IOP Conf. Series: Materials Science and Engineering **190** (2017) 012023 doi:10.1088/1757-899X/190/1/012023

Implementation of Multipattern String Matching Accelerated with GPU for Intrusion Detection System

[1] Rangga Nehemia, [1] Charles Lim, [1] Maulahikmah Galinium and [2] Ahmad Rinaldi Widianto

[1] Department of Information Technology,
Swiss German University,
Tangerang 15339, Indonesia
[2] PT. Garuda Solusi Kreatif, Forsecnet Division,
Ruko The Icon Business Park Blok L/11
BSD, Tangerang, Indonesia

E-mail: rangga.nehemia[at]student.sgu.ac.id
charles.lim[at]sgu.ac.id
maulahikmah.galinium[at]sgu.ac.id
rinaldi[at]forsecnet.com

Abstract. As Internet-related security threats continue to increase in terms of volume and sophistication, existing Intrusion Detection System is also being challenged to cope with the current Internet development. Multi Pattern String Matching algorithm accelerated with Graphical Processing Unit is being utilized to improve the packet scanning performance of the IDS. This paper implements a Multi Pattern String Matching algorithm, also called Parallel Failureless Aho Corasick accelerated with GPU to improve the performance of IDS. OpenCL library is used to allow the IDS to support various GPU, including popular GPU such as NVIDIA and AMD, used in our research. The experiment result shows that the application of Multi Pattern String Matching using GPU accelerated platform provides a speed up, by up to 141% in term of throughput compared to the previous research.

1. Introduction

With the continuous increase of the Internet usage at the rate of 50% yearly [1], the number of security threats also increases sharply. Network Intrusion Detection System (NIDS) monitors the network traffic for harmful packets that is used as an attack vector, as described by Scarfone and Mell [2]. NIDS uses string matching for scanning and detecting patterns of threats contained in packets that match the pattern of the specified rules. Hence, to keep up with this requirement, the need of a higher speed of NIDS becomes crucial.

There are two popular Open Source NIDS, namely Snort developed by Roesch [3] and Bro developed by Paxson [4]. Compared to Snort, Bro is more flexible, customizable, and suitable for Gbps network, but is more complex to deploy. In addition, Bro's rules/signatures is also more sophisticated than the Snort ones [5]. The high flexibility and capability of Bro to handle a heavier network environment becomes the main reason for Bro to be selected to be implemented

in this research. Our research aim to create an accelerated NIDS which is capable to perform efficiently in a a high speed network.

To increase the performance of NIDS, our research utilizes Graphical Processing Units for its parallel computing power to implement a parallel version of string matching algorithm. Parallel Failureless Aho Corasick (PFAC) developed by Lin et al. [6], an improvement of Aho Corasick by Aho and Corasick [7] is being implemented in our research. This research extends the research conducted by Widianto et al. [8], which showed the improved performance of the existing Intrusion Detection System by utilizing the GPGPU to allow multiprocessing of packets, effectively increasing the capacity of the IDS in high throughput environment.

Following are the organization of the paper: Section 2 summarizes the related work, Section 3 discusses our research methods, Section 4 describes our experiment environment setup and experiment results and finally section 5 concludes our research works.

2. Related Works

Many research has been performed to improve the the performance of IDS. There are three general approaches of improvements: Specialized Hardware, GPU utilization, and Algorithm Development. Specialized hardware is easy to develop but is expensive and thus not as popular as the cheaper and more efficient GPU utilization or algorithm development. GPU utilization research mostly includes the implementation of new algorithms.

Young et al [9] designed a deep packet filtering firewall on Field Programmable Gate Array to take advantage of parallelism while maintaining the programmability. Alfred et al [7], and Wu et al. [10] developed new algorithm for string or pattern matching, while others such as Kouzinopoulos et al. [11], Zha and Sahni [12], and Soroushnia et al. [13] improve the performance the pattern matching algorithms using GPU on CUDA [14]. Pyrgiotis et al. [15] perform their research based on OpenCL [16].

Vasiliadis et al. [17] developed Gnort, a GPU based Intrusion Detection System, which utilizes the computational power of GPU to handle the pattern matching operations which is costly if implemented in CPU. The implementation is using CUDA, which is NVIDIA exclusive, while our research is using OpenCL for compatibility to various machine and architecture.

This paper extends the research conducted by Widianto et al. [8] with the objective of this research is to improve the performance of the existing Network Intrusion Detection System by utilizing the GPGPU to allow multiprocessing of packets, effectively increasing the capacity of the NIDS in high throughput environment. Our research implement the Multi Pattern String Matching and compare the performance result with the performance result of the research by Ahmad Rinaldi.

3. Research Methodology

Signatures are the keywords for hinting the probability of malicious packets that NIDS uses as references to inspect packets flowing through the network. If a packet contains any of the listed signature or NIDS rule, the packet will be classified as a malicious packet. Once the packet is identified and captured, the packet is then preprocessed before the packet is sent to GPU for further inspection. Packet is inspected with PFAC algorithm to determine any suspicious or malicious pattern, that found in the NIDS rule that describe the malicious byte pattern that IDS is looking for. Using parallel computation capability offered by GPU, the search can then be executed in parallel fashion, using multiple work unit to accelerate the search.

Result of the packet inspection is sent back to CPU to determine whether the packet is malicious or not. If the packet contain malicious pattern, the reporting system to alert user for the existence of the malicious pattern. The overview of the system architecture used in our research follows the system proposed in the previous work by Widianto et al. [8] and shown in

Figure 1, which can be divided into two parts: Packet Preprocessing and Transfer on CPU, and Multipattern Matching on GPU.

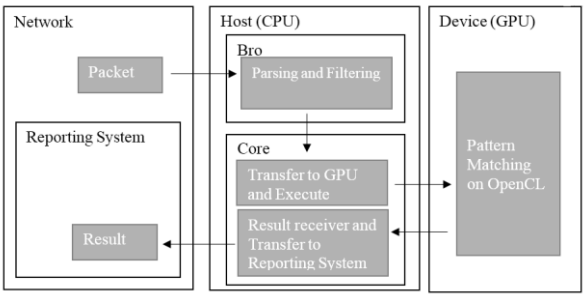

Figure 1: System Architecture Overview

In Packet Preprocessing and Transfer, the packets are captured by IDS and preprocessed before they can be inspected for malicious payload. After the packets are processed, they are sent to the core, that executes GPU specific instructions for Multi Pattern String Matching inspection based on the packets received and transfer the packets to GPU. Multipattern Matching on GPU is performed using Parallel Failureless Aho-Corasick (PFAC) algorithm by Lin et al. [6]. PFAC is an improved Aho-Corasick algorithm to used in parallel processing environment. Based on the rules, PFAC creates a trie in the initialization phase using CPU and copies it to the GPU. In the searching phase, GPU runs the PFAC in parallel fashion, using multiple threads to process different part of the packets.

Using the same method, our research also compares two different GPU platform, i.e. NVIDIA and AMD. Figure 2 shows the testing network architecture using NVIDIA GPU environment for the test subject. The test subject is the accelerated NIDS, and handle the packet capture and packet scanning. DNS query will be generated by the packet generator and sent to NIDS. The network has a single subnet where the data for testing is sent during the testing operation, with sending server connected on one end and receiving server/accelerated NIDS on the other end. After the test, the data captured by NIDS is checked with the sent data and if the two data match each other, this mean the test is successful as no packet is dropped, and the result can be measured and analyzed.

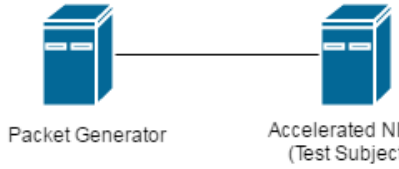

Figure 2: NVIDIA Testing Network Architecture

Figure 3: AMD Environment Testing Network Architecture

Figure 3 shows the testing network architecture using AMD GPU Environment for the test subject. The difference with NVIDIA environment is that in the NVIDIA environment, the packet capture and the service (packet scanning) is handled in the same computer, while in AMD environment the packet capture and service are handled in separate computer. Bro Control handles the packet capturing and send the packet to the test subject for packet scanning (Bro Control handles both packet capture and packet scanning in the first scenario). The criteria to check validity of the data is the same with NVIDIA environment, which the sent data and captured data are compared.

4. Experiment Results

4.1. Experiment Environment

Our research uses Intel(R) Core(TM) i7-3770 CPU equipped with NVIDIA GeForce GTX 970 GPU. Bro, an open source NIDS, is chosen since it allows developer to integrate new feature as a plugin in a more flexible way. Python programming language is used for general programming while C programming language is used for the OpenCL part of the implementation. Jmeter, developed by Apa [18] is used to send multiple packets to test the system and Bro for packet capture in Simulated Benchmark.

4.2. Initial Testing

Table 1: Initial Testing Comparison of GPU Naive and GPU PFAC

Metrics	GPU Naive	GPU PFAC	Improvement
Computation Time(sec)	3.42	2.58	25%
Requests Per Second	2196	2907	32%

Table 1 shows the result of our initial test between the system developed by the author, which is GPU PFAC, compared to the system from previous research by Widianto et al. [8], GPU system based on Naive algorithm. Given the same condition, GPU PFAC shows a computational time improvement of 25%.

4.3. Detailed Experiment Result

To further improve the performance of the system, three major optimization is conducted. The first optimization is to manage the global size to match the local size, allowing the global size to be dividable by local size. This also allows the system to be run with NVIDIA GPU. Second optimization is to perform better memory allocation to enable faster and lighter system. The last optimization is to use better handling of the result counter, reducing the computational cost for transferring results. Table 2 shows the comparison of GPU PFAC computation time before and after the optimization. Optimization of rule management allows the system to handle around 58% more rules or around 515.040 characters, limited only by the hardware. The test also included the AMD implementation of the system with the hardware that is similar to the hardware from previous system by [8]. As the hardware used in AMD is similar to the previous system, the improvement gained in the AMD implementation is the pure algorithm increased performance while the improvement gained from NVIDIA implementation, which has more powerful hardware, is the improvement gained from both algorithm and hardware. The result shows that the base performance of the algorithm has increased by up to 40% for NVIDIA GPU and 56% for AMD GPU. AMD GPU has faster computation time because of the faster GPU preparation time.

Table 2: GPU PFAC Computation Time Comparison (in Seconds)

String Length (character)	500.000	1.000.000	1.500.000	2.100.000
Unoptimized PFAC	2.08	2.11	2.14	2.2
AMD PFAC	0.94(-55%)	0.94(-55%)	0.94(-56%)	0.96(-56%)
NVIDIA PFAC	1.29(-38%)	1.30(-38%)	1.31(-39%)	1.32(-40%)

Table 3: GPU PFAC Searching Time Comparison (in Seconds)

String Length (character)	100.000	500.000	700.000
Unoptimized PFAC	0.0028	0.0048	0.0057
AMD PFAC	0.0152	0.0195	0.0242
NVIDIA PFAC	0.0071	0.0102	0.0119

A test with simulation benchmark shows that the improvements made are beneficial in real time scenario, and make the comparison between the AMD implementation of the algorithm compared to the NVIDIA implementation of the algorithm possible. The result shows computational time improvement of 51% and 58% and throughput by 106% and 141% for AMD and NVIDIA respectively compared to the GPU Naive used in the previous research [8], as shown in table 4.

Table 4: Simulated Benchmark Improvement

	Computation Time (in Second)	Throughput (Request per Second)
GPU Naive	3.42	2196
Unoptimized GPU PFAC	2.58(-25%)	2907(+32%)
AMD GPU PFAC	1.66(-51%)	4518(+106%)
NVIDIA GPU PFAC	1.42(-58%)	5282(+141%)

5. Conclusion

Existing NIDS could not cope with the increased volume of Internet traffic that flow through the organization network. This paper proposes the implementation of Multi Pattern String Matching algorithm using a GPU-based equipped computer platform. Our experiments show that the implementation of GPU accelerated Multi Pattern String Matching algorithm in NIDS, outperforms Naive String Matching algorithm, with an improvement up to 106% in term of request per second and 51% shorter computation time compared to the Naive String Matching, resulting in an NIDS that can perform twice as fast. We also prove that with a better performance GPU, such as NVIDIA GPU, the experiment shows that the performance could improve as much as 141% in term of request per second and 58% shorter computation time compared to the Naive String Matching, with a 33% more performance gain compared with AMD GPU. Due to the limitation of the string matching and scope of our research, the proposed system is suited only to scan the DNS query at the moment, among other type of possible traffics. We hope to include Regular Expression [19] in our future works to scan various type of packet that is common to real world network traffic.

References

[1] Jacob Nielsen. Nielsen's law of internet bandwidth, April 2016. URL https://www.nngroup.com/articles/law-of-bandwidth/.

[2] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). NIST special publication SP 800-94, 2007.

[3] Martin Roesch. Snort-lightweight intrusion detection for networks. In Proceedings of the 13th USENIX conference on System administration, pages 229–238. USENIX Association, 1999.

[4] Vern Paxson. Bro: a system for detecting network intruders in real-time. Computer networks, 31(23):2435–2463, 1999.

[5] Pritika Mehra. A brief study and comparison of snort and bro open source network intrusion detection systems. International Journal of Advanced Research in Computer and Communication Engineering, 1(6):383–386, 2012.

[6] Cheng-Hung Lin, Sheng-Yu Tsai, Chen-Hsiung Liu, Shih-Chieh Chang, and Jyuo-Min Shyu. Accelerating string matching using multi-threaded algorithm on gpu. In Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, pages 1–5. IEEE, 2010.

[7] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. Communications of the ACM, 18(6):333–340, 1975.

[8] A. R. Widianto, C. Lim, and I. E. Kho. Improving performance of intrusion detection system using opencl based general-purpose computing on graphic processing unit (gpgpu). In 2015 3rd International Conference on New Media (CONMEDIA), pages 1–5, Nov 2015. doi: 10.1109/CONMEDIA.2015.7449146.

[9] Young H Cho, Shiva Navab, and William H Mangione-Smith. Specialized hardware for deep network packet filtering. In Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream, pages 452–461. Springer, 2002.

[10] Sun Wu, Udi Manber, et al. A fast algorithm for multi-pattern searching. 1994.

[11] Charalampos S Kouzinopoulos, Panagiotis D Michailidis, and Konstantinos G Margaritis. Multiple string matching on a gpu using cudas. Scalable Computing: Practice and Experience, 16(2):121–138, 2015.

[12] Xinyan Zha and Sartaj Sahni. Multipattern string matching on a gpu. In Computers and Communications (ISCC), 2011 IEEE Symposium on, pages 277–282. IEEE, 2011.

[13] Shima Soroushnia, Masoud Daneshtalab, Juha Plosila, Tapio Pahikkala, and Pasi Liljeberg. High performance pattern matching on heterogeneous platform. Journal of integrative bioinformatics, 11(3):253, 2014.

[14] NVIDIA. NVIDIA CUDA Compute Unified Device Architecture Programming Guide, August 2016. [Online]. Available: `http://docs.nvidia.com/cuda/cuda-c-programming-guide/`.

[15] Themistoklis K Pyrgiotis, Charalampos S Kouzinopoulos, and Konstantinos G Margaritis. Parallel implementation of the wu-manber algorithm using the opencl framework. In Artificial Intelligence Applications and Innovations, pages 576–583. Springer, 2012.

[16] Khronos. OpenCL The open standard for parallel programming of heterogeneous systems, May 2016. [Online]. Available: `https://www.khronos.org/opencl/`.

[17] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P Markatos, and Sotiris Ioannidis. Gnort: High performance network intrusion detection using graphics processors. In Recent Advances in Intrusion Detection, pages 116–134. Springer, 2008.

[18] Apache jmeter, June 2016. URL `http://jmeter.apache.org/`.

[19] Yuichiro Utan, Masato Inagi, Shin'ichi Wakabayashi, and Shinobu Nagayama. A gpgpu implementation of approximate string matching with regular expression operators and comparison with its fpga implementation. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.