# Algorithms for Extracting Frequent Episodes in the Process of Temporal Data Mining

Alexandru PIRJAN
Romanian-American University, Bucharest, Romania
alexparjan@yahoo.com

*An important aspect in the data mining process is the discovery of patterns having a great influence on the studied problem. The purpose of this paper is to study the frequent episodes data mining through the use of parallel pattern discovery algorithms. Parallel pattern discovery algorithms offer better performance and scalability, so they are of a great interest for the data mining research community. In the following, there will be highlighted some parallel and distributed frequent pattern mining algorithms on various platforms and it will also be presented a comparative study of their main features. The study takes into account the new possibilities that arise along with the emerging novel Compute Unified Device Architecture from the latest generation of graphics processing units. Based on their high performance, low cost and the increasing number of features offered, GPU processors are viable solutions for an optimal implementation of frequent pattern mining algorithms.*

***Keywords****: Frequent Pattern Mining, Parallel Computing, Dynamic Load Balancing, Temporal Data Mining, CUDA, GPU, Fermi, Thread*

## 1 Introduction

Most of the data mining processes analyze unordered data collections, but there are also many important applications in which the analyzed data is ordered. Given the importance and usefulness of real time data mining, in recent years numerous researches have been aimed towards the discovery of new hardware architectures that could manage and process huge amounts of data. Real time data mining enables scientists to conduct research at an unimaginable scale.

Not only the hardware architecture but also the implemented data mining algorithms must properly manage and process a huge amount of data, otherwise data analysis risks becoming irrelevant in certain fields, like that of neuroscience. The optimization of a data mining algorithm can be achieved by improving both the quality of the data mining process and by minimizing the response time.

An episode is defined as a partially ordered set of events for consecutive and fixed-time intervals in a sequence. A specific issue of temporal data mining is the one concerning the mining and analysis of frequent episodes, meaning the sequences of frequent appearances for certain groups of events in a time ordered database [1]. The main purpose in mining frequent episodes is to discover relations between different events, relations that could determine a certain event or help to anticipate future results.

Frequent episodes mining is used successfully in different fields such as security analysis and intrusion detection in case of computer systems, biomedical data analysis [2], [3], predicting the evolution of the stock shares, disaster risk management in climatology [4] or in mining significant episodes from statistical models.

A frequent pattern is a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set. The term was first proposed in [5] in the context of frequent item sets and association rule mining. Frequent patterns are widespread in our daily life such as products that are often purchased together, subsequent purchases after buying a computer, establishing what kinds of DNA are sensitive to a new drug, some molecular fragments frequently appearing in a certain class of molecules with similar functions [6], [7]. Frequent pattern mining can be successfully applied to Basket data analysis, cross marketing, catalog design, sale campaign, analysis, Web log (click stream) analysis, and DNA sequence analysis.

In the purpose of defining frequent patterns, it will be considered a transaction database $D = \{t_1, t_2, ..., t_n\}$, where $t_j, j \in \{1, ..., n\}$ are transactions, and a real number $\varepsilon \in (0,1]$ called user-specified minimum support. In a frequent pattern mining problem the aim is to discover all pattern sets contained in a percentage greater than or equal to the user-specified minimum support, ε in the transactions from the database. The input transaction and pattern depends on the type of the studied problem and it can be a graph, a tree, an itemset or a sequence. In frequent pattern mining, patterns are layered depending on their size and sorted in correspondence to a certain order at each layer, modeled as a lattice structure [6]. The problem size influences the dimension of this pattern lattice. Considering a transaction database which contains $m$ distinct items, the number of possible patterns is $2^m$.

On large databases, the frequent pattern mining requires a lot of computational power. Serial frequent pattern mining algorithms cannot scale to large data sets because they are limited to the computing capability of a single processor and to the memory space, which is finite. In order to fix this problem, it is required to use parallel or distributed high-performance computing, which will overcome the problems of sequential algorithms.

When a serial frequent pattern mining algorithm is designed and it is intended to use its parallelizability, it is essential to understand the properties of pattern lattice and techniques of common pattern enumeration. The redundancy or incompleteness in enumerating frequent patterns are eliminated because all serial frequent pattern mining algorithms use a specific method of browsing the lattice.

The main criteria for the algorithms classification are the cutting technique of a serial frequent pattern mining algorithm, along with the support counting method adopted and the type of pattern lattice enumeration. In recent years there have been proposed many efficient algorithms using a wide variety of features and details. Some of them use hash table or other special data structures in order to improve their performance.

Regarding the space enumeration paradigm, frequent pattern mining episodes could be classified into two categories. In the first class of algorithms, it is used the level-wise candidate-generation-and-test method, which consists of the following: it is selected the set of already mined frequent patterns of length $l$, then by joining patterns of length $l$ it is generated the set of all candidates patterns of length $l+1$ and these candidates are tested in order to filter out infrequent patterns. The process is iteratively repeated until the longest pattern is obtained. Representative algorithms in this class include Apriori, GSP and AGM, which correspond to frequent itemset, frequent sequence, and frequent graph mining problems, respectively [6].

The second class of algorithms for frequent pattern mining implements the depth-first pattern growth and database projection method. When a frequent pattern of length $l$ is mined, it is extended in a predefined order by one item in order to obtain a pattern of length $l+1$. Gradually, by increasing the pattern size, it is obtained a projection database. The procedure is repeated recursively until all the elements of the pattern tree are browsed. Representative algorithms in this class are FP-growth for itemset mining, PrefixSpan for sequence mining, and gSpan for graph mining [6].

## 2 About Parallel Pattern Discovery

In order to solve frequent pattern mining problems multiple processors can be used. In this case, the number of processors leads to a significant speedup, but there are some facts (arising from the specificity of the frequent pattern mining problems) that must be taken into account. A first step in order to find the support for a pattern is a count operation that has to be done in the database, against all the transactions. If the transactions are distributed evenly among the processors, during the process of pattern mining each processor performs its operations on its local set of transactions. This method, called count-distribution, has some problems related to communications and synchronization. This is the reason why the method is inconvenient. Because a transaction contains many patterns and a frequent pattern appears in more than one transaction, each transaction is counted for each of the frequent patterns contained.

When it comes to discover an aggregate pattern, this task could be distributed among multiple processors. In fact, the pattern lattice is split on the processors which deal with mining the subset of patterns assigned to them. In this case, almost all the transactions in the database should be available for all the processors.

Non-Uniform Memory Access or Non-Uniform Memory Architecture (NUMA) is a computer memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors. For such a system, the database is replicated on each processor in order to avoid an excessive number of non-local transactions. The benefits of having a large quantity of aggregate memory and a storage space on parallel platforms are affected by database replication. If the database were replicated selectively as to satisfy the required needs, additional statistics process would be required in the database.

A load balancing technique is essential in order to improve the response time in the case of parallel algorithms. For this, a method that approximates the tasks unit relative mining time is required. The approximation must reach at an adequate level of granularity when designing a static load balancing strategy. A strategy for task assignment must be applied afterwards when distributing (in a balanced way) work over multiple processors. In most datasets the workload can only be approximate in a heuristic manner because the pattern tree is unbalanced. The running time necessary to discover a subset of frequent pattern must be approximated accurately.

In order to achieve an accurate approximation, a dynamic load balancing technique can be used. As tasks have to be split among multiple processors, any parallel frequent pattern mining algorithm must partition the task of pattern discovery so that it produces fine granulated or tasks recursively partitioned.

The task transfers introduce an overhead which has to be minimized and this problem is more serious when the overhead is accompanied by a large amount of transaction transfers or when the architecture incorporates a slow network.

The parallelization of frequent pattern mining algorithms must implement sophisticated pattern space pruning techniques and compact data structures. A pruning strategy makes use of patterns that had already been mined by other processors but has the disadvantage of introducing additional costs. An efficient frequent pattern mining algorithm design must balance all these factors very carefully.

## 3 The most important parallel algorithms

In the following are depicted some of the most representative frequent pattern mining algorithms, emphasizing itemset mining, sequence mining, and graph mining.

a) The parallel frequent itemset mining is the simplest of all the frequent pattern mining problems. In this case it is considered a set $J$ of items, and a transaction database $D = \{t_1, t_2, ..., t_n\}$, where $t_j, j \in \{1, ..., n\}$ are subsets of $J$. In the following, the most important methodologies for parallel frequent items mining algorithms (of the candidate-generation-and-test class) are presented. There are three methods for parallel frequent items mining algorithms of the candidate-generation-and-test class [6]: count distribution, data distribution and hybrid methods.

- In the count distribution method, the database must be partitioned evenly and the patterns of length $l$ must be replicated over all processors. The processors generate the set of candidate patterns with length $l+1$ and count them in each of their local database. In the next step, the global count is obtained, by performing a sum reduction. The whole process is in a level-wise approach.

- Regarding the data distribution method, one must take into account that the database is partitioned among the processors, each of them generating a disjoint set of candidates. The processors must intercommunicate in order to have access to the entire database for counting the candidates.
- In hybrid methods, the parallelization strategy of the distributed memory, is the same as in the count-distribution method algorithm. A general purpose parallel data mining middleware processes datasets resident on the disk.

Generally, in the case of parallel frequent itemset mining algorithms, which implement the level-wise candidate-generation-and-test (CGT) method, some steps must be followed. In the beginning, the candidate generation phase must be computed by all the processors and then, through an intensive inter-processors cooperation, a higher level for the next pattern length is achieved. It is difficult to obtain a dynamic load balance strategy due to the running mode presented below, which leads to a cascade of steps and a huge amount of synchronization stages.

An example of frequent itemset mining algorithm, implementing the depth-first pattern growth and database projection paradigm is FP-growth (Frequent Pattern). First, in order to hold all the data in the original database a compact data structure, the FP-tree, is built. In the next stage, on the read-only FP-tree structure is performed a search according to a specific method, named divide-and-conquer [8]. Because FP-growth has an intrinsic divide-and-conquer nature it can easily be parallelized.

A particular case of simple parallel algorithm based on the FP-growth model is the Multiple Local Frequent Pattern Tree (MLFPT) [9]. This is useful for mining frequent itemsets on shared memory machines. First, each processor builds a local FP-tree structure using the part of the transaction assigned to it, which means that a parallel paradigm is used. A global header table is shared by the FP-tree structures and through this table, every processor has access to the entire structure. Using the header table, the frequent items are assigned among processors in a manner that takes into account the best execution time for mining. Then, based on the frequent item assigned, each processor independently mines patterns. As mentioned below, the FP-growth algorithm (on which the Multiple Local Frequent Pattern Tree algorithm is based) uses for mining a divide-and-conquer method, which is essential in this case.

Another parallel algorithm based on the FP-growth model is the PFP-tree [10]. This algorithm uses a distributed memory architecture. In the first stage the construction of the parallel frequent pattern trees similar to that in MLFPT is realized, but in this case multiple FP-trees reside in different physical memory spaces. In the second stage, the mining of these data structures is executed. This task is also divided among processors, similarly as in the Multiple Local Frequent Pattern Tree parallel algorithm. An important difference between this algorithm and the one presented below is that in this case each processor can keep the needed part of the database locally. Both Multiple Local Frequent Pattern Tree and PFP-tree algorithms have a major drawback, related to the estimate of the relative mining time that is done in a coarse way.

Another method for mining frequent itemsets on distributed memory architectures is the Inverted Matrix [11]. First a matrix structure is created, which reorganizes the transaction database into it. This structure, called inverted matrix, is then replicated among all the processors nodes. As in the Multiple Local Frequent Pattern Tree algorithm, the next stage consists in the assignation of frequent items among the processors. Each processor is responsible for mining patterns containing items which have been assigned to it and also items with a larger rank in the ascending order of the items' frequency. In the next stage, for each assigned items, each processor builds a structure named co-occurrence frequent item tree and finally on this structure it is performed a non-recursive mining. The Inverted matrix algorithm has no serial counterpart. This algorithm offers a good static load balancing strategy, more reasonable than that one offered by the Multiple Local Frequent Pattern Tree algorithm or the PFP-tree.

b) The parallel frequent sequence mining algorithms deal with the mining of patterns taking into account the temporal order between items. A sequence is a list of temporally ordered itemsets (events). For the most serial frequent sequence mining algorithms, the methodologies are similar with those implemented in the case of frequent itemset mining algorithms, but in the case of frequent sequence mining special features arising from the temporal order between events must be taken into account.

In the literature, there are less results and studies about parallel frequent sequence mining than about parallel frequent itemset mining. Some of them are depicted below.

In [12] are proposed three variants of parallel algorithms (NPSPM, SPSPM and HPSPM) for distributed memory computer based on GSP (mentioned before, as a classic level-wise candidate-generation-and-test style frequent sequence mining algorithm). A common characteristic of these three algorithms is the fact that they implement a partition of the database among the processor nodes. Two of these algorithms, NPSPM and SPSPM, are similar to Counter Distribution and Data Distribution algorithms designed for parallel frequent itemset mining problems (mentioned previously). The HPSPM algorithm is an improved version of the SPSPM algorithm because it uses a hashing mechanism in order to partition candidate sequences among the processors and also reduces the amount of communication overhead used to count the global support.

Also based on the distributed memory architecture, the algorithm proposed in [13] is a variant of the parallel tree-projection-based frequent sequence mining algorithm. In this case, each processor builds the same pattern tree and this leads to redundancy as in the serial algorithm. An interesting feature of this algorithm is that it switches to task parallel mode after running the data parallel on the first $l$ levels of the pattern tree. Processors generate independently sub-forests rooted at the assigned nodes after the distribution of the nodes at the $l$ level among them. In order to be able to run independently, processors must exchange part of their local databases. This algorithm uses a dynamic load-balancing strategy for overcoming consequences of the inaccurate estimation of the task run time (as the load imbalance).

In [14] it is proposed another parallel algorithm, pSADE, based on hardware distributed shared memory architecture. This algorithm is similar with other parallel pattern growth and database projection algorithms. Each of the processors works on different partitions of the database, but the whole pattern tree is processed synchronously. One copy of the database can be accessed by every processor. For this algorithm, the static load balancing is designed so that top-level tasks are partitioned among processors. In [14] is proposed a strategy for improving the efficiency of the algorithm, the recursive dynamic load balancing strategy. Briefly, the strategy helps the splitting of tasks between busy processors and idle ones. A busy processor that detects an idle processor inserts nodes from its current working class into the global task queue. In the next stage, idle processors choose a task from this list.

A parallel closed sequential pattern-mining algorithm (Par-CSP) is proposed in [15]. This algorithm runs on distributed memory system and is based on depth-first-search and divide-and-conquer strategies. In order to perform the mining, the tasks are distributed among processors. A specific feature of this algorithm is that a selective sampling technique to achieve good load balance is used, but except that, the method is similar to that one implemented in the case of parallel frequent itemset mining algorithms. The sampling technique selects small portions of the projected database. Each of these representative samples is used to approximate the relative mining time of the projected database that it belongs, and the runtime estimated is later used for the static task assignments.

c) The parallel frequent graph mining involves computational complexity of the graph-based algorithms and also graph/subgraph isomorphism test requirements. In a graph database, transactions are usually undirected labeled graphs. Some examples of frequent graph mining algorithms are Subdue, MolFea, FSG, MoSS/MoFa, gSpan, CloseGraph, FFSM and Gaston.

The major problem in parallelizing frequent graph mining algorithms is the risk of severe load imbalance in task partitioning, as a consequence of the irregularity of the graph pattern lattice. In order to solve this problem, there are required a good load balancing strategy and a proper parallelism granularity.

A first example of frequent graph mining algorithms is MoFa [7] which was developed to discover connected discriminative molecular fragments for drug discovery. This algorithm models molecular fragments as attributed graphs. An interesting fragment is infrequent in a non-active set and frequent in another active molecular set. MoFa applies a depth-first search on the frequent fragment search tree and then extends one bond. In order to record the exact position information of the studied fragment, the algorithm uses an embedding list. An interesting feature is that algorithm uses a local order based structural strategy according to which the algorithm does not need to extend an atom inserted before the last extended atom.

Based on the above depicted MoFa algorithm, in [16] it is proposed a distributed frequent subgraph mining algorithm in which each worker machine has local access to the entire active database. An independent sub-task for the worker machine is generated by the job management machine by pruning the search tree. In the task assignment message intermediate mining states are also included. When the worker machine has just finished a job or it is idle, it gets a new task from the job pool.

Similar to the above-depicted algorithm an improved version of it was developed in [17]. This algorithm, based on the fact that every machine is a donor and a worker at the same time, also offers enhanced search space partitioning. Each time a worker finishes his job, a donor is selected by the worker in order to spawn a new job. This is a dynamic load balancing strategy called ranked-random polling. A centralized machine keeps the ranking information.

Another parallel frequent graph-mining algorithm on a shared memory machine is presented in [18]. The algorithm is a parallelized version of MoFa. In this case, the graph database, the idle worker processor list and the global frequent fragment set are the globally shared data structures. In order to track for each processor its depth first search path is used a stack structure. At the proper time, locally mined frequent fragment set of each processor is merged into the global set. At the beginning of the mining process, every processor starts mining on the whole database. Before starting the search step, the processor checks the global idle processor list, finds an available co-worker and donates out a part of its working stack.

Based on the parallelization of the gSpan algorithm [19] another algorithm was developed in [20]. In this case, instead of work donation, it is implemented the so-called work stealing technique which uses a global busy processor list. For part of its work stack, idle processors actively request one of the busy processors.

A revolution in computer architecture technology, Chip Multiprocessing or simply multicore, is a combination of two or more independent processors (multi-core systems) into a single integrated circuit package. The processors share the same memory space. The Chip Multiprocessing allows the device to exhibit some sort of parallelism - thread-level parallelism and/or instruction level parallelism - while enjoying fewer components, lower cost, and less interconnection overheads.

In the case of a Chip Multiprocessing architecture some algorithms have been developed, their design being similar to that for shared memory system. A characteristic of these cases is the lower processor communication cost. An example of a parallelized graph mining algorithm for the Chip Multiprocessing architecture is proposed in [21]. First, the algorithm performs a depth search on the frequent graph tree with each candidate extension from a tree node being a task unit. It is used a distributed task queuing model, each processor core performing operations of enqueue and dequeue on its own task queue. When a processor core's queue is empty, it searches other cores' queue for work and if all queues are empty it waits until a core, which has a nonempty queue, queries it. In order to reduce memory consumption a pointer based compact embedding list is used.

Systems on which are designed most of the parallel frequent pattern mining algorithms are based on shared memory, distributed memory, hybrid systems, heterogeneous environments, chip multiprocessing or simultaneous multithreading.

As a consequence of the importance and usefulness of real time data mining, in recent years researchers intensified their efforts to discover new hardware architectures that can manage and process large volumes of data. A real potential in optimizing the data mining process is offered by graphics processing units (GPUs). They are multithreaded and multicore processing units and that is the reason why a GPU has a computational capacity and memory bandwidth far beyond than those of central processing units (CPU). As a consequence, most of the databases operations are accelerated, the entire data mining process is simplified, the necessary time for extracting knowledge from data analysis is reduced. Combining hundreds of simplified parallel processing cores, these graphics processing units also improve the performance per watt consumed (obtained from the GPU when compared to the CPU processors). Based on their high performance, low cost and on the increasing number of features offered, an increasingly wide range of applications from different fields could be solved by GPU processors, and among them the study of the temporal data mining process and its applications in financial data prediction, telecommunication control, neuroscience, medical data analysis.

Below are depicted some temporal data mining algorithms and as a solution for improving their performances, the algorithms are implemented on the new Compute Unified Device Architecture (CUDA) from the latest generation of graphics processing units (GPU). For each temporal data mining problem it is required to address specific technical issues. The size of the problem and the type of the algorithm implemented on the GPU are important factors used to determine the optimal algorithm, the data access model and the number of threads that are necessary to achieve the desired performance.

Graphics processing units processors have been used in order to accelerate graphics rendering on computers and over time the GPU has evolved through specialized architecture (from one-purpose components to multiple purposes complex architectures). A broad class of applications could be accelerated as a consequence of this development, and the GPU is able to do much more than just provide video rendering.

## 4 The Compute Unified Device Architecture – a viable solution for improving algorithms performances

The Compute Unified Device Architecture (CUDA) is a software and hardware parallel computing architecture (developed by NVIDIA) that allows the NVIDIA graphics processor to execute programs written in C, C++, FORTRAN, OpenCL, Direct Compute and other languages. CUDA gives developers access to the native instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest NVIDIA GPUs effectively become open architectures like CPUs. Unlike CPUs however, GPUs have a parallel "many-core" architecture, each core being capable of running thousands of threads simultaneously - if an application is suited to this kind of an architecture, the GPU can offer large performance benefits. This approach of solving general-purpose problems on GPUs is known as GPGPU.

A CUDA program calls parallel program kernels. A set of parallel threads is executed by the kernel in parallel. These threads are organized by the programmer or compiler into thread blocks and grids of thread blocks. The graphics processor unit instantiates a kernel program on a grid containing parallel thread blocks. Each thread from the block executes an instance of the kernel and has an unique ID associated to registers, to thread’s private memory from the thread block [22].

In the CUDA programming model, when algorithms are developed, the most important concern of developers is to divide the required work in fragments that can be processed by a number of thread blocks, each containing more threads. In order to avoid that the threads within a block will be executed by more cores within a streaming multiprocessor, it is recommended that the number of thread blocks matches the number of processors. The most important factor in achieving performance is the repartition of tasks to be performed between the thread blocks.

An usual method used to discover how certain subsets of elements are associated with other subsets is the technique of data mining through association, and a restricted version of that technique is temporal data mining (in which temporal relationships between elements are taken into account). In a timed ordered database, a specific problem of temporal data mining is the mining of frequent episodes. The purpose in this case is to find sequences of frequent items (episodes) appearances. An episode is defined as a partially ordered set of events for consecutive time intervals, embedded in a sequence [1].

Below are depicted four algorithms based on the CUDA programming model [23]. For each of these algorithms it is implemented some kind of parallelism but a common feature of them is that all are based on the MapReduce programming model [24].

Algorithm 1 uses a search of a single episode by each thread, using data stored in graphics memory. This algorithm doesn’t implement buffering. The database is placed by the first algorithm in the texture memory. For each thread, this feature facilitates the use of the high bandwidth of the GPU. As a consequence, threads are allocated in thread blocks one by one until the maximum number of threads per block is reached.

In the case of Algorithm 2, each thread is looking for a single episode. The second algorithm loads a block of data from the database into a buffer of shared memory. After the data from the buffer is processed, the algorithm then loads another block of data in the buffer and the process is repeated along the entire database. The thread allocation method within the thread blocks is the same as in Algorithm 1.

Algorithm 3 does not use buffering. All the threads in a block search one episode. The data from the graphic card memory is used and different blocks are looking for different episodes. Even if there are similarities with the first algorithm (in both cases threads within each block access data through the texture memory), there are also some characteristics that differentiate them: threads within a block are starting at different positions within the database, while threads with the same ID from different blocks are starting from the same position.

In the fourth algorithm analyzed the same episode is searched by all the threads in a block. A data buffer is created by means of shared memory and then this buffer is used by different blocks which are looking for different episodes. Algorithm 4 uses block-level parallelism with shared memory database buffering. As in Algorithm 3, for each thread the starting point depends on buffer size and not on the size of the database. During all searches, a thread will always access the same area of shared memory, but each time when buffer updates the data content from the shared memory will change.

## 5 Experimental results

In the following are presented the most relevant experimental results and interpretations on the performance of above depicted algorithms implemented on CUDA architecture, for episodes at different levels with different numbers of threads per block. At the L level of an episode, an algorithm searches an episode of length L. In the considered cases, L was chosen 1, 2 or 3.

A test consists of selecting an episode's level, an algorithm, a graphics card and the block size. The execution period (measured in milliseconds) is considered the period between the moment when the kernel is invoked and the moment when it returns the answer.

The configuration used in tests was Intel i7-965 operating at 3.2 GHz with 6 GB (3x2GB) of 1333Mhz Triple Channel Memory. The graphic card chosen was nVIDIA GTX470, based on nVidia's new Fermi architecture. The feature-list is considerable: over 3 billion transistors double the processing units of its predecessors, and a strong emphasis on geometric realism. This card has a huge computational capability and is based on the latest FERMI architecture. Some specifications of this graphic card are: 448 CUDA cores, the graphics clock's frequency 607 MHz, the processor clock's frequency 1,215 MHz, the texture fill rate 34 billion/sec, the memory clock's frequency 1,674 MHz, the memory bandwidth 133.9 GB/sec, the amount of memory 1,280 MB, the maximum number of threads per block is 512.

It was considered a database which consists of 393,019 letters from the capital letters of the English alphabet that repeat themselves. It was chosen a different number of episodes at each level as follows: the first level contained 26 episodes, level 2 contained 650 episodes and level three contained 15,600 episodes [23].

Below is depicted the effect of algorithm selection on execution time. When an algorithm is chosen, it must always be taken into account the size of the considered problem. Mostly, a programmer wants to solve a problem of a certain size and he has access only to a certain type of hardware. Because he can modify only the algorithm and the number of threads that he uses within this algorithm, in order to obtain the best results he will use the fastest algorithm for the problem (Figure 1). Some conclusions of tests are mentioned in the following.

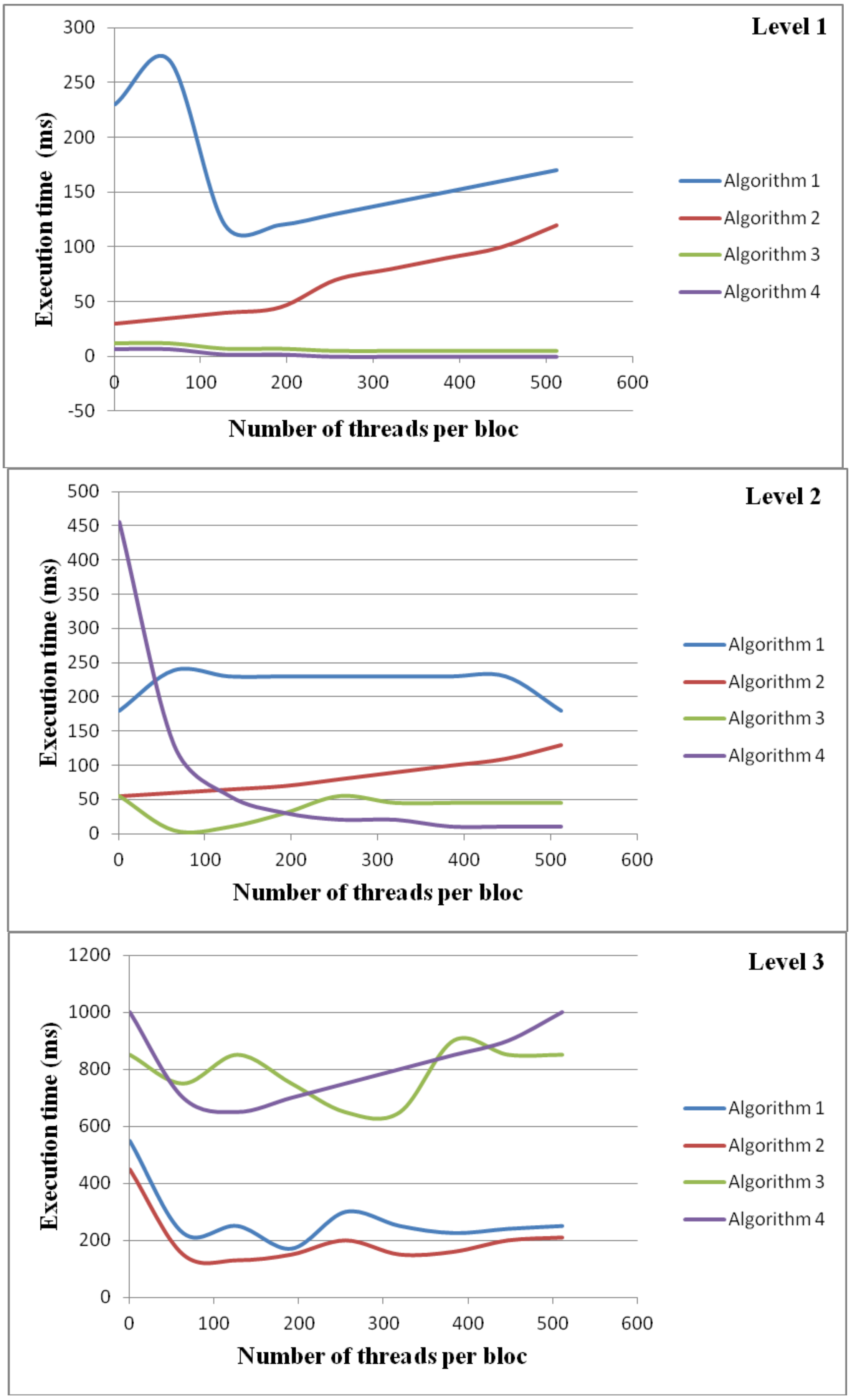

**Fig. 1.** The effect of algorithm selection on execution time

1) For small problems (Level 1), one thread per episode is not enough. In the case of small lower levels problems, the number of threads generated by episodes is low, (because the number of episodes is low) and therefore there are no sufficient threads to use the graphic card's resources. In the case of Algorithms 1 and 2, as the number of episodes is fixed and there is just one thread per episode, one can observe the tendency to increase the execution time along with the number of threads. Unlike these cases, Algorithm 4 obtains a search time of a milliseconds order. The most important observation is that when using the GTX470, practically real-time data mining can be achieved. In the future, for significant size databases, servers incorporating more of these parallel cards and future GPU architectures will reach startling performance.
2) For medium size problems (Level 2), the block level depends on its size. In the case of Algorithms 1 and 2 the number of blocks decreases while the number of threads per block increases. Because there is a fixed number of episodes, there is also a fixed number of threads. Therefore, the number of blocks and the number of threads per block changes in the same time.
3) For large problems (Level 3), the thread level parallelism is enough. In this case there are 25,230 episodes to search [5]. One can observe that Algorithms 1 and 2 (parallel thread processing algorithms) are much faster than Algorithms 3 and 4 (block-level algorithms). In the case of Algorithms 1 and 2 each thread within a block will look for one episode so more episodes may be searched.

## 6 Comparisons among algorithms and conclusions

The algorithms presented in the first part of this paper, designed to solve various types of pattern discovery (for frequent itemset, frequent sequence and frequent graph mining problems), have some common characteristics. The pattern lattice/tree traversal paradigm is useful in the implementation of the parallelism for most of the recently developed parallel frequent pattern mining algorithms. The divide-and-conquer pattern enumeration scheme of the FP-growth algorithm is used by MLFPT, PFP-tree, and Inverted Matrix algorithms. A prefix tree or suffix tree manner is used for many serial frequent sequence mining algorithms when the pattern lattice is traversed. This is the reason why in the case of frequent sequence mining task parallelism the method frequently used is to partition the tree nodes of a certain level.

Regarding the static load balancing, in parallel pattern discovery is required the estimation of the relative mining time for each task unit, but there is a low communication overhead. The dynamic load balancing (recommended especially in the case of graph databases) can be used only if the mining task associated with a pattern can be recursively partitioned into smaller ones.

The database replication is also an important feature in parallel frequent pattern mining. As mentioned before, a database can be partitioned with no overlaps among processors. In order to reduce the communication overhead, some algorithms duplicate the whole database on each processor (Par-CSP, pSPADE, all the reviewed parallel frequent graph mining algorithms). The problem which appears in this case is that this replication does not facilitate the efficient use of aggregate disk or memory space of a parallel platform, consequently a tradeoff is needed. In the future, for the designing of parallel frequent pattern mining algorithms there will be useful techniques as the minimum-cut bipartite graph partition model in pTPSM [13] or the idlist partitioning based data parallel formulation in pSPADE.

In the second part of the article, we analyzed and compared some temporal data mining algorithms implemented on the latest CUDA based architecture Fermi, As experimental results outlined, in order to obtain an increased performance, any implementation (based on the MapReduce framework) must dynamically adapt the type and parallelism level.

Even if the practical implementation of data mining algorithms on a GPU architecture brings a lot of advantages, there are many difficulties and some limitations in this process. First of all it is the fact that a CUDA programmer must have thorough knowledge of how threads work and how thread blocks are mapped, must know in detail six different areas of memory and especially inter-threading communication.

Another problem is caused by the limitations on the performance of temporal data mining algorithms in memory size and the transfer time between the GPU and the memory. When it comes to huge dimensions data warehouses even if current NVIDIA cards support memory sizes up to 6 GB, this size is still insufficient, being far below from the required size. With the launch of the new Fermi architecture, this size was extended: before Fermi was 4 GB and Fermi brings 6 GB. Even if the size of the memory supported by the GPU has increased, 6 GB is still insufficient. In practice, many databases` sizes are of the order of terabytes or even petabytes. Therefore, this remains a significant hardware limitation.

The performance (when applying temporal data mining algorithms) is also influenced by the considerable amount of execution time consumed by the transfer of memory blocks between the CPU and GPU.

Compared to conventional architectures based on CPUs, the results offered by the new Fermi architecture highlight a huge potential for improving the performance of temporal data mining process. Some limitations of the implemented algorithms' performance, caused by hardware issues, can be overcome since the new Fermi hardware architecture has been launched. When dealing with dynamically accessed arrays, there is an important limitation that must be take into account. In this case, because dynamically accessed arrays cannot be accessed directly by an index at compile time, occurs a direct impact on algorithms runtime.

In the CUDA programming model, dynamically accessed arrays are automatically stored in local memory. They cannot be stored in the registry memory. The problem is that the local memory is an abstraction and it has the same latency time as global memory of GPU. As a consequence, it is many times slower than registry memory. A significant restriction in the case of the four algorithms for temporal data mining presented below is the fact that the registry memory cannot be used, as a consequence of the frequent use of this type of arrays addressing.

Most of the limitations mentioned above can be solved by the Fermi architecture, the new generation of NVIDIA architecture, which also allows improved execution times for the algorithms. Studying the experimental results one can state that the new nVIDIA CUDA architecture can be a viable solution for the parallelization of algorithms.

## References

[1] N. Satish, M. Harris and M. Garland, "Designing Efficient Sorting Algorithms for Manycore GPUs," in *Proc. 23rd IEEE International Parallel and Distributed Processing Symposium,* 2009.

[2] G.C. Garriga, "Discovering unbounded episodes in sequential data," in *Proc. Seventh European Conference on Principles and Practice of Knowledge Discovery in Databases* (PKDD), 2003.

[3] N. Meger, N.L.C. Leschi and C. Rigotti, "Mining episode rules in stulong dataset," *ECML/PKDD 2004 Discovery Challenge* (PKDD), 2004.

[4] S. Harms, J. Deogun, J. Saquer and T. Tadesse, "Discovering representative episodal association rules from event sequences," in *Proc. 2001 IEEE International Conference on Data Mining* (ICDM'01), 2001.

[5] R. Agrawal, T. Imielinski and A. Swami, "Mining association rules

between sets of items in large database," in *Proc. ACM SIGMOD International Conference on Management of Data,* pages 207-216, Washington D.C., May 1993.

[6] Z. Yuzhou, W. Jianyong and Z. Lizhu, "Parallel Frequent Pattern Discovery: Challenges and Methodology," *Tsinghua Science And Technology,* ISSN 1007-0214 15/20, pp.719-728, vol. 12, no. 6, December 2007.

[7] C. Borgelt and M.R. Berthold, "Mining molecular fragments: Finding relevant substructures of molecules," in *Proc. 2002 IEEE International Conference on Data Mining,* Maebashi, Japan, 2002, pp.51-58,

[8] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, USA, 2000, pp.1-12.

[9] O.R. Zaïane, M. El-Hajj and P. Lu, "Fast parallel association rule mining without candidacy generation," in *Proc. 2001 IEEE international Conference on Data Mining,* San Jose, USA, 2001, pp.665-668.

[10] A. Javed and A. Khokhar, "Frequent pattern mining on message passing multiprocessor systems," *Distrib. Parallel Databases,* 2004, 16(3), pp.321-334.

[11] M. El-Hajj and O.R. Zaiane, "Parallel association rule mining with minimum inter-processor communication," in *Proc. 14th International Workshop on Database and Expert Systems Applications,* Prague, Czech Republic, 2003: 519.

[12] T. Shintani and M. Kitsuregawa, "Mining algorithms for sequential patterns in parallel: Hash based approach," in *Proc. Second Pacific-Asia Conference on Research and Development in Knowledge Discovery and Data Mining,* Melbourne, Australia, 1998, pp.283-294.

[13] V. Guralnik and G. Karypis, "Parallel tree-projection-based sequence mining algorithms," *Parallel Computing,* 2004, 30(4), pp. 443-472.

[14] M.J. Zaki, "Parallel sequence mining on shared-memory machines," *Journal on Parallel Distributed Computing,* 2001, 61(3), pp.161-189.

[15] S. Cong, J. Han and D. Padua, "Parallel mining of closed sequential patterns," in *Proc. Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining,* Chicago, USA, 2005, pp.562-567.

[16] G.D. Fatta and M.R. Berthold, "Distributed mining of molecular fragments," in *Proc. IEEE International Conference on Data Mining, Workshop on Data Mining and the Grid. Brighton,* UK, 2004, pp.1-9.

[17] G. D. Fatta and M. R. Berthold, "Dynamic load balancing for the distributed mining of molecular structures," *IEEE Transactionson Parallel and Distributed Systems*, 2006, 17(8), pp.773-785.

[18] T. Meinl, I. Fischer and M. Philippsen, "Parallel mining for frequent fragments on a shared-memory multiprocessor," *LWA 2005. German Research Center for Artificial Intelligence*, 2005, pp.196-201.

[19] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," *Proc. IEEE International Conference on Data Mining*, Maebashi, Japan, 2002, pp.721-723.

[20] T. Meinl, M. Worlein, I. Fischer, et al., "Mining molecular datasets on symmetric multiprocessor systems," in *Proc. 2006 IEEE International Conference on Systems, Man and Cybernetics*, Taipei, China, 2006, pp.1269-1274.

[21] G. Buehrer, S. Parthasarathy, D. Kim, et al., "Towards data mining on emerging architectures," in *Proc. 9th SIAM Workshop on High Performance and Distributed Mining,* Bethesda, USA, 2006.

[22] "NVIDIA CUDA Compute Unified

Device Architecture" - *Programming Guide*, Version 3.1, 2010.
[23] J. Archuleta, Y. Cao, W. Feng and T. Scogland, "Multi-Dimensional Characterization of Temporal Data Mining on Graphics Processors," *Technical Report TR-09-01*, Computer Science, Virginia Tech, 2009.
[24] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004.

**Alexandru PIRJAN** has graduated the Faculty of Computer Science for Business Management in 2005. He holds a MA Degree in Computer Science for Business from 2007. He joined the staff of the Romanian-American University as a teaching assistant in 2005 and a Full Teaching Assistant in 2008. He is currently a member of the Department of Informatics, Statistics and Mathematics from the Romanian-American University. He is the author of more than 15 journal articles and a member in 4 national scientific research projects. His work focuses on artificial intelligence, database applications and quality of software applications.