Two-level parallel CPU/GPU-based genetic algorithm for association rule mining

Leila Hamdad*

LCSI,
Ecole Nationale Supérieure en Informatique (ESI),
BP 68M, Oued Smar, El Harrach, Algeria
Email: l_hamdad@esi.dz
*Corresponding author

Zakaria Ournani

Ecole Nationale Supérieure en Informatique (ESI),
BP 68M, Oued Smar, El Harrach, Algeria
Email: bz_ournani@esi.dz

Karima Benatchba

LMCS,
Ecole Nationale Supérieure en Informatique (ESI),
BP 68M, Oued Smar, El Harrach, Algeria
Email: k_benatchba@esi.dz

Ahcène Bendjoudi

DTISI,
CERIST Research Center,
Algiers, Algeria
Email: ahcene.bendjoudi@gmail.com

Abstract: Genetic algorithms (GA) are widely used in the literature to extract interesting association rules. However, they are time consuming mainly due to the growing size of databases. To speed up this process, we propose two parallel GAs (ARMGPU and ARM-CPU/GPU). In ARM-GPU, parallelism is used to compute the fitness which is the most time consuming task; while, ARM-CPU/GPU proposes a two-level-based parallel GA. In the first level, the different cores of the CPU execute a GAARM on a sub-population. The second level of parallelism is used to compute the fitness, in parallel, on GPU. To validate the proposed two parallel GAs, several tests were conducted to solve well-known large ARM instances. Obtained results show that our parallel algorithms outperform state-of-the-art exact algorithms (APRIORI and FP-GROWTH) and approximate algorithms (SEGPU and ME-GPU) in terms of execution time.

Keywords: association rules; parallel genetic algorithm; GPU; CPU.

Reference to this paper should be made as follows: Hamdad, L., Ournani, Z., Benatchba, K. and Bendjoudi, A. (2020) 'Two-level parallel CPU/GPU-based genetic algorithm for association rule mining', *Int. J. Computational Science and Engineering*, Vol. 22, Nos. 2/3, pp.335–345.

Biographical notes: Leila Hamdad is a Lecturer in Ecole Nationale Supérieure en Informatique (ESI) at Algiers, Algeria. She is member of Laboratoire de Communication dans les Systèmes Informatiques (LCSI) in applied mathematics team. She had her PhD on Computer Science in the same school. Her topics of interest are related to data mining, machine learning, spatial statistics, parallel computing.

Zakaria Ournani is an Engineer in Computer Science. He had his diploma in Ecole Nationale Supérieure en Informatique (ESI). Actually, he is researcher on Éco-Conception Logicielle (Green Software Design).

Karima Benatchba is a Lecturer in Ecole Nationale Supérieure en Informatique (ESI), Algiers. She is the Head of Laboratoire des Méthodes de Conception des Systèmes (LMCS) and leader of the optimisation team. Her topics of interests are linked to optimisation problems and machine learning.

Ahcène Bendjoudi is a Full-Time Researcher at the CERIST Research Center within DTISI Laboratory in Algiers since 2012. He is the founder and the head of the parallel computing and applications CAPA Team and responsible of high performance computing platform (IBNBADIS) since september 2013. He received his Master and PhD degrees in Computer Science from the University Abderrahmane Mira of Bejaia, Algeria, and the engineering degree in Computer Science from the High School of Computer Science (ESI), Algiers, Algeria. His research interests include mainly parallel optimisation, multi-core/GPU-Computing, Big Graphs and Road Routing in Smart Cities.

1 Introduction

Association rule mining (ARM) consists of extracting up ARM. As they return solutions of good quality with less useful correlations among items of transactional databases. It is one of the most important task of supervised learning is used in many fields (basket market analysis, drugs handle association in medicine, ...). The most used algorithms to deal with ARMs are APRIORI (Agrawal and Srikant, 1994) and FP-GROWTH (Han et al., 2000). They are two exact algorithms that generate frequent itemsets from which association rules are built. Generating frequent itemsets is an NP-difficult problem as it is time consuming. As a result, many metaheuristics have been proposed, in the literature, to extract ARs such as: bee swarm optimisation for association rules mining (BSO-ARM) (Djenouri et al., 2015), particle swarm optimisation for association rules mining (PSOARM) (Kuo et al., 2011), ant colony optimisation for rules (ACOR) (Moslehi et al., 2011) and gravitational systems (Khademolghorani et al., 2011). However, the most used metaheuristic is genetic algorithm (GA). The proposed GAs use different chromosome encodings and different crossover and mutation operators. Genetic association rules (GENAR) (Mata et al., 2001), genetic association rules (GAR) (Mata et al., 2002), adaptative steedy state genetic algorithm for rules discovery (ASGARD) (Jourdan, 2003) are some works found in the literature. Recently in Hamdad et al. (2019), the impact of GA operators on AR quality was studied.

The main purpose of using metaheuristics has been to speed up ARM, so that they return solutions of good quality with less computational effort (Jourdan, 2003). However, the continuous growth of data keeps this execution time prohibitive. To handle this problem, parallel rule mining algorithms have been proposed. These parallel computing techniques have been used to accelerate the extraction by carrying out several tasks simultaneously (Cui and Guo, 2013).

Nowadays, most processors are composed of two or more independent central processing units (cores) that can run multiple instructions at the same time. The use of these processors allows to achieve impressive performances when an adequate parallelism is adopted. This parallelism has to exploit simultaneously and efficiently the cores. However, best performances depend strongly on the design of used algorithms and their implementation. Indeed, executing basic sequential algorithms on multi-core CPU does not allow to achieve better performances because only one core is exploited. Therefore, even though the used algorithms are efficient, they do not benefit from the real power of the multi-core CPUs if they are not well designed to run on the different cores. In addition, graphics processing units (GPU) are fast and cheap parallel processors traditionally used to accelerate 3D graphic applications. However, since 2007, they have evolved from exclusive graphic devices into massively parallel programmable processors with application programming interfaces (APIs), programming environments, and languages. They are an efficient means to ensure parallel computing due to their high parallel architecture and availability of a great number of execution units (Fang et al., 2009). When optimally exploited, such resources (multi-core CPUs and GPUs) supply a great computing power.

GPUs have been successfully used to improve execution time of intensive scientific applications from different domains. Courtier (2013) explains how to use GPUs and presents their advantages in many applications. Moreover, works have been interested in optimising the performance of GPUs such as Li et al. (2019) who proposed a static analytical kernel performance model to estimate the execution time of GPU kernel. Since GPU has small size internal memory to handle big dataset for big data analysis, Choksuchat and Chantrapornchai (2018) proposed the use of different memory types in resource description framework (RDF) to improve the performance in the context of big data.

This simple and reliable technology has also been investigated by the ARM community. However, only a few works have tackled parallel GPU aspects of ARM in the literature. The exploited parallelism is mostly based on multiprocessors (distributed memories) such as PMOGA algorithms (Dehuri et al., 2006) or on GPU's type of architecture as SE-GPU and ME-GPU (Djenouri et al., 2015). The latter is an adaptation of swarm intelligence algorithms (Bee Swarm Optimisation) on GPU. Furthermore, to the best of our knowledge, no hybrid CPU multicore/GPU-based genetic algorithm has been proposed before for ARM problem.

We propose in our work, two parallel GAs (ARM-GPU and ARM-CPU/GPU) based on a proposed GA (GA-ARM). Two population organisations are used:

1 one population enclosing different size rules.

2 one population divided in subpopulations where each one encloses same size rules (Hamdad et al., 2019).

In ARM-GPU, parallelism is used to compute the fitness which is the most time consuming task. ARM-CPU/GPU is a two-level-based parallel GA. In the first level, the different cores of the CPU execute a GA-ARM on a sub-population. The second level of parallelism is used to compute the fitness in parallel on GPU. To evaluate our proposed algorithms, several tests were conducted on several synthetic and real datasets of different sizes (small, medium and large). The obtained results according to performances have a good speedup results, since, they outperform the state of the art algorithms as APRIORI, FP-GROWTH and ME-GPU.

This paper is organised as follows: In Section 2, we introduce the problem statement. In Section 3, we give a review of works based on GPU in exact and approximate ARM methods. In Section 4, we present a brief review of works on genetic algorithms for ARM and our genetic algorithm (GA-ARM). In Section 5, we present our parallel approaches, describing the different levels of parallelism. Section 6 presents results of experimentations on different sizes synthetic and real datasets. In Section 7, concluding remarks are drawn and some perspectives are presented.

2 Problem statement

ARM can be defined as follows: given a set of items $I = \{i_1, i_2, \ldots, i_n\}$ and a set of N transactions $T = \{t_1, t_2, \ldots, t_N\}$, each transaction contains a number of items, and two item-sets A, B of I, an association rule is defined as follows:

$$R : A \rightarrow B,$$

where A is called antecedent (premise) of the rule, B the consequence (conclusion) and $B \cap A = \varnothing$ (Agrawal and Srikant, 1994).

The quality of a rule can be evaluated through *Support* and *Confidence*. The first one expresses rules' generality while the second expresses rules' validity:

$$Support(A \cup B) = Supp(A, B) = \frac{|A \cup B|}{N}.$$

$$Confidence(R) = Conf(R) = \frac{|A \cup B|}{|A|}.$$

The support gives the proportion of transactions that contains both A and B among N ones. An association rule (AR) $R : A \rightarrow B$ has a *Support S* if S% of transactions contain both A and B. R has a *Confidence C* if C% of transactions that contain A, contain also B. The interesting rules extracted are those with *Support* greater than *Minsup* and *Confidence* greater than *Minconf*, where *Minsup* and *Minconf* are two thresholds given by the user.

Other measures exist in literature to filter and improve the quality of the ARs. The main one is *lift* defined as:

$$lift(R) = \frac{Supp(A, B)}{Supp(A) * Supp(B)}.$$

3 Exact and parallel ARM based on GPU

Parallelism has been exploited both for approximate and exact methods. Indeed, the huge amount of data has led to the intensification of parallel computation in all fields. Consequently, several researchers have focused on approximate methods trying to accelerate them using parallel computation, in particular on GPU (See for example Loukil et al., 2013). Since the high performance technologies are abundant, such as CPU, GPU, FPGA, … Duran et al. (2014) have focused on examining the challenges of quantifying technology characteristics and introduced a framework to compare and detect the most appropriate one for a given application. The first works on GPU, were open source frameworks handling metaheuritcs on GPU such as: PUGAGE (Soca et al., 2010) which implements cellular evolutionary algorithms on GPUs. ParadisEO-MO-GPU (Melab et al., 2011) is a framework developed in 2011, to handle parallel s-metaheuristcs on GPU. It is an extension of ParadisEO (Cahon et al., 2004) developed for parallel and distributed metaheuristics. And libCudaOptimize (Nashed et al., 2012) is an extensible open source library for p-metaheuristics on GPU. The GPU takes in consideration the fitness evaluation. For more details on the different cited frameworks, one can refer to (Loukil et al., 2013). In what follows, we present a review on works that use GPU on exact and approximate ARM methods.

3.1 Parallel ARM on GPU

Many techniques of parallelism have been proposed in the literature to deal with ARM. The aim is to speed up exact methods to provide association rules faster than sequential methods. In Zaki (1999), one can find a survey on parallel algorithms used for the AR extraction. These algorithms are classified according to the type of used memory (distributed or shared). Garg and Mishra (2011) provide a similar classification. Since 2007, GPU technology has started take on a particular importance, due to its low cost and the large number of threads that can run simultaneously. This has led many researchers to use parallelism on GPU for ARM. Almost all the first works consisted of a parallelisation of the algorithm APRIORI. In 2009, Fang et al. proposed two approaches to parallelise the first phase of APRIORI (which consist on extracting frequent itemsets). Recall that it is the most time consuming phase. The first approach, *pure bitmap implementation* (PBI) extracts of frequent itemsets only on GPU. Then, they are sent to the central processing unit (CPU) to apply the second phase of APRIORI and extract the AR. The advantage of PBI is that GPU-CPU communication is avoided. However, the number of threads by block can be insufficient to handle all transactions.

The second approach (TBI: *trie bitmap implementation*) is an improvement of PBI in order to reduce the search time of itemsets. To do so, TBI uses a tree structure, instead of the bitmap, to keep track of the itemsets evolution. Trie is

managed by the CPU, while the transaction bitmap is managed by the GPU in the same way as in PBI. According to the authors, both approaches have given better execution times than sequential APRIORI, but not as good as the ones given by the sequential version of FP-GROWTH on T40I10D100K benchmark (Goethals, 2004).

Adil and Qamar (2009) proposed an algorithm using GPU for the first phase of APRIORI and the CPU for the second phase. The adopted representation for the itemsets is bitmap. Each thread computes the support of several itemsets candidates and stores it in a table shared among threads (to avoid costly access to the main memory). A 100% exploitation of the threads is often reached. Once the itemsets are built, the second phase is executed on CPU sequentially. An improvement (20 times faster) is obtained thanks to this parallelism compared to the sequential version of APRIORI. The drawback of this algorithm is the GPU-CPU communication which can be penalising.

Zhou et al. (2010) proposed another parallel approach on GPU to extract the frequent itemsets in APRIORI, called GPU-FPM (GPU-*frequent pattern mining*). This approach uses a structure to save transaction items (similar to the structure of APRIORI-TId) in order to avoid multiple access to the database. Each computation unit computes the support of an assigned itemset using the pre-constructed structure. This version is at least 10 times faster than the sequential version of APRIORI on the *T40I10D100K* benchmark (Goethals, 2004). However, the construction of this structure can lead to an additional execution time. GPAPRIORI (Zhang et al., 2011) computes items' supports on GPU. Indeed, the authors proposed a classical generation of candidate itemsets on CPU, and supports computation on GPU thereafter. Each itemset is processed by a single block of threads. A transaction is taken into account by a thread in the block. The advantage of the method is that threads are not overloaded for the computation of an itemset. However, the number of transactions processed in parallel is limited by the number of threads per block. GPAPRIORI is ten times faster than sequential APRIORI for *T40I10D100K* benchmark and 100 times faster for the dataset accident (Goethals, 2004). CUDA-APRIORI was proposed in Cui and Guo (2013). Its optimisation principle is to distribute a transaction on several blocks of threads and candidate itemsets supports' are computed by the threads. Synchronisation between blocks is required and has a negative impact on the total execution time. The undertaken tests showed that CUDA-APRIORI is 20 times faster than APRIORI for ten datasets. Recently, a deep parallel APRIORI, D2P-Apriori was proposed by Wang et al. (2018). It is a deep parallel frequent itemset mining algorithm with dynamic queue. According to the authors, their parallel approach outperforms several state-of-the-art frequent itemset mining algorithms on large datasets.

3.2 ARM-GPU-based metaheuristics

One can find few GPU-based approximate methods for ARM in the literature and small numbers uses bio-inspired approaches. In 2013, Cano et al. proposed an evolutionary algorithm to solve the ARM problem on GPUs. After saving the rule in constant memory, the consequent and the antecedent of the rule are evaluated concurrently. In 2014, Djenouri et al. proposed a method for extracting AR on GPU using BSO called SE-GPU. In 2015, Djenouri et al. proposed ME-GPU. These two methods are an adaptation of the sequential BSO-ARM method presented earlier (Djenouri et al., 2014). Firstly, SE-GPU (single evaluation) evaluates two solutions on GPU, since this step is CPU time consuming. Each thread checks if the rule belongs to the transaction (and will be set to 1) or not (and will be set to 0). At the end, a particular thread adds the results of all the threads and sends the evaluation to the CPU. The drawback of this strategy is the treatment of only one solution at a time. This implies that the threads of the GPU are often not used 100%. ME-GPU (multiple evaluation) overcomes this drawback and several rules are evaluated once. Indeed, its principle is to assign the evaluation of a solution to a single block of threads. Hence, the key differences between SE-GPU and ME-GPU are in the number of threads assigned to each solution and the number of GPU-CPU exchanges which is larger in SE-GPU. The experiments carried out have shown the acceleration of the proposed parallel methods compared to BSO-ARM when executed on CPU. In addition, ME-GPU is faster than SE-GPU, especially for large and medium-sized instances.

Recently, GBSO-RSS, a GPU-based BSO for rules space was proposed by Djenouri et al. (2018c). It deals with extraction of AR in big data context using meta-rules discovery that gives to the user the summary of the rules space through a meta-rules representation. Some other works have focused on using high performance computing to extract AR in reasonable time (Djenouri et al., 2018a, 2018b).

4 Genetic algorithm for rule mining

In this section, we present a brief review of works on genetic algorithms for ARM and then present our genetic algorithm (GA-ARM).

4.1 Related works

The majority of metaheuristics proposed in the literature for rule mining are GAs. They use different encodings and/or different genetic operators. One of the first GA was genetic association rules (GENAR) proposed by Mata et al. (2001). A chromosome defines a rule and each gene represents a numerical item (see Figure 1). The first genes are items of the antecedent and the last one is the consequence. In this work, non-numerical items are not considered and chromosomes are of fixed size. Support and confidence are stored for each rule. A uniform crossover and a simple mutation that consists on changing the values of genes are used. Genetic association rules (GAR) was proposed by Mata et al. (2002). It uses the same genetic operators as GENAR with a different chromosome representation. Each chromosome is an itemset. As a consequence, it has a

variable length. To obtain AR, the second phase of APRIORI is to be used.

In adaptive steady state genetic algorithm for rules discovery (ASGARD) introduced by Jourdan (2003), chromosomes that represent rules can be of different sizes as only items appearing in a rule will be represented. The author uses an elitist insertion and a generalisation operator on one of the parents participating in the crossover. The crossover is done by exchanging values between the common items of the two chromosomes. If no item is in common, the crossover is done by increasing the size of one parent by one item of the other parent. To overcome chromosome size increase, the generalisation operator removes a random item from one of the parents after each crossover. Two types of mutation are used: changing the value of an item or changing items. Association rules mining genetic algorithm (ARMGA) (Yan et al., 2009) proposes to represent several rules in a chromosome. As a result, an index is used in the solution representation (Figure 2) to distinguish the antecedent part from the consequent. A two point crossover, a mutation by value and a fitness based on a support measure are used. The initial population is generated randomly and to diversify it, a value mutation is applied before executing the GA.

Adaptative genetic algorithm (AGA) (Wang et al., 2011) is another proposal of a genetic algorithm. It uses the same chromosome representation as ASGARD, but with a fixed size. An adaptive mutation rate – which decreases from one iteration to another – is the feature of this algorithm. This reduction aims to reduce the number of mutations through iterations to stabilise the last populations. The drawback of the method is the additional execution time required to adapt the mutation rate.

Mining Boolean association rules algorithm (MBAREA) was proposed in Kabir et al. (2015). It is a genetic algorithm based on ARMGA, using the same crossover and fitness. The difference is that a binary representation of items is used (if an item is present the corresponding entry is set to 1. 0 otherwise). In addition, the chromosomes are divided into classes according to their fitness, where a different mutation rate is used for each class. The comparison between MBAREA and ARMGA showed that the latter often returns more rules, but that MBAREA is the fastest.

Two works summarise and analyse the performances of several Genetic algorithms (Indira and Kanmani, 2012) and more recently Hamdad et al. (2019). In the latter, the authors studied the impact of all GA operators (types of chromosomes encoding, types of crossover, type of replacement, …) on the quality of returned AR.

4.2 GA-ARM algorithm

The pseudo-algorithm of GA-ARM is presented below. We chose to use for each component of the GA, the strategy that gave the best ARs quality in Hamdad et al. (2019).

Algorithm 1 GA-ARM

Begin
- Generate randomly a population (initial population)
- Compute the fitness of all its chromosomes

For each generation
- Select a pair of individuals

For each selected pair of individuals
- Crossover with a probability PC
- Compute new fitness
- Replacement

End for
- Mutation with a probability PC.
- Compute fitness

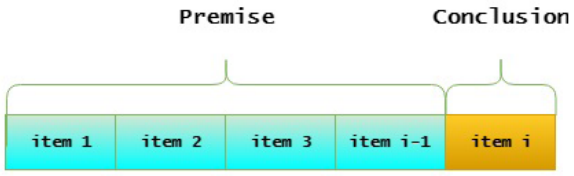
End for.

End

The different used components of our GA-ARM are explained in what follow:

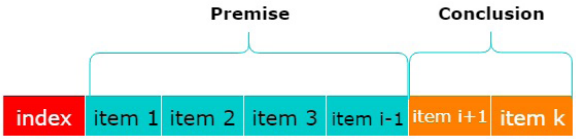
1 *Chromosomes encoding:* we used one rule encoding, where, a chromosome represents a rule as shown in Figure 1 and contains different items (Yan et al., 2009).

Figure 1 One rule encoding (see online version for colours)



Source: Hamdad et al. (2019)

Figure 2 Several rule encoding (see online version for colours)



Source: Hamdad et al. (2019)

2 *Fitness:* usually support and confidence are used to evaluate rules (Indira and Kanmani, 2012, Djenouri et al., 2015). The fitness function is defined as follows:

$$Fitness = \frac{\alpha \times Support + \beta \times Confidence}{\alpha + \beta}$$

where α and β are two parameters in [0, 1].

3 *Genetic operators:* selection, crossover, mutation and replacement are applied on a population of chromosomes for a number of iterations. The following genetic operators are used in our algorithm.

   a *Selection:* at each iteration, chromosomes are selected randomly in pairs on which crossover is applied.

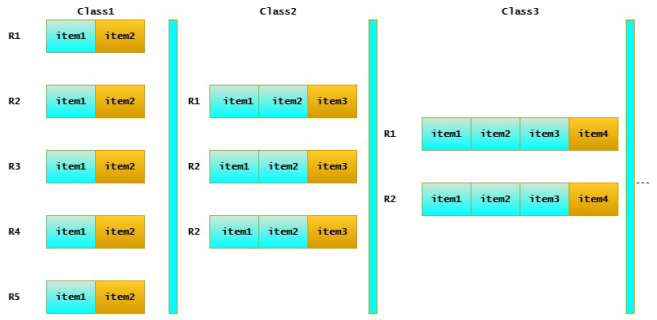
b *Crossover:* the two selected chromosomes are crossed with a probability PC to give two new solutions. Three types of crossover are used in literature (one point, two points and uniform). In our work, we used two-point crossover since it has been shown in Hamdad et al. (2019) it had a good impact on the quality of returned ARs. Each time, a chromosome is generated, its fitness is computed. This value helps to decide whether the new chromosome will be added to the new population.

c *Replacement:* we used the elitist replacement technique after a crossover. It consists of replacing the worst chromosome in the population by the new one if its fitness is better.

d *Mutation:* each chromosome is mutated with a probability PM. It consists of changing a random item in a chromosome with another item selected randomly. This mutation might generate a new association leading to a diversification.

4 *GA's population:* we use, as in Hamdad et al. (2019) two types of population's organisation. The first consists of rules of different sizes generated randomly during initialisation organised in one population. For the second, the population is organised into classes where each class contains same size rules. And, a number of chromosomes are generated randomly while respecting the size of individuals in a class (Figure 3).

Figure 3 Organisation of rules in classes (see online version for colours)



5 Proposed parallel approaches

To speed-up GA-ARM, we present in the following our parallel propositions. The first called ARM-GPU is a GPU-based version where the parallelism is used to compute the fitness on the GPU, as it is the most time consuming task. The second is GA-ARM-CPU/GPU which is a hybrid parallel approach. It is a two level-based parallel GA-ARM. The first level consists on executing several GA-ARMs on different cores of the CPU. In this approach, the initial population is divided into several sub-populations where each sub-population evolves in parallel on a different core of the CPU. The second parallelism level is used to compute the fitnesses on GPU.

5.1 GPU-based GAs (fitness evaluation on GPU)

In this approach, the CPU is used to evolve sequentially the population. Threads of the GPU are used to compute the fitness. The transactions of the database are distributed on the different threads, thus, several transactions are run in parallel and the fitness computation of a rule is often computed in only a few iterations. If the number of transactions is greater than the maximum number of threads of the GPU, the database is split into several parts according to the number of available threads. Figure 4 illustrates the execution of the genetic algorithm between CPU and GPU.

The computation of the two values $|A|$ and $|A \cup B|$ (fitness) requires exclusive thread access to both values. This is done using atomic operations on these values where the incrementation is performed by a single thread at a time, while the other threads are blocked waiting for their turn. Figure 5 shows how threads can be assigned to different transactions and exclusively access to global variables. To minimise run time, a copy of the database is kept in the global memory of the GPU. The database is loaded only once on the GPU so the amount of information exchanged between CPU and GPU is reduced. This communicated data concerns only the rule for which fitness is to be computed.

Figure 4 GPU parallel GA (see online version for colours)

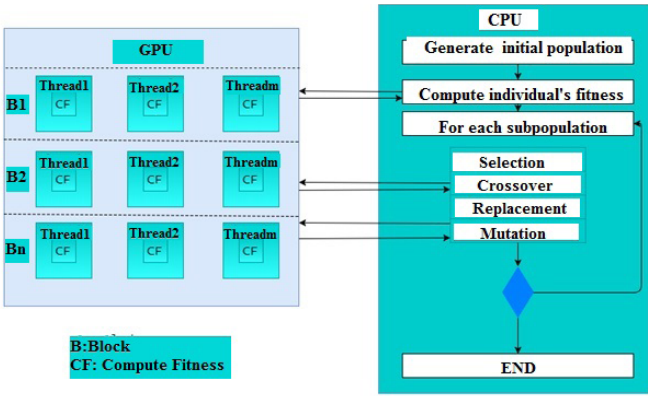
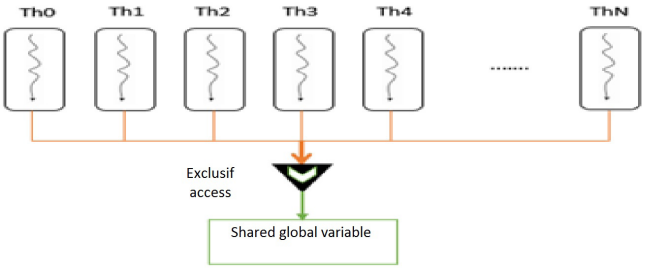


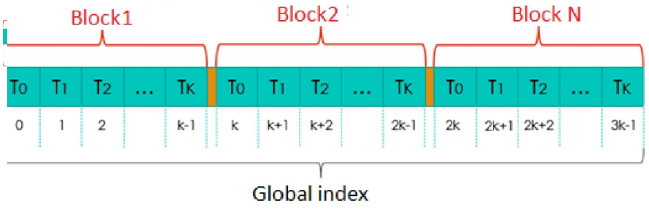
Figure 5 Exclusive access to variables on GPU (see online version for colours)



Each thread is assigned to a transaction regardless of its position in a block. Indeed the transactions are directly distributed on the threads without taking into consideration the block number. To do so, a general index of the threads is computed in order to be able to identify each thread separately and assign to it a particular transaction. Figure 6 gives an example of the GPU's global thread

indexing. If the size of a block is K, the threads of the first block are identified by indices from 0 to $K - 1$. The threads of the second block have indices from K to $2K - 1$, etc. (*Globalindex = blocknumber ∗ blocksize + threadnumberintheblock*).

Figure 6 Global indexation of GPU (see online version for colours)



The SIMD architecture is respected by performing the fitness computation on GPU. Indeed, all threads execute the same instruction on different data. On the other hand, transactions are not always of the same size, which can lead to some thread divergences.

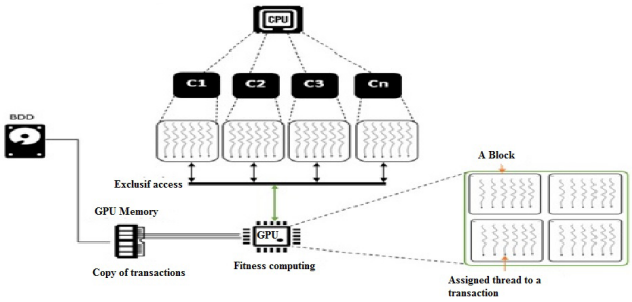
5.2 Hybrid CPU/GPU: two level parallel GA

The developed hybrid algorithm is a two level parallel GA, the first level of parallelism is implemented on the different cores of CPU and the second on GPU. The initial population is divided into sub-populations evolving in parallel on the different cores of the CPU. The parallel evolution of sub-populations is based on Master/Worker paradigm. The Master initiates the algorithm, divides the initial population into sub-populations, launches several workers and distributes the obtained sub-populations among them to be evolved in parallel on the separated cores.

The second level of parallelism exploits the GPU with its large number of threads that can run concurrently. This level of parallelism concerns each worker and the GPU. A master/slave paradigm is then considered. In this case, a worker on a CPU core behaves as a master and the GPU as a slave. When the GPU is available, a worker sends one chromosome at a time to the GPU to evaluate its fitness in parallel by the threads. To do so, each subpopulation is managed by each worker that runs on a CPU core. This worker applies the GA on the sub-population, but computes the fitness on GPU using the previously presented GPU-based GAs. At the end, the different workers join and the Master ends the computation and terminates the algorithm.

The different processes executing in parallel GA-ARMs on CPU cores, access to the GPU for fitness computation concurrently. Indeed, the GPU is considered as a critical resource that must be managed in the program, unlike the main memory and the CPU that are managed by the OS. To do so, access to the GPU is done in critical section, by a single thread at a time. The other processes are queued in case where several fitness computation requests are made at the same time. As soon as a thread releases the GPU, the others can access it one by one. There is no priority among the processes for accessing the GPU (FIFO policy) (see Figure 7). This hybridisation benefits from the advantages of the two previous methods (parallelism of subpopulations on CPU and computation of fitness on GPU).

Figure 7 Concurrent use of the GPU



6 Performance evaluation

To evaluate the performance of our parallel approaches, we have considered well-known ARM instances. Several experiments have been carried out on real and synthetic small, medium, and big datasets (Goethals, 2004). The synthetic datasets are generated from IBM Quest Synthetic database (Bruno, 2008). The size of a dataset is expressed by the number of its transactions. The number of occurrences of the items used in all transactions allows to establish a better ranking of the datasets. Thus, we propose a classification where the small datasets are those using a total number of occurrences of items less than 100,000. A medium size means a total number of occurrences greater than 100,000 and a number of transactions less than 100,000. Large datasets are those with more than 100,000 transactions with more than one million items. Table 1 summarises the datasets used.

Table 1 Dataset characteristics

Size	*Dataset*	*Nbr. of transactions*	*Nbr. of items*	*Total nbr. of items in transactions*	*Type*
Small	SPEC-heart	187	86	8,415	Real
	Forests	246	206	15,332	Real
	C20d10	2,000	386	40,000	Synt
Medium	Chess	3,176	75	121,448	Real
	Mushroom	8,416	128	193,568	Real
	T25i10d10	9,976	1,000	247,148	Synt
	C73d10	10,000	2,178	750,000	Synt
	T10i4d100	98,395	1,000	995,244	Synt
	Retail	88,162	16,469	996,754	Real
	T20i6d100	99,922	1,000	1,988,427	Synt
	Pumsb_star	49,046	7,116	2,524,993	Real
	Connect	67,557	129	2,972,508	Real
	Pumsb	49,046	7,116	3,679,219	Real
Large	Kosarak	990,002	41,270	8,034,379	Real
	Accidents	340,183	468	11,841,053	Real

The experiments have been carried out on the following hardware configuration which is a CPU host coupled with a GPU device (slave). ARM-GPU and ARM-CPU/GPU have been implemented with C-CUDA 4.0 for GPU slave kernel. The CPU host is a 64-bit quad-core Intel Xeon E5520 having a clock speed of 2.27GHz. The GPU device is Nvidia Tesla C2075 with 448 CUDA cores (14 multiprocessors with 32 cores each), a clock speed of 1.15 GHz, a 6 GB global memory, a 16,384 B shared memory, and a warp size of 32. Both the CPU and GPU are used in single precision.

In the following, we experimentally evaluate the ability of the proposed approaches to speedup resolution time regardless of the benchmark used. Indeed, we compare the hybrid approach on both cores of the CPU and the GPU with a classical sequential mono-core approach and with state-of-the-art exact and approximate methods (APRIORI, FP-GROWTH, SE-GPU, and ME-GPU). Three execution schemes have been considered: sequential execution (SE), computation of parallel fitness on GPU (PC-GPU) and parallel hybrid CPU/GPU execution. In the latter, CPU cores are used for the parallel evolution of the sub-populations and the GPU for the parallel evaluation of the fitnesses. Two variants are considered for these approaches (P and SP). P means that the approach is executed on one population and SP that sub-populations are considered.

6.1 Comparison between the different architectures

The aim of the first test is to compare the execution times of (SE, PC-GPU, PC-CPU/GPU) on the different datasets. Table 2 summaries the results obtained for a population size of 200 individuals, a number of iterations of 100, two point exchange crossover and replacement of parent.

Table 2 Execution times (in seconds) for the different approaches on the two type populations

Dataset	*SE-SP*	*PC-GPU-SP*	*CPU/GPU-SP*	*SE-P*	*PC-GPU-P*
SPEC-heart	03	06	07	**02**	05
Forests	04	08	10	**03**	53
C20d10	13	11	13	**09**	11
Chess	43	21	23	35	**20**
Mushroom	74	15	16	72	**14**
T25i10d10	151	20	20	90	**15**
C73d10	309	56	55	180	**28**
T10i4d100	553	24	23	306	**16**
Retail	452	27	27	328	**20**
T20i6d100	1,191	67	68	889	**49**
Pumsb_ star	1,063	198	189	688	**97**
Connect	1,173	187	180	1,027	**148**
Pumsb	1,610	297	296	**144**	161
Kosarak	3,797	396	226	2,705	**204**
Accidents	5,101	750	765	3,188	**367**

The results show that PC-GPU-P gives better results in terms of speedup on medium and large datasets. The speedup is computed as follows:

$$A = \frac{ST}{PT},$$

where *ST* and *PT* are sequential time and parallel execution time respectively. The speedup increases as the dataset size increases (number of transactions). A speedup of 8 on the *Kosarak* dataset and 7 on *Accidents* compared to the sequential execution are achieved. However, the use of GPU gives bad results on very small datasets. This is due to small ratio between the effective fitness computation and CPU/GPU communication times. Indeed, sending requests and recovering take more time than the effective computation time. We cannot benefit from the parallelism offered by the GPU (see Figure 8).

Figure 8 Evolution of execution time according to the datasets size for one population (see online version for colours)

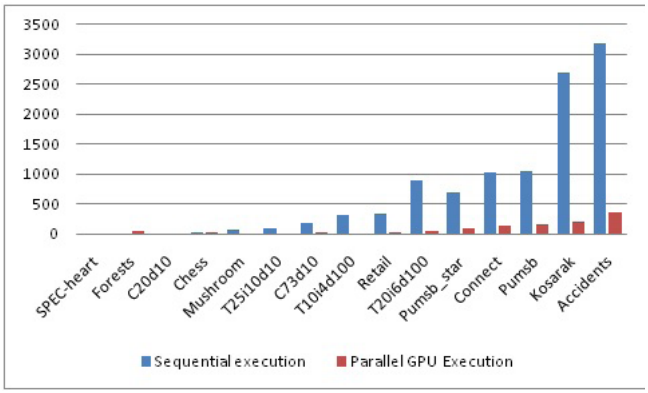
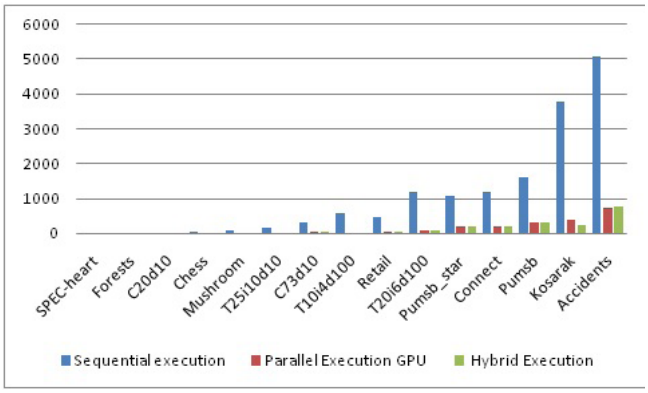


Figure 9 Evolution of the execution time according to datasets size when considering subpopulations (see online version for colours)

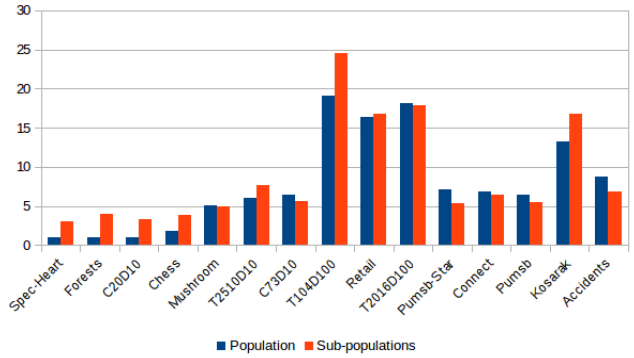


The hybrid version CPU/GPU is based on the progress of the GA-ARM in sub-populations on the CPU and the computation of fitness in parallel on GPU. The execution time taken by the hybrid version can provide a speedup of almost 2 compared to the GPU version on *Kosarak* and a speedup of 16 compared to the sequential one. But it can be also worse than the parallel version on GPU. The reason mainly depends on the average number of items in the transactions. If this number is high, the fitness computation

will be longer. This will increase the risk of simultaneous concurrent requests for the GPU. The simultaneous requests generate additional time due to the blocking and synchronisation of the threads. Figure 9 compares the different execution times for SE-SP, PC-GPU-SP et CPU/GPU-SP.

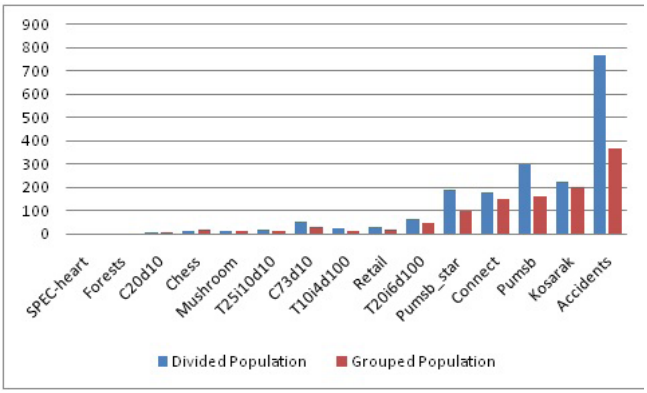
Figure 10 shows the best obtained speedups compared to sequential versions for each population organisation. We clearly note that our approaches were able to achieve a speedups up of 25. The other speedups are not negligible and vary from 6 to 17 for the medium and large instances. This allows to obtain an average speedup of 8.5.

Figure 10 Obtained speedup for each population type (see online version for colours)



Comparing the performances on whole population and sub-populations, we note from Table 2 that GA-ARM on whole population is faster than the execution on sub-populations. Figure 11 compares the best times obtained by GA-ARM on sub-populations and whole population. In general, the times obtained for whole population are better for small database instances. While execution on sub-populations gives better times on medium and large size datasets.

Figure 11 Comparison of the best obtained time for the two population organisation (see online version for colours)



6.2 *Comparison between CPU execution and GPU execution*

In what follows, we compare the results of parallel GA-ARM execution on CPU presented in Hamdad et al. (2019): computation of parallel fitness on CPU (PC-CPU(fitness)-SP), the parallel GA-ARM algorithm on sub-populations (PC-CPU-SP) with our approaches. The size of the population is fixed to 200 chromosomes and the number of iterations to 100. Table 3 shows the results of the different executions on several datasets. Table 3 shows that all types of parallel GPU architectures outperform the parallel CPU one for both population organisations. Note that, parallel CPU execution on subpopulation (PC-CPU-SP) has a shorter execution time for small instances.

Table 3 Comparison of execution time (in seconds) between CPU and GPU parallel versions on the two types of populations

Dataset	*PC-CPU-SP (fitness)*	*PC-CPU-SP (classes)*	*PC-GPU-SP*	*HE-SP*	*PC-CPU (fitness)-P*	*PC-GPU-P*
SPEC-heart	55	01	06	07	54	05
Forests	53	01	08	10	8	53
C20d10	63	04	11	13	58	11
Chess	78	11	21	23	78	20
Mushroom	104	21	15	16	81	14
T25i10d10	136	37	20	20	101	15
C73d10	216	86	56	55	168	28
T10i4d100	333	152	24	23	235	16
Retail	277	127	27	27	215	20
T20i6d100	525	315	67	68	435	49
Pumsb_ star	482	305	198	189	382	97
Connect	520	345	187	180	477	148
Pumsb	638	434	297	296	508	161
Kosarak	1,295	1,072	396	226	1,040	204
Accidents	1,521	1,544	750	765	1,568	367

6.3 Comparative tests with state-of-the-art methods

In this section, we compare our proposed approach GA-ARM-GPU on whole population with the existing exact and approximate methods. The results of the existing methods are often available for fixed thresholds of supports. Taking a smaller *Minsup* will further increase the execution time considerably. In our case, the *Minsup* is only used to filter the association rules and does not affect the extraction time. Table 4 compares the execution times of the APRIORI and FP-GROWTH algorithms (derived from Agrawal and Srikant, 1994; Djenouri et al., 2014) and the execution times obtained by our GA-ARM-GPU method with a whole population (100 iterations applied to a population of 200 chromosomes).

Table 4 Comparison of execution times (in seconds) of GAARM-GPU against APRIORI and FP-GROWTH

Dataset	*APRIORI*	*FP-GROWTH*	*GA-ARM-GPU-P*
Pumsb-star	600	300	**97**
Retail	4,500	3,800	**20**
Connect	2,600	2,900	**148**
Kosarak	3,000	2,900	**204**
T20i6d100	500	-	**49**

Table 5 Execution times (in seconds) using GPU with approximate methods

Dataset	*BSO-ARM*	*SEG*[a]	*ME*[b]	*GA-S*[c]	*GA-GPU*[d]
Chess	149,956	845	95	8	**2**
Mushroom	28,815	1,064	150	13	**2**
Pumsb_star	144,120	2,475	550	85	**19**
Retail	299,658	4,910	1,185	51	**3**
Connect	300,985	5,815	1,485	152	**15**
Kosarak	3258,451	4,525	948	447	**44**

Notes: [a]SE-GPU.
[b]ME-GPU.
[c]Sequential GA-ARM.
[d]GA-ARM-GPU.

The obtained results show that our approach outperforms the two algorithms (APRIORI and FP-GROWTH). For comparison with the approximate methods, unfortunately, most of the tests in literature were carried out only on small datasets. Table 5 compares the results of the BSO-ARM, ME-GPU and SE-GPU (Djenouri et al., 2015) methods which were tested on the same hardware configuration as ours. The number of iterations is set to 100, the size of the population to 200, two-point exchange crossover and replacement of parent were used for all algorithms.

The results clearly show that execution time of both versions of GA-ARM (sequential and parallel) is faster compared to other algorithms. The best reduction is obtained by GA-ARM-GPU.

7 Conclusions

In this work, we have proposed two parallel GAs (ARM-GPU and ARM-CPU/GPU) to speed-up the execution of our genetic algorithm GA-ARM, used to extract a set of relevant rules. This task is time consuming mainly due to the growing size of databases. For this purpose, GA with one rule codifications of chromosomes and population organisations (whole or sub-populations) were considered. In ARM-GPU, we used parallelism to compute the fitness which is the most consuming task. However, running the GA on GPU does not respect the SIMD architecture, causes many thread divergences and involves very little use of the GPU's capabilities. Our second proposition, GA-ARM-CPU/GPU is a two levels-based parallel approach which uses CPU/GPU for hybrid parallelism. It uses CPU for parallel execution on sub-populations and GPU for the parallel computation of fitness. The performance tests allowed us to observe the accelerations given by the different GA-ARM. Parallelism on GPU has been advantageous, especially on large datasets, reaching an acceleration of up to 25. The comparison with exact algorithms (APRIORI and FP-GROWTH) has been largely in favour of our parallel algorithms. In addition, the comparison with state-of-the-art parallel metaheuristics (ME-GPU and SE-GPU) also showed the contribution of our GA-ARM algorithm.

References

Adil, S.H. and Qamar, S. (2009) 'Implementation of association rule mining using CUDA', *Emerging Technologies. ICET 2009*, IEEE.

Agrawal, R. and Srikant, R. (1994) 'Fast algorithms for mining association rules', *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, Vol. 1215, pp.487–499.

Bruno, E. (2008) *Building Multi-Core Ready Java Applications* [online] https://dzone.com/articles/building-multi-core-ready-java (accessed 13 April 2016).

Cahon, S., Melab, N. and Talbi, E.G. (2004) 'ParadisEO: a framework for the reusable design of parallel and distributed metaheuristics', *Journal of Heuristics*, Vol. 10, No. 3, pp.357–380.

Cano, A., Luna, J.M. and Ventura, S. (2013) 'High performance evaluation of evolutionary-mined association rules on GPUs', *The Journal of Supercomputing*, Vol. 66, No. 3, pp.1438–1461.

Choksuchat, C. and Chantrapornchai, C. (2018) 'Various GPU memory utilisation exploration for large RDF search', *International Journal of Computational Science and Engineering*, Vol. 17, No. 4, pp.398–410, Inderscience.

Courtier, R. (2013) *Designing Scientific Applications on GPUs*, Chapman and Hall/CRC Press, Francis and Taylor Group.

Cui, Q. and Guo, X. (2013) 'Research on parallel association rules mining on GPU', *Proceedings of the 2nd International Conference on Green Communications and Networks 2012 (GCN 2012): Volume 2. Lecture Notes in Electrical Engineering*, Vol. 224, Springer, Berlin, Heidelberg.

Dehuri, S., Jagadev, A.K. Ghosh, A. and Mall, R. (2006) 'Multi-objective genetic algorithm for association rule mining using a homogeneous dedicated cluster of workstations', *American Journal of Applied Sciences*, Vol. 3, No. 11, pp.2086–2095.

Djenouri, Y., Bendjoudi, A., Mahdi, M. et al. (2015) 'GPU-based bees swarm optimization for association rules mining', *The Journal of Supercomputing*, Vol. 71, No. 4, pp.1318–1344.

Djenouri, Y., Bendjoudi, A., Mehdi, M. et al. (2014) 'Parallel association rules mining using GPUS and bees behaviors', *2014 6th International Conference on Soft Computing and Pattern Recognition (SoCPaR)*, pp.401–405, IEEE.

Djenouri, Y., Belhadi, A., Fournier-Vigerd, F. and Fujitae, H. (2018a) 'Mining diversified association rules in big datasets: a cluster/GPU/genetic approach', *Information Science*, Vol. 459, pp.117–134.

Djenouri, Y., Djenouri, D., Habbas, Z. and Belhadi, A. (2018b) 'How to exploit high performance computing in population-based metaheuristics for solving association rule mining problem', *Distributed and Parallel Databases*, Vol. 36, No. 2, pp.369–397.

Djenouri, Y., Djenouri, D., Belhadi, A. and Fournier-Vigerd, F. (2018c) 'GBSO-RSS: GPU-based BSO for rules space summarization', *ICBDL 2018: Big Data Analysis and Deep Learning Applications*, pp.123–129.

Duran, R., Chen, D., Saraswat, R. and Hallmark, A. (2014) 'A framework for comparing high performance computing technologies', *International Journal of Computational Science and Engineering*, Vol. 9, Nos. 1–2, pp.119–129, Inderscience.

Fang, W., Lu, M. et al. (2009) 'Frequent itemset mining on graphics processors', *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, ACM.

Garg, R. and Mishra, P.K. (2011) 'Exploiting parallelism in association rule mining algorithms', *Int. J. Adv. Technol.*, Vol. 2, pp.222–232.

Goethals, B. (2004) *Frequent Itemset Mining Implementations Repository* [online] http://.mi.ua.ac.be/data/ (accessed 20 April 2016).

Hamdad, L., Benatchba, K., Bendjoudi, H. and Ournani, Z. (2019) 'Impact of genetic algorithms operators on association rules extraction', in Rojas, I., Joya, G. and Catala, A. (Eds.): *Advances in Computational Intelligence. IWANN 2019. Lecture Notes in Computer Science*, Vol. 11507, pp.747–759, Springer, Cham.

Han, J., Pei, P. and Yin, Y. (2000) 'Mining frequent patterns without candidate generation', *ACM Sigmod Record*, Vol. 29, No. 2, pp.1–12.

Indira, K. and Kanmani, S. (2012) 'Performance analysis of genetic algorithm for mining association rules', *International Journal of Computer Science Issues*, Vol. 9, No. 2, pp.368–376.

Jourdan, L. (2003) *Métaheuristiques pour l'extraction de connaissances: Application Ãă la génomique*, PhD, Université des Sciences et Technologie de Lille I.

Kabir, M.M.J., Xu, S., Kang, B.H. and Zhao, Z. (2015) 'A new evolutionary algorithm for extracting a reduced set of interesting association rules', *International Conference on Neural Information Processing*, pp.133–142, Springer International Publishing.

Khademolghorani, F., Baraani, A. and Zamanifar, K. (2011) 'Efficient mining of association rules based on gravitational search algorithm', *International Journal of Computer Science Issues*, Vol. 8, No. 2, pp.51–58.

Kuo, R.J., Chao, C.M. and Chiu, Y.T. (2011) 'Application of particle swarm optimization to association rule mining', *Applied Soft Computing*, Vol. 11, No. 1, pp.326–336.

Li, J., Chen, Q. and Liu, B. (2019) 'A static analytical performance model for GPU kernel', *International Journal of Computational Science and Engineering*, Vol. 18, No. 2, pp.201–210, Inderscience.

Loukil, L., Mehdi, M., Bendjoudi, A. and Melab, N. (2013) 'Parallel GPU-accelerated metaheuristics', *Designing Scientific Applications on GPUs*, CRC Press, Taylor and Francis Group.

Mata, J., Alvarez, J.L. and Riquelme, J.C. (2001) 'Mining numeric association rules with genetic algorithms', *Artificial Neural Nets and Genetic Algorithms*, Springer, Vienna.

Mata, J., Alvarez, J.L. and Riquelme, J. C. (2002) 'Discovering numeric association rules via evolutionary algorithm', in Chen, M.S., Yu, P.S. and Liu, B. (Eds.): *Advances in Knowledge Discovery and Data Mining. PAKDD 2002. Lecture Notes in Computer Science*, Vol. 2336, Springer, Berlin, Heidelberg.

Melab, N., Luong, T.V., Boufaras, K. and Talbi, K.J. (2011) Towards ParadisEOMO-GPU: a framework for GPU-based local search metaheuristics', *Advances in Computational Intelligence*, Vol. 6691 of *Lecture Notes in Computer Science*, pp.401–408.

Moslehi, P., Bidgoli, B.M., Nasiri, M. and Salajegheh, A. (2011) 'Multi-objective numeric association rules mining via ant colony optimization for continuous domains without specifying minimum support and minimum confidence', *International Journal of Computer Science*, Vol. 8, Issue 5, No. 1, pp.34–41.

Nashed, Y.S.G., Ugolotti, R., Mesejo, P. and Cagnoni, S. (2012) 'libCudaOptimize: an open source library of GPU-based metaheuristics', *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion, GECCO Companion '12*, pp.117–124.

Soca, N., Blengio, J.L., Pedemonte, M. and Ezzatti, P. (2010) 'PUGACE, a cellular evolutionary algorithm framework on GPUs', *IEEE Congress on Evolutionary Computation (CEC)*.

Wang, M., Zou, Q. and Liu, C (2011) 'Multi-dimension association rule mining based on adaptive genetic algorithm', *2011 International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE)*, IEEE.

Wang, Y., Xu, T., Xue, S., Shen, Y. et al. (2018) 'D2P-Apriori: a deep parallel frequent itemset mining algorithm with dynamic queue', *Tenth International Conference on Advanced Computational Intelligence (ICACI)*, IEEE.

Yan, X., Zhang, C. and Zhang, S. (2009) 'Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support', *Expert Systems with Applications*, Vol. 36, No. 2, pp.3066–3076.

Zaki, M.J. (1999) 'Parallel and distributed association mining: a survey', *IEEE Concurrency*, Vol. 4, No. 14, pp.14–25.

Zhang, F., Zhang, Y. et al. (2011) 'Gpapriori: GPU-accelerated frequent itemset mining', *2011 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE.

Zhou, J., Yu, K.M. et al. (2010) 'Parallel frequent patterns mining algorithm on GPU', *2010 IEEE International Conference on Systems, Man and Cybernetics*.