algorithms

MDPI

Article

Length-Bounded Hybrid CPU/GPU Pattern Matching Algorithm for Deep Packet Inspection

Yi-Shan Lin [1], Chun-Liang Lee [2,*] and Yaw-Chung Chen [1]

[1] Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan; yishanl@cs.nctu.edu.tw (Y.-S.L.); ycchen@cs.nctu.edu.tw (Y.-C.C.)

[2] Department of Computer Science and Information Engineering, School of Electrical and Computer Engineering, College of Engineering, Chang Gung University, Taoyuan 33302, Taiwan

* Correspondence: cllee@mail.cgu.edu.tw; Tel.: +886-3-211-8800 (ext. 5196)

Academic Editor: Andras Farago

Received: 29 November 2016; Accepted: 11 January 2017; Published: 18 January 2017

Abstract: Since frequent communication between applications takes place in high speed networks, deep packet inspection (DPI) plays an important role in the network application awareness. The signature-based network intrusion detection system (NIDS) contains a DPI technique that examines the incoming packet payloads by employing a pattern matching algorithm that dominates the overall inspection performance. Existing studies focused on implementing efficient pattern matching algorithms by parallel programming on software platforms because of the advantages of lower cost and higher scalability. Either the central processing unit (CPU) or the graphic processing unit (GPU) were involved. Our studies focused on designing a pattern matching algorithm based on the cooperation between both CPU and GPU. In this paper, we present an enhanced design for our previous work, a length-bounded hybrid CPU/GPU pattern matching algorithm (LHPMA). In the preliminary experiment, the performance and comparison with the previous work are displayed, and the experimental results show that the LHPMA can achieve not only effective CPU/GPU cooperation but also higher throughput than the previous method.

Keywords: network security; pattern matching algorithm; deep packet inspection; intrusion detection system; general-purpose graphics processing unit; compute unified device architecture

1. Introduction

Deep packet inspection (DPI) is a technique that examines the packet content to ensure the network security. While packet headers can be examined for blocking malicious packets, DPI is responsible for checking packet contents for application awareness [1–3] demanded by modern networks. The network intrusion detection system (NIDS) is one of the DPI applications for network security. The anomaly-based and signature-based NIDSs are two forms of NIDSs to monitor the network traffic. Anomaly-based NIDSs determine whether the behaviors in the network are normal or anomalous [4–6]. In contrast, signature-based NIDSs inspect whether the packet payloads contain any malicious content. Specific data strings considered as malicious contents and appearing in the packet payloads are called "signatures" or "patterns". A collection of those malicious strings is called a "pattern set". Once any pattern is recognized in a packet payload, the NIDS will alert or drop this packet in order to protect the resources in the network. In the system, the pattern matching algorithm is needed to recognize the patterns, which is the core mechanism for searching patterns in the payloads.

Designing an efficient pattern matching algorithm has become a priority because pattern matching is very time-consuming and can occupy around 70% of system execution time [7,8]. Several pattern matching algorithms for signature-based NIDSs designed on software platforms have been proposed. In order to satisfy the required processing speed in high speed networks, some research focused on

implementing the pattern matching algorithms on the graphic processing units (GPUs) [9]. Since the capability of central processing units (CPUs) cannot meet the performance requirement for such networks nowadays, GPUs with more parallel processing power than CPUs may be deployed. However, although GPU processing power is superior, the overall performance of GPUs is limited due to the GPU overhead. The GPU overhead consists of kernel launch latency and data transfer latency; this limitation must be taken into account during the development. As a consequence, designing an efficient cooperative pattern matching algorithm between CPUs and GPUs becomes an essential issue.

Here, we focus on designing the software-based and collaborative pattern matching algorithms. Initially, a hybrid CPU/GPU pattern matching algorithm (HPMA) [10] has been proposed. We designed HPMA based on the considerations that CPU features performance degradation with the computation and memory-intensive operations such as pattern matching algorithms, and GPU efficiency is limited with the data transfer overhead via the peripheral component interconnect express (PCIe) channel. Hence, an HPMA procedure was divided into "pre-filtering" and "full pattern matching", which is executed, respectively, by CPUs and GPUs. CPUs provide a rapid and high-filtration-ratio pre-filtering to classify the "normal" and "suspicious" packets first, and then the suspicious packets are delivered to GPU for full pattern matching. The normal packets can pass through directly. This mechanism can not only lessen the computation and memory-intensive operations for CPU, but also reduce CPU–GPU data transfer overhead to achieve the task allocation balancing. The experiment showed that the HPMA brought higher efficiency than the CPU-only and GPU-only full pattern matching algorithms, indicating that such collaboration is effective. However, the HPMA performance may be restricted under a few conditions. For example, the input traffic with varying length packets can cause a large variation of thread execution time in parallel processing, resulting in throughput degradation.

In this paper, a length-bounded hybrid CPU/GPU pattern matching algorithm (LHPMA) is proposed to deal with this problem. Once an incoming packet is loaded, the LHPMA categorizes the packet in advance according to a pre-determined payload length bound. The packets whose lengths exceed the bound are delivered to the CPU rapid prefiltering; otherwise, the packets are assigned to the GPU full pattern matching directly. Higher performance can be obtained by means of reducing the payload length diversity. In the experiment, the previous methods are also implemented and compared with the LHPMA, and the results demonstrate that the LHPMA can enhance the performance.

The rest of this paper is organized as follows: Section 2 describes typical pattern matching algorithms and some previous studies of software-based implementation on CPU and GPU platform applied to NIDSs. Section 3 illustrates the proposed LHPMA, including the overall architecture, components, procedure and key algorithm. Section 4 demonstrates the experiment setup and results. The conclusions and future work are presented in Section 5.

2. Related Work

Pattern matching algorithms can be categorized as single-pattern matching and multi-pattern matching. The former finds a single pattern within a string at one time; for instance, the Knuth–Morris–Pratt (KMP) [11] and the Boyer–Moore (BM) [12] algorithms are well known. The latter can recognize multiple patterns within a string simultaneously, in which the Aho–Corasick (AC) [13] and Wu–Manber (WM) [14] algorithms are widely used.

Implementing the algorithms by software on general-purpose processors (GPPs) is a trend that presents the advantages of lower cost and higher scalability than implementing it by hardware. Earlier, some algorithms were carried out on the central processing units (CPUs) [15–17] to handle the incoming packets; nevertheless, such processing did not satisfy the required performance for high speed networks. Therefore, some studies began to focus on developing pattern matching algorithms using graphical processing units (GPUs), which offer superior parallel processing power compared to CPUs. Related studies with regard to designing algorithms on GPUs include Jacob and Bordely [18] proposing a modified KMP algorithm "PixelSnort", which was based on the conventional Snort [19] system. PixelSnort involved off-loading packet processing to the GPU by respectively encoding the

packets and patterns into pixels and textures to perform full matching. The result showed that their method outperformed Snort by up to 40% in terms of packet processing rate, but the improvement occurred only in heavy-load conditions. Vasiliadis et al. also modified the Snort and proposed a novel system "Gnort" [20], in which the AC algorithm was ported to the GPU. The input packets were transferred to the GPU for full pattern matching, and the matched results were sent back to the CPU. The maximum throughput of 2.3 Gbps in the synthetic traffic was achieved, which was two times faster than the Snort in real traffic. Vasiliadis et al. further proposed one more parallel processing architecture named "MIDeA" [21] that optimized the parallelism of network interface card (NIC), CPUs and GPUs to improve the processing performance. With real traffic input, MIDeA presented the throughput of 5.2 Gbps in off-the-shelf equipment. Vespa and Weng [22] presented an optimized pattern matching algorithm "GPEP", which achieves higher efficiency by means of low operational complexity and small-size state transition tables. Jamshed et al. presented a software-based pattern matching method named "Kargus" [23] processed by CPU and GPU, which also involved off-loading packet processing to the GPU. The result presented that Kargus outperformed the existing state-of-the-art software IDS by 1.9 to 4.3 times. Zu et al. [24] focused on porting nondeterministic finite automaton (NFA) to the GPU platform. Their proposed data structure and parallel processing achieved a throughput of 10 Gbps. Yu and Becchi [25] studied more optimal porting to the GPU platform for deterministic finite automaton (DFA) and NFA. They pointed that using small data tables was not the only way to improve the matching efficiency; the pros and cons of porting automaton to GPU were also described. On the other hand, some literature focused on porting algorithms to other processors [26] and optimizing NFA [27] to further improve efficiency.

However, even though GPUs feature superior parallel processing power, the data transfer latency between CPU and GPU impacts the overall performance of pattern matching algorithms. Literature [28] showed that, in GPU program development, the GPU overhead (i.e., kernel launch latency and data transfer latency) should be taken into account. The former indicates the latency of the triggering kernel (the function executed by GPU), and the latter indicates the latency of data transfer between CPU and GPU via PCIe channels. Especially, the data transfer latency is higher than the kernel launch latency. In other words, allocating all tasks to GPU may not achieve the best efficiency in certain cases, even with its powerful parallel processing. Thus, other studies began to concentrate on CPU and GPU collaboration design to improve the efficiency.

For this purpose, we focused on designing effective cooperation methods based on CPUs and GPUs. Our previous work HPMA [10] divides the processing tasks into two parts: rapid packet prefiltering and full pattern matching, as shown in Figure 1. First, the incoming packets are delivered to the CPU host memory, and then the CPU accessed the packets and started the prefiltering according to the lookup tables (T_1, *Base Table*, *Indirect Table* and T_2). The table size is sufficiently small so that it can reduce the memory access latency to decrease the pre-filtering time. In addition, a high filtration ratio is also achieved so that the number of packets can be decreased to reduce the data transfer latency. Next, the prefiltered packets considered "suspicious" are sent to the GPU for the full pattern matching process. We adopted the AC algorithm into the GPU platform, and the AC lookup tables were stored in the GPU texture memory for the algorithm use, since the access latency of texture memory is much lower than that of device memory. The matched result is copied back to the CPU host memory after the full pattern matching is completed. By using the pre-filtering method in HPMA, the CPU can perform rapid pre-filtering and reach a high filtration ratio so that the GPU latency can be decreased and make the processing more efficient. HPMA outperformed the CPU-only and GPU-only pattern matching algorithms by 3.4 and 2.7 times, respectively, with input traffic of 1460-byte random payloads, indicating that the collaboration between the CPU and GPU can improve overall processing speed.

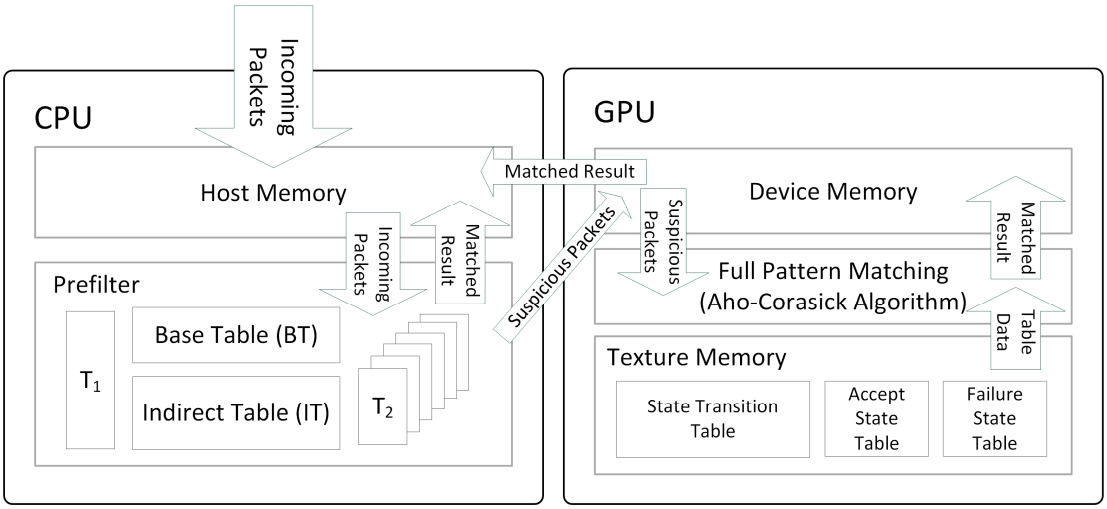

Figure 1. Hybrid CPU/GPU pattern matching algorithms.

Nevertheless, the HPMA performance may be restricted under the following conditions: (a) all incoming packets are finally delivered to the GPU, causing redundant operations rather than directly delivering packets to the GPU; and (b) the input traffic consists of varying length packets that cause execution time inconsistency of each thread in parallel processing. For these limitations, we have proposed another design, a capability-based hybrid CPU/GPU pattern matching algorithm (CHPMA) [29] to face the former condition. The CHPMA first estimates the CPU and GPU processing capability of the system, and, during runtime, the CHPMA can automatically perform the processing selection according to the historical filtration ratio and capability estimation result. To face the latter condition, we propose a length-bounded hybrid CPU/GPU pattern matching algorithm in this paper, as illustrated in the next section.

3. Length-Bounded Hybrid CPU/GPU Pattern Matching Algorithm

3.1. Overall Architecture and Procedure

Figure 2 illustrates the architecture of LHPMA. As for the cooperation, the CPU is responsible for pre-filtering the packets, and the GPU is responsible for inspecting the packets by full pattern matching. Here, an additional process, the "length-bounded separation algorithm" (LBSA), is executed by the CPU. Different from the previous work, the incoming packets pass the LBSA before they are sent to the prefilter buffer in the CPU host memory, in which those packets to be processed by the prefilter are stored. The task of LBSA is checking the payload lengths and determining the allocation of those packets. The LBSA is described in the next subsection. After the LBSA process, some packets are stored in the prefilter buffer and thus delivered to the CPU prefiltering (dashed-line arrow); on the other hand, the other packets are directly sent to the full-matching buffer in the CPU host memory (dotted-line arrow), which is used for storing the packets to be processed by the GPU for full pattern matching. In addition, when the prefilter buffer becomes full, the prefiltering is executed; prefiltered packets are also sent to the full-matching buffer (dashed arrow).

Once the full matching buffer is full, the buffered packets are transferred to the GPU device memory and accessed by the full pattern matching algorithm. Here, the full pattern matching is implemented using the AC algorithm, which requires the lookup tables (state transition table, accept states table and failure states table) being copied to the GPU texture memory to accelerate the inspection speed because of the much lower access latency. Finally, the matched result from the full pattern matching algorithm is sent back to the CPU. Figure 3 illustrates the overall procedure of LHPMA.

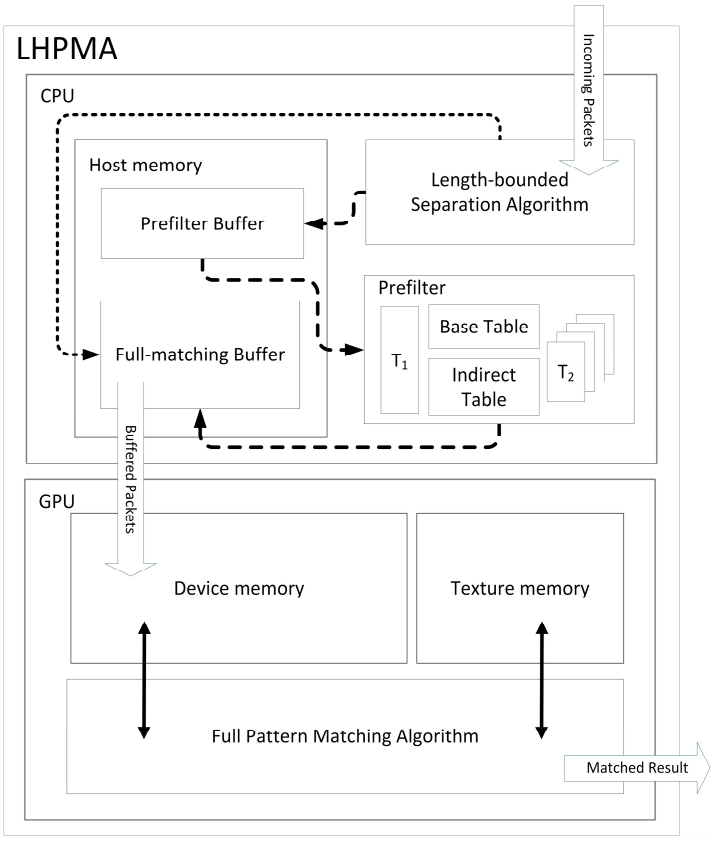

Figure 2. LHPMA architecture.

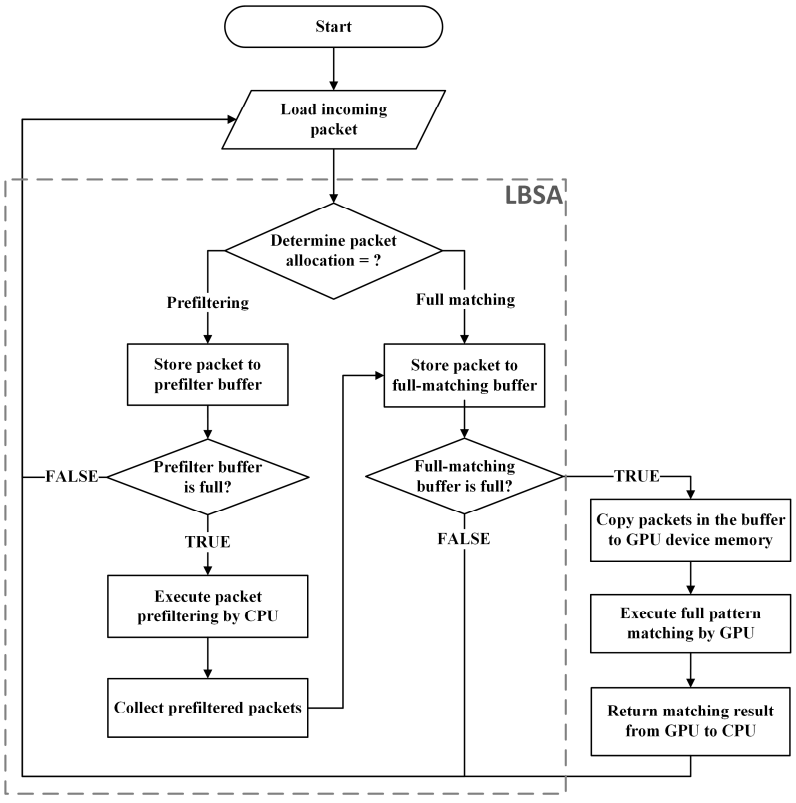

Figure 3. LHPMA overall procedure.

3.2. Length-Bounded Separation Algorithm

Algorithm 1 describes the LBSA. This algorithm categorizes the incoming packets in advance according to a pre-determined payload length bound in order to reduce the payload length diversity for parallel processing. Given the current processing of packet payload set P and the length bound L_b, the LBSA can determine the payloads in P with L_b.

Initially, the prefilter buffer T_{PB} and the full matching buffer T_{FB} are empty. In addition, a payload set $P_{filtered}$ is declared to temporarily store the prefiltering results, which is also empty (Lines 1–3). A variable idx_{PB}, the index of prefilter buffer entries, is set to zero (Line 4). For each payload in P, the LBSA measures the payload length of packet P_i (denoted as L_i) for comparing it with the length bound L_b (Lines 5 and 6). If the payload length L_i exceeds the bound (longer than L_b), the system stores this packet in the prefilter buffer T_{PB} and waits for the CPU prefiltering process (Lines 7 and 8). After the packet is stored in T_{PB}, the index of prefilter buffer entries idx_{PB} increases for the next insert (Line 9). Otherwise, this packet will be stored in the full matching buffer T_{FB} to wait for the GPU full pattern matching (Lines 10 and 11).

Once the prefilter buffer T_{PB} is full, all of the packets in this buffer are delivered to the prefilter for processing. Then, the resulting payload set $P_{filtered}$ is obtained and stored in the full matching buffer T_{FB} (Lines 14–16), and the system clears the data stored in $P_{filtered}$, T_{PB} and idx_{PB} and continues the process for the next batch of incoming packets (Lines 17–19). On the other hand, once the full matching buffer T_{FB} becomes full, all of the packets in this buffer are copied to the GPU device memory and the full pattern matching process is executed. Finally, the matching result is returned, and system clears the data stored in T_{FB} (Lines 22–24).

Algorithm 1: Length-Bounded Separation Algorithm (LBSA).

```
     Input: P (packet payload set) and L_b (length boundary).
1    T_PB ← empty; // Prefilter buffer
2    T_FB ← empty; // Full matching buffer
3    P_filtered ← empty; // Prefiltered payloads
4    idx_PB ← 0; // Index of prefilter buffer
5    foreach P_i do
6        L_i ← payload length of P_i;
7        if L_i > L_b then
8            T_PB [idx_PB] ← P_i; // store P_i in prefilter buffer
9            idx_PB ← idx_PB + 1;
10       else
11           T_FB ← T_FB ∪ P_i; // store P_i in full matching buffer
12       end
13
14       if T_PB is full then
15           P_filtered ← prefiltering(T_PB); // execute prefiltering algorithm and
16           T_FB ← T_FB ∪ P_filtered; // store P_filtered in full matching buffer
17           P_filtered ← empty;
18           T_PB ← empty;
19           idx_PB ← 0;
20       end
21
22       if T_FB is full then
23           result ← fullmatching(T_FB); // execute full matching algorithm and
24           T_FB ← empty; // return the matching result
25       end
26   end
```

In this design, shorter packet payloads will be processed by the GPU that benefits the overall efficiency. Since using short packets is a typical way to attack [30], a large number of packets can be generated and sent in a shorter time; shorter packets are more likely malicious. If such packets are sent to the GPU directly, the redundant operations of CPU prefiltering caused by these packets can be reduced. Moreover, shorter packets can be collected into a larger number of packets that is favorable for the GPU because of much higher parallel processing power. On the other hand, longer packets may possibly pass the prefilter without being delivered to the GPU for complete inspection. Longer packet payloads—namely, most of the packet bytes—can be reduced by the prefilter to offload the GPU processing and decrease the data transfer latency.

4. Experiments

In this section, the experiments are demonstrated, including the experimental setup and results. In the experiments, both the performance and comparison are illustrated.

4.1. Experimental Setup

The prefilter and full pattern matching algorithms were implemented respectively based on the platform of Open Multi-Processing (OpenMP) [31] and NVIDIA's (Nvidia Corporation, Santa Clara, CA, USA) Compute Unified Device Architecture (CUDA) [32–35] programmed with multithreading for parallel processing. The hardware for this experiment is shown in Table 1. The CPU and GPU used here were respectively Intel Core i7-3770 (Intel Corporation, Santa Clara, CA, USA) with four processor cores and NVIDIA GeForce GTX680 (NVIDIA Corporation, Santa Clara, CA, USA) with 1536 processor cores. For the parameters, the number of blocks and the number of threads per block were, respectively, 128 and 64 for the CUDA setting. In addition, the sizes of full matching buffer and pre-filter buffer were, respectively, 256 MB and 1 MB for the configuration.

Table 1. Hardware specification in the experiments.

Device	Specification
CPU	Intel Core i7-3770 (3.40 GHz) Number of cores: 4 Host memory: 8 GB DDR3
GPU	NVIDIA GeForce GTX680 (1058 MHz) Number of cores: 1536 Device memory: 2 GB GDDR5

We took the Snort rules 2008 [19], from which we extracted and generated the pattern set for the experiment. The pattern lengths range from 1 to 208 bytes with an average of 13.8 bytes. The lookup tables for prefilter and full pattern matching algorithms were generated with the generated pattern set and loaded to our system. A real packet trace was used as the input of incoming packets, which is the web traffic extracted from the portal websites of Google, Amazon and Yahoo. Table 2 lists the payload length statistics of the used packet trace, showing that this trace consists of various-length packets.

When the system started, those packets from the trace were loaded, allocated and inspected. The execution times of LHPMA and other methods being compared were measured and recorded by the system during the experiment; finally, the throughput results were calculated. In addition, the system also recorded other information such as the number of total incoming packets, full-matched packets and their lengths, in order to observe the distribution and relationship between them.

Table 2. Payload length distribution of input traffic.

Payload Length (Bytes)	Count	Ratio
<100	3334	0.009
100–300	6664	0.018
300–500	32,919	0.090
500–700	14,670	0.040
700–900	4326	0.012
900–1100	3946	0.011
1100–1300	4441	0.012
>1300	297,419	0.809

4.2. Experimental Results

Figures 4–8 show the preliminary experiment results, in which the notations "LHPMA-AC" and "HPMA-AC", respectively, indicate the LHPMA and HPMA matching methods, and both methods ported the AC algorithm to handle the full pattern matching task. On the other hand, the notations "AC-GPU" and "AC-CPU" are the matching methods that directly inspect the packets by the AC algorithm using the GPU and CPU, respectively. Before exhibiting the result of LHPMA, we first discuss some factors that affect the GPU processing performance. Figure 4 illustrates the data transfer rate between the CPU and GPU with different size of the full matching buffer. The *x*-axis represents the buffer sizes, and the *y*-axis indicates the average data transfer rates performed in Gbps. It can be observed that the data transfer rate presented was very low when the buffer size was small; namely, few packet payloads were transferred at one time, and the overall GPU overhead would be significantly high. For instance, when the buffer size was 0.25 MB, the data transfer rate was 1.95 Gbps. On the other hand, when the buffer size was large enough, more packet payloads were transferred together at one time, and the performed data transfer rate would be higher. For instance, when the buffer size was 256 MB, the data transfer rate was 35.1 Gbps. Note that the optimal solution is not setting the full matching buffer size as large as possible, since too large a buffer size may cause the packet payloads in this buffer to wait for too much time and thus affect the overall performance. Therefore, the proper choice was located at the turning point of the data transfer rate convergence, 256 MB.

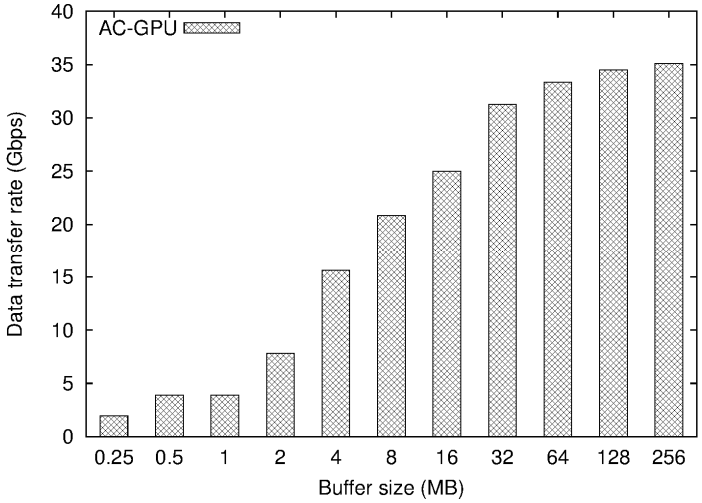

Figure 4. The data transfer rate between CPUs and GPUs with different full matching buffer sizes.

Figure 5 illustrates the GPU performance with different sizes of the full matching buffer. The *x*-axis represents the buffer sizes, and the *y*-axis indicates the average throughputs performed in Gbps. Here, both short (100-byte) and long (1460-byte) packet payload sets were prepared, which were represented with "AC-GPU-100" and "AC-GPU-1460", respectively. As Figure 4 presented, the

performance became low with small buffer size, especially in the case of AC-GPU-1460 with 2 MB buffer size. In addition to the data transfer rate, the number of packet payloads were few in this case, resulting in low GPU thread utilization and thus degrading overall performance. In other words, the input payload set length also affects the GPU efficiency. Shorter packets can be collected into a larger number of packets, which is favorable for the parallel processing power of GPU—hence, the efficiency of AC-GPU-100 displayed from 5.37 Gbps to 9.03 Gbps, which was higher than that of AC-GPU-1460, from 2.71 Gbps to 6.10 Gbps.

Figure 6 presents the GPU performance with different lengths of payload sets. Here, a total of six payload sets were prepared, which are 250-byte to 1460-byte (the maximum length of payload) long, as represented in the *x*-axis. The *y*-axis indicates the average throughputs performed in Gbps. The result shows that shorter payloads are favorable to the GPU efficiency, in which the GPU performed up to 8.0 Gbps with a 250-byte set. As the payload length increased, an obvious performance degradation occurred in the case of 750-byte set, and the throughput gradually decreased to 5.9 Gbps with a 1460-byte set. Since the number of short payloads in the GPU device memory is more than that of longer payloads, more benefits from multithread parallel processing can be made due to the number of GPU cores being much larger than the number of CPU cores.

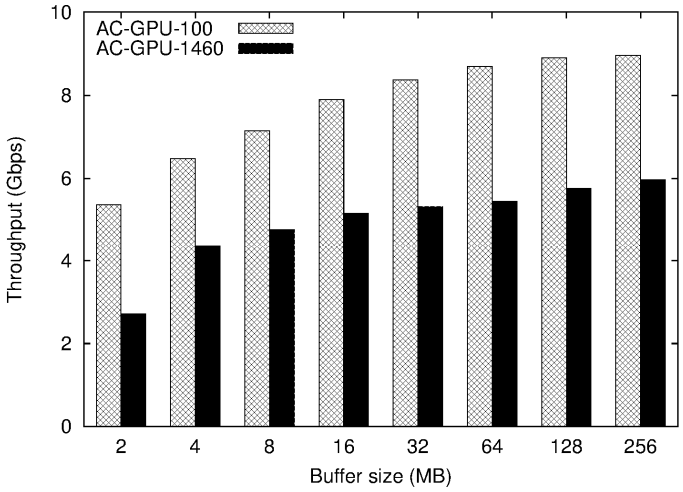

Figure 5. The GPUs with different full matching buffer sizes.

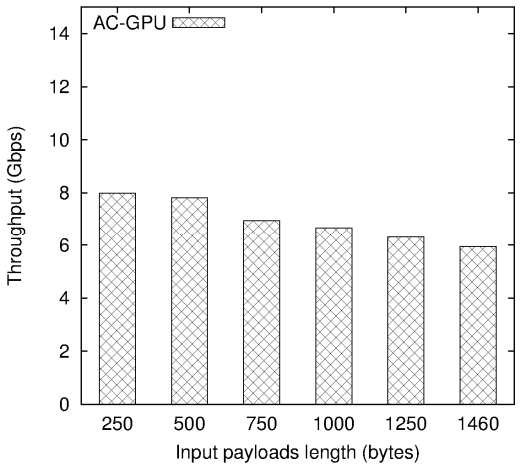

Figure 6. The performance of GPU processing with different lengths of input payloads.

With respect to the LHPMA results, we initially observe the effect of payload length bound on LHPMA. The LHPMA performance, with the packet trace described in the previous subsection as

input, and different settings of bound L_b are displayed in Figure 7. The x-axis represents the selected L_b values, and the y-axis indicates the average throughputs in Gbps. According to this figure, the case of $L_b = 0$ indicates that all of the incoming packets passed the prefilter with performance 12.7 Gbps. As the L_b value increased, the performance upgraded to 14.0 Gbps. In particular, a specific case appeared with $L_b = 750$, which corresponded to the result in Figure 6. In other words, since the payload length diversity is reduced, some performance gains were thus obtained. On the other hand, the case of $L_b = 1460$ revealed a performance degradation of 5.9 Gbps. This is because all of the incoming packets were sent to the GPU without passing the prefiltering. The degradation was caused by both data transfer latency and payload length diversity.

For demonstrating the improvement to the previous work HPMA and the CPU/GPU cooperation that is more efficient than using either CPU only or GPU only, the comparison is displayed in Figure 8. The input traffic is also the packet trace described in the previous subsection. The x-axis represents the different matching methods, and the y-axis indicates the average throughputs performed in Gbps. Here, the middle of the bound, $L_b = 750$, is chosen for the LHPMA-AC. According to this figure, the LHPMA-AC, HPMA-AC, AC-GPU and AC-CPU, respectively, performed 13.8, 12.7, 5.9 and 4.6 Gbps, indicating that the LHPMA enhanced performance more than the HPMA and outperformed the AC-GPU and the AC-CPU methods by 2.3 and 3.0 times, respectively.

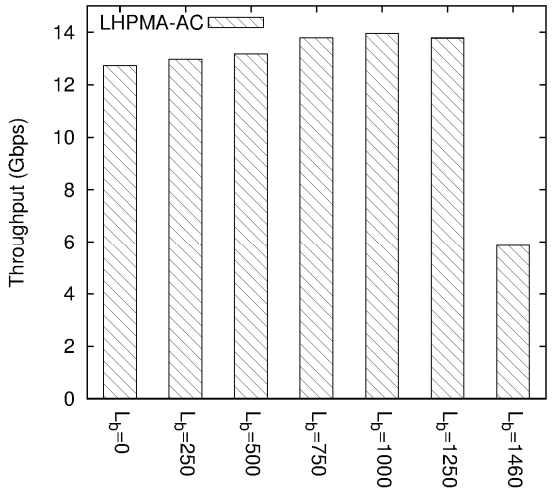

Figure 7. The performance of LHPMA with different payload length boundaries.

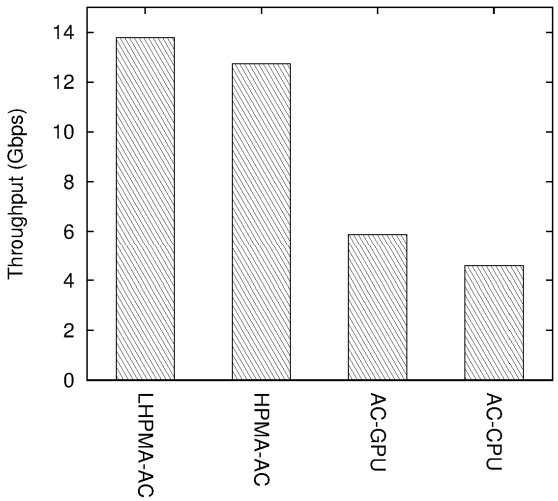

Figure 8. The performance of LHPMA and comparison with the previous methods.

Since the conditions of other related work were quite different each other (e.g., configuration setting, input packet traffic set, pattern set and hardware specification), it is hard to make comparisons. Therefore, we provide a presentation referred from [26] that illustrates the efficiency per cost of our method, as presented in Table 3. According to the resulting LHPMA throughput (13.8 Gbps) and the total processor cost of CPU and GPU being $875, the throughput per dollar of the LHPMA was 16.15 Mbps/$.

Table 3. LHPMA efficiency per cost.

Processor Cost ($)	Throughput (Gbps)	Throughput per Dollar (Mbps/$)
$875	13.8	16.15

Table 4 presents the full-matched (considered intrusive) packet distribution of the input packet trace traffic. The result shows that most of the short packets are intrusive; even the number of longer packets are much more in this case, and the ratio of intrusion in the shorter packets are up to 88% higher than that in longer packets. It indicates that shorter packets are usually selected as the way to generate the intrusion.

Table 4. The ratio of number of full-matched packets and total number of packets in input traffic.

L_b = 750	Full-Matched/Total Packets	Ratio
$L_i \leq L_b$	54,511/61,929	88%
$L_i > L_b$	80,549/328,991	24%

5. Conclusions

In this paper, the LHPMA, a pattern matching algorithm based on CPU and GPU cooperation for packet inspection is proposed, which is an enhanced version of our HPMA to improve efficiency in the case of various payload lengths in input traffic. This work achieves the goal by automatically categorizing and allocating the incoming packets to the corresponding process, in which the categorizing depends on the pre-determined payload length bound to reduce the payload length diversity. We present the experiments, which consists of the GPU efficiency affected by the payload sets with different lengths, the LHPMA performance affected by different payload length bounds and the performance comparison with other methods. The results demonstrate that the LHPMA can outperform the HPMA and the CPU and GPU individual processing methods. Since the LHPMA is an initial design, future work involves analyzing the input traffic with different payload length distributions and processing overhead to have a more advanced design of the LHPMA method.

Acknowledgments: This work was supported in part by the High Speed Intelligent Communication (HSIC) Research Center of Chang Gung University, Taoyuan, Taiwan, and by grants from the Ministry of Science and Technology of Taiwan (MOST-101-2221-E-009-004-MY3 and MOST-102-2221-E-182-034) and Chang Gung Memorial Hospital (BMRP 942).

Author Contributions: Yi-Shan Lin and Chun-Liang Lee conceived and designed the experiments; Yi-Shan Lin performed the experiments; Yi-Shan Lin and Chun-Liang Lee analyzed the data; Yi-Shan Lin contributed reagents/materials/analysis tools; and Yi-Shan Lin, Chun-Liang Lee and Yaw-Chung Chen wrote the paper. All authors have read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Handley, M.; Paxson, V.; Kreibich, C. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In Proceedings of the Symposium on USENIX Security, Washington, DC, USA, 13–17 August 2001; pp. 115–131.

2. Kruegel, C.; Valeur, F.; Vigna, G.; Kemmerer, R. Stateful intrusion detection for high-speed networks. In Proceedings of Symposium on Security and Privacy, Oakland, CA, USA, 12–15 May 2002; pp. 285–293.
3. Paxson, V. Bro: A system for detecting network intruders in real-time. *Comput. Netw.* **1999**, *31*, 2435–2463. [CrossRef]
4. Tian, D.; Liu, Y.H.; Xiang, Y. Large-scale network intrusion detection based on distributed learning algorithm. *Int. J. Inf. Secur.* **2009**, *8*, 25–35. [CrossRef]
5. Beghdad, R. Critical study of neural networks in detecting intrusions. *Comput. Secur.* **2009**, *27*, 168–175. [CrossRef]
6. Wu, J.; Peng, D.; Li, Z.; Zhao, L.; Ling, H. Network intrusion detection based on a general regression neural network optimized by an improved artificial immune algorithm. *PLoS ONE* **2015**, *10*, e0120976. [CrossRef] [PubMed]
7. Antonatos, S.; Anagnostakis, K.G.; Markatos, E.P. Generating realistic workloads for network intrusion detection systems. *ACM SIGSOFT Softw. Eng. Notes* **2004**, *29*, 207–215. [CrossRef]
8. Cabrera, J.B.; Gosar, J.; Lee, W.; Mehra, R.K. On the statistical distribution of processing times in network intrusion detection. In Proceedings of the Conference on Decision and Control, Woburn, MA, USA, 14–17 December 2004; Volume 1, pp. 75–80.
9. General-Purpose Computation Using Graphics Hardware. Available online: http://www.gpgpu.org (accessed on 24 November 2016).
10. Lee, C.L.; Lin, Y.S.; Chen, Y.C. A hybrid CPU/GPU pattern matching algorithm for deep packet inspection. *PLoS ONE* **2015**, *10*, e0139301. Available Online: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0139301 (accessed on 24 November 2016). [CrossRef] [PubMed]
11. Knuth, D.E.; Morris, J.; Pratt, V. Fast pattern matching in strings. *SIAM J. Comput.* **1977**, *6*, 127–146. [CrossRef]
12. Boyer, R.S.; Moore, J.S. A fast string searching algorithm. *Commun. ACM* **1977**, *20*, 762–772. [CrossRef]
13. Aho, A.V.; Corasick, M.J. Efficient string matching: An aid to bibliographic search. *Commun. ACM* **1975**, *18*, 333–340. [CrossRef]
14. Wu, S.; Manber, U. *A Fast Algorithm for Multi-Pattern Searching*; Department of Computer Science, University of Arizona: Tucson, AZ, USA, 1994.
15. Scarpazza, D.P.; Villa, O.; Petrini, F. Exact multi-pattern string matching on the cell/B.E. processor. In Proceedings of the Conference on Computing Frontiers, Ischia, Italy, 5–7 May 2008; pp. 33–42.
16. Schuff, D.L.; Choe, Y.R.; Pai, V.S. Conservative vs. optimistic parallelization of stateful network intrusion detection. In Proceedings of the International Symposium on Performance Analysis of Systems and Software, Philadelphia, PA, USA, 20–22 April 2008; pp. 32–43.
17. Vallentin, M.; Sommer, R.; Lee, J.; Leres, C.; Paxson, V.; Tierney, B. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In Proceedings of the International workshop on Recent Advances in Intrusion Detection, Queensland, Australia, 5–7 September 2007; pp. 107–126.
18. Jacob, N.; Brodley, C. Offloading IDS computation to the GPU. In Proceedings of the Computer Security Applications Conference, Miami Beach, FL, USA, 11–15 December 2006; pp. 371–380.
19. Snort.Org. Available online: http://www.snort.org (accessed on 24 November 2016).
20. Vasiliadis, G.; Antonatos, S.; Polychronakis, M.; Markatos, E.P.; Iasnnidis, S. Gnort: High performance network intrusion detection using graphics processors. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Cambridge, MA, USA, 15–17 September 2008; pp. 116–134.
21. Vasiliadis, G.; Polychronakis, M.; Ioannidis, S. MIDeA: A multi-parallel intrusion detection architecture. In Proceedings of the Conference on Computer and Communication Security, Chicago, IL, USA, 17–21 October 2011; pp. 297–308.
22. Vespa, L.J.; Weng, N. GPEP: Graphics processing enhanced pattern-matching for high-performance deep packet inspection. In Proceedings of the International Conference on Internet of Things and International Conference on Cyber, Physical and Social Computing, Dalian, China, 19–22 October 2011; pp. 74–81.
23. Jamshed, M.A.; Lee, J.; Moon, S.; Yun, I.; Kim, D.; Lee, S.; Yi, Y.; Park, K. Kargus: A highly-scalable software-based intrusion detection system. In Proceedings of the ACM conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 317–328.
24. Zu, Y.; Yang, M.; Xu, Z.; Wang, L.; Tian, X.; Peng, K.; Dong, Q. GPU-based NFA implementation for memory efficient high speed regular expression matching. *ACM SIGPLAN Not.* **2012**, *47*, 129–140. [CrossRef]

25. Yu, X.; Becchi, M. GPU acceleration of regular expression matching for large datasets: Exploring the implementation space. In Proceedings of the ACM International Conference on Computing Frontiers, Ischia, Italy, 14–16 May 2013; p. 18.
26. Jiang, H.; Zhang, G.; Xie, G.; Salamatian, K.; Mathy, L. Scalable high-performance parallel design for network intrusion detection systems on many-core processors. In Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, CA, USA, 21–22 October 2013; pp. 137–146.
27. Valgenti, V.C.; Kim, M.S.; Oh, S.I.; Lee, I. REduce: Removing redundancy from regular expression matching in network security. In Proceeding of the International Conference on Computer Communication and Networks, Las Vegas, NV, USA, 3–6 August 2015; pp. 1–10.
28. Han, S.; Jang, K.; Park, K.; Moon, S. PacketShader: A GPU-accelerated software router. *ACM SIGCOMM Comput. Commun. Rev.* **2011**, *40*, 195–206. [CrossRef]
29. Lin, Y.S.; Lee, C.L.; Chen, Y.C. A capability-based hybrid CPU/GPU pattern matching algorithm for deep packet inspection. *Int. J. Comput. Commun. Eng.* **2016**, *5*, 321–330. [CrossRef]
30. Douligeris, C.; Serpanos, D.N. *Network Security: Current Status and Future Directions*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
31. OpenMP. Available online: http://openmp.org (accessed on 24 November 2016).
32. Fatahalian, K.; Houston, M. A closer look at GPUs. *Commun. ACM* **2008**, *51*, 50–57. [CrossRef]
33. Nickolls, J.; Buck, I.; Garland, M.; Skadron, K. Scalable parallel programming with CUDA. *ACM Queue* **2008**, *6*, 40–53. [CrossRef]
34. NVIDIA. CUDA Architecture Introduction & Overview. Available online: http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf (accessed on 24 November 2016).
35. NVIDIA. CUDA C Programming Guide. Available online: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (accessed on 24 November 2016).