frequent itemsets mining taxonomy based on hardware devices used in the reviewed literature.

implemented in hardware efficiently after a proper transformation.

In the revised literature, the most of algorithms for frequent itemsets mining are based in Apriori [3], FP-Growth [28] and Eclat [29]. Fig.11 shows a taxonomy of the differentworksthat compute frequent itemsets mining focused on the algorithms that they are based on.

Fig.11 shows that Apriori is preferred instead FP-Growth or Eclat. Such behavior could be explained by the fact that Apriori is easier to implement and older than FP-Growth and Eclat. Based on the hardware device that implements those architectures for frequent itemsets mining, the reviewed literature can be classified as is shown in Fig.12.

In Fig.12 we can notice that FPGAs is preferred to be used for implementing frequent itemsets mining tasks.

Fig.13 shows the trends of frequent itemsets mining on databases and data streams. As the figure shows, just one architecture is reported for mining frequent itemsets on data streams and several researches are referred to databases.

Using the presented taxonomies, the reviewed literature can be organized as it is shown in table 4.

Table 4: Principal algorithms and architectures for the frequent itemsets mining problem in hardware.

Title	**Author**	**Year**	**Device**	**Based**	**Source**
Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs. [4]	Baker	2005	GPU	Apriori	BD
An Architecture for Efficient Hardware Data Mining Using Reconfigurable Computing Systems. [5]	Baker	2006	FPGA	Apriori	BD
Hardware Enhanced Mining for Association Rules. [8]	Liu	2006	FPGA	Apriori	Stream
Parallel Data Mining on Graphic Processors. [30]	Fang	2008	GPU	Apriori	BD
Hardware-Enhanced Association Rules Mining With Hashing and Pipelining. [14]	Wen	2008	FPGA	Apriori	BD
Novel Strategies for Hardware Acceleration of Frequent Itemset Mining With the Apriori Algorithm. [1]	Thöni	2009	FPGA	Apriori	BD
Frequent Itemset Mining on Graphic Processors. [31]	Fang	2009	GPU	Apriori	BD
GPApriori: a GPU-accelerated Frequent Itemset Mining. [15]	Zhang	2011	GPU	Apriori	BD
Mining Association Rules with Systolic Trees. [10]	Sun	2008a	FPGA	FP-Growth	BD
A Reconfigurable Platform for Frequent Pattern Mining. [9]	Sun	2008b	FPGA	FP-Growth	BD
A Highly Parallel Algorithm for Frequent Itemset Mining. [16]	Mesa	2010	FPGA	FP-Growth	BD
Design and Analysis of a Reconfigurable Platform for Frequent Pattern Mining. [11]	Sun	2011	FPGA	FP-Growth	BD
Accelerating frequent itemset mining on graphics processing units. [32]	Zhang	2013a	GPU	Eclat	BD
An FPGA-Based Accelerator for Frequent Itemset Mining. [33]	Zhang	2013b	FPGA	Eclat	BD
FPGA Acceleration for Intersection Computation in Frequent Itemset Mining. [34]	Shi	2013	FPGA	Eclat	BD

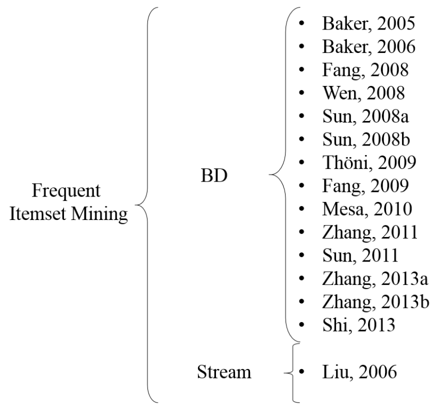


Figure 13: Frequent itemsets mining taxonomy based on data sources used by the hardware architectures.

3.1. Frequent itemsets mining based on hardware acceleration.

The following algorithms were designed to solve the frequent itemsets mining problem. Frequent itemsets mining in general is a more challenging task than frequent items mining. Only one of the revised papers was applied to data streams, which can be explained by the high computational complexity required for that task. In frequent itemsets mining, there are three main approaches: algorithms that use Apriori as the starting point, algorithms that use FP-Growth and those that use Eclat. Following the reviewed state-of-the-art for each algorithm is presented.

3.1.1. Apriori-based.

Apriori [3] is perhaps the most popular correlation-based Data Mining algorithm for the frequent itemsets mining problem. However, it is a computationally expensive algorithm and the running time can extend up to days for large databases in the order of gigabytes. Apriori uses the downward closure [3] property of itemsets support that any subset of frequent itemsets must also be frequent. Thus during each iteration of the algorithm's execution only the itemsets found to be frequent in the previous iteration are used to generate a new candidate set, C_k. Before inserting an itemset into C_k, Apriori test whether all its $(k-1)$-subsets are frequent. This pruning step can eliminate a lot of unnecessary candidates. To enable fast support counting, the candidates are stored in a hash tree. An internal node of the hash tree at depth d contains a hash table whose cells point to nodes at depth $d+1$. All the itemsets are stored in the leaves. The insertion procedure starts at the root and hashing on successive items, insert a candidate in a leaf. For counting C_k, for each transaction in the database, all k-subsets of the transaction a regenerated in lexicographical order. Each subset is searched in the hash tree, and the count of the candidate incremented if it matches the subset. This is the most compute intensive step of the algorithm.

Baker et al. were the first authors to study the problem of efficient hardware implementations of the Apriori algorithm [4], which mainly involves efficient implementation of the set membership functions efficiently as well as the algorithm control. Authors addressed these issues by using a hybrid systolic array[3]-microcontrolled data paths and efficient design principles. They also presented a strategy called *Systolic Injection*, a contribution to the general use of systolic arrays that can be used in many other applications. Through the use of the systolic array, the proposed architecture allows for increased frequency performance, decreased number of interconnection, and simple and easily scalable processing units.

The main issue with the Apriori algorithm is the data complexity. Each candidate must be compared against every transaction set. This results in a high running time for a single generation, $O(\|T\|\|C\|\|t\|)$(where T is a transaction and C are the items in the transaction T), assuming the subset function can be implemented in time $\|t\|$. However, the parallelism contained in the loops allows to explore hardware acceleration. In the proposed architecture, the computation units are fully interconnected, forming a chain (or linear array). Data flows in one direction and stall information flows in the opposite direction. Each unit contains memory locations to store the candidates whose support was calculated and to allow for temporal stalling. A processing unit is composed of the candidate memory, an index counter, and a comparator, which allows the output of the candidate memory to be compared with an incoming item.

In 2006 Baker and Prasanna proposed a new hardware architecture [5] for accelerating the frequent itemsets mining based on Apriori algorithm. They ran several experiments and realized that the behavior of the Apriori algorithm has certain characteristics that allow some redundancy between candidates to be extracted for use in hardware. The common candidates (that are stored into memories) in a transaction can be reutilized in later iterations by using a specific heuristic. This allows to keep the memories consumption low, and this is an extremely valuable characteristic. Also, experiments revealed that the performance bottleneck for the Apriori algorithm is determining if each candidate is a subset of each transaction sets (the support computing operation in the Apriori algorithm [3]). To do this, each candidate is compared against every transaction set resulting in an intense computing operation. Based on this fact, Baker focused his research on accelerating the support computing operation.

The main contribution in [5] is the design of a highly parallel custom architecture using Content Addressable Memories (CAMs) organized into a bitmap structure. The bitmapped CAMs emulate a linked list data structure in order to determine subset satisfaction for a large number of candidates simultaneously. Using CAMs, the time and area required for executing the subset operations fundamental to Data Mining can be significantly reduced. The success of this approach depends on the similarity of the candidates processed. However, determining the appropriate ratio number of CAM elements to candidates in a unit is not a trivial task. The experiments performed using industry-standard databases showed that the proposed architecture provides a minimum of 24x

[3]Systolic Array is a pipe network of processing blocks called cells. Systolic Arrays is a specialized parallel computing, where cells (i.e. processors) compute data and store it independently of each other.

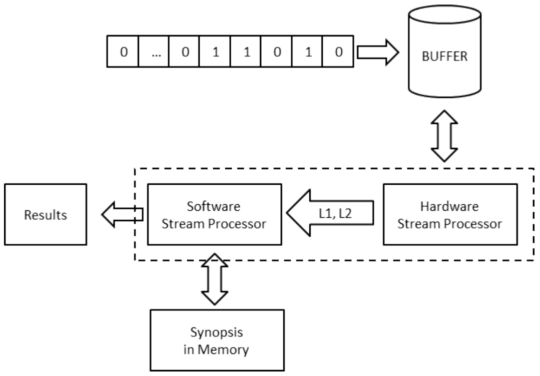


Figure 14: The “Receiving-storing-processing” approach proposed by Liu et al. in [8] for the frequent temporal patterns mining process.

(and often much higher) time performance advantage over the fastest software Apriori implementation [35].

Papers explained above were focused on processing databases. In the revised literature, only one paper that performs frequent itemsets mining in data streams was found. Liu et al. in [8] proposed a hardware-enhanced mining framework and an Apriori-like algorithm to mine frequent temporal patterns[4] from data streams. This architecture is specially designed to mine those itemsets of length 1 and 2 because the computing of L1- and L2-itemsets is the most time-consuming task in their algorithm, so they propose to offload this operation to hardware to improve the overall performance. They describe in their work the main issues that must be dealt with when data streams are processed which are those that were explained in section 2.3. According to the authors, the novelty in this hardware-enhanced approach resides in the transformation of the items transactions in a data streams into a matrix structure and the efficient mapping of operations for discovering frequent items into highly efficient hardware processing units. The “Receiving-storing-processing” approach shown in Fig.14 is used to perform the transformation of transactions of streams into matrix. Experiments on synthetic data set showed that the throughput is two orders of magnitudes larger than that of its software counterpart.

Fang et al. propose in 2008 a novel parallel Data Mining system called GPUMiner [30]. After a simple performance profiling of the Apriori algorithm, they concluded that a support counting is a hot spot in the Apriori execution, spending around 90% of the execution time in this function. As many others, Fang et al. rely their work on the massively multi-threaded SIMD architecture provided by GPUs, and the use of bitmap memories structures to improve the counting capability. The bitmapped memories make easier the data parallelism in the SIMD execution. Also, they enhance their implementation using memory optimizations and thread parallelism. The memory optimizations include the local memory optimization for temporal locality and the coalesced access optimization for spatial locality.

[4]Frequent temporal patterns are referred to those items or itemsets that are frequent in some time period.

GPUMiner is composed by three modules: a module to handle the IO data transfer between the GPU and the GPP efficiently; a GPU-GPP co-processing parallel mining module and a GPU-based visualization module. The most attractive module for the purpose of this PhD. Proposal is the second module: the k-means [36] and the Apriori algorithm that was implemented within this module. K-means is a largely parallelizable algorithm that has been hardware accelerated in many previous works. In their paper, Fang et al. focused in the bitmapped Apriori which represents the transactions in a bitmap structure and the counting operations were implemented using simple logical operations like *And*s and *Or*s.

To demonstrate the validity of GPUMiner's Apriori implementation, the FIMI'03 best implementation [35] was used as the baseline. Also, three FIMI'03 datasets were used (a small, medium, and large data sets used in FIMI'03). To prove the viability of the proposed bitmapped Apriori algorithm, two implementations were developed: a GPP and GPU implementation. Both implementations were faster than FIMI's: Apriori, achieving a speedup up to 10.4x and 7.5x, respectively when the support used is about 1%. Furthermore, the speedup of the bitmapped Apriori algorithm with the CPU or the GPU implementations, increases as the data size increases. As conclusion, both implementations of the proposed bitmapped Apriori algorithm are faster than the FIMI implementations throughout a range of support thresholds.

Apriori is also used in association rules mining. In such manner Wen et al. proposes HAPPI (HAsh-based and PiPelIned) architecture [14] for the association rules mining in hardware, using FPGAs. In the design of HAPPI architecture authors identified those functions of the mining process that are appropriate to be implemented in hardware. Also, authors incorporated the pipeline methodology to compare itemsets and gather useful information that allowed to reduce the number of candidate itemsets and items in the database simultaneously. HAPPI implements five operations: support counting, transaction trimming, hash table building, candidate generation, and candidate pruning (the sketch of Apriori is shown in those five operations). First of all, the database is fed into the hardware and, at the same time, the candidate itemsets are compared with the items in the database using a systolic array. Second, HAPPI determines the frequency on which each item occurs in the candidate itemsets in the transactions at the same time. Infrequent items can be eliminated using this information. Third, the itemsets from transactions are generated and hashed them into the hash table. The hash table is used to filter out unwanted candidates itemsets. After the hardware compares candidate itemsets with the items in the database, the trimming information is collected, and the hash table is built. Based on the trimming information, items are trimmed if their corresponding occurrence frequencies are not larger than the length of the current candidate itemsets. In addition, after the candidate itemsets are generated by merging frequent sub-itemsets, they are sent to the hash table filter. If the number of itemsets in the corresponding bucket of the hash table is less than the minimum support, the candidate itemsets are pruned.

In order to demonstrate the feasibility of HAPPI, some different experiments were conducted. The T10I4D100 dataset was used with different numbers of items in the database. The minimum support was set to 0.5 percent.

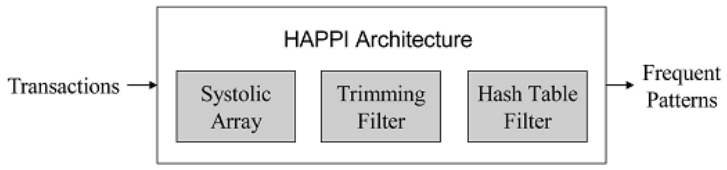


Figure 15: A sketch of HAPPI architecture.

Wen et al. also implemented an approach based on [4] (called Direct Comparison, DC) for comparison proposes and HAPPI outperforms DC in some orders of magnitude. The main improvement of the execution time results from the hash table filter and the pipelining strategy.

Another attempt to speed-up Apriori algorithm was conducted by Thöni et al. in [1]. Thöni et al. proposed an architecture that uses the block RAMs of a Virtex V FPGA device instead of using logic cells. They also present a strategy to store candidates, effectively shifting resource utilization from logic to block RAMs, thereby freeing up scarce logic resources for actual execution of logic operations. The proposed architecture is composed of a systolic array that comprises a set of data path units (or cells) sequentially connected by a parallel data and control bus. Sequential and control-intensive operations are left for implementation in a controller which can be realized in either hardware or software. In this way, the proposed architecture only addresses the support counting stage. To the correct performing of the architecture, memories depth must be chosen to equal or exceed the number of frequent items of any database being processed. This approach is unrealistic beyond out of some cases in scientific researches. To test the performance evaluation of the proposed architecture, authors used the same dataset that were used by Baker and Prasanna in [4] and [5]. Baker and Prasanna targeted their architecture to a Virtex II FPGA Device whereas Thöni el al. used a Virtex V FPGA device. As a direct comparison was unfair, Thöni compared both implementations in terms of area utilization and frequency of operation obtaining better results than those reported by Baker and Prasanna.

In 2009, Fang et al. proposed new architecture to speed-up frequent itemsets mining using an algorithm based in Apriori and a GPUs [31]. Authors developed two Apriori's implementations named *Pure Bitmap-based Implementation* (PBI) and *Trie-based Implementation* (TBI). As in other works, the advantage of the GPU's massively multi-threaded SIMD architecture is used as main acceleration source. Both implementations employed a bitmap data structure to exploit the GPU's SIMD parallelism and to accelerate the frequency counting operations which facilitates the fast set intersection to obtain transactions containing a particular itemset. The bitmap structure to store the occurrences of items in transactions was selected because it is more efficient to be partitioned and used with the SIMD processors. Besides, a lookup table was used to facilitate support counting since that operation is the most time-consuming component in the Apriori algorithm.

Both of PBI and TBI implementations follow the workflow of the original Apriori stated in [3]. First, PBI implementation features regular data access patterns, which are a best fit to the GPU; however, it may cause redundant computation and data access between frequent itemsets of different sizes. To solve such issue, TBI was implemented. TBI adopts a trie structure to represent itemsets and utilizes the CPU for trie traversal and incremental maintenance. Both implementations were evaluated using a synthetic and real-world datasets, and both implementations had proven to be up to two orders of magnitude faster than optimized CPU-based Apriori implementations. The time for data transfer between the GPU memory and the CPU memory, candidate generation, and support counting dominates the total running time.

Besides Fang's work, in 2011 Zhang et al. [15] proposed another architecture based on GPU to speed-up Apriori, taking advantage of SIMD architecture named GPApriori. Authors designed a new memory structure to represent the items transactions, named "static bitset". This data structure improves upon the traditional approach of the vertical data layout in state-of-the-art Apriori implementations. To improve the performance of Apriori, they parallelized the support counting on the GPU. GPApriori has two keystones:

- **Data representation.** Here, Zhang et al. proposed a new way to represent the input items transaction and they named it "static bitset". The bitset representation requires more memory space than the vertical transaction list representation but is more suitable for designing a parallel set join operation, which is better suited for GPUs. Joining two bit-represented transaction lists can be performed by a "bitwise and" operation between the two bit vectors.

- **Support counting.** In Apriori, support ratio is computed by multiple passing over the transactions database. This requires considerable binary searches and trie traversal which is not suitable to be exported to GPUs. The GPU support counting proposed by Zhang is based on complete intersection, where candidates are copied from main memory to graphic memory by host code, the GPU calculates their support ratio value by executing bitwise intersections on their vertical transaction lists, and the resulting support values are copied back to main memory. This data transfer into GPU and GPP could involve an execution delay in GPApriori. To improve this task, they perform the support counting using CUDA [37] and several optimization techniques (as candidates preloading, hand-tuned loop unrolling to improve the kernel speed and hand-tuned block size further) were introduced.

To prove the suitability of GPApriori, the Borgelt's Apriori [38] was used as baseline. Several experiments were conducted using UCI [39] dataset. The comparison of GPApriori and CPU-based showed that on the smaller dataset called Chess, the GPU version can achieve a 10X speed up while for the dataset Accident (which is larger than Chess dataset), the speed up ranges from 50X to 80X. In general, the performance scales with the size of the dataset. Experimental results show that GPApriori outperforms Borgelt's Apriori on the most of moderate sized datasets with 4X-10X speed up and on a large dataset Accident, the speed up ratio can reach up to 80X.

3.1.2. Partial Remarks.

The algorithms that mimic the Apriori-based schemes in hardware require loading the candidate itemsets and the database into the hardware. This strategy is limited by the capacity of the chosen platform: if the number of items to manage is larger than the hardware capacity the items must be loaded separately in many consecutive times degrading performance. In consequence, the support counting must be executed several times. Since the time complexity of those steps that need to place candidate itemsets or database items into the hardware is in proportion to the number of candidate itemsets and the number of items in the database, this approach is very time consuming. In addition, several candidate itemsets and a large database may cause a bottleneck in the system.

In the revised algorithms, researchers focused their contributions in technical issues rather than theoretical enhancements. Basically, the reported improvements are listed as follow:

1. Enhanced data structures and new strategies for data partitioning.
2. Parallelize certain functions of algorithms.

Apriori-based algorithms require many passes over the database. This is forbidden in data streams mining due to the Expiration restriction (see section 2.2 for further information).

3.1.3. FP-Growth-based.

Another approach to process frequent itemsets is using a tree-based database of transactions representation. One of the most successfully exponent of this approach is FP-Growth algorithm [28]. FP-Growth is one of the fastest and efficient algorithms reported for frequent itemsets mining. It has been implemented in several ways, including sequential and parallel approaches. FP-Growth is based on a prefix tree representation of the given database of transactions (called FP-tree). The FP-tree representation allows saving a considerable amount of memory for storing the transactions. In FP-tree representation, every transaction is stored as a string in a trie along with its frequency. Fig.16 shows the FP-tree presentation for transactions shown in table 5.

In the building process of FP-tree data structure if one transaction is a prefix of another transaction, it will share the same path in the FP-tree (the counter of such transaction will be increased). The maximum number of children in FP-tree will be, in the worst case, 2^n, where n is the number of items in the database.

FP-Growth performs two passes over databases: in the first one the a FP-tree is built up. The frequent itemsets are generated in the second pass using a recursive elimination scheme, where all the items that do not exceed a defined threshold will be deleted.

In 2008 Song Sun and Joseph Zambreno proposed an architecture [10] to speed up the association rules mining process based on FP-Growth. To emulate the FP-tree data structure they proposed a new hardware structure

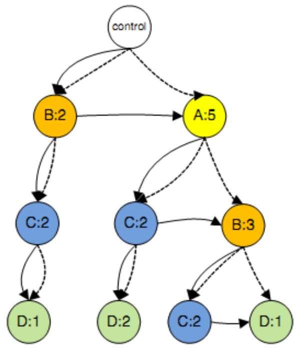


Figure 16: The FP-tree data structure.

Table 5: An example transaction database. Each row in the table corresponds to a transaction.

ID	Items
1	B, C, D
2	B, C
3	A, C, D
4	A, C, D
5	A, B, C
6	A, B, C
7	A, B, D

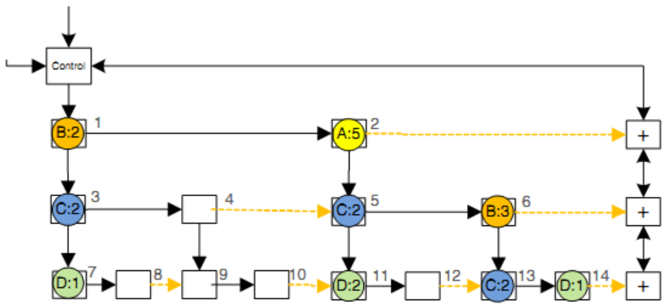


Figure 17: Systolic tree architecture that emulates the FP-tree data structure represented in Fig. 16.

named *systolic tree*[5]. A systolic tree could be seen as a tree where each node is a processing unit which has their own logic. Fig.17 shows the systolic tree built for the FP-tree represented in Fig.16.

The main difference between the FP-tree and systolic tree structures is that the systolic tree has a control node which governs their behavior. All inputs and outputs of the systolic tree pass through and they are controlled by the control node. Systolic tree structure contains three different nodes (or processing elements PE) types: the Root PE, which controls the architecture; the Counting PE which performs counting and calculating; and the General PE. Each General PE has one input from its parent and two outputs (to its child and siblings respectively). The main idea implemented in this architecture is to build a lexicographic tree while items flow through the systolic tree. When the complete database is mapped into a systolic tree, each PE will contain the frequency of the respective item.

There are three processing modes for the PEs: Write mode, Scan mode and Count mode. In the Write mode, the systolic tree is created and initialized. The Scan mode verifies if an itemset is frequent and Count mode return all the frequent itemsets. The systolic tree architecture was simulated and synthesized, and to prove their feasibility a series of experiments were run based on the simulation results and FP-Growth algorithm. Experiments demonstrate that the run time of the systolic tree implementation depends only on the number of frequent items while the run time of FP-Growth is closely related to the size of the FP-tree. When the number of frequent items in the systolic tree is greater than 11 then the performance is worse than FP-Growth. When the size of the systolic tree is 10, the mining speed is 24 times faster than FP-Growth. The throughput obtained in simulation was around 3Gbps.

Also in 2008, Sun and Zambreno propose a new hardware architecture for frequent itemsets mining using a systolic tree [9]. Similar to [10], the goal of this architecture is to emulate the original FP-Growth algorithm while achieving a much higher throughput. The main contribution in this paper is that Sung et al. modified the original scheme introduced in [10] by eliminating the counting nodes (see Fig.18), and provide a new count

[5]In VLSI terminology, a systolic tree is an arrangement of pipelined processing elements in a multi-dimensional tree pattern.

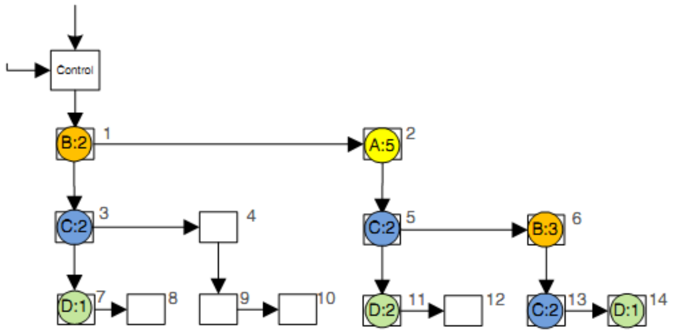


Figure 18: Enhanced systolic tree architecture that emulates the FP-tree data structure represented in Fig. 16. Notice that the Counting PE was eliminated in this representation.

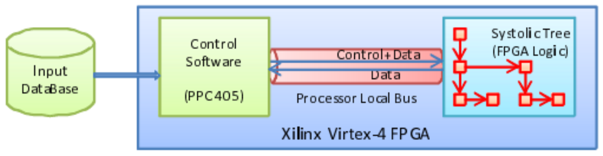


Figure 19: Hardware-software architecture for frequent itemsets mining using systolic trees proposed by Sun in [9].

mode algorithm.

Sun et al. implemented their architecture using a software-hardware coprocessing approach. Following the FP-Growth processing scheme, two scans of the database are required. In the first scan, both the set of frequent items and the support count of each frequent item are computed. This task is implemented in the software component of the system as shown in Fig.19. To compute the frequent itemsets, each candidate frequent itemset is passed by the software module into the systolic tree using the Processor Local Bus (PLB) which is a high speed and bandwidth bus. The systolic tree then reports the support count of the itemset back to the software module.

Several experiments were carried out to prove the feasibility of the proposed architecture. As baseline, a java implementation of FP-Growth algorithm was used over three benchmark datasets (Accident, Chess and Retail, taken from UCI Repository [39]). The projection and partition of the database were necessary in some cases. Experiments demonstrate that, in the worst case, the proposed architecture outperform FP-Growth in almost 12x.

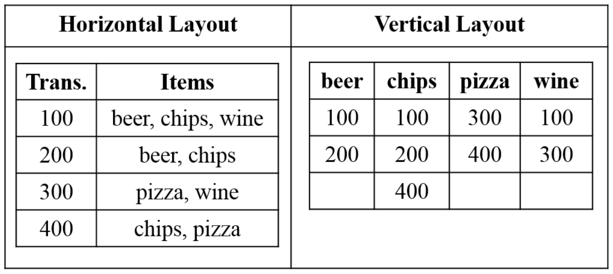
In 2010 Mesa et al. [16] proposed a novel architecture that used FP-Growth as the starting point. They proposed a vertical bit vector data layout to represent items and transactions. This layout allows to calculate the support by using logical AND and OR operations. Mesa et al. defined a two-dimensional matrix to store the database where the columns represent the elements of the dataset and the rows represent the transactions. Using such data representation, Mesa et al. proposed an algorithm that performs a search over the solution space through the equivalence class. This search presumes a lexicographical order over the items. This is a two-dimensional search (bottom-down, right-left) both breadth and depth are performed concurrently on the FPGA. The proposed algorithm does not need a candidate generation stage, and it uses a binary tree structure of processing elements. This structure represents a systolic tree, and it was chosen because it allows increasing the concurrent operations at each processing step exponentially. The size of the proposed architecture grows according to the number of frequent items that it is capable to handle. For the chosen hardware device, only 11 items can be processed. Experiments demonstrate that the proposed architecture outperforms [9] almost in one order of magnitude. Also, the architecture performs better when the density of the database and the number of frequent itemsets increases.

In [9], Sun et al. explained in a detailed manner the systolic tree architecture and their approach [11]. In this paper, the authors described the same idea that in [9], but they extended their paper with more detailed explanations about systolic tree and the working modes of their architecture. Also, new experiments were performed demonstrating that the systolic tree architecture can outperform best known FP-Growth implementation [40]. The performance measure in those experiments was the execution time of the whole systems on the benchmarks with different support thresholds. The execution time only included the time for disk reading and memory IO but excluded the disk writing time. The baseline FP-Growth algorithm was implemented in C++, and it was taken from [40]. Experiments demonstrate that systolic tree is a valid architecture to mine frequent patterns. For those datasets that have sizes who can be placed directly in the device, the systolic tree architecture always outperforms FP-Growth. In such cases that the dataset could not be placed directly in the FPGA, a dataset projection must be used and chosen. In this work, Sun et al. proposed one projection dataset strategy and prove their feasibility. In such cases, FP-Growth outperforms systolic tree structure. This behavior is caused by the overhead introduced by the projection strategy for database transaction mapping over FPGA. If the overhead is not amortized by the run-time reduction, the systolic tree algorithm is slower than the original FP-Growth algorithm.

3.1.4. Partial Remarks.

As well as Apriori-based algorithm, the FP-Growth-based algorithms need to download the mining data base to FPGA. They also need two passes over the data base except Mesa et al. [16] but this one still need to download the data base to the hardware device. This is impractical in data stream mining scenario due to the Expiration restriction listed in section 2.2. Like others reviewed algorithms, authors focused their attention in better data structures rather than substantial theoretical contributions.

As rule, FP-Growth based algorithm can handle a limited number of itemsets, less than 11 in the better [16] cases which is inadequate for real-life applications. Although strategies for device re-utilization are proposed, no concluding results of timing, throughput or performance are presented. Nevertheless, those algorithms based on FP-Growth use the FP-Tree data structure which is very well suited for data stream mining applications.



Horizontal Layout

Trans.	Items
100	beer, chips, wine
200	beer, chips
300	pizza, wine
400	chips, pizza

Vertical Layout

beer	chips	pizza	wine
100	100	300	100
200	200	400	300
	400		

Figure 20: Horizontal and Vertical database layout of table 1.

3.1.5. Eclat-based.

Eclat[29] is the first algorithm for the frequent itemset mining that uses a depth-first search using an intersection set. Also, Eclat is the first that uses a vertical database representation instead of listing all transactions. In this representation, each item is stored together with its support and uses an intersection based approach to compute the support of itemsets. In consequence, the support of an itemset X can be determined intersecting the supports of any two subsets $Y, Z \subseteq X$ such that $Y \bigcup Z = X$. This type of representation is especially beneficial when is used with depth-first search candidate generation because reduces the memory requirements and processing time. Also, vertical representation is especially useful in hardware because the intersection process can be performed using only a bit-wise AND operation, which is organic to be performed in hardware. Fig. 20 shows the vertical and horizontal layout of table 1.

Eclat handles a huge number of itemsets, but these itemsets were not generated completely in breadth-first mode. Here arise the main issue in Eclat: the processed itemsets can not be pruned because the algorithm does not have all frequent itemsets for the same level at each iteration. Eclat was studied in [38] and there was shown that the occupancy of sorted-set intersection during the computation will increase rapidly while the support value is decreasing. Because of that the speed of intersection of two sorted-sets is the key point affecting the performance of Eclat. Nevertheless, there have been proposed efficient implementations of Eclat who solved this issue, e.g. dExclat [41]. Due to the descendent strategy and the search tree characteristics, Eclat can find previous maximal itemsets [6] which they will be supersets of some others where the traverse will be in descendent order. Using this feature unnecessary traverse will be avoid, and this is the starting point of more efficient implementations [38].

In 2013, approaches to perform frequent itemsets mining based on Eclat were proposed. Zhang et al. [32] proposed a new algorithm named *Frontier Expansion* which uses a hardware-software co-design with a GPU as coprocessor. Frontier Expansion uses Eclat as a starting point and therefore, uses a vertical-bit database representation and using data parallelism. As it was mentioned, this type of data representation is especially

[6]An itemset X is maximal if it is not subset of any other known frequent itemset.

useful for hardware designs. On the proposed co-design, which implement the Frontier Expansion algorithm, the software on the host GPP dynamically controls the search boundary and the expansion rate on behalf of the GPU kernel. This allows for the efficient utilization of the computational and memory resources of the GPU co-processor. Zhang et al. also redesigned and optimized the GPU's memory allocation strategy and added a data preprocessing procedure to improve the memory utilization of the algorithm. The proposed co-design uses a complex data structures which are controlled by software in host GPP while the hardware in the GPU performs the intersections (logical *AND* operations) of the candidates transactions vectors sent by the software.

The key contributions of this work, according to the authors are:

- The implementation of a parallel GPU kernel for calculating the support using the vertical data representation.
- A dynamic frontier expansion approach that maps candidates to block of GPU while enforcing a tight bound on memory usage.
- An approach for scaling to multiples GPUs.

Also, it can added:

- This work is the first that propose and implement an algorithm based on an approach different of Apriori and FP-Growth, used so far in the revised literature.

After the proper implementation of the Frontier Expansion algorithm (which is available here[7]), several experiments were conducted using state-of-the-art Eclat [38] and FP-Growth implemented by Goethal et al. in [42] as baseline. Experiments demonstrate that the proposed Frontier Expansion algorithm obtains better performance than state-of-the art Eclat and FP-Growth for dense datasets with low support value, which is the circumstance under which sequential algorithms become expensive. This is because lower support ratios result in an extra execution time required for support counting (GPU kernel), which is highly parallelized. The obtained results indicate that the proposed approach can achieve up to 30 speedup over state-of-the-art CPU-only serial implementations.

Although the Frontier Expansion algorithm improves the performance of ECLAT, it cannot be used for mining data streams. Frontier Expansion uses a vertical database representation, where the database should be stored before the processing start. Data stream management systems only can access current transaction, and therefore algorithms that use vertical database representation can not be implemented straightforwardly to mine data streams. Some buffering strategies can be used to get an excerpt of stream and handle it as a database to be used by Frontier Expansion, but this will introduces unwanted delays in overall execution time.

[7]http://tachyon.cse.sc.edu/gpufim.html

Zhang et al. presented [33], where is described a FPGA-based coprocessor architecture for Eclat algorithm. Unlike other attempts to implement FIM algorithms in hardware, which use performance-limiting strategies such as iterative database loading and runtime logic unit reconfiguration, the proposed architecture does not impose limits on the maximum set size as a function of available FPGA logic resources. Besides, like original Eclat algorithm, here is used the vertical database representation. According to the authors, the main contributions of this paper are:

- Is the first FPGA implementation of the Eclat FIM algorithm.
- The proposed accelerator can handle large datasets without iterative database loading or logic reconfiguration required by previous FPGA FIM designs.
- The proposed accelerator design employs a novel on-chip caching and associated data compression technique to reduce frequency of off-chip accesses for intermediate results.
- Their design is scalable to multiple FPGAs.

In previous papers, it was evident that the systolic arrays-based designs introduce serious limitations (concerning the memory consumptions and multiples passes over database) for mining large databases. To avoid this issue, Zhang et al. chose to take a radically different approach. Instead of using a systolic array, they implemented their processing element consisting of a controller unit (implemented by a finite state machine connected multiple on chip memories. The Eclat algorithm is executed in the controller unit which performs a depth-first search using an on-chip stack, generating its own sequence of addresses for accessing off-chip memory when the support counting is calculated (this emulates the traverses over the search tree of Eclat).

To prove the performance of their design, 4 FPGAs (where each FPGA uses two 2GB of SODIMM and one 256 MB DDR2 onboard DRAM) were interconnected. This offer 15GB of total memory for the entire system allowing to handle large databases without iterative loadings. The Eclat implemented cited in [38] was used on a Dell PowerEdge R710 server as baseline. The server contains two Intel Nehalem Xeon 5520 CPUs and each CPU runs at 3.2GHz, and the system memory is 16 GB. They used 4 synthetic datasets with 500K and 1000K transactions and 300 - 500 items in each transaction. Due to limitations of the memory controller, the maximum word length that can it handle is 256. The 256-bit processing element achieves a speedup of 4 to 5x compared with the baseline, and the 128-bit PE achieves a speedup of 23x. The entire system achieves a speedup of 30 to 40x compared to the baseline. Also, Zhang et al. used the OpenMP framework described in their previous work [43], to compare their result with a 16-thread X86 implementation. The speedup against 16-thread parallel version ranges between 4.71 and 7.28x.

The proposed design employs off-chip memories to store the results. The communication time is a major issue in this type of designs, and it was not reported. As previous work based on Eclat, this approach can not be applied straightforwardly to data stream scenarios due to the database representation. Using the vertical database representation is needed to store all database in the device. This does not fulfill the restriction of Expiration and Infinity of section 2.2.

Also using the Eclat algorithm as a starting point, Shi et al. [34] proposed an accelerator for certain functions of Eclat. They performed an execution profile of original Eclat cited in [38] to analyze its features and get its execution time in details. The test tool used was Intel Vtune Amplifier 2013, and the test platform was Intel Core I7 920 Quadcore processor with 3.0GHz. The test data adopted was T40I10D100K (15MB), which has an average of 40 elements in a transaction over 100,000 entries. The test results demonstrate that the intersection computation is the most time consumption part in Eclat, and the percentage of intersection computation time increases with the support value decreases (range in 70% for 5% of support and 93% for 0.5% of support). Accelerating the intersection computation with parallel approach will reduce the overall execution time of Eclat algorithm.

To accelerate the intersection computation, the authors propose an hybrid approach between software (on GPP) and hardware (on FPGA) where software executes the control and simple functions and hardware executes the intersection computation. To accomplish that, a full comparison matrix structure to perform the parallel intersecting of two groups of data elements was proposed. In addition, a data generator to control the index updating strategy was proposed. This data generator control guarantees the right load of the elements to be intersected. The proposed approach can be describe as it is stated in the following enumeration:

1. Parallel load the data elements of two different sorted set from memory modules to two data vectors.
2. Send the these two vectors of data element to full comparison unit (in hardware) to find the common values, and update the index according the last data in vector;
3. Write back the common data to the memory modules.

In order to compare the performance of software, the T40I10D100K data set was adopted to evaluate the proposed hardware architecture. The selected hardware device was the Xilinx ML605 board which includes a Virtex LX240T-1 FPGA, cannot provide enough input data bandwidth for the right database handling, so the results were obtained by simulation. The experiment results show that the proposed solution can achieve 6x to 26.7x speedup under different support value for intersection computation in Eclat algorithm. This comparison was performed between the original Eclat algorithm in software and the original Eclat algorithm using the hardware accelerator. Due to the vertical database representation, this approach can not be neither used to process data stream

3.1.6. Partial Remarks.

Eclat-based algorithm uses the vertical database representation in order to save memory and processing time. This representation is adequate to be used in hardware design. It use the intersection of items to compute the support, and it is more efficient than hash-trees. All the Eclat-based implementations propose an hybrid approach, where the most consuming functions were download to hardware while software controls the execution flow and data structures.

Due to the bandwidth limitations of used hardware devices, very large transactions must be segmented. In the reviewed papers, no segmentation strategies were reported.

Although the vertical database representation allows to save memory and processing time, it is not compatible with the Expiration restriction as it was explained before. Also, the pruning strategy in Eclat is inefficient and introduces delays that affect the performance of the algorithms. This two issues make Eclat impractical to be used as a starting point for data stream mining algorithms.

4. Methodological foundations.

In this section the methodological formalization of the current research is presented.

4.1. Research Problem.

In recent times, it is evident that the classical paradigms for knowledge acquisition are not suitable to obtain information from modern data sources. Databases are growing exponentially, and it is necessary to create new methods and algorithms that can handle those huge data amounts. With the apparition of multicore processing and distributed computing, parallel algorithms are gaining more attention, and they have been used to deal with this growing data trend. Modern applications, such as web click stream, telephone records, network traffic analysis, network intrusion detection, retail market analysis, stock market prediction and sensor networks generate huge data volumes represented as streams. Due to the increase of this kind of applications it is necessary to obtain useful knowledge from those data streams. As it was previously defined, data streams are a continuous, ordered and potentially infinite sequence of items in real time where data arrives without interruptions at a high speed. Also, data can be accessed just once, and the only assumption that can be made about bounds of streams is that the total number of data is unbounded. It is unrealistic to store all items of data streams to process them offline. These characteristics impose extra difficulties to algorithms and systems that process data streams.

Due to the high incoming rate, the impossibility to store the data and the huge volumes of items in streams, software that analyzes such data streams can not process exhaustively all items. The supporting hardware and software are not capable to deal with such intense processing. Instead, commercial applications that mine data stream use an “approximate” processing approach. That is, they do not analyze all items that are present in a flow; instead, they use some heuristic or probabilistic approach to determine which item is the most likely to contain the desired information. There are applications that need intense processing requirements, e.g. intrusion detection systems or network analysis systems. In this kind of applications, the immediate data analysis and near-real-time response are extremely valuable. To fulfill these requirements it is needed to propose new parallel algorithms running on high-performance computing devices such as FPGAs. FPGAs can perform tasks in a high parallel fashion, and this is very useful in data streams processing applications.

Frequent itemsets mining is one technique that is commonly used in data knowledge extraction and have been used with success in databases scenario. To mine frequent itemsets in data streams efficiently, an alternative would be to develop new parallel approaches that use custom hardware architectures. In the reviewed literature, there is only one architecture to mine frequent itemsets on data streams.

Summarizing, data streams are a modern data source that are gaining interest in recent applications. Traditional approaches for frequent itemset mining are not suitable to be used in data streams, and this situation introduces new challenges to this task.

4.2. Research Questions.

Based on the research problem exposed before then arise the following research questions:

1. How to perform the frequent itemsets mining on data streams efficiently and effectively outperforming the state-of-the-art algorithms?
2. Which features of frequent itemsets mining process and data streams must be taken into account to develop new efficient and effective algorithms for these tasks?
3. How to adapt the state-of-the-art algorithms to perform frequent itemset mining on data streams?
4. Which of the reported data structures can be used to mine frequent itemsets on data streams? How to improve such data structures?
5. Which development platform is better suited to implement algorithms for mining frequent itemsets in data streams?

4.3. General Aims.

The general aim of this research work is:

To develop parallel methods for frequent itemsets mining in data streams that outperform the state-of-the-art algorithms for data streams analysis and that are suitable for being implemented in hardware-accelerated platforms. The proposed methods must outperform in one order of magnitude (at least) the state-of-art algorithms implemented in software.

4.4. Specific Aims.

1. To propose a flexible method for separating the incoming data stream into windows that it can be used by the support counting algorithm.
2. To adopt data structures that can be used in frequent itemsets mining on data streams.
3. To develop new algorithms for frequent itemsets mining that use the separation method selected and the data structures adopted.

4. To obtain parallel hardware implementation of the algorithms mentioned above that can perform frequent itemsets mining at least 1 order of magnitude faster (without compromising effectively) than state-of-the-art software implementations.

4.5. Expected Contributions.

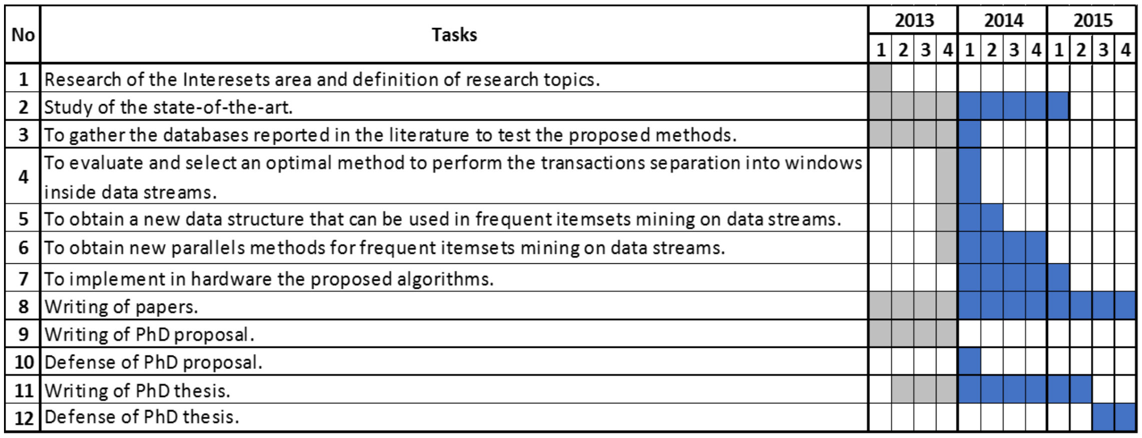
The expected contributions of this proposal are listed as follows:

1. A new method for frequent itemsets mining on data streams.
2. A design of parallel one-pass algorithms to mine frequent itemsets on data streams.
3. A custom hardware architecture that implements the proposed algorithms. This custom architecture will take advantage of inner parallelism provided by the hardware device used in its implementation.

4.6. Methodology.

To this research, according to the stated aims, the following phases are defined:

1. To gather the databases reported in the literature to test the proposed methods.
2. To evaluate methods to perform the transactions separation into windows inside data streams.
   (a) To study the methods reported in the literature to separate transactions into windows inside data streams.
   (b) To identify on the studied methods which ones enhance the performance of the reported algorithms.
   (c) To select one (or many) of the separation method that can be used in the new approach that will be proposed.
3. To select a data structure that can be used in frequent itemsets mining on data streams.
   (a) To study different data structures reported to store the itemsets on data streams.
   (b) Select those data structures that can be used by the new method that will be proposed.
   (c) Propose a new approach for itemsets representation based on selected data structures.
   (d) Implement in hardware the proposed data structures and evaluate its performance.
   (e) Analyze the experimental results obtained and identify limitations or elements that can affect the performance of the proposed data structures. If such issues are detected, thenmust be corrected.
4. To develop new algorithms for frequent itemsets mining that use the separation method selected in phase 2.c and the data structures proposed in phase 3.c.
   (a) To propose new algorithms that use the separation method and data structures in phases 2.c and 3.c. These algorithms must be designed to be executed in parallel.
   (b) To analyze the computational complexity of the proposed algorithms.
   (c) To implement in software the proposed algorithms to verify their behavior.



No	Tasks	2013				2014				2015			
		1	2	3	4	1	2	3	4	1	2	3	4
1	Research of the Interesets area and definition of research topics.	▨											
2	Study of the state-of-the-art.	▨	▨	▨	▨	🟦	🟦	🟦	🟦	🟦			
3	To gather the databases reported in the literature to test the proposed methods.	▨	▨	▨	▨	🟦							
4	To evaluate and select an optimal method to perform the transactions separation into windows inside data streams.				▨	🟦							
5	To obtain a new data structure that can be used in frequent itemsets mining on data streams.				▨	🟦	🟦						
6	To obtain new parallels methods for frequent itemsets mining on data streams.				▨	🟦	🟦	🟦	🟦				
7	To implement in hardware the proposed algorithms.					🟦	🟦	🟦	🟦	🟦			
8	Writing of papers.	▨	▨	▨	▨	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
9	Writing of PhD proposal.	▨	▨	▨	▨								
10	Defense of PhD proposal.					🟦							
11	Writing of PhD thesis.		▨	▨	▨	🟦	🟦	🟦	🟦	🟦	🟦		
12	Defense of PhD thesis.											🟦	🟦

Figure 21: Proposed task schedule.

(d) To compare the obtained results by software with the results reported in literature to verify inconsistencies.

(e) To analyze the experimental results obtained and identify limitations or elements that can affect the performance of the proposed algorithms. If such issues are detected, thenmust be corrected.

5. To obtain a parallel hardware implementation of the methods proposed for frequent itemsets mining on data streams.

    (a) To implement in hardware the proposed methods in phases 2, 3 and 4.

    (b) To validate the correctness of the algorithms implemented in phase 5.a.

    (c) To analyze the experimental results obtained and identify limitations or elements that can affect the performance of the proposed algorithms. If such issues are detected, thenmust be corrected.

4.7. Work Plan.

Fig. 21 shows a schedule of major activities to be undertaken for this proposal.

5. Preliminary Results.

In this section, the initial considerations taken in account to propose a solution to the investigation problem presented in section 4.1 are presented. Some restrictions are identified and presented, as well as the preprocessing strategy to transform the incoming data stream into a form that can be handled by the proposed algorithms. Also in this section the adopted data structure is presented and explained. After these mandatory initial considerations, the proposed method is presented.

5.1. Restrictions.

As it was explained in section 2.3, data streams are a challenging data source. To mine frequent itemsets on data streams, some restrictions must be imposed:

- Transactions and items must be delimited with some specials (and different) symbols or characters.
- The duplicate items (if exist) in transactions must be removed.

5.2. Preprocessing streams.

To accomplish with the second restriction, data preprocessing must be performed as the initial stage in the proposed method. When the stream is received, all spaces between items must be removed. Also, all of the duplicated items (if exist) must be removed. Also, all items in data stream that not fulfill the restrictions adopted will be removed from the stream.

5.3. Transactions separation strategy.

The window model selection is crucial in order to determine which items can be regarded as frequent. In practice, the window model establishes how to separate a data stream into portions named *windows*. It is inside a window where an itemset can be regarded as frequent. The proposed method gets as an input the window model to use in the mining process. This allows researchers to determine which model is better suited to be used in a particular problem. After the literature was reviewed, the conclusion observed is that the selected window model should not be an issue. The proposed method, and therefore, the hardware designs derived, must work fine regardless of the window model selected.

5.4. Prefix tree data representation.

Due to the issues of data stream mining presented in section 2.3, a single-pass method is required to mine efficiently data streams. The method presented in this PhD. Proposal adopts the Frequent Pattern tree (FP-tree) approach presented on [9], which is based on [28]. The basic idea is to develop a tree structure of processing units where the itemsets of data streams flow from the root node to leaf nodes. Although the FP-tree structure is similar to the baseline algorithm, the logic implemented in each node is different.

The FP-tree structure is represented as a tree (named *systolic tree*) where each node has one child and one sibling. For leaf nodes, the child and sibling nodes are null nodes. In this structure, the child node contains, as a prefix, the itemset handled by its parent. Fig. 22 represents the FP-tree structure used.

In Fig. 22, the root node is represented by the label a, where its sibling is the node represented by the label b. The child node of a is represented by the label $\underline{a}b$, and this node shares the prefix a of its parent. The same happens with the sibling node of $\underline{a}b$ (the node $\underline{a}c$). Nodes $\underline{a}b$ and $\underline{a}c$ compose the *Equivalent class* of node a. Fig. 23 shows the concept of Equivalent Class.

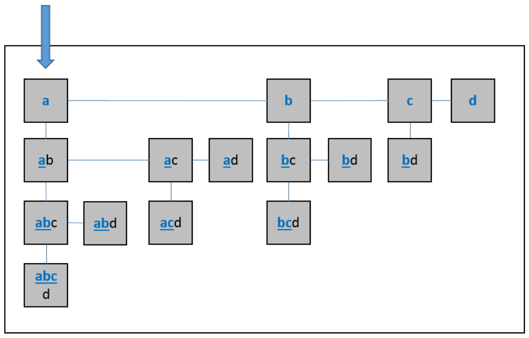


Figure 22: FP-tree structure to mine frequent itemsets on data streams.

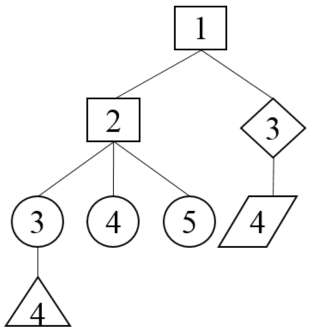


Figure 23: Equivalent class for items 1, 2, 3, 4, and 5.

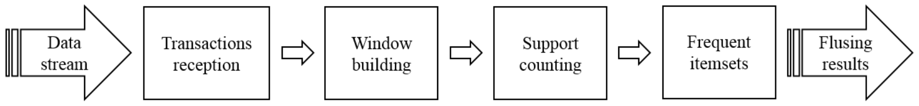


Figure 24: Behavior of the proposed algorithm.

Equivalent class can be defined by a set of items of the same size, assumed k, sharing the common $k-1$ prefix. For example, itemsets $(1,2,3),(1,2,4)$ and $(1,2,5)$ are in the same equivalent class $(1,2,-)$. $(1,2,3)$ and $(1,3,4)$ are not in the same equivalent class because they have the different 2-prefix $(1,2)$ and $(1,2)$. $(1,2,3,4)$ and $(1,2,3)$ are not in the same equivalent class because they have different size. All 1-item sets are in the same equivalent class.

Using the Apriori property, which states that any subset of frequent itemset must be frequent [3] if a node is regarded as frequent then its parent is frequent too with equal or greater frequency counting. This property is specially useful for the flushing strategy implemented in Algorithm 3 and explained in section 5.5.

The size (in number of nodes) of the systolic tree is determined by the maximum length of the itemsets in the incoming transactions that it must process, but as this can not be established a priori, 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, the size of the systolic tree will be:

$$nodes = 2^n - 1 \tag{3}$$

Section 5.6 shows the experiments conducted to determine the behavior of the systolic tree while the length of the incoming itemsets grows.

Nodes in the systolic tree have its own processing logic, that is presented in Algorithm 2. Previous approaches [9, 10, 16] proposed architectures with a single control unit that governs all the algorithm's behavior, and multiples types of processing elements. The systolic tree presented in this PhD. Proposal implements a distributed control scheme: the processing and control logic are distributed in each node of the systolic tree. This allows saving computational resources due to the logic reduction.

5.5. Description of the proposed method.

The proposed method is formed by 3 algorithms. The diagram in Fig. 24 describes the 4 modules that compose the frequent itemset mining on data stream approach presented in this PhD. Proposal.

1. A module that receives and preprocesses the current transaction in data stream. In this module, the itemsets are transformed into a form that fulfills the restrictions indicated in section 5.1 (named *Transactions reception*).
2. A module for window building according to the selected window model (named *Window building*).

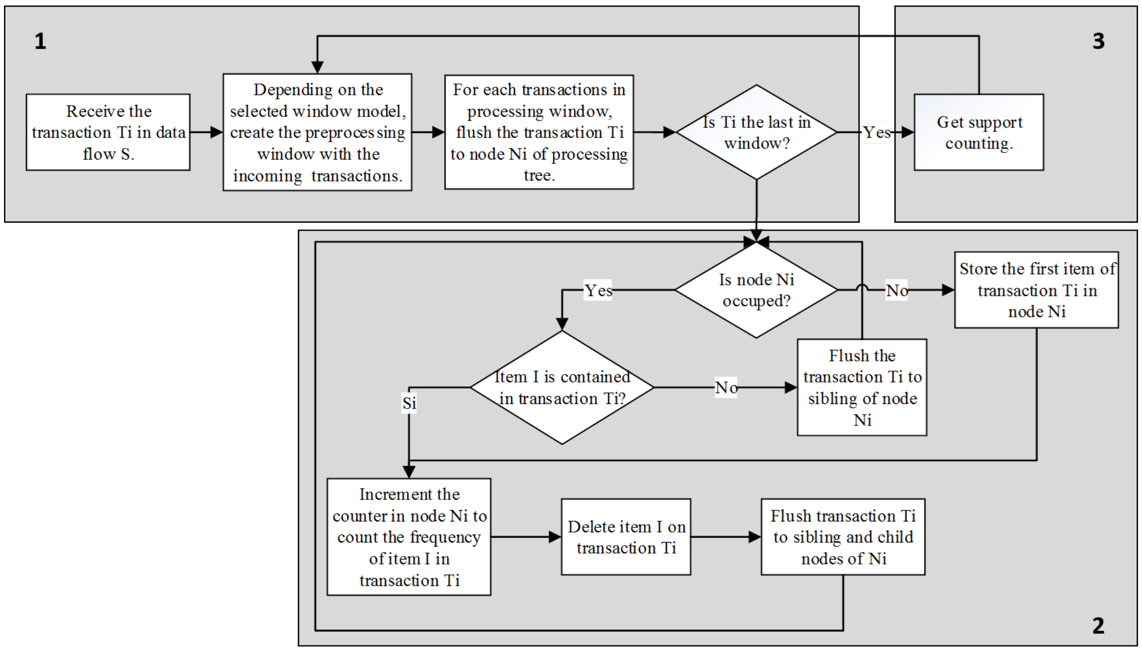

1
Receive the transaction Ti in data flow S.
Depending on the selected window model, create the preprocessing window with the incoming transactions.
For each transactions in processing window, flush the transaction Ti to node Ni of processing tree.
Is Ti the last in window?
Yes
3
Get support counting.
Is node Ni occuped?
Yes
No
Store the first item of transaction Ti in node Ni
Item I is contained in transaction Ti?
No
Flush the transaction Ti to sibling of node Ni
Si
Increment the counter in node Ni to count the frequency of item I in transaction Ti
Delete item I on transaction Ti
Flush transaction Ti to sibling and child nodes of Ni
2

Figure 25: Flow diagram of the proposed method.

3. A module to support counting using a FP-tree structure (named *Support counting*).
4. A module to determine which itemset is frequent (named *Frequent itemsets*).

The inputs of the proposed method are:

- Input data stream.
- Support threshold (can be presented as percent or as a frequency value).
- Window model to use.

The output of the proposed method is:

- The frequent itemsets and its frequency counting (if the support threshold was expressed as a frequency value) or support (if the support threshold was expressed as a percent value).

Fig. 25 shows in detail the flow diagram of this approach. The diagram is divided in 3 areas. Area 1 performs the window creation, area 2 preforms the frequency counting and area 3 performs the support calculation.

The three areas of Fig. 25 indicate the algorithms that compose the proposed method. In area 1, the mechanism to create the processing window is depicted. First, the data stream is received and according to the selected window model and the chosen window size, the processing window is created. In area 2, the frequency counting of itemsets is performed. The frequency is calculated for the processing window constructed in area 1 by mean of systolic tree. Process in area 3 uses the frequency counting calculated and determines, based on the Apriori property [3], which itemset can be regarded as frequent.

Algorithm 1 implements the processes that took place in area 1 of the flow diagram presented in Fig. 25. This algorithm uses one of the following window model: *landmark_window*, *sliding_window*, *damped_window*. Also, the window size is needed, and it must be an integer value. The window size is the number of transactions to be contained in the window. The output is the processing window $window\_buffer$, which is the input of algorithm 2. Algorithm 2 get as input the processing window created by algorithm 1. Then each transaction is flowed into the systolic tree to determine the frequency of each item. The output of this algorithm is the systolic tree with the frequency counting of each node calculated.

The algorithm 2 or Frequency Counting is executed in parallel by each node of the systolic tree. Nodes in the systolic tree must be in one of the following states:

- State 1 (*Empty*): Current node is not occupied (*Occupied* flag of the current node is set to $false$). In this state, the current node is empty and it can store an itemset and therefore, it can count the frequency of the stored itemset.
- State 2 (*Occupied*): Current node is occupied (*Occupied* flag of the current node is set to *true*). In this state, the current node contains an itemset and therefore, its frequency counting.

When a node is in *Empty* state, it can accept the first item $S_i[0]$ of the incoming itemset S_i, which is stored in the *Label* variable. Then, the *Occupied* flag is set to *true* for next itemsets. Also, the $Counter$ variable is incremented and a reduced itemset $\widetilde{S_i}$ is created by removing the itemset stored in the *Label* variable from the original itemset S_i. Following, if the reduced itemset $\widetilde{S_i}$ is not empty, then it is flushed to sibling and child nodes of the current node, which are in *Empty* state. The process is repeated for sibling and child nodes simultaneously in parallel.

When a node is in *Occupied* state the itemset stored in *Label* variable is searched in the incoming itemset. If it is found, the $Counter$ variable is incremented, and a reduced itemset $\widetilde{S_i}$ is created by removing the itemset stored in the *Label* variable from the received itemset S_i. If the reduced itemset is not empty, then it is flushed to sibling and child nodes of the current node, which are in *Empty* state. The process is repeated for sibling and child nodes simultaneously in parallel. If the the itemset stored in *Label* variable is not found in the incoming itemset I, then the incoming itemset I will be flushed in parallel to the sibling node of the current node.

When windows are processed, the systolic tree is given as input to algorithm 3, which traverse the systolic tree recursively from the root to the leaf nodes. When the leaf nodes are reached, the variable $Counter$ is compared against the minimum support value in variable *min_sup* to determine if the itemset stored in the current node can be regarded as frequent. If variable $Counter$ is greater or equal of variable *min_sup*, then the itemset stored

Algorithm 1: Window creation.

```
Input: Data stream S.
Selected window model ($window\_model$).
Selecetd window size ($window\_size$).
Output: Transaction's window ($window\_buffer$).
1  $processed\_windows \longleftarrow 0$;
2  $window\_buffer.Capacity \longleftarrow window\_size$;
3  while Arriving transactions on data stream S do
4      while window_buffer.IsFull() == false do
5          switch window_model do
6              case sliding_window
7                  Get transaction $T_i$ from S;
8                  if window_buffer.Contains($T_i$) == false then
9                      window_buffer.Add($T_i$);
10                 end
11             case landmark_window
12                 $window\_buffer.Cappacity += window\_size$;
13                 Get transaction $T_i$ from S;
14                 if window_buffer.Contains($T_i$) == false then
15                     window_buffer.Add($T_i$);
16                 end
17             case damped_window
18                 $window\_buffer.Cappacity += window\_size$;
19                 Get transaction $T_i$ from S;
20                 if window_buffer.Contains($T_i$) == false then
21                     window_buffer.Add($T_i$);
22                 end
23                 $window\_buffer.Weight = 1/processed\_window$;
24             end
25         end
26         $processed\_window++$;
27     end
28 end
29 return $window\_buffer$;
```

Algorithm 2: Frequency counting.

```
Input: Transaction's window ($window\_buffer$)
Output: Systolic tree with the counting frequency of each item in its corresponding node ($systolic\_tree$).
1  $n_i \longleftarrow systolyc\_tree.RootNode$;
2  foreach $itemset\ S_i\ in\ window\_buffer$ do
3      Flush $S_i$ into $n_i$;
4      if $n_i.IsOccupied == false$ then
5          $n_i.IsOccupied = true$;
6          $n_i.Label.Add(S_i[0])$ ;                 /* Node $n_i$ stores the first item of itemset $S_i$. */
7          $n_i.Counter++$;
8          $\widetilde{S_i} = S_i.Exclude(n_i.Item)$ ;  /* $\widetilde{S_i}$ contains all items of $S_i$ except the item $n_i.Item$. */
9          if $\widetilde{S_i}.IsEmpty == false$ then
10             StartParallalelBlock:
11             $n_i \longleftarrow n_i.ChildNode$;
12             Flush $\widetilde{S_i}$ to $n_i$ and go to step 4;
13             $n_i \longleftarrow n_i.SiblingNode$; Flush $\widetilde{S_i}$ to $n_i$ and go to step 4;
14             EndParallelBlock;
15         end
16     else
17         if $S_i.Contain(n_i.Label) == true$ then
18             $n_i.Counter++$;
19             $\widetilde{S_i} = S_i.Exclude(n_i.Item)$;
20             if $\widetilde{S_i}.IsEmpty == false$ then
21                 StartParallalelBlock:
22                 $n_i \longleftarrow n_i.ChildNode$; Flush $\widetilde{S_i}$ to $n_i$ and go to step 4;
23                 $n_i \longleftarrow n_i.SiblingNode$; Flush $\widetilde{S_i}$ to $n_i$ and go to step 4;
24                 EndParallelBlock;
25             end
26         else
27             $n_i \longleftarrow n_i.SiblingNode$; Flush $S_i$ to $n_i$ and go to step 4;
28         end
29     end
30 end
31 return $systolic\_tree$;
```

in *Label* and counting in *Counter* variables is stored in the *frequent* list. When the recursion is finished, the variable *frequent* will contains all the frequent itemsets and its frequency or support counting.

Algorithm 3: Flushing strategy.

```
Input: Systolic tree with the counting frequency of each item in its corresponding node ($systolic\_tree$).
Minimum support value ($min\_sup$).
Output: Frequent itemsets and its support counting ($frequents$).
1  $n_i \longleftarrow systolyc\_tree.RootNode$;
2  if ($n_i.SiblingNode$ == null) and ($n_i.ChildNode$ == null) then
3      if $n_i.Count == min\_sup$ then
4          frequent.Add($n_i.Item$, $n_i.Count$);
5      end
6  else
7      StartParallalelBlock:
8      $n_i \longleftarrow n_i.ChildNode$;
9      Go to step 1;
10     $n_i \longleftarrow n_i.SiblingNode$;
11     Go to step 4;
12     EndParallelBlock;
13 end
14 return $frequents$;
```

It is valid to notice that the processing scheme that is presented can be scaled to multiple devices. With proper interconnections, multiples processing units that implement the proposed method can be connected and therefore, larger transactions can be handled.

5.6. Preliminary experimental results.

The present PhD. Proposal describes a new method for the frequent itemset mining on data streams. This method, which is composed by 3 algorithms, is designed to be implemented in a custom hardware architecture. To validate the concept introduced in this research, the algorithms were programmed sequentially in software using C# language over the .Net Framework platform. The resulting prototype introduces visual elements to facilitate its use and its experimentation.

As it was explained in section 5.4, the systolic tree can process a limited number of items which is determined by the maximum length of the incoming transactions which contains itemsets. The processing method proposed is insensitive to the support value selected, but not of length of the incoming transactions which is determined

Dataset	# Trans.	ave. length	Max. length.	Min. length	Alphabet
MSNBC	989818	2.825	4	1	1 - 17

Table 6: Characteristics of the chosen datasets.

by the computational resources of the selected development platform. If the chosen development platform can hold a systolic tree with 1024 nodes, the maximum length of itemsets that it can be process will be 10. If the length of incoming itemsets is greater, some itemsets will be not processed and therefore, the mining process will be approximate. However, if the development platform can hold all the possibles itemsets, the mining process will be exact.

To verify the correct functioning of the presented approach, some experiments were conducted. The pursued objectives were:

- Verify the correct performance of the proposed algorithms.
- To measure how the systolic tree grows according to the length of incoming transactions.

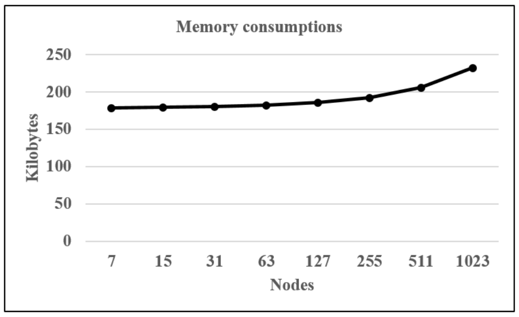
To accomplish these goals, a dataset from UCI repository [39] was used. Table 6 describes this dataset after preprocessed. The preprocessing consists of removing all the repeated items in transactions and spaces, as it was stated in section 5.2. Due to MSNBC dataset is not designed for stream mining, the entire dataset was considered as a window. The MSNBC dataset is formed from the logs for msnbc.com site. Each sequence in the dataset corresponds to page views of a user during that twenty-four hour period. The FP-Growth reported in [40] was chosen as a baseline, and the mining process was performed inside one window.

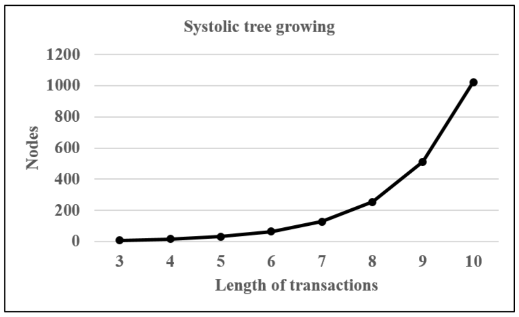
In order to validate that the frequency counting computed by algorithm 2 is correct, it was assumed that the systolic tree can handle all possibles itemsets for used dataset. For various support values selected, experiments demonstrate that the frequent itemsets detected by the proposed method, and its frequency counting, was the same that the one obtained by the baseline FP-Growth.

Table 7 shows the behavior of the systolic tree while the incoming transactions grows, and Fig. 26 shows graphically the same results. Experiments show that the systolic tree grows exponentially concerning to the length of the incoming transactions. This effect can be attenuated using larger hardware devices or using external memories, but it is still an issue to be taken in account. Memory consumption is a hot spot for hardware designs. Experiments demonstrate that the memory needed for systolic trees of different sizes grows in almost an exponential way. One systolic tree with 131 071 processing nodes requires near of 6Mb of memory. These values are calculated for the software implementation, which may not be the same trend when it is calculated for the hardware implementation. The processing time also grows linear concerning of the length of the incoming transactions. Once again, these values are calculated for the sequential software implementation. The proposed

Length	Nodes	Memory (Kb)	Times (s) Tree building	Itemset flushing	FIS flushing	Total
3	7	178.6	0.001	0.940	0.002	0.943
4	15	179.3	0.001	1.253	0.002	1.257
5	31	180.2	0.001	1.667	0.002	1.670
6	63	182.1	0.001	2.003	0.002	2.006
7	127	185.5	0.001	2.534	0.002	2.537
8	255	192.0	0.001	2.909	0.002	2.912
9	511	205.6	0.001	3.376	0.002	3.379
10	1023	232.1	0.002	3.739	0.003	3.744
17	131071	6445.0	0.056	6.274	0.099	6.429

Table 7: Behavior of the proposed method while the length of the incoming transactions grows.

a)


b)


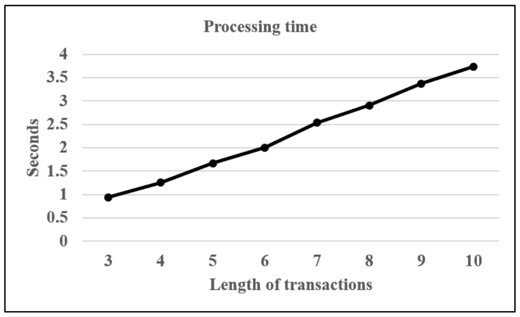
c)


Figure 26: Behavior of the proposed method while the length of the incoming transactions grows. a) Graph of the memory consumption as the systolic tree is growing. b) Graph of the systolic tree size while the incoming transactions grows and c) Graph of the processing time while the incoming transactions grows.

algorithms are designed to be implemented in parallel, so the processing will be executed simultaneously and after some initial time, the results will arrive continuously.

5.7. Analysis of the preliminary results.

The software implementation of the proposed method pursuits the main objective of determining whether it is a valid solution for frequent itemset mining on data streams. Experiments demonstrate that for different support values, the frequent itemsets and its frequency counting are the same that obtained by the baseline software. The length of the incoming transactions, and therefore the systolic tree size, can affect these results. For transactions of maximum size of 5, the systolic tree can handle 31 itemsets. Transactions of length 6 will be treated like transactions of length 5, and the mining process will be approximate. In this case, not all of the frequent itemsets will be returned, but those itemsets that are regarded as frequents by the proposed method will be regarded as frequent with the same frequency counting by the baseline FP-Growth. In other words, if the available computing resources of the development platform selected can handle any length of the incoming transactions, the mining process will be exact. Otherwise, the mining process will be approximate with no false positives. The software implementation validates the correct functioning of the proposed method and allows to understand its functioning before implement it in hardware.

The Fig. 26 gives an approximate trend of the computing resources consumptions. After analyzing such result it is necessary to introduce some optimizations to the proposed method in order to reduce the resources consumption in the selected hardware device. The timing analysis indicates that the hot spot in the sequential software implementation is the itemsets flushing. This issue would be solved in the hardware implementation due to the parallel nature of the algorithms and the selected hardware device.

6. Conclusions.

The frequent itemset mining is a widely used Data Mining technique with outstanding results in database scenario. Data stream mining is a recent research field where frequent itemsets are introducing. Due to the continuity, expiration and infinity characteristic of data streams it is necessary to explore alternatives that allow to increase the efficiency of the mining process in such datasets. One alternative could be the design of parallel algorithms to be implemented in custom hardware architectures.

This PhD. Proposal introduce a new parallel method for frequent itemset mining in data streams which is designed to be implemented in a custom hardware architecture. The proposed method is based on the FP-tree data structure and it is composed of 3 pipelined algorithm. Some experiments were conducted and it can be concluded that the proposed method correctly performs this task. When it is executed in a device with no resources restrictions then the exact mining process is performed. By the contrary, when restrictions are imposed, then the approximate mining process with no false positives is performed. From the experiments conducted it is derived that some adequations must be done to the proposed method in order to save computing resources of the selected hardware device.

Based on the current state of this research and the results obtained in the experimentation it can be concluded that the objectives presented will be accomplished, with the expected results and in the proposed time according with the methodology and the presented work plan.

7. References

[1] D. W. Thöni, A. Strey, Novel strategies for hardware acceleration of frequent itemset mining with the apriori algorithm, in: M. Danek, J. Kadlec, B. E. Nelson (Eds.), 19th International Conference on Field Programmable Logic and Applications, FPL 2009, August 31 - September 2, 2009, Prague, Czech Republic, IEEE, 2009, pp. 489–492. `doi:http://dx.doi.org/10.1109/FPL.2009.5272494`.

[2] A. McAfee, E. Brynjolfsson, Big data: the management revolution., Harvard Bussines Review 90 (10) (2012) 60–6, 68, 128.

[3] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, pp. 487–499.
URL `http://dl.acm.org/citation.cfm?id=645920.672836`

[4] Z. K. Baker, V. K. Prasanna, 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 '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 3–12. `doi:10.1109/FCCM.2005.31`.
URL `http://dx.doi.org/10.1109/FCCM.2005.31`

[5] Z. K. Baker, V. K. Prasanna, An architecture for efficient hardware data mining using reconfigurable computing systems, in: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 67–75. `doi:10.1109/FCCM.2006.22`.
URL `http://dx.doi.org/10.1109/FCCM.2006.22`

[6] U. Erra, B. Frola, Frequent items mining acceleration exploiting fast parallel sorting on the gpu., Procedia CS 9 (2012) 86–95.
URL `http://dblp.uni-trier.de/db/journals/procedia/procedia9.html#ErraF12`

[7] N. K. Govindaraju, N. Raghuvanshi, D. Manocha, Fast and approximate stream mining of quantiles and frequencies using graphics processors, in: Proceedings of the 2005 ACM SIGMOD international conference

on Management of data, SIGMOD '05, ACM, New York, NY, USA, 2005, pp. 611–622. doi:10.1145/1066157.1066227.
URL http://doi.acm.org/10.1145/1066157.1066227

[8] W.-C. Liu, K.-H. Liu, M.-S. Chen, Hardware enhanced mining for association rules, in: Proceedings of the 10th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining, PAKDD'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 729–738. doi:10.1007/11731139_85.
URL http://dx.doi.org/10.1007/11731139_85

[9] S. Sun, M. Steffen, J. Zambreno, A reconfigurable platform for frequent pattern mining, in: Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs, RECONFIG '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 55–60. doi:10.1109/ReConFig.2008.80.
URL http://dx.doi.org/10.1109/ReConFig.2008.80

[10] S. Sun, J. Zambreno, Mining association rules with systolic trees., in: FPL, IEEE, 2008, pp. 143–148.
URL http://dblp.uni-trier.de/db/conf/fpl/fpl2008.html#SunZ08

[11] S. Sun, J. Zambreno, Design and analysis of a reconfigurable platform for frequent pattern mining, IEEE Transactions on Parallel and Distributed Systems 22 (2011) 1497–1505. doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.34.

[12] J. Teubner, R. Mueller, G. Alonso, Frequent item computation on a chip., IEEE Trans. Knowl. Data Eng. 23 (8) (2011) 1169–1181.
URL http://dblp.uni-trier.de/db/journals/tkde/tkde23.html#TeubnerMA11

[13] J. Teubner, R. Mueller, G. Alonso, Fpga acceleration for the frequent item problem, in: Data Engineering (ICDE), 2010 IEEE 26th International Conference on, IEEE, 2010, pp. 669–680.

[14] Y.-H. Wen, J.-W. Huang, M.-S. Chen, Hardware-enhanced association rule mining with hashing and pipelining, IEEE Trans. on Knowl. and Data Eng. 20 (6) (2008) 784–795. doi:10.1109/TKDE.2008.39.
URL http://dx.doi.org/10.1109/TKDE.2008.39

[15] F. Zhang, Y. Zhang, J. Bakos, Gpapriori: Gpu-accelerated frequent itemset mining, in: Proceedings of the 2011 IEEE International Conference on Cluster Computing, CLUSTER '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 590–594. doi:10.1109/CLUSTER.2011.61.
URL http://dx.doi.org/10.1109/CLUSTER.2011.61

[16] A. Mesa, C. Feregrino-Uribe, R. Cumplido, J. Hernndez-Palancar, A highly parallel algorithm for frequent itemset mining, in: J. Martnez-Trinidad, J. Carrasco-Ochoa, J. Kittler (Eds.), Advances in Pattern Recognition, Vol. 6256 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 291–300.

doi:10.1007/978-3-642-15992-3_31.
URL http://dx.doi.org/10.1007/978-3-642-15992-3_31

[17] C. C. Aggarwal (Ed.), Managing and Mining Sensor Data, Springer, 2013.
URL http://dblp.uni-trier.de/db/db/books/collections/mmsd2013.html

[18] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '02, ACM, New York, NY, USA, 2002, pp. 1–16. doi:10.1145/543613.543615.
URL http://doi.acm.org/10.1145/543613.543615

[19] M. J. Zaki, Spade: An efficient algorithm for mining frequent sequences, Machine learning 42 (1-2) (2001) 31–60.

[20] L. Golab, M. T. Özsu, Issues in data stream management, SIGMOD Rec. 32 (2) (2003) 5–14. doi:10.1145/776985.776986.
URL http://doi.acm.org/10.1145/776985.776986

[21] G. Yang, The complexity of mining maximal frequent itemsets and maximal frequent patterns, in: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '04, ACM, New York, NY, USA, 2004, pp. 344–353. doi:10.1145/1014052.1014091.
URL http://doi.acm.org/10.1145/1014052.1014091

[22] K. Compton, S. Hauck, Reconfigurable computing: a survey of systems and software, ACM Comput. Surv. 34 (2) (2002) 171–210. doi:10.1145/508352.508353.
URL http://doi.acm.org/10.1145/508352.508353

[23] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. J. Purcell, A survey of general-purpose computation on graphics hardware, Eurographics 2005, State of the Art Reports (2005) 21–51.
URL http://graphics.idav.ucdavis.edu/publications/print_pub?pub_id=844

[24] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, O. O. Storaasli, State-of-the-art in heterogeneous computing, Sci. Program. 18 (1) (2010) 1–33.
URL http://dl.acm.org/citation.cfm?id=1804799.1804800

[25] C. Cullian, C. Wyant, T. Frattesi, Computing performance benchmarks among cpu, gpu, and fpga, Internet: www. wpi. edu/Pubs/E-project/Available/E-project-030212-123508/unrestricted/Benchmarking_Final.

[26] S. Che, J. Li, J. W. Sheaffer, K. Skadron, J. Lach, Accelerating compute-intensive applications with gpus and fpgas, in: Proceedings of the 2008 Symposium on Application Specific Processors, SASP '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 101–107. `doi:10.1109/SASP.2008.4570793`.
URL `http://dx.doi.org/10.1109/SASP.2008.4570793`

[27] T. El-Ghazawi, D. Bennett, D. Poznanovic, A. Cantle, K. Underwood, R. Pennington, D. Buell, A. George, V. Kindratenko, Is high-performance reconfigurable computing the next supercomputing paradigm?, in: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, SC '06, ACM, New York, NY, USA, 2006. `doi:10.1145/1188455.1188530`.
URL `http://doi.acm.org/10.1145/1188455.1188530`

[28] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, SIGMOD Rec. 29 (2) (2000) 1–12. `doi:10.1145/335191.335372`.
URL `http://doi.acm.org/10.1145/335191.335372`

[29] M. J. Zaki, Scalable algorithms for association mining, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING 12 (2000) 372–390.

[30] W. Fang, K. K. Lau, M. Lu, X. Xiao, C. K. Lam, P. Y. Yang, B. He, Q. Luo, P. V. Sander, K. Yang, Parallel data mining on graphics processors, Tech. rep., The Hong Kong University of Science and Technology (Oct. 2008).

[31] W. Fang, M. Lu, X. Xiao, B. He, Q. Luo, Frequent itemset mining on graphics processors, in: Proceedings of the Fifth International Workshop on Data Management on New Hardware, DaMoN '09, ACM, New York, NY, USA, 2009, pp. 34–42. `doi:10.1145/1565694.1565702`.
URL `http://doi.acm.org/10.1145/1565694.1565702`

[32] F. Zhang, Y. Zhang, J. D. Bakos, Accelerating frequent itemset mining on graphics processing units, The Journal of Supercomputing 66 (1) (2013) 94–117.

[33] Y. Zhang, F. Zhang, Z. Jin, J. D. Bakos, An fpga-based accelerator for frequent itemset mining, ACM Trans. Reconfigurable Technol. Syst. 6 (1) (2013) 2:1–2:17. `doi:10.1145/2457443.2457445`.
URL `http://doi.acm.org/10.1145/2457443.2457445`

[34] S. Shi, Y. Qi, Q. Wang, Fpga acceleration for intersection computation in frequent itemset mining, in: Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2013 International Conference on, IEEE, 2013, pp. 514–519.

[35] F. Bodon, A fast apriori implementation, in: B. Goethals, M. J. Zaki (Eds.), FIMI 03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining

Implementations, 19 December 2003, Melbourne, Florida, USA, Vol. 90 of CEUR Workshop Proceedings, CEUR-WS.org, 2003. `doi:http://SunSITE.Informatik.RWTH-Aachen.de/Publications/CEUR-WS//Vol-90/bodon.pdf`.

[36] J. A. Hartigan, M. A. Wong, A K-means clustering algorithm, Applied Statistics 28 (1979) 100–108.

[37] NVIDIA, Cuda zone (Jan. 2014).
URL `http://www.nvidia.es/object/cuda_home_new_es.html`

[38] C. Borgelt, Efficient implementations of apriori and eclat., in: FIMI, Vol. 90 of CEUR Workshop Proc., CEUR-WS.org, 2003.

[39] A. Frank, A. Asuncion, UCI machine learning repository (2010).
URL `http://archive.ics.uci.edu/ml`

[40] C. Borgelt, An implementation of the fp-growth algorithm, in: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations, OSDM '05, ACM, New York, NY, USA, 2005, pp. 1–5. `doi:10.1145/1133905.1133907`.
URL `http://doi.acm.org/10.1145/1133905.1133907`

[41] M. J. Zaki, K. Gouda, Fast vertical mining using diffsets, in: KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, New York, NY, USA, 2003, pp. 326–335. `doi:http://doi.acm.org/10.1145/956750.956788`.
URL `http://portal.acm.org/citation.cfm?doid=956750.956788#`

[42] B. Goethals, Frequent pattern mining implementations (Jan. 2014).
URL `http://adrem.ua.ac.be/~goethals/software/`

[43] Y. Zhang, F. Zhang, J. Bakos, Frequent itemset mining on large-scale shared memory machines, in: Cluster Computing (CLUSTER), 2011 IEEE International Conference on, IEEE, 2011, pp. 585–589.