Parallel Rabin-Karp Algorithm for String Matching using GPU

Masoumeh Moeini
School of Electrical Engineering
Iran University of Science and Technology
Tehran, Iran
m_moeini@elec.iust.ac.ir

Hadi Shahriar Shahhoseini
School of Electrical Engineering
Iran University of Science and Technology
Tehran, Iran
shahhoseini@iust.ac.ir

Abstract— **String matching algorithms play an important role in many aspects. In this paper, a new method is proposed for parallel execution of Rabin-Karp string matching algorithm. In the proposed method, all patterns are divided into different groups. This categorization has been used to prevent redundant comparisons and accelerate the matching process. This procedure takes two advantages: a reduction in the number of comparisons of hash values and a decline in the number of pattern reading from global memory. It is recommended to implement the algorithm in various cases on NVIDIA GPUs using CUDA Platform to demonstrate the efficiency of the proposed algorithm. Experimental results depict that the execution time of the algorithm has decreased significantly. In the best case, the proposed method is approximately 27 times faster than the series mode.**

Keywords-String Matching Algorithm; Rabin-Karp Algorithm; GPU; Categorization

I. INTRODUCTION

The rapid growth of computer networks and the Internet and its numerous applications results in producing massive data in many applications. In these big data environments many challenges have been appeared, among them searching is ones of the most important one and high-speed string matching algorithms are the core of the solution. String matching has been widely used in many real-world applications including biological signal processing [1,2], network processing [3], task scheduling [4,5], pattern matching [6,7] and intrusion detection system [8-10]. The process of string matching is a way of finding a particular string pattern from a large volume of text. Currently, most applications use the concept of string matching to retrieve data or to match patterns with a large amount of data. Generally, the process of string matching finds all the occurrences of a pattern $x=[x_0x_1x_2\ldots x_{m-1}]$; $x_i\in\Sigma; i=0,1,2,\ldots m-1$, in a text of $y=[y_0y_1y_2\ldots y_{n-1}]$; $y_j\in\Sigma; j=0,1,2,\ldots,n-1$.

It is so crucial to perform the process of string matching in a specified time. Nowadays, one of the best solutions to accelerate any time consuming process is parallel processing. However, it is necessary to modify the process based on the capabilities of the used parallel computing platform. There are various structures of parallel processing, each with its advantages and disadvantages [11]. Our goal is to implement parallel processing in the graphical processing unit (GPU) [12]. There are some important string matching algorithms like Rabin-Karp [13], Aho-Corasick [14], Boyer Moore [15] and KMP [16], which are already executed in parallel. The implementation of Aho-Corasick algorithm on the GPU [17] has led to DPAC [18] method although there were some drawbacks. When the data is split into different parts, each one is given to the processor and each of these processors works separately; hence, there is a problem on the boundaries where a pattern might be potentially placed on two contiguous parts [19]. Moreover, they proposed another method that eliminates the invalid transfer and each processor is only responsible for processing its associated information. The DPAC and KMP algorithms have been efficiently paralleled and implemented on GPUs, obtaining significant speedups [20-21]. In addition, one of the most important algorithms that is implemented on a GPU is Boyer Moore algorithm. This algorithm has been taken into consideration due to its very high speed. The method of parallelism the Boyer-Moore algorithm is based on the division of data into several parts and giving them to parallel processors [22]. Furthermore, Dayarathne and Ragel has paralleled Rabin-Karp algorithm to implement the DNA finder sequence on NVIDIA GPUs by CUDA platform [23]. Additionally, Sharma and Singh implemented Rabin-Karp algorithm on GPUs for deep packet inspection and then use shared memory to optimize the use of memory bandwidth [24].

Our proposed method is based on the fact that it is not necessary to compare all patterns. To reduce the number of comparisons, we proposed a method for categorizing patterns which aims to a reduction in the total computational time.

The following sections are organized as follows; section II gives a survey on GPU architecture. Section III provides a detailed view of the basic method of Rabin-Karp algorithm. Section IV introduces the proposed algorithm. In section V, we provide the result and discussion for the proposed algorithm. Section VI contains the conclusion.

II. GPU ARCHITECTURE

In recent years, the graphics processing unit (GPU) has emerged as a powerful computation device. The use of general purpose graphics processors (GPGPUs) has grown and has become more common. GPUs can provide great performance at very low prices, while they can be used in high-performance computing systems without much trouble [25]. These types of graphics processing units, instead of those designed exclusively for graphic operations, use the computational power of a modern graphic shader to compute many parallel tasks such as ray tracing, computational fluid dynamics, and climate modeling. Generally, this tool is used for computers requiring high computing power, since GPUs have high parallelism due to their special architecture. The general purpose of using the GPU in these cases is that a specific program is broken into several small sections and then executed in parallel. It is worth mentioning that only programs that are inherently data parallel can have a profit of this technology.

NVIDIA (2010) developed a software platform named compute unified device architecture (CUDA), which allows almost the direct translation of C code onto the GPU [26]. CUDA is a powerful computing engine (processing) of NVIDIA graphics cards, that allows software developers to use a CUDA-enabled GPU for processing purposes.

CUDA gives developers a direct access to memory and instruction set in the graphical processing units. The GPU has several multiprocessor cores, each multiprocessor runs a single instruction, with multiple data known as SIMD programming model that run by CUDA. It enables us to implement the massive data programs on GPU. The hierarchical structure of memory in GPU is shown in Fig. 1. It is notable that the GPU memory bandwidth is higher than CPU. In CUDA, the function of a program must be written as a kernel.

III. THE BASIC RABIN-KARP ALGORITHM

The main idea of Rabin-Karp algorithm for implementing on GPU is to select a window of input data string and give it to different threads in the GPU. This method is shown in Fig. 2.

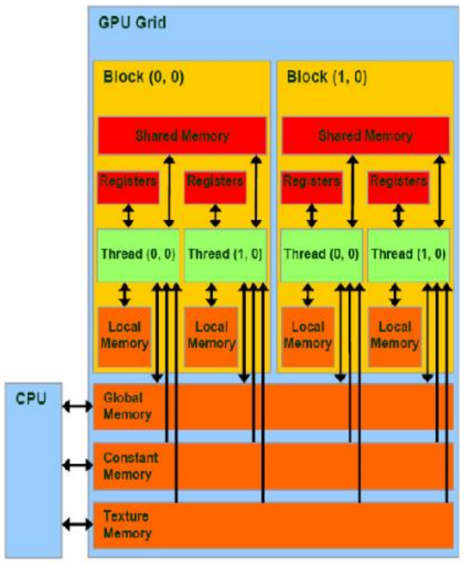


Figure 1. Hierarchical structure of GPU memory [27]

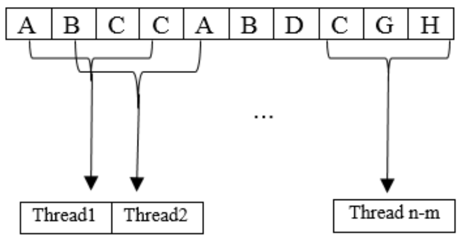


Figure 2. The basic method of Rabin-Karp algorithm

As can be seen, a window with the length of pattern (m) is selected from the beginning of the input data and also other windows are chosen in the same way. Each window is allocated to one thread. Then, these threads perform the pattern matching algorithm independently. First, the hash value of the pattern and all windows are calculated. Next, the hash value of each window is compared with the hash value of the pattern. If these values are the same, the verification process is performed by using Brute-Force algorithm. The number of nodes in the basic Rabin-Karp algorithm is n-m and the size of input data string is n.

There are various strategies to optimize the implementation of the basic Rabin-Karp algorithm. The GPU has various types of memory, so a solution can be using shared memory and transferring the data window to the shared memory. In this case, once the selected data is read from the global memory, it is transferred to the shared memory. This strategy has some disadvantages. It would be useful only when the memory is used for several times; otherwise an overhead of transferring data is imposed to the whole process. Therefore, using shared memory is reasonable only when the number of patterns in the input data string is very large.

IV. THE PROPOSED METHOD FOR IMPROVING PARALLELISM IN BASIC RABIN-KARP ALGORITHM

In order to improve the performance of basic Rabin-Karp algorithm, the main idea employed in this paper, is to split the input data and then dedicate it to parallel processors. Nevertheless, the basic method calculates the hash values of the input data windows and the pattern. Subsequently, the hash value of a pattern is compared with the hash values of the windows. It can need extra time, reducing the overall performance. If you can reduce the number of comparisons, then output performance will increase. According to the above, the Rabin-Karp algorithm is beneficial in case of multi-pattern matching.

Thus, in order to reduce the system response time, we proposed a pattern grouping method that it reduces the number of comparisons between the hash values. The categorization method is based on the first letter of the patterns; that is, the patterns starting with a same letter put into a same class. All patterns starting with "A", for example, establish the group "A". Other patterns also follow this rule. In addition, we make a slight change to the Rabin-Karp algorithm to select the desired group; that is, the selection of the group is based on the first letter of each window.

Therefore, the main trend of the proposed Rabin-Karp algorithm is the use of pattern categorization. First, a group is selected based on the first letter of the window. Next, the hash value of the window is compared with the hash values of the patterns fallen into the associated group. As a result, it is not necessary to compare with all of the patterns, which leads to a significant reduction in the number of comparisons in the Rabin-Karp algorithm; consequently, more performance can be expected. Additionally, the pattern categorization method takes another advantage. Since fewer patterns are compared in pattern categorization method, fewer memory calls will happen and consequently the number of memory access operations will be reduced. As a result, the memory access constraint can be relaxed by the proposed method. The pattern categorization is shown in the Fig. 3. The point should be taken into consideration that the selection of the desired group leads to an extra overhead to the process; however, two benefits can be acquired: a drop in the number of comparisons and the memory read operations.

V. Experimental results

Before describing different modes of implementations used in this research, we offer the hardware requirements. To run the series mode, the Rabin-Karp algorithm was implemented by the C++ programming language; furthermore, the parallel Rabin-Karp algorithm was implemented by CUDA platform. The central processing unit used in this implementation is Intel Core i7. In addition, we used the NVIDIA GeForce GT 630 graphics processing unit. To prove the efficiency of the proposed algorithm in different aspects, three schemes have been used based on the input data string and pattern properties. The first one is to increase the input data size. As GPUs are a kind of SIMD (single instruction multiple data) processors that run an instruction on a large number of data, by increasing the input data size, the level of parallelism will increase. The second one is changing the number of patterns. Due to the different applications, the number of patterns can be different. For example, in the intrusion detection system, the number of patterns is very high and is constantly growing. So, we need to reach an acceptable performance in this system. Therefore, we examine the increasing or decreasing in the number of patterns when running the Rabin-Karp algorithm. The third one is to increase the length of the pattern. In different applications, the length of the patterns may vary.

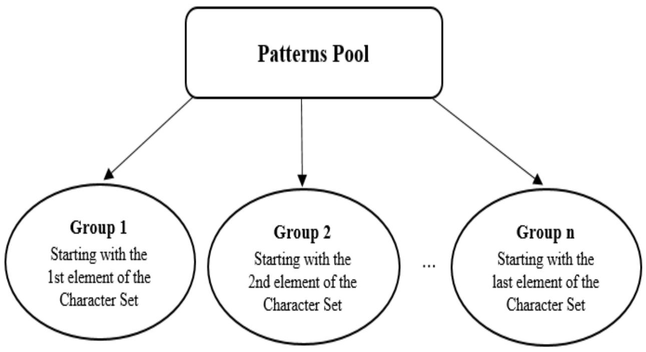
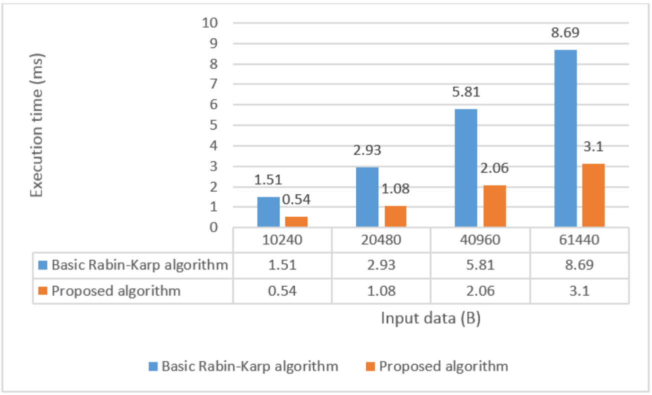

Patterns Pool
Group 1
Starting with the 1st element of the Character Set
Group 2
Starting with the 2nd element of the Character Set
...
Group n
Starting with the last element of the Character Set

Figure 3. Patterns categorization in the proposed method

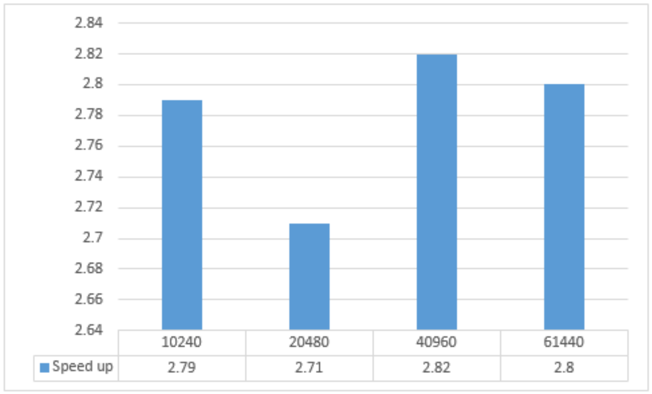
Table 1. The execution time of the Rabin-Karp algorithm for different sizes of input data

Input data (B)	**Execution time (ms)**		
	The series mode	*The basic method*	*The proposed method*
10240	7	1.51	0.54
20480	14	2.93	1.08
40960	29	5.81	2.06
61440	44	8.69	3.1



	10240	20480	40960	61440
Basic Rabin-Karp algorithm	1.51	2.93	5.81	8.69
Proposed algorithm	0.54	1.08	2.06	3.1

Figure 4. The execution time of the parallel implementations of the Rabin-Karp algorithm on GPU for different sizes of input data



	10240	20480	40960	61440
Speed up	2.79	2.71	2.82	2.8

Figure 5. The speed up rate of the proposed method over the basic method for different sizes of input data

A. *Different sizes of input data*

In this case, the length of the pattern is 4 bytes and the number of patterns is 260. The results of this implementation are listed in Table 1. In addition, the results of execution time and speedup rate are respectively shown in Fig. 4 and Fig. 5. It can be observed that the proposed method is approximately up to 14 times faster than the series mode. Moreover, in terms of execution time, the proposed method is superior than the basic method for different sizes of input data.

B. *Different numbers of patterns*

In this case, the length of the pattern is 4 bytes and the size of the input data is 40960 B. The results acquired by this implementation are presented in Table 2. In addition, the results of execution time and speedup rate are respectively shown in Fig. 6 and Fig. 7. It can be observed that the proposed method is approximately up to 27 times faster than the series mode.

This case evidently reveals the main advantage of the proposed method for improving the Rabin-Karp algorithm. As can be observed in Table 2, by increasing the number of patterns, the execution time of the proposed method stays almost constant while the execution time of the basic one linearly rises. As a consequence, the speed up of our method over the basic one is approximately proportional to the number of patterns (Fig. 7). Consequently, increasing the number of patterns will make our method more efficient and interesting to use. This superiority is due to the fact that in the proposed categorization method all new patterns are divided among different groups, so only a few of the new patterns are compared.

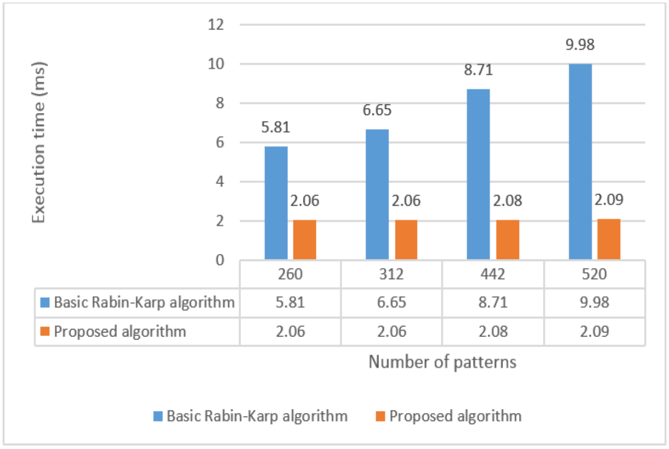
As noted above, there are many patterns in the intrusion detection systems and also their number are increasing; thus, the proposed method is so useful in these systems.

C. Different lengths of patterns

In this case, the number of patterns is 260 and the size of the input data is 40960 B. Table 3 depicts the execution time of the parallel implementations of the Rabin-Karp algorithm on GPU for different lengths of patterns. In addition, the results of execution time and speedup rate are respectively shown in Fig. 8 and Fig. 9. It can be observed that the proposed method

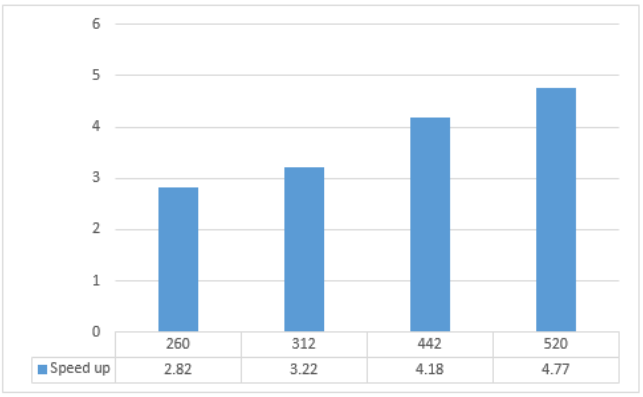
Table 2. The execution time of the parallel implementations of the Rabin-Karp algorithm on GPU for different numbers of patterns

Number of patterns	**Implementation time (ms)**		
	The series mode	*The basic method*	*The proposed method*
260	29	5.81	2.06
312	35	6.65	2.06
442	49	8.71	2.08
520	57	9.98	2.09



	260	312	442	520
🟦 Basic Rabin-Karp algorithm	5.81	6.65	8.71	9.98
🟧 Proposed algorithm	2.06	2.06	2.08	2.09

Fig. 6. The execution time of the parallel implementations of the Rabin-Karp algorithm on GPU for different numbers of patterns



	260	312	442	520
🟦 Speed up	2.82	3.22	4.18	4.77

Fig. 7. The speed up rate of the proposed method over the basic method for different numbers of patterns

for parallelism of the Rabin-Karp algorithm is approximately up to 14.5 times faster than the series mode. In different applications, the pattern length can vary depending on their protocols. Therefore, the proposed method should be checked for different cases. It can be seen that with an increase in the average length of patterns, the speedup rate increases slightly, and this is not very significant because with the pattern's length increasing, the matching time of both basic and proposed algorithm rises. Hence, we can not expect an increase in the speedup rate for this situation. It can be said that the change in the length of the pattern does not have a significant impact on the speedup rate of the proposed method versus the basic method of the Rabin-Karp algorithm.

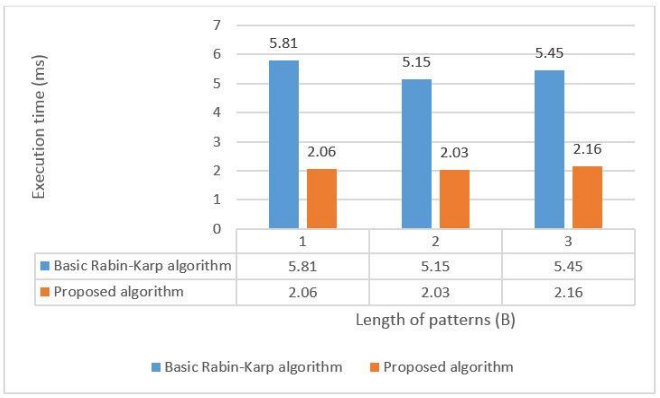
D. Thread variation

Thread is the smallest part of a parallel program on GPUs. NVIDIA recommends that the threads should be properly defined to utilize the maximum computational power of a GPU.

To optimize a CUDA program, it is essential to establish an optimum balance between the number of blocks and their size.More threads per block will be useful in masking the latency of the memory operations, at the same time, the number of registers available per thread is reduced. So, NVIDIA advises using blocks of 128 to 256 threads, which offers the best tradeoff between masking latency and the number of registers needed for most kernels. This point is investigated in the proposed method. The results for different sizes of input data and different numbers of threads are shown in Table 4. As can be seen, the results listed in the table show the validity of NVIDIA's recommendation.

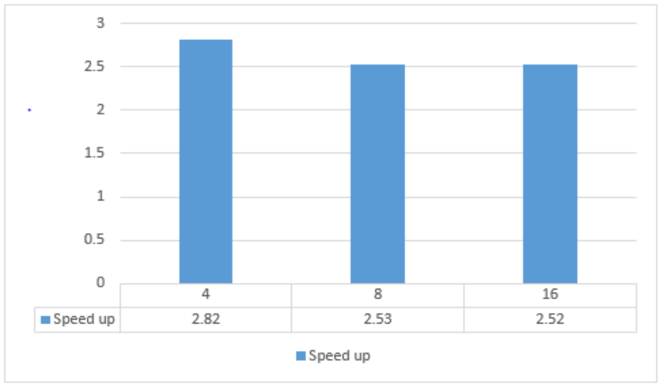
Table 3. The execution time of the parallel implementations of the Rabin-Karp algorithm on GPU for different lengths of patterns

Length of patterns (B)	**Implementation time (ms)**		
	The series mode	*The basic method*	*The proposed method*
4	29	5.81	2.06
8	29.5	5.15	2.03
16	30.5	5.45	2.16



	1	2	3
🟦 Basic Rabin-Karp algorithm	5.81	5.15	5.45
🟧 Proposed algorithm	2.06	2.03	2.16

Fig. 8. The execution time of the parallel implementations of the Rabin-Karp algorithm on GPU for different lengths of patterns



	4	8	16
🟦 Speed up	2.82	2.53	2.52

Fig. 9. The speed up rate of the proposed method over the basic method for different lengths of patterns

VI. CONCLUSION

Rabin-Karp algorithm is one of the pattern matching algorithms that can be used in both single-mode matching algorithms and multiple matching algorithms. It is a fact that the execution time of the algorithm plays an important role in some security applications, such as the intrusion detection system. The most important and cost-effective way to accelerate the pattern matching process in today's world is to use parallel processing and implementing on GPU. In this paper, a pattern categorization method was proposed for parallel implementation of the pattern matching process in the Rabin-Karp algorithm. The proposed method was implemented on GeForce GT 630 for different scenarios to be comprehensively verified. The experimental results demonstrated the efficiency and superiority of the proposed method compared to the basic one, especially in case of applications with a large number of patterns.

Table 4. The execution time of proposed algorithm for different sizes of input data and different numbers of threads

Input data (B)	**Implementation time (ms)**			
	128 threads	*256 threads*	*512 threads*	*1024 threads*
10240	0.53	0.54	0.54	0.54
20480	0.98	0.99	1.01	1.08
40960	1.92	1.93	1.92	2.06
61440	2.82	2.88	2.86	3.1

REFERENCES

[1] P. Rastogi and R. Guddeti, “GPU accelerated inexact matching for multiple patterns in DNA sequences,” International Conference Advances in Computing on Communications and Informatics (ICACCI), pp. 163-167, 2014.

[2] D.R.V.L.B Thambawita, R. G. Ragel and D. Elkaduwe, “An optimized parallel failure-less Aho-Corasick algorithm for DNA sequence matching,” 2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS), 2016.

[3] H. Naderi, H. S. Shahhoseini, A. H. Jafari, “Evaluation MCDM multi-disjoint paths selection algorithms using fuzzy-copeland ranking method,” International Journal of Communication Networks and Information Security, Vol. 5, No. 1, pp. 59-67, 2013.

[4] M. M. Bassiri, H. S. Shahhoseini, “Configuration reusing in on-line task scheduling for reconfigurable computing systems, ” Journal of Computer Science and Technology Vol. 26, No. 3, pp. 463-473, 2011.

[5] M. M. Bassiri, H. S. Shahhoseini, “A new approach in on-line task scheduling for reconfigurable computing systems,” Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, pp. 321-324, 2010.

[6] A. Tabatabaei, M. R. Mosavi, A. Khavari, H. S. Shahhoseini, “Reliable urban Canyon navigation solution in GPS and GLONASS integrated receiver using improved Fuzzy weighted Least-Square method,” Wireless Personal Communications Vol. 94, No. 4, pp. 3181-3196, 2017.

[7] H. J. Rad, M. Azarafrooz, H. S. Shahhoseini, B. A. Abolhassani, “New adaptive power optimization scheme for target tracking wireless sensor networks,” 2009 IEEE Symposium on Industrial Electronics and Applications, ISIEA 2009.

[8] B. M. Bidgoli, M. Analoui, M. H. Rezvani, H. S. Shahhoseini, “Performance evaluation of decision tree for intrusion detection using reduced feature spaces,” Trends in Intelligent Systems and Computer Engineering, 2008 Lecture Notes in Electrical Engineering, Vol. 6, pp. 273-284 Springer, 2008.

[9] M. Saeed, H. S. Shahhoseini, “APPMA - An Anti-phishing protocol with mutual authentication,” Proceedings - 2010 IEEE Symposium on Computers and Communications, pp. 308-313, 2010.

[10] T. AbuHmed, A. Mohaisen, and D. Nyang, “Deep packet inspection for intrusion detection systems: A survey,” Magazine of Korea Telecommunication Society, Vol. 24, pp. 25-36, 2007.

[11] H. S. Shahhoseini, M. Naderi, R. Buyya, “Shared memory multistage clustering structure, an efficient structure for massively parallel processing systems,” Proceedings of the 4th International Conference on High Performance Computing in the Asia-Pacific Region, 2000, Beijing.

[12] C. S. Kouzinopoulos and K. G. Margaritis, “String matching on a multicore GPU using CUDA,” In 13th Panhellenic Conference on Informatics, ser. PCI '09.IEEE, pp. 14-18, Sep. 2009.

[13] R. M. Karp and M. O. Rabin, “Efficient randomized pattern-matching algorithms,” IBM J. Res. Dev., Vol. 31, No. 2, pp. 249–260, 1987.

[14] A. V. Aho and M. J. Corasick, “Efficient string matching: An aid to bibliographic search,” Commun. ACM, Vol. 18, No. 6, pp. 333–340, Jun. 1975.

[15] R. S. Boyer and J. S. Moore, “A fast string searching algorithm,” Commun. ACM, Vol. 20, No. 10, pp. 762–772, 1977.

[16] D. E. Knuth, J. H. Morris, and V. R. Pratt, “Fast pattern matching in strings,” SIAM J. Comput., Vol. 6, No. 2, pp. 323–350, Jun. 1977.

[17] A. Tumeo, O. Villa, and D. Sciuto, “Efficient pattern matching on GPUs for intrusion detection systems,” Proc. Seventh ACM Int’l Conf. Computing Frontiers, 2010.

[18] P. R. Jelenkovic, X. Kang and A. Radovanovic, “Near optimality of the discrete persistent access caching algorithm,” DMTCS proc. AD, pp. 201–222, 2005.

[19] C. H. Lin, C. H. Liu, L. S. Chien, S. C. Chang, “Accelerating pattern matching using a novel parallel algorithm on GPUs,” IEEE Transactions On Computers, Vol. 62, No. 10, october. 2013.

[20] C. H. Lin, S. Y. Tsai, C. H. Liu, S. C. Chang, and J. M. Shyu, “Accelerating string matching using multi-threaded algorithm on gpu,” In Proceedings On IEEE Global Telecommunications Conference (GLOBECOM 2010), pp. 1–5, 2010.

[21] A. Rasool and N. Khare, “Parallelization of KMP string matching algorithm on different SIMD architectures: multi-core and GPGPU’s,” Int. J. Comput. Appl., Vol. 49, No. 11, pp. 26–28, 2012.

[22] X. Zha and S. Sahni, “GPU-to-GPU and Host-to-Host multipattern string matching on a GPU,” IEEE Transaction On Computer, Vol. 62, No. 6, pp. 1156–1169, 2013.

[23] N. Dayarathne and R. Ragel, “ Accelerating Rabin Karp on a Graphics Processing Unit (GPU) using Compute Unified Device Architecture (CUDA),” Proceeding in International Conference on Information and Automation for Sustainability (ICIAfS), pp. 1–6, 2014.

[24] J. Sharma and M. Singh, “CUDA based Rabin-Karp pattern matching for deep packet inspection on a multicore GPU,” International Journal Computer Network and Information Security, Vol. 10, pp. 70-77, 2015.

[25] Y. Tan and K. Ding, “A survey on GPU-based implementation of Swarm intelligence algorithms,” IEEE Transactions on Cybernetics, Vol. 46, No. 9, Sept. 2016.

[26] NVIDIA Corporation, “NVIDIA CUDA C programming guide,” Sep. 2015, pG-02829-001_v7.5.

[27] CUDA NVIDIA, “NVIDIA CUDA programming guide,” (version 1.0), NVIDIA: Santa Clara, CA (2007).