STM JOURNALS

Journal of
Operating Systems
Development & Trends

ISSN: 2454-9355
Volume 8, Issue 2, 2021
DOI (Journal): 10.37591/JoOSDT

http://computers.stmjournals.com/index.php?journal=JoOSDT&page=index

Research JoOSDT

Parallelization of Metaheuristics for the Optimization of Permuted Perceptron Problem

Anurag Yadav*

Abstract

Parallel computing has found a mainstream backing in graphics processing units (GPUs).These resources have enormous processing capacity, are energy efficient, and are broadly available, unlike grids. Since the advent of CUDA (Compute Unified Device Architecture) developed by NVIDIA that permits GPU programming in C, C++ language, GPUs are now being used in a variety of fields, including scientific computing. This thesis focuses on the optimization of the solution of a NP-complete problem called Permuted Perceptron Problem based on cryptographic identification scheme, which particularly meets the requirements for resource restricted devices like smart cards. We will study about GPU optimization techniques for parallel metaheuristics in order to achieve an efficient solution.

Keywords: Permuted perceptron problem, ILS, Tabu Search, simulated annealing, metaheuristics

INTRODUCTION

Optimization problems in the realm of academia and industries are usually complex and NP-hard and finding a near-optimal solution is becoming a prominent topic among scholars. For small size instances, heuristics methods can address these optimization difficulties, but for large size instances, it may take a long time. There are numerous metaheuristics that can be utilized to achieve a near-optimal solution to an optimization problem using massively parallel devices like GPUs. In this paper, we sought to optimize the solution of the Permuted Perceptron Problem, an NP-complete problem proposed by [1] David Point cheval in 1995. The Permuted Perceptron Problem is a variation of the Perceptron Problem that emerges in both physics and artificial intelligence with the study of Ising's Perceptron. This problem is NP-complete problem, based on cryptographic identification system, which particularly meet the requirements of resource restricted devices like smart cards. Perceptron Problem is inspired from a widely known perceptron in neural network. We will study about several metaheuristic approaches that can be implemented to solve the problem and we will also study about how to use different parallel programming techniques to parallelize the metaheuristics for the optimization of the solution.

***Author for Correspondence**
Anurag Yadav

Student, Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati, Assam, India

Received Date: November 29, 2021
Accepted Date: December 09, 2021
Published Date: December 15, 2021

Citation: Anurag Yadav. Parallelization of Metaheuristics for the Optimization of Permuted Perceptron Problem. Journal of Operating Systems Development & Trends. 2021; 8(2): 32–37p.

What is Permuted Perceptron Problem?

[2] Perceptron Problem arises in physics as well as in artificial intelligence with the machine learning. Perceptron is widely known as Single layer Neural Network that executes certain computations to discover features or business intelligence in the input data.

Beginning with the perceptron problem before moving on to the permuted perceptron problem. A matrix in which all elements are either +1 or -1 is

called ∈-matrix. Similarly, a vector in which all elements being either +1 or -1 is called ∈-vector.

Definition 1 *[1] Given an* ∈-matrix *A of size m × n, find an* ∈-vector *V of size n such that* $AV \geq 0$.

It is a very complex problem to solve. Permuted Perceptron Problem which is a variation of Perceptron Problem was introduced in order to develop a zero-knowledge identification protocol:

Definition 2 *[1] Given an s-matrix A of size m × n and a multiset S of non-negative integers of size m, find an s-vector V of size n such that* $\{\{(AV)_j | j = \{1, ..., m\}\}\} = S$.

Permuted Perceptron problem is even more difficult to solve as it is evident that PPP is a solution for PP.

What is Metaheuristic?

A metaheuristic is an algorithmic framework that provides a set of principles or approaches for building heuristic optimization algorithms at a high level, regardless of the task. Simulated Annealing, Genetic algorithms, Variable neighborhood search, large neighborhood search, Tabu search are only a few examples of metaheuristics. A problem-specific implementation of a heuristic optimization algorithm that follows principles established in a metaheuristic framework is known as metaheuristic.

Parallelization of Metaheuristics

Metaheuristics are well-known for delivering near-optimal results in a short amount of time. However, most metaheuristics suffer from a lack of scalability, when dealing with high-dimensional problems, performance suffers in terms of both time complexity and effectiveness. GPU-based parallel metaheuristics have been evolved to overcome this limitation. It minimizes the execution time and increase the quality of the solutions found. The parallel design of metaheuristics is classified into three classes (Figure 1):

- *Algorithmic Level:* This level permits multiple algorithms to execute in parallel. The algorithms can also communicate with one another, allowing the metaheuristics' behavior to evolve and increase the quality of the solutions produced.
- *Iteration level:* This level permits a parallelization in each iteration.
- *Solution level:* This level permits a parallelization of a single solution.

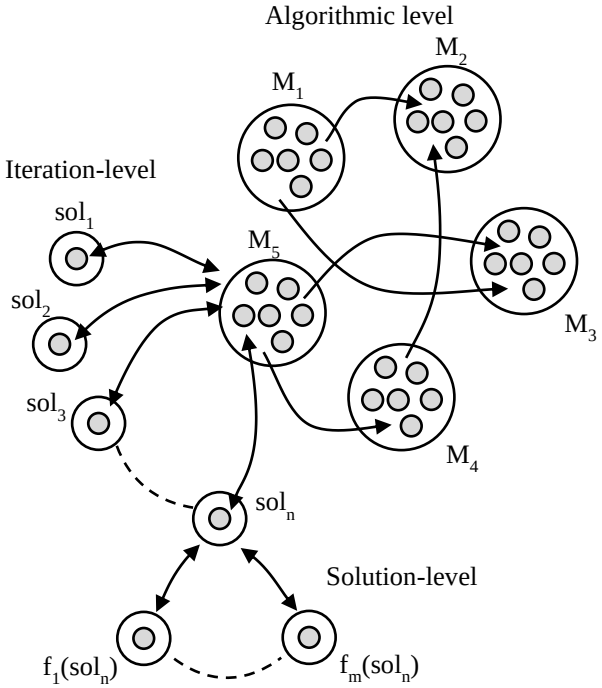

Figure 1. Parallel models of metaheuristics.

We will look at a various metaheuristic method that can be used to solve the PPP. Following that, we will look at the implementation of GPU based parallel metaheuristics to solve PPP.

LITERATURE SURVEY

Various works done in the area of optimization involving metaheuristics and studies done to address Permuted Perceptron Problem has been discussed in this section.

A New Identification Protocol Based on the Perceptron's Problem

In 1995, Point cheval [2] developed the permuted perceptron problem, a new NP-complete problem derived from the perceptron problem. In this paper he aimed to provide some zero-knowledge interactive identification protocol based on PPP, along with a security assessment. These protocols were meeting the requirements for smart card applications.

The size of the problem (m, n) set so that n = m + 16, is more difficult to solve, according to the author. He also stated that in order to find *s*-vector Y, one should first select an arbitrary *s*-vector Y of size n and a *s*-matrix A of size m × n. Then make the following changes:

- If $(AY)_j < 0$, then j^{th} row of matrix A is multiplied by −1
- If $(AY)_j \geq 0$, then no need to change

Then, compute $S = \{\{(AY)_j | j = 1, ..., m\}\}$. As a result, (A, S) is a Permuted Perceptron Problem instance, with Y as a solution. Author tried several attacks to assess the security of a possible protocol. First attack is majority vector attack M: for all j,

$$\begin{cases} M_j = +1 & if\{i|A_{i,j} = +1\} > n/2 \\ M_j = -1 & Otherwise \end{cases}$$

But this attack proved to be ineffective, due to which author tried a well-known probabilistic algorithm which originates from artificial intelligence called simulated annealing. He claimed that simulated annealing is not suitable for PPP due to multiset, but that it is ideal for PP. He used simulated annealing to obtain the energy function for solving PP.

$$E(Y) = \frac{1}{2}\sum_{i=1}^{m}(|(AY)_i| - (AY)_i)$$

In [3] Knudsen and Meier improved the simulated annealing search for the solution of PPP by deriving a new energy function.

$$E(Y) = g1\sum_{i=1}^{m}(|(AY')_i| - (AY')_i) + g2\sum_{i=1}^{n}(|H_i - H'_i|)$$

Here H denotes the histogram vector over the integers. If m, n is odd then H_i is set for only odd values of i, $1 \leq i \leq n$. Y' is the candidate vector, *Hj* is the histogram for finding a ∈-vector Y , and $g1 \geq 30$, g2 = 1. In [4] authors modified the cost function for finding a solution of PPP as in equation.

$$Cost(Y') = g\sum_{i=1}^{m}(max\{K - (AY')_i, 0\})^R + \sum_{i=1}^{n}(|H_i - H'_i|)^R$$

Here g, K and R is the constant which is varied for distinct instances. Authors used for instance (201, 217) the value of K is: 20, 15, 10 and R=2.0, For (401, 417) the value of K is: 30, 25, 20, 15 and R=2.0 and for other instances R=3.0. The value of g = 20 is used throughout.

Algorithm 1 Simulated Annealing (SA) for PPP on GPU

Input: size of matrix *m* and *n*.
Output: desired solution.
1: Generate a random ∈ *A* and *Y* matrix
2: Evaluate the objective function of the solution
3: Generate random initial candidate solution
4: Set the initial parameters for SA
5: **for all** neighbor solutions Y' **do**

6: **while** not Termination_Criteria *i.e., T<1* **do**
7: **Repeat**
8: Evaluate the multi-set matrix AY' on GPU in parallel.
9: Results are sent back from GPU to CPU.
10: $\Delta C = g_1(|f(AY') - f(AY)|) + g_2(|H - H'|)$; /* H and H' are histogram of initial and neighbor solution */
11: **if** $\Delta C = 0$ **then**
12: $Y = Y'$ /* Accept the neighbor solution */
13: **else**
14: Accept the neighbor solution Y' with a probability $e^{\frac{-\Delta C}{T}}$
15 **until** Absolute Temperature (Equilibrium condition) is reached
16: *T=g(T)*; /* Update the temperature = temperature × coolingRate */

Parallel Metaheuristics on GPU

Parallel computing has found a mainstream backing in graphics processing units (GPUs). These resources have enormous processing capacity, are energy efficient, and are broadly available, unlike grids. Since the advent of CUDA (Compute Unified Device Architecture) developed by NVIDIA that permits GPU programming in C, C++ language, GPUs are now being used in a variety of fields, including scientific computing.

With some control units and a small cache, the GPU has a huge number of arithmetic units. This enables the GPU to produce small and independent parts in a huge and simultaneous manner while processing a big amount of data. Because more transistors are devoted to data processing rather than data caching and flow management, GPUs are specialized for compute-intensive and highly parallel tasks (Figure 2).

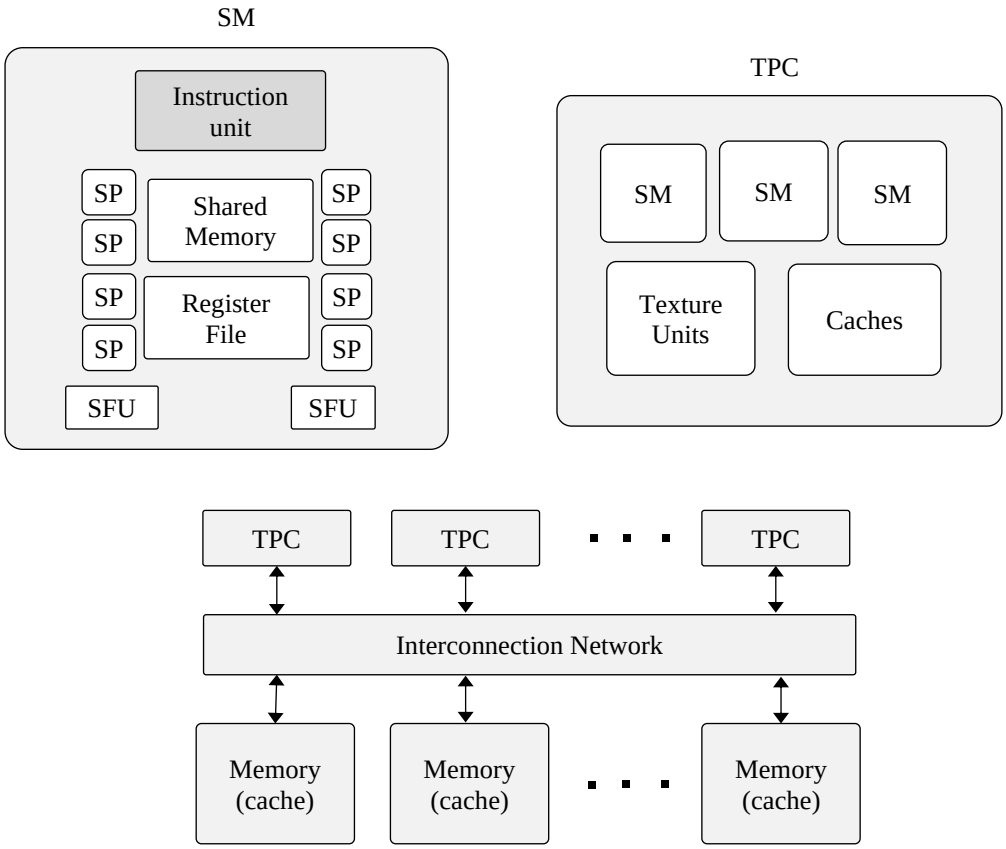

Figure 2. General GPU architecture composed of streaming multiprocessors.

GPU Challenges for Metaheuristics

The optimal allocation of data processing between GPU and CPU, data transfer optimization across multiple memories, thread synchronization and memory capacity limits are the key challenges for parallel combinatorial optimization on GPU.

Challenges in order to develop GPU-based parallel metaheuristics that are effective:

- *CPU and GPU collaboration:* This stage necessitates the definition of task distribution in metaheuristics. In order to get the best performance, the data flow between the two components must be optimized.
- *Parallelism Control:* To meet the memory limits, effective thread control is required. It is necessary to construct an effective mapping between a thread designated by a unique identifier delivered at GPU runtime and each potential solution.
- *Memory Management:* The various optimization structures must be efficiently arranged on the various memory, considering their sizes and access times.

Author has proposed various solution to address these challenges in this paper.

Local Search Algorithms on GPU

[5] PPP was utilized as a case study for GPU-based local search algorithms. He employed binary encoding to construct neighbor solutions from the first solution in this article. To solve PPP, he employed the tabu search metaheuristic. The author created a neighbor solution for one hamming distance and ran it on GPU, comparing GPU and GPU with texture memory. He also created a neighbor solution with two hamming distances and compared the results between GPU and GPU with texture memory, finding that texture memory was up to 40 times faster.

EXPERIMENTAL RESULTS AND DISCUSSIONS

After reading the publications mentioned above, it is evident that parallel combinatorial optimization on GPU faces a number of obstacles. For the Permuted Perceptron Problem, we have seen two possibly effective solutions. The first method is to use Simulated Annealing metaheuristics, while the second method is to use ILS metaheuristics (Table 1).

For ILS metaheuristics, in the Table 1 we can see that as we increase the size of the instance CPU execution time also increases rapidly. But it is also evident from the table that GPU execution time is quite high for large size instances even higher than CPU execution time because of overhead. So, we come to this conclusion that in case of ILS metaheuristic GPU does not improve the execution time over CPU (Table 2).

For simulated annealing, Table 2 shows that CPU execution time increases with the increase in the size of the instance. But for GPU, execution time increases at a very low rate and as the size of instance increases the speedup of GPU also increases. Result shows that GPU can be up to 80 times faster than CPU. In this case GPU is facilitating us with a huge advantage over CPU.

Table 1. Experimental results of ILS metaheuristics for PPP.

Instance	CPU		GPU		Speed-up
	No. of sols.	*Time (in sec.)*	*No. of sols.*	*Time (in sec.)*	
6–9	26	0.001	69	0.008	
73–73	5258	26.748	5258	37.336	-
81–81	6482	44.416	6482	64.837	-
101–117	13512	226.604	13574	383.017	-
121–137	18634	575.928	18634	797.448	-
151–167	27724	1579.239	27724	2100.334	-

Table 2. Experimental results of SA metaheuristics for PPP.

Instance	CPU		GPU		Speed-up
	No. of sols.	*Time (in sec.)*	*No. of sols.*	*Time (in sec.)*	
73–73	9	0.048	30	0.010	4.86
81–81	27	0.148	40	0.014	10.74
101–101	27	0.217	50	0.024	8.91
101–81	35	0.140	70	0.026	5.41
121–81	35	0.294	67	0.036	8.28
101–117	16	0.168	70	0.034	4.87
121–137	18	0.433	50	0.036	11.87
151–167	20	0.702	40	0.047	14.90
301–317	22	5.806	47	0.209	27.80
601–617	31	47.519	66	1.133	41.93
801–817	30	118.524	67	2.074	57.16
1001–1017	30	224.113	68	3.137	71.45
1301–1317	42	573.927	90	7.118	80.63

In the simulated annealing approach, there is a scope for improvement. The simulated annealing algorithm is extremely sensitive to parameter values, and even a tiny change in 'β' can generate drastically different results and as a consequence behavior of PPP search algorithm also changes. The value of the parameter *'t'* is crucial in the PPP-search algorithm. PPP solutions can be found quickly with a low value of *'t'* in certain cases, but a higher value of *'t'* appears to be beneficial in others. Using more than one energy function could help to improve the PPP-search process. Either we need to switch between energy functions from one simulated annealing algorithm run to the next one or we can use one energy function for first *'t'* runs and another energy function for next *'t'* runs.

CONCLUSION AND FUTURE WORK

This report reviews different papers on Permuted Perceptron Problem as well as on Parallel Metaheuristic optimization using GPU. Every solution has its pros and cons. We can strive to learn from previous work and propose a new solution that inherits the benefits of previous work while attempting to minimize the drawbacks. As a part of future work, I will implement the approach discussed in the previous section as well as try to improve the solution as much as possible utilizing GPU optimization and CUDA programming.

REFERENCES

1. Pointcheval D. A new identification scheme based on the perceptron problem. Lecture Notes in Computer Science International Conference on the Theory and Applications of Cryptographic Techniques. 1995:319-28. doi: 10.1007/3-540-49264-X_26.
2. Essaid M, Idoumghar L, Lepagnot J, Brévilliers M. GPU parallelization strategies for metaheuristics: a survey. Int J Parallel Emergent Distrib Syst. 2019;34(5):497-522. doi: 10.1080/17445760.2018.1428969.
3. Knudsen LR, Meier W. Cryptanalysis of an identification scheme based on the permuted perceptron problem. Lecture Notes in Computer Science International Conference on the Theory and Applications of Cryptographic Techniques. 1999:363-74. doi: 10.1007/3-540-48910-X_25.
4. Clark JA, Jacob JL. Fault injection and a timing channel on an analysis technique. Lecture Notes in Computer Science International Conference on the Theory and Applications of Cryptographic Techniques. 2002:181-96. doi: 10.1007/3-540-46035-7_12.
5. Van Luong T, Melab N, Talbi E-G. Local search algorithms on graphics processing units. a case study: the permutation perceptron problem. Lecture Notes in Computer Science. 2010:264-75. doi: 10.1007/978-3-642-12139-5_23.

