Parallel Association Rules Mining Using GPUs and Reptile Search Algorithm

Abderrahim Boukhalat[1,2]✉, KamelEddine Heraguemi[3], Mohamed Benouis[1] ,Brahim Bouderah[1] and Samir Akhrouf[1]
[1]*University of M'Sila, Computer Science Department, M'Sila, Algeria*
{Abderrahim.Boukhalat, Mohamed Benouis, Brahim Bouderah, Samir Akhrouf} @univ-msila.dz
[2] *LIM Laboratory, ,University of Souk Ahras ,Computer Science Department, Algeria*
[3] *National School of Artificial Intelligence, Algiers, Algeria Algeria*
kameleddine.heraguemi@ensia.edu.dz

Abstract

This paper proposes a novel approach to accelerate association rule mining using the Reptile Search Algorithm (RSA) in conjunction with GPU-based parallel processing. Traditional association rule mining techniques can be computationally expensive, especially with large datasets. By utilizing the inherent parallelism of Graphics Processing Units (GPUs), we significantly speed up the fitness evaluation process, a core component of the Reptile Search Algorithm. Our results show a marked improvement in the performance of RSA on large datasets, making it feasible for real-time or large-scale applications such as market basket analysis, healthcare for drug interaction analysis, and web usage mining. We also analyze the impact of various GPU optimizations and present a comparison with CPU-based execution.

.

Keywords: *Data mining, Association rule mining, parallel computing, GPU, Reptile Search Algorithm*

1 Introduction

Association Rule Mining (ARM) consists of extracting useful correlations among items in large transactional databases. It is one of the most important tasks in supervised learning. It is used in many fields, including market basket analysis, healthcare for drug interaction analysis, web usage mining, and more. The two most widely used algorithms for ARM are Apriori. [1] and FP-Growth [2]. These are exact algorithms that generate frequent itemsets from which association rules are built. However, generating frequent item sets is an NP-hard problem, making it time-consuming, especially as the size of the dataset grows. Consequently, many metaheuristic algorithms have been proposed in the literature to extract association rules more efficiently. Examples include bee swarm optimization for ARM (BSO) [3], particle swarm optimization (PSO) [4], ant colony optimization (ACO) [5], and Reptile search algorithm [6].

The Reptile Search Algorithm (RSA) is a bio-inspired optimization technique that simulates the hunting and adaptive behavior of reptiles. RSA has been successfully applied to a variety of NP-complete problems across fields such as engineering, finance, and logistics due to its ability to balance exploration and exploitation in search spaces. RSA has been explored in the context of ARM by Abderrahim et al.[7] .Applying RSA-ARM (Reptile Search Algorithm for Association Rules Mining) offers a promising new direction for optimizing rule-mining processes. However, the computational complexity involved in using RSA-ARM on large datasets necessitates innovative approaches.

Association rule mining, especially when combined with metaheuristic algorithms like RSA, is computationally expensive due to the large number of candidate rules that must be evaluated in each iteration of the algorithm. Traditional CPU implementations need help to scale effectively with the increasing size of datasets, especially for ARM tasks combined with complex optimization algorithms like RSA-ARM.[8]. This is due to the inherently sequential nature of CPU computation, which limits parallelism and slows down the process as the dataset grows.[9]. On the other hand, Graphics Processing Units (GPUs)[10], designed for high-throughput parallel computation, provide a promising solution to accelerate the fitness evaluation process by evaluating multiple rules in parallel. Graphics Processing Units (GPUs) offer a powerful solution due to their ability to handle massive parallelism, allowing multiple calculations to occur simultaneously. This makes GPUs well-suited for the high-dimensional computations required by ARM, offering a scalable solution that significantly reduces execution time compared to CPUs.

GPUs have been widely adopted in various computationally intensive tasks, such as deep learning and high-performance computing, due to their ability to perform parallel operations across thousands of cores. Recent studies have demonstrated the significant speedups that can be achieved by parallelizing frequent pattern mining and ARM algorithms on GPUs [11]. To address the computational challenges posed by large datasets, we propose a novel approach that integrates the Reptile Search Algorithm for ARM (RSA-ARM) with GPU parallelism to accelerate computations. In our work, we focus on parallelizing the fitness evaluation function of RSA-ARM, which is the most computationally expensive part of the algorithm. By offloading the fitness evaluation to the GPU using Nvidia CUDA(Compute Unified Device Architecture)[12], we can evaluate multiple candidate rules simultaneously, significantly reducing the time required to mine association rules.

This paper presents the following contributions:

- We present the first implementation of the Reptile Search Algorithm (RSA) for Association Rule Mining (ARM) in GPUs.
- By leveraging the parallel architecture of GPUs, we significantly speed up the fitness evaluation process, enabling RSA to scale to larger datasets.
- We evaluate the performance of the proposed approach on both GPU and CPU, highlighting the speedup achieved through GPU parallelism and analyzing the scalability of the algorithm across different datasets.

The structure of this paper is as follows: Section 2 provides a review of related work on association rule mining, RSA, and parallel GPU computing. Section 3 outlines the methodology, including the integration of RSA-ARM with GPU-based parallelism. Section 4 proposes the parallel approach employed in this study. Section 5 presents the performance evaluation of experimental results and compares the efficiencies of CPU and GPU implementations. Finally, Section 6 concludes the paper and discusses potential directions for future research.

2 Related Work

2.1 Sequential ARM Algorithms:

Association Rule Mining (ARM) has long been studied using sequential algorithms, most notably exact approaches such as Apriori, FP-Growth, DIC, and DHP. These algorithms are highly effective for small to medium-sized datasets but are limited when dealing with large datasets due to their time and space complexity. Apriori, for example, involves multiple passes over the dataset to generate candidate itemsets, making it prohibitively time-consuming for larger datasets. Similarly, FP-Growth, which compresses the dataset into a compact tree structure, suffers from significant memory overhead when handling large-scale data.

To address these limitations, researchers have explored metaheuristic-based approaches for ARM. Algorithms like Genetic Algorithms for Association Rules (GAR)[13], Particle Swarm Optimization for ARM (PSO-ARM)[14], and Bees Swarm Optimization for ARM (BSO-ARM)[15] Have demonstrated improved performance for large datasets, balancing solution quality and execution time. These methods aim to navigate the vast solution space more efficiently but encounter difficulties with extremely large datasets, such as the WebDocs benchmark, which contains over 1.5 million transactions.

2.2 Algorithms For Parallel Processing:

With the increasing size of datasets, parallelization has become a critical solution for enhancing the performance of ARM algorithms. Early attempts at parallelizing ARM focused on exact methods like Apriori. [16]Agrawal and Shafer introduced parallel Apriori algorithms using strategies like count distribution (CD) and data distribution (DD), where the workload is divided among several processors. Extensions of these methods, such as the Intelligent Data Distribution (IDD) and Hybrid Distribution (HD) models, sought to optimize communication between processors.[17]. However, the limitations of the underlying hardware still bound these approaches.

FP growth has also been parallelized to improve performance on large datasets.[18]. Algorithms such as Multiple Local Frequent Pattern Trees (MLFPT) and distributed FP-Growth achieved some success by distributing the workload across multiple processors and merging local results.[19]. However, the frequent itemset mining task, even when parallelized, remained challenging for large datasets due to the memory requirements of storing FP trees and the computational complexity of pattern discovery.

In addition to exact approaches, parallel metaheuristic algorithms have been developed. Melab et al.[20] They introduced a parallel genetic algorithm for ARM called PGARM, using a master-slave architecture. However, this approach required replicating the entire database on each processor, leading to significant memory usage and potential inefficiency when dealing with very large datasets. Similarly, Mishra et al. [21] IPMOGA divided the search space into regions and assigned them to different processors. However, partitioning the search space efficiently remains a challenge.

In parallel and distributed association rule mining in life science: A novel parallel algorithm to mine genomics data. This paper introduces BPARES (Balanced Parallel Association Rule Extractor from SNPs)[22], a novel algorithm designed to improve the performance of Association Rule Mining (ARM) on biological datasets, particularly those containing Single Nucleotide Polymorphisms (SNPs). Traditional ARM algorithms face significant challenges with execution time, even on small datasets. BPARES leverages parallel computing and a new balancing strategy to optimize response time, efficiently utilizing computational power and reducing memory consumption while maintaining accuracy. The paper also includes a comparison of various sequential, parallel, and distributed ARM techniques.

2.3 GPU resources for parallel processing

With the rise of GPU computing, ARM algorithms have been significantly accelerated by leveraging the massive parallelism that GPUs offer. Wenbin et al. introduced two parallel versions of Apriori for GPU architectures: Pure Bitmap Implementation (PBI) and Trie Bitmap Implementation (TBI) [23] . These implementations transformed datasets into bitmap structures to optimize memory usage and parallelism, achieving speedups of 10x to 15x in their experiments. Syed et al. further improved GPU-based Apriori by implementing a two-step process that first computes itemset support on the GPU before transferring the results back to the CPU for rule generation. This approach showed a 20x speedup but was limited by the cost of CPU-GPU communication. Other GPU-based implementations, such as the

GPApriori algorithm [24], used compact data structures to accelerate item set counting and reduce memory usage, achieving up to 100x speedups on small datasets. However, as the number of transactions increases, the performance benefits diminish due to thread divergence and memory bottlenecks.

Parallel metaheuristic algorithms have also benefited from GPU acceleration. Cano et al.[25] They proposed a GPU-accelerated evolutionary algorithm for ARM, parallelizing the fitness evaluation step to allow multiple rules to be evaluated simultaneously on the GPU. This approach reduced computation time, but the number of rules that could be evaluated in parallel was limited by the GPU's memory capacity. Strategies to overcome these limitations, such as using multiple GPUs or more efficient memory management techniques, have shown promise in recent research.

Recent works have focused on optimizing the performance of ARM algorithms on GPUs by addressing the memory and communication bottlenecks inherent in large-scale data mining. For instance, Jiayi et al.[26] They introduced the Mempack strategy for compressing itemsets and reducing memory fetching operations during ARM on GPUs. While this approach showed speedups of up to 15x, it was limited by the complexity of the data structures involved. Other approaches have explored clustering techniques, such as the probabilistic frequent itemset mining approach.[27], which partitioned the dataset across GPU clusters. However, load balancing between nodes remained a challenge, impacting overall performance.

The introduction of swarm intelligence algorithms, such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), has also been adapted for GPU architectures. These algorithms exploit the multi-threading capabilities of GPUs, allowing large numbers of swarm agents to explore the solution space concurrently. Weiss et al. implemented ANtMinerGPU.[28], an ACO algorithm where each ant is mapped to a GPU thread, demonstrating that larger swarms significantly improved solution quality. Similarly, Zhou et al.[29] We have developed a fully GPU-implemented PSO algorithm, where each dimension of the fitness function is evaluated in parallel, showing that the size of the swarm plays a crucial role in the parallel efficiency of the algorithmS. In the work by Cecilia et al.[30], they applied this hierarchical approach to solving the Traveling Salesman Problem (TSP) using a GPU-based ACO algorithm. In this setup, two categories of artificial ants were defined: queen ants and worker ants. The queen ants are associated with CUDA thread blocks, and the worker ants are individual CUDA threads that assist the queen ants in making decisions during the tour construction step of the algorithm. This collaboration between queen ants and worker ants helps accelerate the decision-making process, especially in selecting the next city to visit in the TSP. By using this strategy, the algorithm achieves better GPU resource utilization without requiring an excessively large swarm, which is beneficial for improving both performance and efficiency.

This hierarchical organization contrasts with the flat implementations, like those used in ANtMinerGPU , where each thread is assigned a single agent, potentially leading to under-utilization of the GPU's computational power if the swarm size is not large enough. By organizing threads hierarchically, the computational load is more evenly distributed, resulting in higher GPU occupancy and faster execution, particularly for complex tasks like ACO.

In [31], in parallel Association rule mining using GPUs and Bees behaviors, the authors propose a parallel GPU-based extension of the Bees Swarm Optimization algorithm for Association Rule Mining. While the CPU handles the solution generation and search processes, the evaluation of solutions is parallelized on the GPU to take advantage of its massive threading capabilities. The experimental results demonstrate that this parallel approach significantly outperforms the sequential version in terms of efficiency and execution time, showcasing the potential of GPU-based optimization in ARM.

In GPU-based Bees Swarm Optimization for Association Rules Mining[32], the paper introduces two GPU-based Bees Swarm Optimization algorithms for Association Rule Mining: SE-GPU and ME-GPU. SE-GPU evaluates one rule at a time, assigning each thread to a transaction, while ME-GPU evaluates multiple rules concurrently, with each thread handling several transactions. These approaches leverage the parallel processing power of GPUs to address the computational challenges posed by large datasets, such as the WebDocs benchmark. Experiments conducted on an Intel Xeon processor coupled with an Nvidia Tesla GPU demonstrate up to 100× speed improvement compared to sequential versions. Additionally, the GPU-based algorithms outperformed multi-core CPU versions, proving more efficient across various core counts. This study highlights the significant advantages of GPU-based parallelism for large-scale ARM.

In 2020[33], researchers introduced two parallel Genetic Algorithms for ARM: ARM-GPU and ARM-CPU/GPU. ARM-GPU accelerates fitness evaluation using GPU parallelism, while ARM-CPU/GPU applies a two-level parallelism where CPU cores evolve sub-populations and offload fitness computation to the GPU. These methods significantly outperform traditional exact algorithms (Apriori, FP-Growth) and other parallel approaches (SE-GPU, ME-GPU) in terms of execution time, especially for large datasets. This demonstrates the effectiveness of combining CPU and GPU parallelism to handle the computational demands of large-scale ARM tasks.

In [34], the work introduced NRFP-Growth, a non-recursive version of FP-Growth, and GPFP-Growth, a parallel GPU-based variant. NRFP-Growth reduces time and space complexity by using FP-array and ItemPoss-map structures, while GPFP-Growth leverages GPU parallelism to accelerate frequent itemset mining. Tests on various datasets showed that NRFP-Growth improved performance over the classical FP-Growth, and GPFP-Growth provided significant speedups and scalability, making it suitable for large datasets.

Despite these advances, several challenges remain. Memory limitations on GPUs continue to constrain the size of datasets that can be processed efficiently. Additionally, the overhead of frequent data transfers between the CPU and GPU limits the potential speedups in some applications . Researchers are now investigating strategies to minimize these communication costs, such as by performing most computations on the GPU and reducing the need for CPU intervention.

3 Reptile Search Algorithm for ARM

The Reptile Search Algorithm (RSA) was introduced by Abualigah et al. in 2022 as a novel metaheuristic inspired by the hunting strategies of reptiles, particularly their encircling and cooperation behaviors. RSA has shown remarkable success in solving various NP-complete optimization problems, ranging from energy optimization to image processing.

Recently, RSA was applied to ARM by Boukhalat et al. in their work titled "Reptile Search Algorithm for Association Rule Mining."[7]. Their approach demonstrated that RSA could outperform traditional metaheuristics, such as BSO and genetic algorithms, in terms of rule quality, execution time, and memory efficiency. This application of RSA to ARM introduced a novel way to address the computational complexity of mining association rules from large datasets.

In the context of ARM, RSA is used to find the optimal association rules by evaluating candidate rules based on two key metrics:

- **Support**: The support of an association rule $X \rightarrow Y$ is determined by the frequency of occurrence of a rule's antecedent and consequent in the dataset, as shown in Eq (1).

$$\text{Support}(X \rightarrow Y) = \frac{P(X \cup Y)}{|D|}$$

- **Confidence**: The confidence (Conf) for the rule $X \rightarrow Y$, as outlined in Eq. (2), The likelihood that the consequent of the rule occurs given the antecedent.

$$\text{Confidence}(X \rightarrow Y) = \frac{P(X \cup Y)}{P(X)}$$

The core components of RSA in ARM are:

- **Rule Representation**: Each rule is represented as a candidate solution in RSA. The rule's antecedent and consequent form the search space, which RSA explores.
- **Fitness Function**: The fitness of each rule is evaluated based on the support and confidence metrics as in Eq (3). The goal is to maximize both metrics to generate high-quality association rules.

$$F(r) = \begin{cases} \alpha * \text{Sup}(r) + \beta * \text{Conf}(r) & \text{if accepted} \\ -1 & \text{otherwise} \end{cases}$$

4 Proposed parallel approaches

4.1 Motivation for GPU-Accelerated RSA-ARM

Despite the success of metaheuristic algorithms like our previous RSA-ARM, as shown in Algorithm 1[7], the scalability of these algorithms remains a challenge, particularly when applied to large datasets. To address this, researchers have turned to parallel computing, utilizing multi-core CPUs and GPUs to reduce execution times. Graphics Processing Units (GPUs), with their high degree of parallelism, are particularly well-suited for tasks that involve repetitive computations, such as the evaluation of candidate rules in ARM.

Several studies have demonstrated the effectiveness of using GPUs to parallelize ARM algorithms. For instance, Djenouri et al. proposed a GPU-based implementation of BSO-ARM, which showed significant performance improvements over CPU-based implementations. Similarly, Syed and Qamar [35] demonstrated the effectiveness of parallelizing the Apriori algorithm using CUDA to achieve substantial speedups in support counting. However, no previous work has applied GPU parallelism to RSA for ARM, leaving an important gap in the literature for further performance optimization.

The fitness function in RSA-ARM, which evaluates the quality of association rules based on support and confidence, is computationally expensive. With larger datasets, the iterative nature of RSA further increases the complexity, making it difficult to achieve real-time performance using traditional CPU-based implementations. Therefore, by offloading this computationally intensive part of the algorithm to a GPU, the evaluation of candidate rules can be parallelized, resulting in a significant reduction in execution time.

Our work seeks to address this gap by implementing RSA-ARM on GPUs, parallelizing the evaluation of candidate rules and fitness functions to enhance scalability and performance on large datasets.

4.2 Parallelizing RSA-ARM Using the Master-Slave Paradigm

Parallelizing the RSA-ARM algorithm is based on the Master-Slave paradigm, also known as Master-Worker or Manager-Worker.[36]. This is a common parallel computing design pattern used in distributed systems to divide computational tasks across processors effectively. In this strategy, the CPU acts as the master, while the GPU functions as the slave.

The Master (CPU) coordinates the entire computational process, managing the generation of candidate solutions (rules represented by reptiles), distributing computationally intensive tasks (like fitness evaluations) to the GPU, and updating global information such as the best fitness or solution based on results received from the GPU. The Slaves (GPU) handle the execution of these computationally intensive tasks, distributing the workload across thousands of GPU cores. Each core performs computations in parallel, significantly speeding up the overall process.

4.3 Implementation in RSA-ARM

The CPU(Master) generates candidate solutions represented as reptiles in RSA-ARM. It manages the iterative flow of the algorithm, offloading computationally heavy parts (like fitness evaluation) to the GPU for parallel execution. Once results are received from the GPU, the CPU updates global information, such as the best fitness or solution found so far.

The GPU(Slave) performs the actual computations, including evaluating the support and confidence of candidate rules. The computational workload is distributed across the thousands of GPU cores, ensuring efficient parallel execution. In RSA-ARM, tasks such as fitness function evaluation and rule updates (high walking, belly walking, and hunting cooperation) are executed concurrently by the GPU cores.

As shown in Figure 1, the CPU (master) generates candidate solutions (rules represented by reptiles) and manages the iterative flow of the algorithm. It sends computationally expensive parts, such as fitness evaluations, to the GPU (slave) for parallel execution. After receiving results from the GPU, the CPU updates global information, such as the best fitness or solution. The GPU (slave) distributes the computational workload across its cores, performing calculations in parallel to optimize performance.

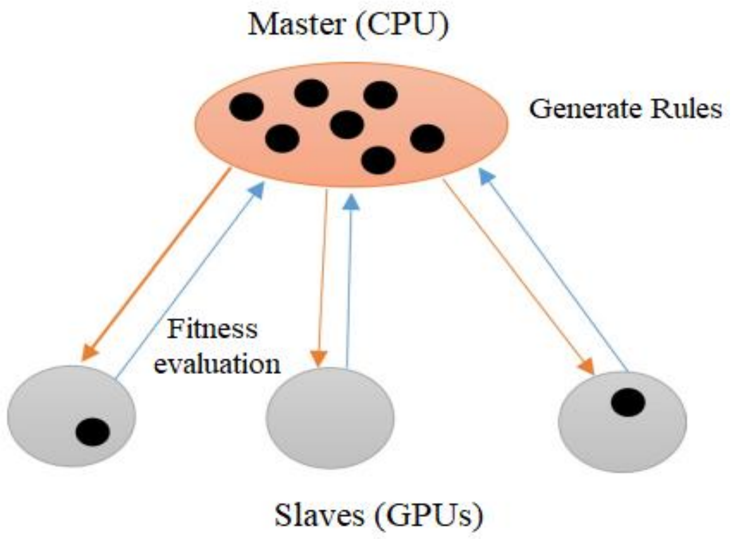


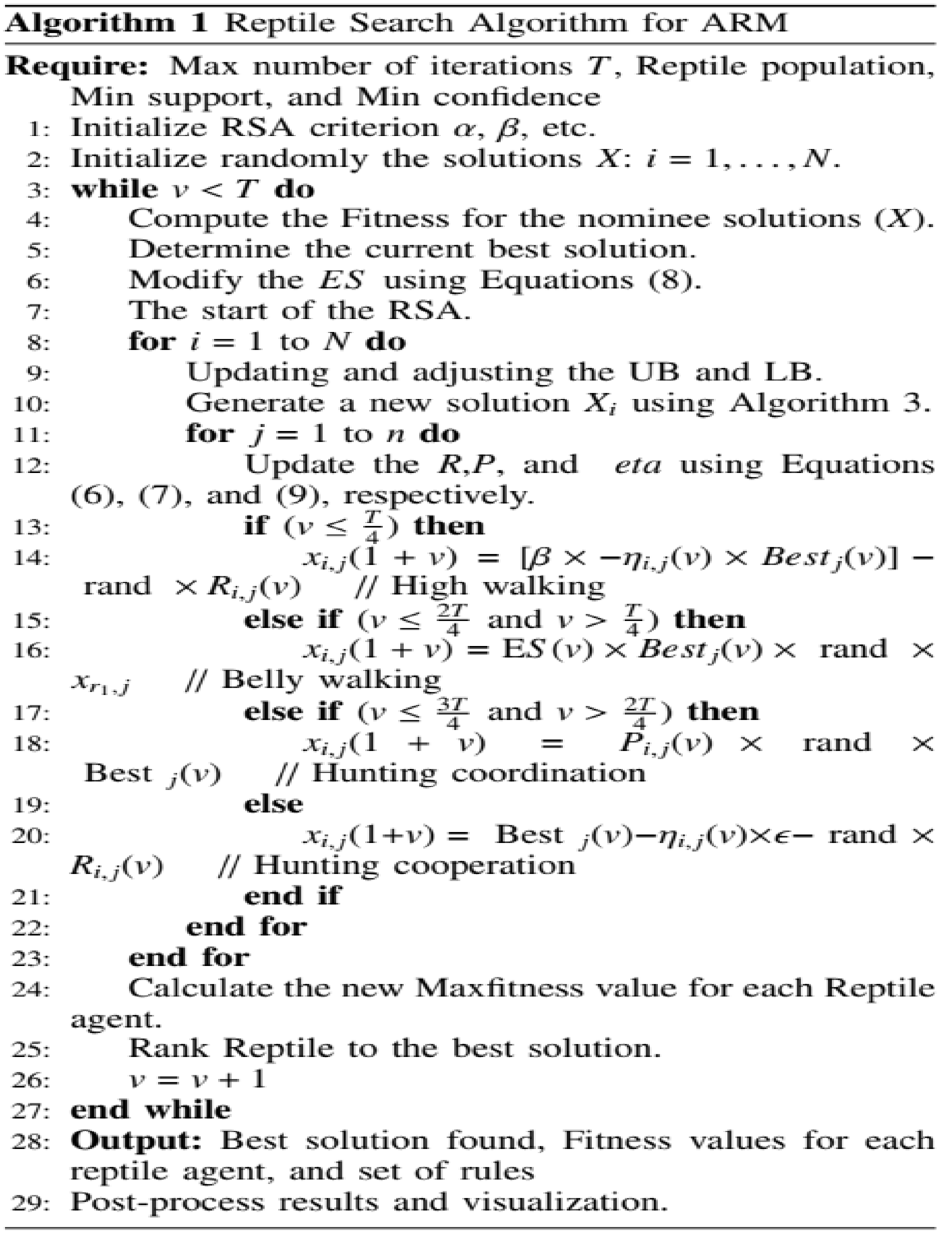
Figure 1. Master/Slave paradigm

4.4 Parallel Fitness Evaluation on GPU

Algorithm 2 shows that the fitness function evaluation is parallelized using the GPU to enhance the computational efficiency of RSA-ARM. At the start of the computation, the transaction data is transferred to the GPU memory to minimize overhead. This data remains in GPU memory throughout the process, ensuring minimal data transfer between the CPU and GPU. The dataset is split among multiple GPU threads, with each thread processing a portion of the dataset to calculate the support and confidence for different candidate rules. Each thread runs the fitnessKernel (Algorithm 3), which computes support and confidence for a candidate rule, and the results are aggregated across threads, enabling faster fitness evaluations compared to a serial CPU implementation. The GPU's parallelization distributes fitness calculations across thousands of cores, significantly reducing execution time compared to CPU-only implementations.

4.5 Parallel Reptile Movements on GPU

In RSA-ARM, reptiles explore and exploit the search space using specific strategies like high walking, belly walking, and hunting cooperation. These strategies involve updating candidate solutions (rules) during each iteration. The movement strategies of RSA are parallelized by assigning each GPU thread to a reptile agent, ensuring that rule updates are performed concurrently. A separate CUDA kernel is designed to handle these reptile movements, with each thread applying the rule update logic in parallel, making the algorithm more scalable and efficient. In RSA-ARM, parallelization of rule updates ensures that computationally intensive parts of the algorithm are efficiently handled by the GPU, allowing concurrent updates of multiple candidate solutions.



```
Algorithm 1 Reptile Search Algorithm for ARM
Require: Max number of iterations $T$, Reptile population,
    Min support, and Min confidence
1: Initialize RSA criterion $\alpha$, $\beta$, etc.
2: Initialize randomly the solutions $X$: $i = 1, \dots, N$.
3: while $v < T$ do
4:     Compute the Fitness for the nominee solutions ($X$).
5:     Determine the current best solution.
6:     Modify the $ES$ using Equations (8).
7:     The start of the RSA.
8:     for $i = 1$ to $N$ do
9:         Updating and adjusting the UB and LB.
10:        Generate a new solution $X_i$ using Algorithm 3.
11:        for $j = 1$ to $n$ do
12:            Update the $R$,$P$, and  $eta$ using Equations
    (6), (7), and (9), respectively.
13:            if ($v \leq \frac{T}{4}$) then
14:                $x_{i,j}(1+v) = [\beta \times -\eta_{i,j}(v) \times Best_j(v)] -$
    rand $\times R_{i,j}(v)$    // High walking
15:            else if ($v \leq \frac{2T}{4}$ and $v > \frac{T}{4}$) then
16:                $x_{i,j}(1+v) = ES(v) \times Best_j(v) \times$ rand $\times$
    $x_{r_1,j}$    // Belly walking
17:            else if ($v \leq \frac{3T}{4}$ and $v > \frac{2T}{4}$) then
18:                $x_{i,j}(1+v) = P_{i,j}(v) \times$ rand $\times$
    Best $_j(v)$    // Hunting coordination
19:            else
20:                $x_{i,j}(1+v) =$ Best $_j(v) - \eta_{i,j}(v) \times \epsilon -$ rand $\times$
    $R_{i,j}(v)$    // Hunting cooperation
21:            end if
22:        end for
23:    end for
24:    Calculate the new Maxfitness value for each Reptile
    agent.
25:    Rank Reptile to the best solution.
26:    $v = v + 1$
27: end while
28: Output: Best solution found, Fitness values for each
    reptile agent, and set of rules
29: Post-process results and visualization.
```

Figure 2. RSA-ARM[7]

Algorithm 2 Parallel RSA-ARM Using the Master-Slave Paradigm

```
Require: Dataset $D$, Number of reptiles $R$, Min support minsup, Min confidence minconf, Max number of generations $G$, GPU configuration with $T$ threads
 1: Initialize RSA parameters $\alpha$, $\beta$, $\alpha_{\text{Reptile}}$, $\beta_{\text{Reptile}}$
 2: Generate initial set of candidate solutions (reptiles) $X_i$, for $i = 1, 2, ..., R$
 3: Transfer dataset $D$ to GPU memory
 4: Set generation $g = 0$
 5: while $g < G$ do
 6:     Master (CPU) Coordination
 7:     for each reptile $i = 1, 2, ..., R$ do
 8:         Generate candidate solutions and manage updates using movement strategies
 9:         Offload computationally heavy tasks (fitness evaluation) to GPU
10:     end for
11:     Parallel Fitness Evaluation on GPU
12:     for each GPU thread $t = 1, 2, ..., T$ do
13:         Split dataset $D$ among $T$ GPU threads
14:         Calculate support and confidence for each rule
15:         if support $\geq$ minsup and confidence $\geq$ minconf then
16:             Compute fitness: Fitness($r$) = $\beta \times$support($r$)+ $\alpha \times$ confidence($r$)
17:         end if
18:     end for
19:     Aggregate fitness results across GPU threads and return to CPU
20:     Master (CPU) Updates
21:     for each reptile $i = 1, 2, ..., R$ do
22:         Update global best fitness value and solution
23:         for each attribute $j = 1, 2, ..., n$ do
24:             Update reptile positions based on movement strategies:
25:             if $g \leq \frac{G}{4}$ then
26:                 High Walking: $X_i[j] = X_i[j] - \eta_i \times \beta_{\text{Reptile}} - R_i[j] \times \text{rand}()$
27:             else if $\frac{G}{4} < g \leq \frac{2G}{4}$ then
28:                 Belly Walking: $X_i[j] = X_i[j] \times X_{\text{random}}[j] \times \text{ES} \times \text{rand}()$
29:             else if $\frac{2G}{4} < g \leq \frac{3G}{4}$ then
30:                 Hunting Coordination: $X_i[j] = X_i[j] \times P_i[j] \times \text{rand}()$
31:             else
32:                 Hunting Cooperation: $X_i[j] = X_i[j] - \eta_i \times \text{Double.MIN} - R_i[j] \times \text{rand}()$
33:             end if
34:         end for
35:     end for
36:     Reptile Movement Parallelization on GPU
37:     for each GPU thread $t = 1, 2, ..., T$ do
38:         Assign reptile agent to thread and apply movement strategies concurrently
39:     end for
40:     Update reptile solutions and increment generation: $g = g + 1$
41: end while
42: Output: Best solution found, Fitness values for each reptile agent, and set of association rules
43: Post-process results and visualization
```

Figure 3. Parallel RSA-ARM Algorithm

Algorithm 3 Fitness Calculation for Parallel RSA-ARM

```
Require:
    $lhs$: Array representing the left-hand side items of the rule
    $lhsSize$: Number of items in $lhs$
 3: $rhs$: Array representing the right-hand side items of the rule
    $rhsSize$: Number of items in $rhs$
    $\alpha, \beta$: Parameters for the fitness calculation
 6: minsup, minconf: Minimum support and confidence thresholds
Ensure:
    $result$: The fitness value for the candidate rule
    Steps:
    Thread ID Calculation:
 9: Calculate the thread ID as:
        $$\text{tid} = \text{blockIdx.x} \times \text{blockDim.x} + \text{threadIdx.x}$$
    This calculation assigns each GPU thread to process one item in the $lhs$ array.

    Check Bounds:
12: if tid $\geq lhsSize$ then
        Terminate the current thread
    else
15:     Proceed to the next step
    end if

    Initialize Support and Confidence:
18: Set $support \leftarrow 0.5$    ▷ Example value for support calculation
    Set $confidence \leftarrow 0.6$ ▷ Example value for confidence calculation
    Check Minimum Thresholds:
21: if $support \geq$ minsup and $confidence \geq$ minconf then
        Proceed to the next step
    else
24:     Set $fitness \leftarrow -1$ and terminate the current thread
    end if
    Calculate Fitness:
27: Compute the fitness of the candidate rule as:
        $$\text{Fitness} = \beta \times \text{support} + \alpha \times \text{confidence}$$

    Atomic Update of Result:
    Use an atomic addition operation to safely update the global $result$:
        $$\text{result} \leftarrow \text{result} + \text{fitness}$$
30: This ensures that concurrent threads do not cause race conditions when updating the shared variable.
```

Figure 4. Fitness calculation for parallel RSA-ARM

5 Performance Evaluation

In this section, we first evaluate the performance of the proposed Parallel RSA-ARM using GPUs against the original CPU-based RSA-ARM. In the second part, we will make comparisons with state-of-the-art GPU-accelerated ARM algorithms, such as Multiple Evaluation GPU-based BSO. The comparison was conducted using the same datasets to ensure a fair and accurate evaluation.

5.1 Experimental Setup

Table 1 shows hardware specifications and software framework environment. The experiments were conducted on a system configured as follows:

Table 1: Hardware specifications and software framework environment

Category	Specification
Hardware Specifications	**CPU**: Intel Core i7 **Cores/Threads**: 8 cores, 8 threads 1. **RAM**: 16 GB DDR4 **GPU**: NVIDIA GeForce RTX 2070, 8 GB
Software Frameworks	**CUDA Toolkit 11.4**: Libraries and tools for parallel execution on NVIDIA GPUs. **JCuda Library**: Java binding for CUDA enabling Java applications to interact with CUDA. **Java Development Kit (JDK) 11**: For implementing RSA-ARM algorithm (both CPU and GPU versions).

5.2 Datasets and Performance Metrics

The evaluation uses the same benchmark datasets from the original CPU-based RSA-ARM experiments. The datasets represent varying sizes and complexities, allowing us to measure performance across small, medium, and large datasets. To compare the CPU-based and GPU-based RSA-ARM implementations, metrics such as Execution Time, Speedup Ratio, which is the ratio of execution times between the CPU and GPU implementations, and scalability are considered.

5.3 Results

The execution times and Speedup Ratio of the CPU-based RSA-ARM, Parallel RSA-ARM ME-GPU-BSO for each dataset are presented in Table 2

Table 2: CPU-based RSA-ARM vs Parallel RSA-ARM vs ME-GPU-BSO

Dataset	Transactions	Item Size	CPU-based RSA-ARM Time (s)	GPU-based RSA-ARM Time (s)	ME-GPU-BSO Time (s)	Speed up Ratio
Basketball	96	5	0.17	0.04	0.69	4.25
IBM Quest	1,000	20	0.13	0.03	1.73	4.33
Quake	2,178	4	0.10	0.02	1.38	5.00
Chess	3,196	75	0.30	0.08	16.38	3.75
Mushroom	8,124	119	1.42	0.35	25.86	4.06
BMS-Web-1	59,602	497	40.78	8.5	106	3.88
BMS-POS	515,597	1,657	1,185	125	2166	9.48
WebDocs	1,692,082	526,765	Blocked	1,500	6202	/

5.4 Discussion

5.4.1 Comparing with CPU-based RSA-ARM

The Parallel GPU-based RSA-ARM consistently outperforms the original CPU-based RSA-ARM across all datasets, demonstrating clear performance improvements due to parallelization. The speedup ratios range from 3.75x to 5.00x, which aligns with common speedup trends in GPU-accelerated data mining tasks. This indicates that the algorithm is effectively leveraging the parallel architecture of the GPU. The results validate the potential of parallel computing to handle the computational bottlenecks in RSA-ARM, especially in evaluating the fitness function across large datasets. For smaller datasets like Basketball, IBM Quest, and Quake, the GPU implementation achieves speedup ratios of around 4x to 5x. For medium-sized datasets like Chess and Mushroom, the speedup ratios are slightly lower, around 3.75x to 4.06x.

For medium-sized datasets like Chess and Mushroom, the speedup ratios are slightly lower, around 3.75x to 4.06x. This is due to increased memory overhead and more complex rule mining, which requires more GPU resources but still maintains significant speedup compared to the CPU. In large datasets like BMS-Web-1 and BMS-POS, the GPU-based RSA-ARM achieves speedup ratios of approximately 9.48x, which reflects the ability of the GPU to manage larger transaction volumes and item sizes through efficient parallel computation.

The execution time for BMS-POS is reduced from 1185 seconds on the CPU to 125 seconds on the GPU, demonstrating a substantial reduction in mining time. The CPU-based implementation could not process the WebDocs dataset due to memory limitations, but it was feasible on the GPU within 1,500 seconds. This shows that parallelization not only accelerates computation but also enables the processing of previously intractable datasets.

The results demonstrate the scalability of the GPU-based RSA-ARM, as it maintains consistent speedup ratios across datasets of varying sizes. However, there is a slight reduction in speedup for larger datasets due to Handling larger transaction data and frequent itemsets, which can lead to more memory access delays, impacting speedup, and transferring data between CPU and GPU can add overhead, especially for extremely large datasets. Despite these limitations, the Parallel RSA-ARM shows strong potential to efficiently mine association rules even in complex, large-scale datasets.

5.4.2 Comparing with other GPU-accelerated approaches

GPU-RSA-ARM is extremely fast for small datasets; ME-GPU-BSO lags significantly, with runtimes 17–69 times slower than GPU-RSA-ARM. As the dataset size increases, GPU-RSA-ARM maintains its performance with minimal increases in runtime. ME-GPU-BSO slows down significantly, with runtimes 74–204 times slower, demonstrating poor scalability compared to GPU-RSA-ARM.
GPU-RSA-ARM scales efficiently to large datasets, maintaining reasonable runtimes even for extremely complex datasets like WebDocs. ME-GPU-BSO, while feasible, becomes 4–17 times slower, with runtimes that are impractical for real-time or large-scale applications.

6 Conclusion

The results strongly support the use of GPU-based parallelization for RSA-ARM, showing significant performance gains across all dataset sizes. The approach not only accelerates mining but also extends the feasibility of processing larger datasets that are beyond the capacity of traditional CPU-based implementations. This work demonstrates that parallel computing is a promising strategy for enhancing the speed and scalability of association rule mining, making it more applicable to big data scenarios. These findings illustrate the effectiveness of the Parallel RSA-ARM and highlight its advantages in terms of speed, scalability, and handling of complex datasets.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. Very large databases, VLDB*, Citeseer, 1994, pp. 487–499.

[2] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 1–12, Jun. 2000, doi: 10.1145/335191.335372.

[3] Y. Djenouri, H. Drias, and Z. Habbas, "Bees swarm optimisation using multiple strategies for association rule mining," *International Journal of Bio-Inspired Computation*, vol. 6, no. 4, p. 239, 2014, doi: 10.1504/IJBIC.2014.064990.

[4] Z. Kou and L. Xi, "Binary Particle Swarm Optimization-Based Association Rule Mining for Discovering Relationships between Machine Capabilities and Product Features," *Math Probl Eng*, vol. 2018, pp. 1–16, Oct. 2018, doi: 10.1155/2018/2456010.

[5] M. M. Dorigo Vittorio; Colorni Alberto, "Ant system: optimization by a colony of cooperating agents," *IEEE Trans Syst Man Cybern B Cybern*, vol. 26, no. 1, pp. 29–41, 1996, doi: 10.1109/3477.484436.

[6] L. Abualigah, M. A. Elaziz, P. Sumari, Z. W. Geem, and A. H. Gandomi, "Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer," *Expert Syst Appl*, vol. 191, p. 116158, Apr. 2022, doi: 10.1016/j.eswa.2021.116158.

[7] A. Boukhalat, K. E. Heraguemi, M. Benouis, B. Bouderah, and S. Akhrouf, "Reptile Search Algorithm for Association Rule Mining," *International Journal of Computing and Digital Systems*, vol. 15, no. 1, pp. 1729–1744, 2024, doi: 10.12785/ijcds/1501122.

[8] S. S. Aljehani and Y. A. Alotaibi, "Preserving Privacy in Association Rule Mining Using Metaheuristic-Based Algorithms: A Systematic Literature Review," *IEEE Access*, vol. 12, pp. 21217–21236, 2024, doi: 10.1109/ACCESS.2024.3362907.

[9] Y. Zhang *et al.*, "Parallel Processing Systems for Big Data: A Survey," *Proceedings of the IEEE*, vol. 104, no. 11, pp. 2114–2136, Nov. 2016, doi: 10.1109/JPROC.2016.2591592.

[10] M. Madiajagan and S. S. Raj, "Chapter 1 - Parallel Computing, Graphics Processing Unit (GPU) and New Hardware for Deep Learning in Computational Intelligence Research," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, A. K. Sangaiah, Ed., Academic Press, 2019, pp. 1–15. doi https://doi.org/10.1016/B978-0-12-816718-2.00008-7.

[11] H. Bavarsad Salehpour, H. Haj Seyyed Javadi, P. Asghari, and M. E. Shiri Ahmad Abadi, "Improvement of Apriori Algorithm Using Parallelization Technique on Multi-CPU and GPU Topology," *Wirel Commun Mob Comput*, vol. 2024, pp. 1–14, May 2024, doi: 10.1155/2024/7716976.

[12] S. H. Adil and S. Qamar, "Implementation of association rule mining using CUDA," in *2009 International Conference on Emerging Technologies*, IEEE, Oct. 2009, pp. 332–336. doi: 10.1109/ICET.2009.5353149.

[13] M. Kaya and R. Alhajj, "Utilizing Genetic Algorithms to Optimize Membership Functions for Fuzzy Weighted Association Rules Mining," *Applied Intelligence*, vol. 24, no. 1, pp. 7–15, Feb. 2006, doi: 10.1007/s10489-006-6925-0.

[14] C. A. C. Coello and M. S. Lechuga, "MOPSO: A proposal for multiple objective particle swarm optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, IEEE, 2002, pp. 1051–1056.

[15] Y. Djenouri, H. Drias, and Z. Habbas, "Bees swarm optimisation using multiple strategies for association rule mining," *International Journal of Bio-Inspired Computation*, vol. 6, no. 4, p. 239, 2014, doi: 10.1504/IJBIC.2014.064990.

[16] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Trans Knowl Data Eng*, vol. 8, no. 6, pp. 962–969, 1996, doi: 10.1109/69.553164.

[17] D. Boley *et al.*, "Partitioning-based clustering for Web document categorization," *Decis Support Syst*, vol. 27, no. 3, pp. 329–341, Dec. 1999, doi: 10.1016/S0167-9236(99)00055-X.

[18] B. K. Sarker, T. Mori, T. Hirata, and K. Uehara, "Parallel Algorithms for Mining Association Rules in Time Series Data," 2003, pp. 273–284. doi: 10.1007/3-540-37619-4_28.

[19] O. R. Zaiane, M. El-Hajj, and P. Lu, "Fast parallel association rule mining without candidacy generation," in *Proceedings 2001 IEEE International Conference on Data Mining*, IEEE Comput. Soc, pp. 665–668. doi: 10.1109/ICDM.2001.989600.

[20] N. Melab and E. Talbi, "A parallel genetic algorithm for rule mining," in *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, IEEE Comput. Soc, pp. 1347–1352. doi: 10.1109/IPDPS.2001.925112.

[21] J. Eggermont, J. N. Kok, and W. A. Kosters, "Genetic Programming for data classification," in *Proceedings of the 2004 ACM symposium on Applied computing*, New York, NY, USA: ACM, Mar. 2004, pp. 1001–1005. doi: 10.1145/967900.968104.

[22] G. Agapito, P. H. Guzzi, and M. Cannataro, "Parallel and distributed association rule mining in life science: A novel parallel algorithm to mine genomics data," *Inf Sci (N Y)*, vol. 575, pp. 747–761, Oct. 2021, doi: 10.1016/j.ins.2018.07.055.

[23] W. Fang, M. Lu, X. Xiao, B. He, and Q. Luo, "Frequent itemset mining on graphics processors," in *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, New York, NY, USA: ACM, Jun. 2009, pp. 34–42. doi: 10.1145/1565694.1565702.

[24] K.-W. Chon, S.-H. Hwang, and M.-S. Kim, “GMiner: A fast GPU-based frequent itemset mining method for large-scale data,” *Inf Sci (N Y)*, vol. 439–440, pp. 19–38, May 2018, doi: 10.1016/j.ins.2018.01.046.

[25] A. Cano, J. M. Luna, and S. Ventura, “High-performance evaluation of evolutionary-mined association rules on GPUs,” *J Supercomput*, vol. 66, no. 3, pp. 1438–1461, Dec. 2013, doi: 10.1007/s11227-013-0937-4.

[26] X. J. A. Bellekens, C. Tachtatzis, R. C. Atkinson, C. Renfrew, and T. Kirkham, “A Highly-Efficient Memory-Compression Scheme for GPU-Accelerated Intrusion Detection Systems,” in *Proceedings of the 7th International Conference on Security of Information and Networks*, New York, NY, USA: ACM, Sep. 2014, pp. 302–309. doi: 10.1145/2659651.2659723.

[27] Y. Djenouri, A. Belhadi, P. Fournier-Viger, and H. Fujita, “Mining diversified association rules in big datasets: A cluster/GPU/genetic approach,” *Inf Sci (N Y)*, vol. 459, pp. 117–134, Aug. 2018, doi: 10.1016/j.ins.2018.05.031.

[28] R. Skinderowicz, “The GPU-based parallel Ant Colony System,” *J Parallel Distrib Comput*, vol. 98, pp. 48–60, Dec. 2016, doi: 10.1016/j.jpdc.2016.04.014.

[29] Y. Zhuo, T. Zhang, F. Du, and R. Liu, “A parallel particle swarm optimization algorithm based on GPU/CUDA,” *Appl Soft Comput*, vol. 144, p. 110499, Sep. 2023, doi: 10.1016/j.asoc.2023.110499.

[30] X. Geng, Z. Chen, W. Yang, D. Shi, and K. Zhao, “Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search,” *Appl Soft Comput*, vol. 11, no. 4, pp. 3680–3689, Jun. 2011, doi: 10.1016/j.asoc.2011.01.039.

[31] Y. Djenouri, A. Bendjoudi, M. Mehdi, N. Nouali-Taboudjemat, and Z. Habbas, “Parallel association rules mining using GPUS and bees behaviors,” in *6th International Conference on Soft Computing and Pattern Recognition, SoCPaR 2014*, Institute of Electrical and Electronics Engineers Inc., Jan. 2014, pp. 401–405. doi: 10.1109/SOCPAR.2014.7008040.

[32] Y. Djenouri, A. Bendjoudi, M. Mehdi, N. Nouali-Taboudjemat, and Z. Habbas, “GPU-based bees swarm optimization for association rules mining,” *Journal of Supercomputing*, vol. 71, no. 4, pp. 1318–1344, Apr. 2015, doi: 10.1007/s11227-014-1366-8.

[33] L. Hamdad, Z. Ournani, K. Benatchba, and A. Bendjoudi, “Two-level parallel CPU/GPU-based genetic algorithm for association rule mining,” *International Journal of Computational Science and Engineering*, vol. 22, no. 2/3, p. 335, 2020, doi: 10.1504/IJCSE.2020.107366.

[34] B. Silva, L. G. Lopes, and F. Mendonça, “Parallel GPU-Acceleration of Metaphorless Optimization Algorithms: Application for Solving Large-Scale Nonlinear Equation Systems,” *Applied Sciences*, vol. 14, no. 12, p. 5349, Jun. 2024, doi: 10.3390/app14125349.

[35] S. H. Adil and S. Qamar, “Implementation of association rule mining using CUDA,” in *2009 International Conference on Emerging Technologies*, IEEE, Oct. 2009, pp. 332–336. doi: 10.1109/ICET.2009.5353149.

[36] A. Shenneld, P. Fleming, J. Allan, and V. Kadirkamanathan, “Optimisation of Maintenance Scheduling Strategies on the Grid,” in *2007 IEEE Symposium on Computational Intelligence in Scheduling*, IEEE, Apr. 2007, pp. 231–237. doi: 10.1109/SCIS.2007.367695.