Full paper

Parallel Keyword Searching utilizing a CUDA GPU for Network Forensics

Napath Pitaksirianan[1], Noppadol Wanichworanant[2*]
Department of Computer Engineering, Faculty of Engineering, Mahidol University
999 Phuttamonthon 4 Road, Salaya, Nakhonpathom 73170, Thailand
Email: noppadol.wan@mahidol.ac.th[*]

Abstract

Nowadays, there is an increasing number of cybercrime on the Internet. Governments, businesses and individuals are in great risk. In order to trace back to criminals, lawful interception may be required. However, it is a complex and computation intensive task. The advent of Graphics Processing Units (GPUs) can offer highly parallel computing power with great performance at a low cost to such a task. This paper; therefore, focus on using a GPU for keyword searching in network traffic. The KMP skip searching algorithm was selected as the experimental string searching algorithm. Various parameters of the GPU were also investigated to obtain the optimal value for this application.

Key Words: Cybercrime, Internet, Criminals, GPUs, Parallel computing, KMP skip searching algorithm

1. Introduction

Nowadays, the Internet is the largest network that connects people around the world and it has become part of our living. People can share knowledge with each other. Governments, businesses and individuals depend upon it. Criminals can also use the Internet as a channel for committing a cybercrime; hence people on the Internet are in a great risk. Therefore, a powerful network forensic tool is needed for investigating criminal activities on the Internet. The advent of Graphics Processing Units (GPUs) can offer highly parallel computing power with a great performance at a low cost. In this paper, a NVIDIA CUDA was chosen as a main parallel processor in conjunction with a traditional CPU. The KMP skip searching algorithm was also selected as the experimental string searching algorithm. It has high performance capability in short keyword searching. With the high performance algorithm and powerful parallel computing of GPUs, parallel keyword searching on the Internet traffic became possible. This research investigated various parameters of a GPU to obtain the optimal value of each parameter for a network forensic application. The application started with sniffing packets from the network in multiple sessions and dispatched them to the GPU cores along with a specified keyword. Numerous threads in GPU cores would then search incoming traffic for the keyword in parallel. If a session contained the specified keyword, it would be stored in a database for further review.

2. Literature Review

2.1 Graphics Processing Units

GPUs have presently been developed up to thousands of core processors for handling complex graphic computing. However, they can be used for general-purpose application. GPUs are alternative choices of high computing hardware. The CUDA or Compute Unified Device Architecture is the NVIDIA architecture used to manage NVIDIA GPUs. Hongjian Li, Bing Ni, Man-Hon Wong and Kwong-sak Leung [1] implemented the k-difference agrep algorithm in the CUDA system for nucleotide sequence matching and claimed that the CUDA could minimize computing time from seconds to milliseconds. This showed that a GPU could offer highly parallel computing power with higher performance than a CPU. G. Vasiliadis, M. Polychronakis and S. Ioannidis [2] integrated the DFA and GPUs and showed that using of hierarchical memory in modern GPUs could reduce the overhead of data transfer to the GPU. Jiangfeng Peng, Hu Chen, Shaohuai Shi [3] implemented the AC string matching algorithm with a GPU on NVIDIA Tesla C1060 and claimed that a GPU could speedup the AC algorithm 28 times faster than a CPU. Therefore, GPUs are suitable for string searching tasks. Xinyan Zha and Sartaj Sahni [4] implemented the Aho-Corasick string matching using multiple threads on a GPU. This research emphasized that a GPU could speedup 9 times faster than a single-threaded string matching on a CPU. On the other hand, multi-threaded string-matching on the CPU can speedup 3 times faster than a single-threaded one on the CPU. In conclusion, GPUs have advantages in multi-threaded programming over the one on the CPUs.

2.2 Algorithms

The algorithm used in this network forensic system was the KMP skip searching algorithm [5]. The algorithm is a combination between the KMP algorithm and the skip searching algorithm. This algorithm has two steps; preprocessing and searching process. The preprocessing process made the preprocessing table for the searching process. A-Ning Du, Bing-Xing Fang, Xiao-Chun Yun, Ming-Zing HU and Xiu-Rong Zheng [6] claimed that the skip searching algorithm was more efficient in a short searched pattern. The combined KMP skip algorithm would give the better performance. Additionally, Tararat Nowsaeng, Chanokrirdee Sriwichai, Chatchamai Sakolchai and Aunyarat Champreeda [7] suggested that the KMP skip searching algorithm had the best performance in short keyword searching and it was the most suitable for network forensic application. Panwei Cao and Suping Wu [8] implemented a parallel KMP algorithm on MPI's multiprocessor with high performance. This showed that the KMP algorithm could be improved its efficiency by making it as a parallel algorithm.

3. Research Methodology

This paper emphasized on capturing enormous data traffic on network and subsequently employed highly parallel computing power of a GPU for a keyword searching on that traffic. The C++ programming language was used in the CPU programming part and C language was used in the CUDA programming part. These two programming languages have advantages in low-level memory access and execution speed. For the user interface, a web application was developed to support in multiple platforms. Our network forensic application was developed on the Ubuntu Desktop 10.04. Moreover, the g++ compiler was used to compile the CPU program and the nvcc compiler was used to compile the CUDA program.

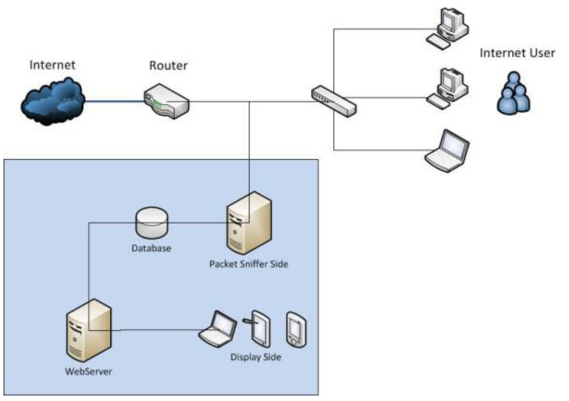


Figure 1. System overview

3.1 System Overview

According to Figure 1, the system began with capturing packets from the network and analyzed TCP packets. After that, the system sent data to be analyzed by a GPU with a user keyword. The GPU would then search for the user keyword in parallel with the KMP skip searching algorithm. If there were a match in a packet, the system would save the entire of session to a database. Reviewing the logs could be done via the web interface.

The system composed of 4 main components: CPU process, CUDA process, Control process and Web application as shown in Figure 2. The web application would generate an xml file contained a user configuration and sent to the control process. After the control process read the xml configuration file, it would start or stop the CPU process and the CUDA process accordingly.

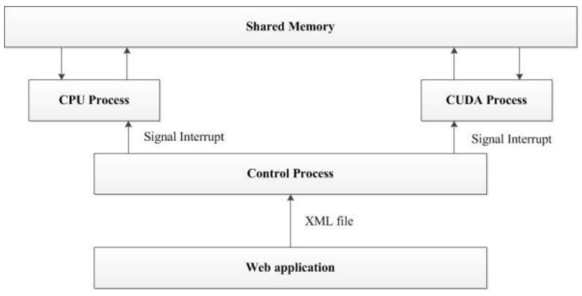


Figure 2. CPU process, CUDA process, control process and web application

3.1.1 The CPU process

The CPU process started with packet sniffing on the network. After a packet was received, the process would create a session according to the packet's port number, the source IP address and the destination IP address. The sequence number of each packet was also used to sort packets in the session. Each new session was created as a linked-list of packets as shown in Figure 3. When the process detected the FIN flag in the TCP segment, it would decompress the session payload compressed with GZIP. At this point, the process would generate a reference number for each session. A reference number was used to associate the session in the CPU process and the same session in the CUDA process. In Figure 4, the reference number and the size of payload were used to identify the session in the CUDA process. The 20 MB of shared memory was also allocated for each session payload.

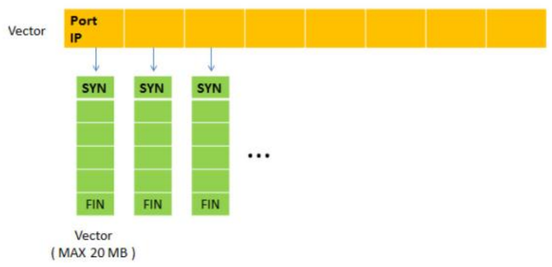


Figure 3. Handle multiple sessions

After the CUDA process had finished its searching tasks, the results would be recorded in the shared memory. If there were a match, the CPU process would save that session to its database.

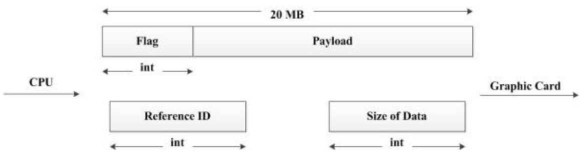


Figure 4. Shared memory implementation

3.1.2 The CUDA process

The CUDA process needed to identify the CUDA card number. Subsequently the CUDA process would create multiple threads for its searching task as in Figure 5. When a session was created in the shared memory, the CUDA process would forward a session to threads in order to search with the KMP skip algorithm.

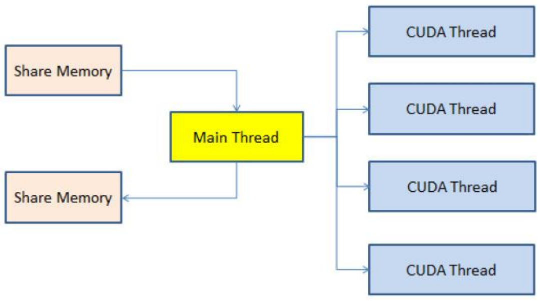


Figure 5. Creating kernel on the graphic card

In the CUDA card, the CUDA thread would fetch a specified keyword and do the preprocessing of KMP skip searching to make a preprocessing table. With the session data and preprocessing tables, the CUDA threads were ready for keyword matching in the computing core. Before a kernel (the KMP skip searching function) was executed, the preprocessing table and the keyword were transferred to the high speed CUDA shared memory. The session data would be divided into small particles as in Figure 6.

Finding the optimal particle size (data per thread) is one of our objectives in this research. After a particle was given to each thread in a CUDA block, the kernel would begin searching for a specified keyword with the KMP skip searching algorithm as in Figure 7. In order to cover the word that might be split to different particles. A particle must contain the overlap data of another particle by a keyword size. After the searching process was finished, each thread would return matched positions of a keyword in the particle and saved them in the shared memory.

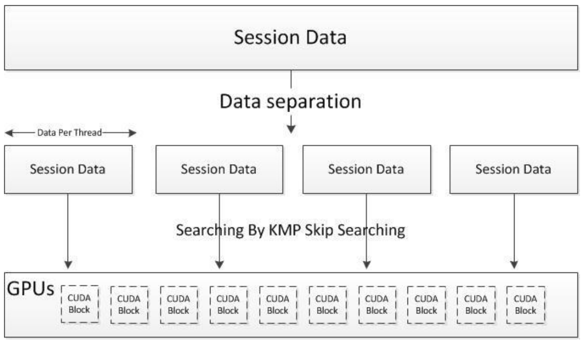


Figure 6. Separation of session data

Subsequently, the CPU process would collect the matched positions from the kernel for each session and recorded them to its database along with the session data.

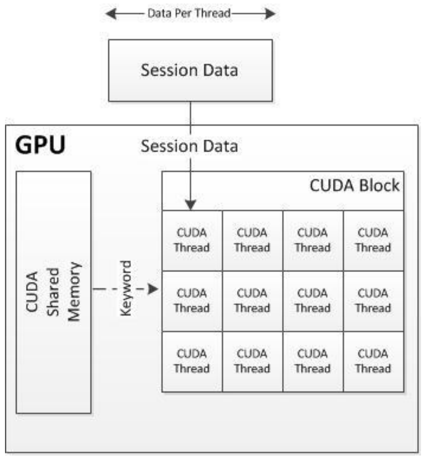


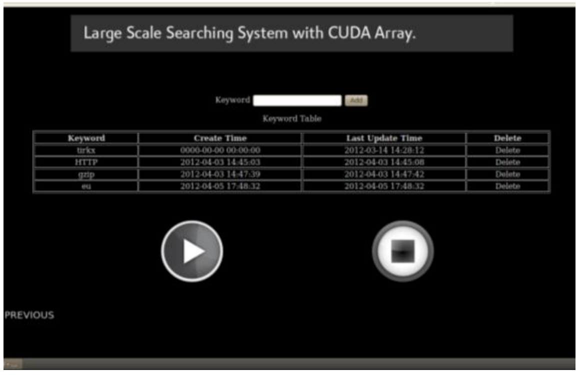
Figure 7. Parallel computing of a GPU

3.1.3 The control process

In order to control the CPU process and the CUDA process, this process would read the XML configuration written by the web application. After that the control process would send a start or stop signal to CPU process and CUDA process.

3.1.4 The web application

The web application was used as an interface to show logs, insert or remove keywords and start or stop network forensic system as in Figure 8. This web application used a database and an XML configuration file as a medium for communicating with the CPU and CUDA processes in our network forensic system.



Large Scale Searching System with CUDA Array.

Keyword [Add]

Keyword Table

Keyword	Create Time	Last Update Time	Delete
tirkx	0000-00-00 00:00:00	2012-03-14 14:28:12	Delete
HTTP	2012-04-03 14:45:03	2012-04-03 14:45:08	Delete
gzip	2012-04-03 14:47:39	2012-04-03 14:47:42	Delete
eu	2012-04-05 17:48:32	2012-04-05 17:48:32	Delete

Figure 8. A control page of the web application

4. Experiment and Results

4.1 Experiment

This research examined two important parameters: the size of block and the data per thread. These two parameters affected the efficiency of the searching process. The size of block is the number of threads in a CUDA blocks. The data per thread is the size of partial session data for each thread. A file with a size of 200 MB was used in our experiment. The size of block was varied from 4 threads per block to 512 threads per block for each specified data size per thread. The data per thread was also varied from 100 wchars (4 bytes) to 10000 wchars for each specified block size. Our system used double NVIDA 450 GT cards and the Intel i3 with 8 GB of memory. The network bandwidth in the experiment was 100 Mbps.

4.2 Results

4.2.1 Size of block

According to our experiment, the 64 threads per block had the lowest average search time of 36.769 ms as shown in Figure 9. This indicated that the 64 threads per block were the most suitable for our network forensic system.

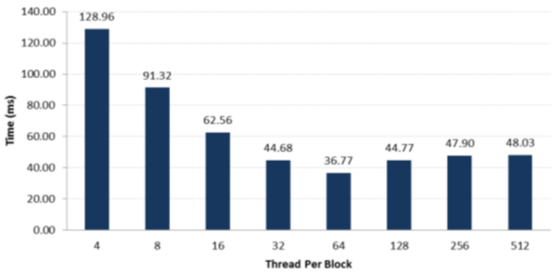


Figure 9. Experimental results of average search time for each thread per block

4.2.2 Data per thread

From the experiment, the results of the optimal data per thread for each specified block size can be shown in Table 1.

Table 1. Experimental results of the optimal data per thread for each block size

Size of Block	Average Search Time (ms)	Optimal Data per thread (wchar)
4	83.95	6500
8	54.05	7800
16	38.79	7800
32	33.58	1300
64	35.79	1300
128	42.97	300
256	46.92	300
512	46.77	300

The experimental results showed that the data size of 1300 wchars gave the lowest average search time of 33.58 ms and 35.79 ms for the block size of 32 and 64 threads accordingly as shown in Table 1 and Figure 10. This showed that 1300 wchars per thread is the optimal data size for our searching task.

To conclude our experiment, we compared the search time of the single-threaded CPU searching task with our highly parallel multi-threaded GPU searching task. The results showed that our CUDA system can speed up about 28 times.

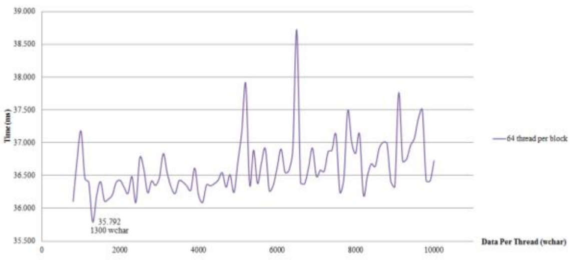


Figure 10. Experimental results of average search time of various block size

5. Conclusion

We have developed the network forensic system as a powerful tool for supporting an investigation of criminal activities on a network. This system can analyze data traffic on a network by searching for malicious keywords. The GPUs have been used to speed up the searching process. We also investigated the optimal value for CUDA parameters.

According to our experiment, the optimal value for the block size on our CUDA system was 64 threads per block and the optimal value for the data size per thread is 1300 wchars.

When the optimal value was applied for each CUDA parameters, our network forensic system could speed up about 28 times faster than the system with the tradition CPU.

6. References

[1] Hongjian Li, Bing Ni, Man-Hon Wong and Kwong-Sak Leung "A Fast CUDA Implementation of Agrep Algorithm for Approximate Nucleotide Sequence Matching", IEEE 9th Symposium on Application Specific Processors (SASP), 2011, pp.74-77

[2] G. Vasiliadis, M. Polychronakis, S. Ioaanidis "Parallelization and Characterization of Pattern Matching using GPUs" IEEE, 2011, pp.216-255

[3] Jiangfeng Peng, Hu Chen, Shaohuai Shi "The GPU-based string matching system in adavanced AC algorithm" IEEE CIT 2010

[4] Xinyan Zha and Sartaj Sahni "Multipattern String Matching On A GPU" IEEE 2011

[5] Christian C. Thierry L., and Joseph D.P. "KMP Skip Search Algorithm; very fast sring matching algorithm for small alphabets and long patterns", Lecture Notes in Comoputer Science, Vol. 1448, 1998, pp.55-64

[6] A-Ning Du, Bing-Xing Fang, Xiao-Chun Yun, Ming-Zing HU and Xiu-Rong Zheng "Comparison of string matching algorithms: an aid to information content security", Proceedings of the Second International Conference on Machine Learning and Cybernetics, Xi'an, 2-5 November 2003, pp.2996-3001

[7] Tararat Nowsaeng, Chanokrirdee Sriwichai, Chatchamai Sakolchai and Aunyarat Champreeda, "Parallel String Searching for Network Traffic Filtering". Computer Department, Faculty of Engineering, Mahidol University 2010

[8] Panwei Cao, Suping Wu "Parallel Research on KMP Algorithm" IEEE 2011, pp. 4252-4255