Advances in Computer Science Research, volume 62
2nd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2017)

Parallel Design and Performance Optimization based on OpenCL Snort

Hongying Xie, Yangxia Xiang †, Caisen Chen †*

**Unit 61175, China*
†Academy of Armored Force Engineering, China

Keywords: OpenCL snort, GPU, AC algorithm, parallel programming

Abstract

With the rapid increasement of the network speed and number of threats which hide in the network poses enormous challenges to network intrusion detection systems (NIDS). As the most popular NIDS, snort can run as a single threaded application. However, it may not be able to detect intrusions in real-time especially in networks with high traffic. In this paper, a parallel module OpenCL Snort (OCLSnort) is introduced: realize parallel pattern matching algorithm using GPU and innovate new architecture which is more suitable for the parallel algorithm. The result showed that OCLSnort can detect the attacks correctly and effectively, the new system not only has markedly improved on throughput, also effectively reduced the CPU utilization and memory usage.

1 Introduction

Intrusion detection systems (IDSs) are of critical importance to the integrity of computer networks due to massive growth in the data transmission speed and the frequency of attacks. With the rapid development of computer network, more and more data need to be searched, analyzed and detected whether they have threat or not. Such as network monitoring application snort, which is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. Combining the benefits of signature, protocol, and anomaly-based inspection, and as so far, Snort is the most widely deployed IDS/IPS technology worldwide [1].

In snort, they are using pattern matching algorithm such as AC, BM algorithm to detect thread. Pattern matching is one of the core operations used by applications such as traffic classification [2], intrusion detection systems [3] and content monitoring filters [1]. Unfortunately, packet detecting part occupies the most of the time of the whole processing time in modern NIDSes [4,5] and this operation has significant overheads in terms of both memory space and CPU cycles, so when the data or packet which will be detected is very large, there will be packet-losing problem about snort.

Several research efforts have used GPU for security purposes, including cryptography [6], data carving [7] and intrusion detection [8]. And Jacob and Brodley were the first that tried to use the GPU as a pattern matching engine for NIDS in PixelSnort [8]. They changed KMP algorithm to parallel version but the performance result is not very ideal.

This paper is organized as follows: In Section 2 and 3, two methods to realize OpenCL snort are presented. In section 4, we evaluate our implementation and compare with the original snort. Experimental results and analysis are given. Finally, conclusions are given in Section 5.

2 Architecture

The overall architecture of Snort NIDS is shown in Fig.1 and the OpenCL version Snort's architecture is showed in Fig.2. From Fig.1 and Fig.2, there are some differences between the original snort and the new version snort, one is collecting packets at packet classification part; the other is detecting packet content at packet detecting part.

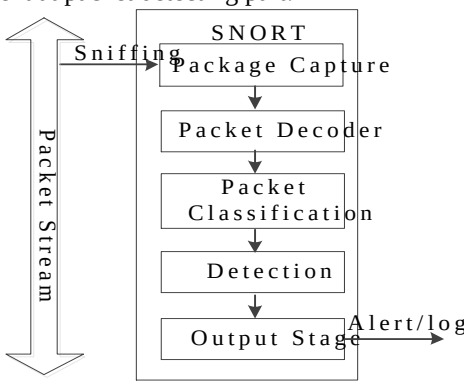

Fig.1 Packet process flow in original snort

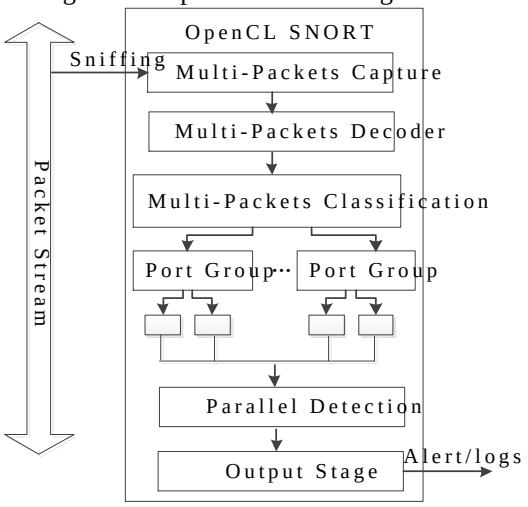

Fig.2 Packet process flow in OpenCL snort

2.1 Packet detecting using OpenCL AC algorithm

For the multi-pattern matching algorithm, the first thing is to build DFA such as Fig.3, and this section is finished before

the beginning of the packet detected in snort. In our design of the OpenCL version Snort, the realized DFA is represented as a two-dimensional state table array that is mapped on the memory space of the GPU. The dimensions of the array are equal to the number of states and the size of the alphabet (256 in our case), respectively. Each cell contains the next state to move, as well as an indication of whether the state is a final state or not.

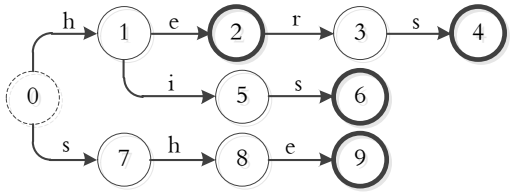

Fig.3 AC State Machine of Patterns “he”, “hers”, “his”, “she”

Fig.3 shows a state machine of patterns which used in our OpenCL AC algorithm, from this figure we can see that the difference between original AC state machine and OpenCL AC state machine is that whether it is needed about failure transitions. The failure transitions are used to back-track the state machine to recognize patterns in any location of an input stream. Given a current state and an input character, the original AC machine first looks up the valid transition table to check whether there is a valid transition for the input character; otherwise, the machine looks up the failure transition table and jumps to the failure state where the failure transition points. Then the machine regards the same input character until the character causes a valid transition.

In our OpenCL version snort, we used OpenCL to change the AC algorithm for parallelism based on PFAC [9]. The idea of the parallel algorithm of AC is: Give an input stream have N byte, we will create N threads corresponding to N byte. And for each thread, it is only responsible for identifying the pattern starting at the thread starting position. So in OpenCL AC algorithm, the failure transitions of the AC state machine can all be removed as well as the self-loop transition of the initial state. And the whole process of the OpenCL AC is showed by Fig.4.

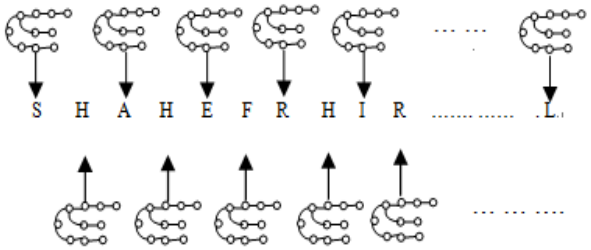

Fig.4 OpenCL AC Algorithm Execution Process

There are several characteristics of the OpenCL AC algorithm. First, although it creates huge amounts of threads, most threads have a high probability of terminating very early because a thread in OpenCL AC is only responsible for matching the pattern beginning at its starting position. Second, the maximum detection length of each thread is the maximum length in whole patterns, and based on this, when the larger the input stream is, the faster the detect speed is. And finally, the failure transitions are all removed when we are using OpenCL AC, and this simplifies the algorithm and the thread can detect the input stream automatically without rollback.

2.2 Packet collecting and transfer to GPU

Before the packet detecting in GPU, the first thing must to consider is how and how many packets will be transferred from the network to the GPU memory. The simplest method is according to the original snort architecture, transfer one packet to GPU for processing once time. However, as we know, the TCP or UDP packet size is usually hundreds byte, the performance is much better batching many small transfers into a large one than making each transfer separately [16]. Thus, we realized the two methods (1) using original snort architecture, transferring one packet to GPU once time, and (2) change packet classification part, transferring more than one packets to GPU once time and get the performance comparison based on two methods.

As we know, the process flow of original snort is showed as Fig.1: capture a packet from network once time, then packet analysis and classification, detecting packet and output the result finally. Using method (1), the process flow can be changed as follows: capture a packet from network, packet analysis and classification is not changed, then transfer the packet to GPU and detecting it using OpenCL AC algorithm, then transfer the results to CPU and output the result finally. So using method (1), we changed Detection part, using OpenCL AC take the place of original ac algorithm and the other part of snort’s architecture are not changed, processing packet one by one.

And using method 1, the performance improvement is not exciting, there are two reasons: (1) the DMA time occupied most of the time; (2) the input stream transferring to GPU only have hundreds byte each time. It does not make full use of GPU resources. Based on this, we proposed a new method that can transfer more than one packets to GPU, the architecture of OpenCL Snort are showed by Fig.2.

From Fig.2 we can see the difference between OpenCL Snort and original snort is processing packet number once time. In OpenCL version Snort, we change the interface to realize capture multi-packets at the beginning of snort and then deal with packets, transfer multi-packets to GPU once time, and finally output alerts/logs.

3 Implementation

In this section, we are showed the implementation details about the OpenCL version snort. In snort, they are using different rules to detecting whether the packet has threatened or not according to packet type. Different rules create different state transition tables. So we are focus on the packets collecting and the state transition tables correspond to packets part when using method (2) transfer multi-packets to GPU once time.

3.1 Transferring a Single Packet to GPU

In this approach, when capturing a packet from network, snort will decode and classify it, then send it to GPU for detecting, send the result to CPU and finally output the result.

Assume the packet has N characters, the algorithm will create N threads in GPU if the device has this ability, and else they will create maximum threads which under the device's ability, then each thread will loop many times to detecting the whole packets. The process flow is showed by Fig.5.

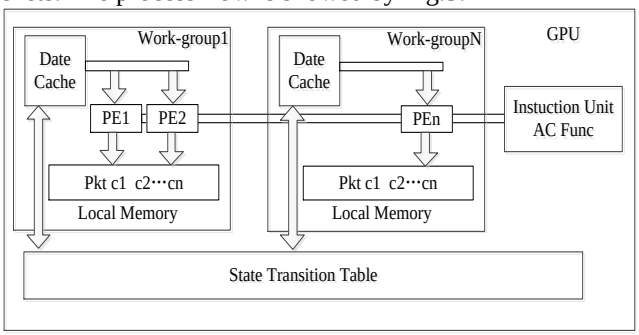

Fig.5 One Packet to GPU

This method is very simple, because there is only one packet, and the state transition table which transferred to GPU also has one. So this method need not to find out which state transition table is corresponding to which packet. A drawback of this approach is that the input stream is very small and the DMA time occupied most of the time, so the GPU is not utilized effectively.

3.2 Transferring Multi-packets to GPU

In this approach, we will mark the packets which we interfered to GPU as unique packetID, and give a unique tableID for each state transition table which finished creation process and transfer all the state transition tables to GPU. The whole process will be finished at the initialization phase of snort.

Using this approach to detect packets, the way to creating threads is the same as method (1), and the difference is the packet must correspond to the state transition table. And this could be solved adding elements packetID and tableID to struct ACSM, and we will also transfer those elements to GPU. In the OpenCL algorithm, we must to judge the packet boundaries in order to get the correct results. The process flow is showed by Fig.6, and example of packets collecting process is showed by Fig.7. From Fig.7, each packet corresponds to a state transition table, so when we transfer packets to GPU, we must to determine the transition table's address corresponding to each packet.

Although this method is complicated comparing with method (1), the input stream transferring into GPU once time is much more than method (1), and the GPU is utilized effectively.

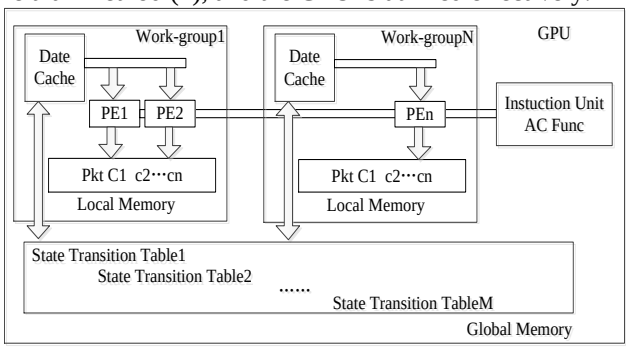

Fig.6 Multi-Packets to GPU

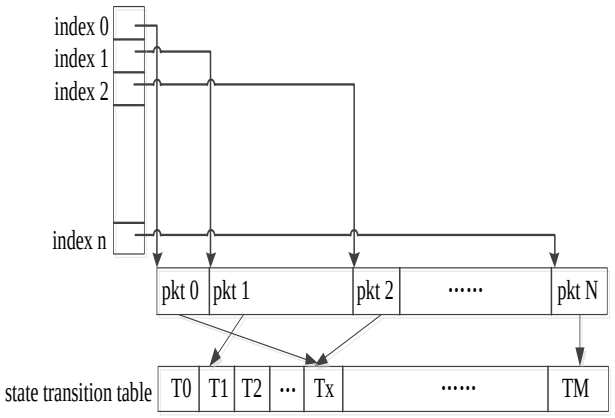

Fig.7 Packet Collecting

4 Evaluation

Pattern matching is the most critical operation in the snort system. Usually pattern matching algorithm can be classified into single pattern matching algorithm (such as KMP) and multi-patterns algorithm (such as AC).

In this section, we explore the performance of our implementation. We realize the two approaches in Snort and compare the two methods with the original snort respectively. In processing multi-packets method, we change the parameter about the collecting packet number once time then get the average time about processing one packet.

In our experiments we used an AMD A10-4600m computer, the CPU in this computer is 2.3GHz APU with Radeon™ HD Graphics 4 processor , 8G memory and GPU is AMD Radeon HD 7660G card, the operating system is Ubuntu 12.04 64-bit.

We get the packets data LLS DDOS 1.0-inside.dump from MIT Lincoln Laboratory [17] as the detected data, we also using snort to dump some small packets date set using the detected data LLS DDOS 1.0-inside.dump, such as contain 200 packets date set, 1000 packets date set, 10000 packets date set and 20000 packets date set, and we read the packets from disk rather than network in order to get the same speed of capture packet in different experiments. We also using the default rules file when using different version snort and this can ensure the correctness of the result.

For all experiments, we disregard the time spent in the initialization phase of snort as well as the log of the alerts to the disk or terminal. Even though it only takes just a few seconds to load rule files and build its internal structures. And we used the full AC implementation to measure the performance in original snort.

4.1 Performance Comparison between One Packet OpenCL snort and Original snort

In this experiment, input1, input2 and input3 are three different size detected packets and the packets size is 200, 1000 and 10000 respectively. We change the input packet numbers to get the performance data about one packet OpenCL snort and original snort and the performance data is showed by Fig.8. From Fig.8, (1) with the increase of input packet size, the throughput of two methods becomes large; (2)

using one packet OpenCL snort, the throughput is not batter than the original snort's throughput, because the local memory is not large enough, the state transition table is stored in Global memory, when judge the current character meet the conversion criteria or not each time, the algorithm must access the global memory once time; and most of threads are terminated at the beginning of the algorithm, and the GPU's utilization is not high.

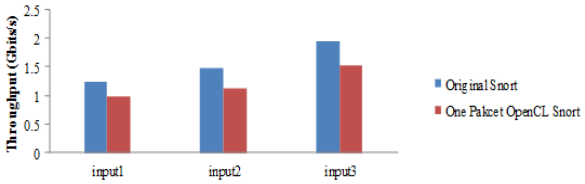

Fig.8 Performance Comparison

4.2 Performance Analysis about Multi-Packets OpenCL snort

In this experiment, we get the performance comparison about multi-packets OpenCL snort and one packet OpenCL snort. Before this comparison, first thing we must to ensure is when we transfer how many packets to GPU, the algorithm will get the best performance and maximum throughput. Fig.9 showed the algorithm's performance comparison when transferring different number of packets to GPU. From Fig.9 we can see with the number's difference, the throughput has some difference as well. When the number which transfers to GPU once time is 30, the throughput is 4.78Gbits/sec, when the number is 100, the throughput is up to 6.43Gbits/sec. And when the number changes from 150 to 200, the throughput grows slowly and then it has a downward trend. So we select 200 as the number which transfers to GPU once time.

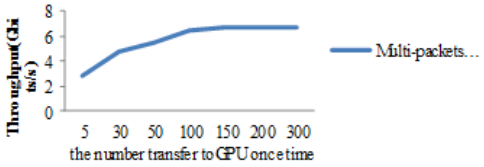

Fig.9 Performance Comparison of Multi-packets OpenCL Snort

The next experiment we are focus on is the performance comparison about the three version snort: original snort, one packet OpenCL snort and multi-packets OpenCL snort. And the result is showed by Fig.10. In this figure, input1, input2 and input3 are three different size detected packets as the same as Fig.9, the packets size is 150, 1200 and 10000 respectively. From the result, we can see the multi-packets OpenCL snort's throughput is about two times faster than other two methods. And the GPU's utilization in multi-packets OpenCL snort is much higher than the one packet OpenCL snort.

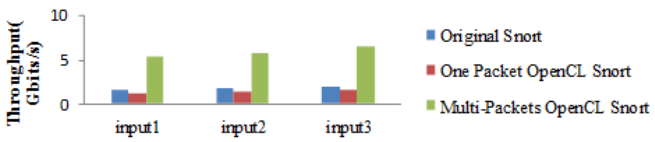

Fig.10 Performance Comparison on Different Versions

5 Conclusions

In this paper, we have proposed two OpenCL version snort, one packet OpenCL snort and multi-packets OpenCL snort in order to accelerate packet detecting by GPU. And the result showed that although one packet OpenCL snort's throughput is about 20% slower than original snort, multi-packets OpenCL snort is about 2 times faster than original snort, and this system was able to achieve a maximum throughput of 6.758Gbit/s.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No. 61402528, all support is gratefully acknowledged.

References

[1] Snort: : Home Page. http://www.snort.org/.
[2] Application Layer Packet Classifier for Linux. http://17-filter.sourceforge.net/.
[3] Clam AntiVirus. http://www.clamav.net/.
[4] S. Antonatos, K. Anagnostakis, and E. Markatos. Generating realistic workloads for network intrusion detection systems. In Proceedings of the 4th ACM Workshop on Software and Performance, (2004).
[5] J. B. D. Cabrera, J. Gosar,W. Lee, and R. K. Mehra. On the statistical distribution of processing times in network intrusion detection. In 43rd IEEE Conference on Decision and Control, 75-80, (2004).
[6] D. L. Cook, J. Ioannidis, A. D. Keromytis, and J. Luck. Cryptographics: Secret key cryptography using graphics cards. In Proceedings of RSA Conference, Cryptographer's Track. 334-350, (2005).
[7] G. G. R. I. Lodovico Marziale and V. Roussev. Massive threading: Using GPUs to increase the performance of digital forensics tools. Digital Investigation. 73–81.
[8] N. Jacob and C. Brodley. Offloading IDS computation to the GPU. In Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference, Washington, DC, USA, IEEE Computer Society. 371–380, (2006).
[9] Lin CH, Tsai SY, Liu CH, Chang SC, Shyu. JM Accelerating string matching using multi-threaded algorithm on gpu. In: GLOBECOM, 1-5, (2010).
[10] C. IOS. IPS deployment guide. http://www.cisco.com/.