行政院國家科學委員會專題研究計畫 成果報告

以多核心圖形處理器為基礎之並行化正規表示樣式比對演算法與架構設計(I)
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 98-2221-E-003-016-
執 行 期 間 ：98 年 08 月 01 日至 99 年 07 月 31 日
執 行 單 位 ：國立臺灣師範大學工業科技教育學系（所）

計 畫 主 持 人：林政宏

計畫參與人員：碩士班研究生-兼任助理人員：林佩蓉
碩士班研究生-兼任助理人員：郭榮杰
大專生-兼任助理人員：黃俊程
大專生-兼任助理人員：蔡睿烝
大專生-兼任助理人員：吳鎬宇
大專生-兼任助理人員：洪嘉彣
大專生-兼任助理人員：張耀穗
大專生-兼任助理人員：粘懿瑩
大專生-兼任助理人員：林慧雯

報 告 附 件 ：出席國際會議研究心得報告及發表論文

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 99 年 08 月 18 日

行政院國家科學委員會補助專題研究計畫成果報告

以多核心圖形處理器為基礎之並行化正規表示樣式比對演算法與架構設計(I)

計畫類別：■個別型計畫　□整合型計畫
計畫編號：NSC 98-2221-E-003-016-
執行期間：98 年 8 月 1 日至 99 年 7 月 31 日

執行機構及系所：國立台灣師範大學科技應用與人力資源發展學系

計畫主持人：林政宏
共同主持人：
計畫參與人員：

成果報告類型(依經費核定清單規定繳交)：■精簡報告　□完整報告

本計畫除繳交成果報告外，另須繳交以下出國心得報告：
□赴國外出差或研習心得報告
□赴大陸地區出差或研習心得報告
■出席國際學術會議心得報告
□國際合作研究計畫國外研究報告

處理方式：除列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

中 華 民 國 九十九 年 七 月 卅一 日

一、 中文摘要

網絡入侵偵測系統已被廣泛用於保護計算機系統免受網路攻擊。由於越來越多的攻擊和網路的複雜性，傳統的軟體方法對單一核心處理器已經不適用於當前的高速網路。本計畫的主要目的提出以多核心圖形處理器(graphics processing unit , GPU)加速正規表示法比對之演算法與架構設計，本計畫所提出之平行演算法利用GPU多核心架構有效加速字串比對，相較實現Aho-Corasick演算法於CPU的方法，加速約4000倍，相較其他硬體架構，亦有2-3倍的效能改善。此外，我們所提出的演算法亦可降低傳統Aho-Corasick演算法的記憶體需求。

研究成果將發表於2010 IEEE Globecom國際會議和第21超大型積體電路設計暨計算機輔助設計研討會。

關鍵字：字串比對、多核心圖形處理單元

二、 英文摘要

Network Intrusion Detection System has been widely used to protect computer systems from network attacks. Due to the ever-increasing number of attacks and network complexity, traditional software approaches on uni-processors have become inadequate for the current high-speed network. In this project, we propose a novel parallel algorithm to speedup string matching performed on GPUs. We also innovate new state machine for string matching, the state machine of which is more suitable to be performed on GPU. We have also described several speedup techniques considering special architecture properties of GPU. The experimental results demonstrate the new algorithm on GPUs achieves up to 4,000 times speedup compared to the AC algorithm on CPU. Compared to other GPU approaches, the new algorithm achieves 3 times faster with significant improvement on memory efficiency. Furthermore, because the new Algorithm reduces the complexity of the Aho-Corasick algorithm, the new algorithm also improves on memory requirements.

The research results have been accepted by IEEE GLOBAL COMMUNICATIONS CONFERENCE (IEEE GLOBECOM 2010) and the 21st VLSI Design/CAD Symposium.

Keywords: ***string matching, graphics processing unit***

三、 計畫緣由與目的

Network Intrusion Detection Systems (NIDS) have been widely used to protect computer systems from network attacks such as denial of service attacks, port scans, or malware. The string matching engine used to identify network attacks by inspecting packet content against thousands of predefined patterns dominates the performance of an NIDS. Due to the ever-increasing number of attacks and network complexity, traditional string matching approaches on uni-processors have become inadequate for the high-speed network.

To accelerate string matching, many hardware approaches are being proposed that can be classified into logic-based [1][2][3][4] and memory-based approaches [5][6][7][8][9]. Recently, Graphic Processor Unit (GPU) has attracted a lot of attention due to their cost-effective parallel computing power. A modified Wu-Manber algorithm [10] and a modified suffix tree algorithm [11] are implemented on GPU to accelerate exact string matching while a traditional DFA approach [12] and a new state machine XFA [13] are proposed to accelerate regular expression matching on GPU.

In this project, we study the use of parallel computation on GPUs for accelerating string matching. A direct implementation of parallel computation on GPUs is to divide an input stream into multiple segments, each of which is processed by a parallel thread for string matching. For example in Fig. 1(a), using a single thread to find the pattern “AB” takes 24 cycles. If we divide an input stream into four segments and allocate each segment a thread to find the pattern “AB” simultaneously, the fourth thread only takes six cycles to find the same pattern as shown in Fig. 1(b).

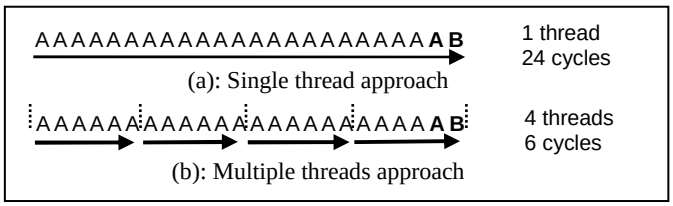

Figure 1. Single vs. multiple thread approach

However, the direct implementation of dividing an input stream on GPUs cannot detect a pattern occurring in the boundary of adjacent segments. We call the new problem as the “boundary detection” problem. For example, in Fig. 2, the pattern “AB” occurs in the boundary of segments 3 and 4 and cannot be identified by threads 3 and 4. Despite the fact that boundary detection problems can be resolved by having threads to process overlapped computation on the boundaries (as shown in Fig. 3), the overhead of overlapped computation seriously degrades performance.

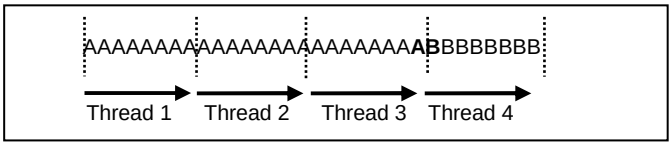

Figure 2. Boundary detection problem that the pattern “AB” cannot be identified by Thread 3 and 4.

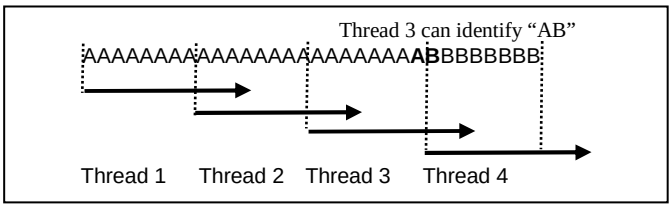

Figure 3. Every thread scans across the boundary to resolve the boundary detection problem.

四、 研究方法

This project addresses the algorithm and architecture design of string matching on GPUs. In this section, we first describe the problems of direct implementation of AC Algorithm on GPU and then propose our parallel algorithm and experimental results.

4.1 Problems of Direct Implementation of AC Algorithm on GPU

Among string matching algorithms, the AC algorithm [5][8][9][14][16][17] has been widely used for string matching due to its advantage of matching multiple patterns in a single pass. Using the AC Algorithm for string matching consists of two steps. The first step is to compile multiple patterns into a composite state machine. The second step is to use a single thread to recognize attack patterns by traversing the state machine. For example, Fig. 4 shows the state machine of the four patterns, “AB”, “ABG”, “BEDE”, and “EF”. In Fig. 4, the solid lines represent the valid transitions whereas the dotted lines represent the failure transitions.

The failure transitions are used to back-track the state machine to recognize patterns in different locations. Given a current state and an input character, the AC machine first checks whether there is a valid transition for the input character; otherwise, the machine jumps to the next state where the failure transition points. Then, the machine regards the same input character until the character causes a valid transition. For example, consider an input stream which contains a substring “ABEDE”. The AC state machine first traverses from state 0, state 1, to state 2 which is the final state of pattern “AB”. Because state 2 has no valid transition for the input “E,” the AC state machine first takes a failure transition to state 4 and then regards the same input “E” leading to state 5. Finally, the AC state machine reaches state 7 which is the final state of pattern “BEDE”.

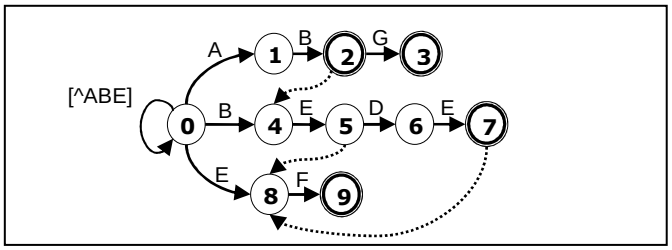

Figure 4. AC state machine of the patterns “AB”, “ABG”, “BEDE”, and “EF”

One approach to increase the throughput of string matching is to increase the parallelism of the AC algorithm. A direct implementation to increase the parallelism is to divide an input stream into multiple segments and to allocate each segment a thread to process string matching. As Fig. 5 shows, all threads

process string matching on their own segments by traversing the same AC state machine simultaneously. As discussed in introduction, the direct implementation incurs the boundary detection problem. To resolve the boundary detection problem, each thread must scan across the boundary to recognize the pattern that occurs in the boundary. In other words, in order to resolve the boundary detection problem and identify all possible patterns, each thread must scan for a minimum length which is almost equal to the segment length plus the longest pattern length of an AC state machine. For example in Fig. 5, supposing each segment has eight characters and the longest pattern of the AC state machine has four characters, each thread must scan a minimum length of eleven (8+4-1) characters to identify all possible patterns. The minimum length is calculated by adding the segment length and the length of the longest pattern, and then subtracting one character. The overhead caused by scanning the additional length across the boundary is so-called overlapped computation.

On the other hand, the throughput of string matching on GPU can be improved by deeply partitioning an input stream and increasing threads. However, deeply partitioning will cause the probability of the boundary detection problem to increase. To resolve the boundary detection problem, the overlapped computation increases tremendously and leads to throughput bottleneck.

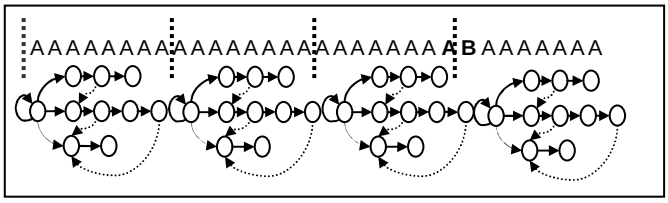

Figure 5. Direct implementation which divides an input stream into multiple segments and allocates each segment a thread to traverse the AC state machine.

4.2 Parallel Failureless-AC Algorithm

In order to increase the throughput of string matching on GPU and resolve the throughput bottleneck caused by the overlapped computation, we propose a new algorithm, called Parallel Failureless-AC Algorithm (PFAC). In PFAC, we allocate each byte of an input stream a GPU thread to identify any virus pattern starting at the thread starting location.

The idea of allocating each byte of an input stream a thread to identify any virus pattern starting at the thread starting location has an important implication on the efficiency. First, in the conventional AC state machine, the failure transitions are used to back-track the state machine to identify the virus patterns starting at any location of an input stream. Since in the PFAC algorithm, a GPU thread only concerns the virus pattern starting at a particular location, the GPU threads of PFAC need not back-track the state machine. Therefore, the failure transitions of the AC state machine can all be removed. An AC state machine with all its failure transitions removed is called Failureless-AC state machine. Fig. 6 shows the diagram of the PFAC which allocates each byte of an input stream a thread to traverse the new Failureless-AC state machine.

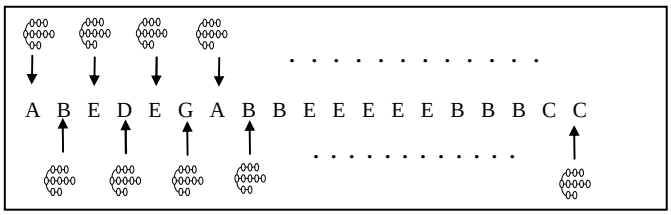

Figure 6. Parallel Failureless-AC algorithm which allocates each byte of an

input stream a thread to traverse the Failureless-AC state machine.

We now use an example to illustrate the PFAC algorithm. Fig. 7 shows the Failureless-AC state machine to identify the patterns “AB”, “ABG”, “BEDE”, and “EF” where all failure transitions are removed. Consider an input stream which contains a substring “ABEDE”. As shown in Fig. 8, the thread tn is allocated to input “A” to traverse the Failureless-AC state machine. After taking the input “AB”, thread tn reaches state 2, which indicates pattern “AB” is matched. Because there is no valid transition for “E” in state 2, thread tn terminates at state 2. Similarly, thread tn+1 is allocated to input “B”. After taking input “BEDE”, thread tn+1 reaches state 7 which indicates pattern “BEDE” is matched.

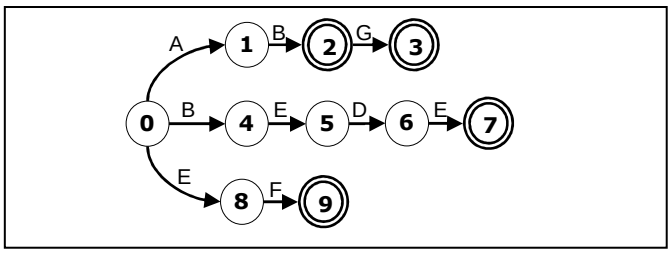

Figure 7. Failureless-AC state machine of the patterns “AB”, “ABG”, “BEDE”, and “EF”.

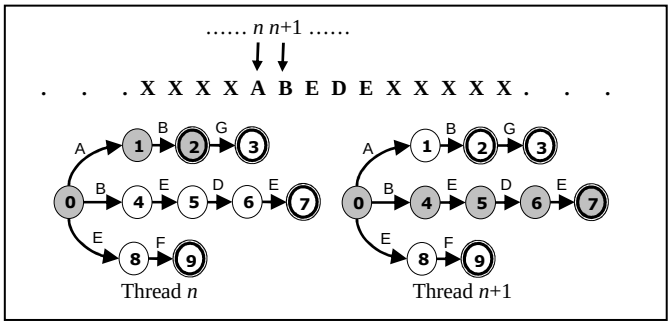

Figure 8: Example of PFAC

There are three reasons that the PFAC algorithm is superior to the straightforward implementation in Section II. They are described as follows. First, there is no boundary detection problem, as with the straightforward implementation. Second, both the worst-case and average life times of threads in the PFAC algorithm are much shorter than the time needed for the straightforward implementation. As shown in Fig. 9, threads tn to tn+3 terminate early at state 0 because there are no valid transitions for “X” in state 0. The threads tn+6 and tn+8 terminate early at state 8 because there are no valid transitions for “D” and “X” in state 8. Although the PFAC algorithm allocates a large number of threads, most threads have a high probability of terminating early, and both the worst-case and average life-time of threads in the PFAC algorithm are much shorter than the direct implementation. Third, the memory usage of the PFAC algorithm is smaller, due to the removal of failure transitions.

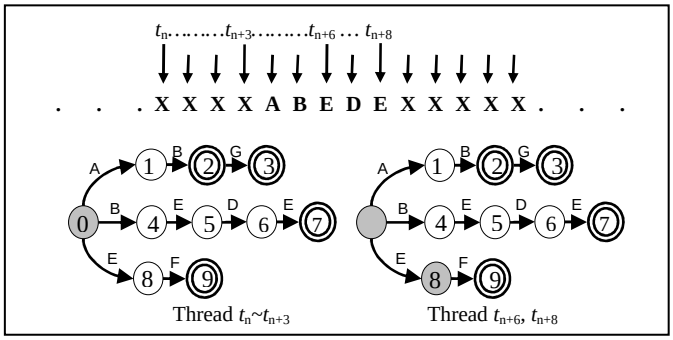

Figure 9. Most threads terminate early in PFAC

4.3 Experimental Results

We have implemented the proposed algorithm on a commodity GPU card and compare with the recent published GPU approaches. The experimental configurations are as follows:

- CPU: Intel® Core™2 Duo CPU E7300 2.66GHz
  - System main memory: 4,096 DDR2 memory
- GPU card: NVIDIA GeForce GTX 295 576MHz
  - 480 cores with 1,792 MB GDDR3 memory
- Patterns: string patterns of Snort V2.4

In order to evaluate the performance of our algorithm, we implement three approaches described in this project for comparisons. As shown in Table 1, the CPU_AC denotes the method of implementing the AC algorithm on CPU, which is the most popular approach adopted by NIDS systems, such as Snort. The Direct_AC approach denotes the direct implementation of the AC algorithm on GPU. The PFAC denotes the Parallel Failureless-AC approach on GPU. Table 1 shows the results of these three approaches for processing two different input streams. Column one lists two different input streams, the normal case denotes a randomly generated sequence of 219Kbytes comprising 19,103 virus patterns, whereas the virus case denotes a sequence of 219 Kbytes comprising 61,414 virus patterns.

Column 2, 3, 4, and 5 list the throughput of the three approaches, CPU_AC, Direct_AC, and PFAC, respectively. For processing the normal case of input streams, the throughput of CPU_AC, Direct_AC, and PFAC are 997, 6,428, and 3,963,966 KBps (Kilo Bytes per second), respectively. The experimental results show that the PFAC performs up to 4,000 times faster than the CPU_AC approach while the Direct_AC can only perform 6.4 times as fast. In other words, the PFAC also achieves up to 600 times faster than Direct_AC approach on GPU.

Furthermore, because the new algorithm removes the failure transitions of the AC state machine, the memory requirement can also be reduced. Table 2 shows that the new algorithm can reduce the number of transitions by 50%, and therefore achieve a memory reduction of 21% for Snort patterns. Table 3 compares with several recent published GPU approaches [10][11][12][13]. In Table 3, columns 2, 3, 4, and 5 shows the character number, memory size, the throughput, and the memory efficiency which is defined as the following equation.

$$\textit{Memory efficiency} = \textit{Throughput} / \textit{Memory} \qquad (1)$$

As shown in Table 3, our results are faster than all [10][11][12][13] with efficient memory usage. We

would like to mention that the memory-based approach requires the design and fabrication of a dedicated hardware whereas the GPU approach is more general in the sense that both software and the virus patterns can be easily updated.

TABLE 1: THROUGHPUT COMPARISON OF THREE APPROACHES

Input streams	CPU_AC	Direct_AC	PFAC
	Throughput (KBps)	***Throughput (KBps)***	***Throughput (KBps)***
Normal Case	997	6,428	3,963,966
Virus Case	657	4,691	3,656,217
Ratio	1	~6.4	**~4000**

TABLE 2: MEMORY COMPARISON

	Conventional AC			PFAC			
	states	***transitions***	***memory (KB)***	***states***	***transitions***	***memory (KB)***	***Reduction***
Snort rule*	8,285	16,568	143	8,285	8,284	114	21%
Ratio	1	1	1	1	0.5	0.79	

* The Snort rules contain 994 patterns and total 22,776 characters.

TABLE 3. COMPARISONS WITH PREVIOUS GPU APPROACHES

Approaches	Character number of rule set	Memory (KB)	Throughput (GBps)	Memory Efficiency (Throughput/memory)	Notes
PFAC	22,776	114	3.9	34,210	NVIDIA GeForce GTX 295
Huang *et al.* [10] Modified WM	1,565	230	0.3	1,304	NVIDIA GeForce 7600 GT
Schatz *et al.* [11] Suffix Tree	200,000	14,125	~0.25	17.7	NVIDIA GTX 8800
Vasiliadis *et al.* [12] DFA	N.A.	200,000	0.8	4	NVIDIA GeForce 9800 GX2
Smith et al. [13] XFA	N.A.	3,000	1.3	433	NVIDIA GeForce 8800 GTX

五、 重要執行成果和價值

In this project, we attempt to speedup string matching using GPU. Our major contributions are summarized as follows.

1. We propose a novel parallel algorithm to speedup string matching performed on GPUs. The new parallel algorithm is free from the boundary problem.
2. We also innovate new state machine for string matching, the state machine of which is more suitable to be performed in the parallel algorithm.
3. Considering special architecture properties of GPU, we have also described several speedup techniques.
4. Finally, we perform experiments on the Snort rules. The experimental results show that the new algorithm on GPU achieves up to 4,000 times speedup compared to the AC algorithm on CPU. Compared to other GPU [10][11][12][13] approaches, the new algorithm achieves 3 times faster with significant improvement on memory efficiency. In addition, because the new Algorithm reduces the complexity of the Aho-Corasick (AC) [14] algorithm, we achieve an average of 21% memory reduction for all string patterns of Snort V2.4 [15].

All techniques have been accepted and published in international conferences and journals. In the following, we list the related papers of above techniques.

[1] ***Cheng-Hung Lin****, Sheng-Yu Tsai, Chen-Hsiung Liu, Shih-Chieh Chang, and Jyuo-Min Shyu, "Optimization of String Matching Algorithm on GPU," in Proc. of 21st VLSI Design/CAD*

Symposium, Kaohsiung, Taiwan, August 3-6, pp.163-166, 2010. (Best Paper Nominee)

[2] ***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," to appear in IEEE GLOBAL COMMUNICATIONS CONFERENCE (IEEE GLOBECOM 2010), Miami, Florida, USA, December 6-10, 2010.*

六、 結論

Graphics Processor Units (GPUs) have attracted a lot of attention due to their cost-effective and dramatic power of massive data parallel computing. In this project, we have proposed a novel parallel algorithm to accelerate string matching by GPU. The experimental results show that the new algorithm on GPU can achieve a significant speedup compared to the AC algorithm on CPU. Compared to other GPU approaches, the new algorithm achieves 3 times faster with significant improvement on memory efficiency. In addition, because the new algorithm reduces the complexity of the AC algorithm, the new algorithm also improves on memory requirements.

七、 參考文獻

[1] R. Sidhu and V. K. Prasanna, “Fast regular expression matching using FPGAs,” in *Proc. 9th Ann. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2001, pp. 227-238.

[2] B.L. Hutchings, R. Franklin, and D. Carver, “Assisting Network Intrusion Detection with Reconfigurable Hardware,” in *Proc.10th Ann. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2002, pp. 111-120.

[3] C. R. Clark and D. E. Schimmel, “Scalable Pattern Matching for High Speed Networks,” in *Proc. 12th Ann. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2004, pp. 249-257

[4] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, “Implementation of a Content-Scanning Module for an Internet Firewall,” in *Proc. 11th Ann. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2003, pp. 31–38.

[5] M. Aldwairi*, T. Conte, and P. Franzon, “Configurable String Matching Hardware for Speeding up Intrusion Detection,” in *ACM SIGARCH Computer Architecture News*, 2005, pp. 99–107

[6] S. Dharmapurikar and J. Lockwood, “Fast and Scalable Pattern Matching for Content Filtering,” *in Proc. of Symp. Architectures Netw. Commun. Syst. (ANCS)*, 2005, pp. 183-192

[7] Y. H. Cho and W. H. Mangione-Smith, “A Pattern Matching Co-processor for Network Security,” in *Proc. 42nd Des. Autom. Conf. (DAC)*, 2005, pp. 234-239

[8] L. Tan and T. Sherwood, “A high throughput string matching architecture for intrusion detection and prevention,” in *proc. 32nd Ann. Int. Symp. on Comp. Architecture, (ISCA)*, 2005, pp. 112-122

[9] H. J. Jung, Z. K. Baker, and V. K. Prasanna, “Performance of FPGA Implementation of Bit-split Architecture for Intrusion Detection Systems,” in *20th Int. Parallel and Distributed Processing Symp. (IPDPS)*, 2006.

[10] N. F. Huang, H. W. Hung, S. H. Lai, Y. M. Chu, and W. Y. Tsai, “A gpu-based multiple-pattern matching algorithm for network intrusion detection systems,” in *Proc. 22nd International Conference on Advanced Information Networking and Applications (AINA)*, 2008, pp. 62–67.

[11] M. C. Schatz and C. Trapnell, “Fast Exact String Matching on the GPU,” Technical report.

[12] G. Vasiliadis , M. Polychronakis, S. Antonatos , E. P. Markatos and S. Ioannidis, “Regular Expression Matching on Graphics Hardware for Intrusion Detection,” In *Proc. 12th International Symposium on Recent Advances in Intrusion Detection*, 2009.

[13] R. Smith, N. Goyal, J. Ormont, K. Sankaralingam, C. Estan, “Evaluating GPUs for network packet signature matching,” in *Proc. of the International Symposium on Performance Analysis of Systems and Software*, ISPASS (2009).

[14] A. V. Aho and M. J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. In *Communications of the ACM*, 18(6):333–340, 1975.

[15] M. Roesch. Snort- lightweight Intrusion Detection for networks. In Proceedings of LISA99, the *15th Systems Administration Conference*, 1999.

[16] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. “Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection,” in *Proc. 23nd Conference of IEEE Communication Society (INFOCOMM)*, Mar, 2004.

[17] S. Kumar, S.Dharmapurikar, F.Yu, P. Crowley, and J. Turner, “Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection,” in *ACM SIGCOMM Computer Communication Review*, ACM Press, vol.36, Issue. 4, Oct. 2006, pp. 339-350.

[18] F. Yu, R. H. Katz, and T. V. Lakshman, “Gigabit Rate Packet Pattern-Matching Using TCAM,” in *Proc. the 12th IEEE International Conference on Network Protocols (ICNP’04)*, 2004.

[19] http://www.nvidia.com.tw/object/cuda_home_tw.html

[20] http://www.nvidia.com.tw/page/home.html

出席國際學術會議心得報告

計畫名稱：以多核心圖形處理器為基礎之並行化正規表示樣式比對演算法與架構設計(I)

計畫編號：NSC 98-2221-E-003-016-

報告人：林政宏副教授

會議名稱：International SOC Design Conference (ISOCC)

會議地點：Busan, Korea

會議時間：November 22-24

出國目的：發表論文

論文名稱：Design and Verification for Dual Issue Digital Signal Processor

與會心得：

It's a really good experience to attend the International SOC Design Conference (ISOCC) held on November 22-24, 2009 in Busan, Korea. The International SoC Design Conference (ISOCC) aims at providing the world's premier SoC design forum for leading researchers from academia and industries. My paper titled "Design and Verification for Dual Issue Digital Signal Processor" was accepted as poster presentation. Because many attendances work on relative researches, I obtained a lot of positive feedback. During my presentation, many professors and students discussed with me about my researches. In addition, I also listened several interesting research topics including network intrusion detection, multicore design and verification, and so on.

Finally, I am very thankful for NSC supporting me to attend this conference.

無研發成果推廣資料

98 年度專題研究計畫研究成果彙整表

計畫主持人：林政宏	計畫編號：98-2221-E-003-016-
計畫名稱：以多核心圖形處理器為基礎之並行化正規表示樣式比對演算法與架構設計(I)	

成果項目			量化			單位	備註（質化說明：如數個計畫共同成果、成果列為該期刊之封面故事...等）
			實際已達成數（被接受或已發表）	預期總達成數(含實際已達成數)	本計畫實際貢獻百分比		
國內	論文著作	期刊論文	0	0	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	1	1	100%		
		專書	0	0	100%		
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（本國籍）	碩士生	2	2	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		
國外	論文著作	期刊論文	1	1	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	1	1	100%		
		專書	0	0	100%	章/本	
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（外國籍）	碩士生	0	0	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		

其他成果 (無法以量化表達之成果如辦理學術活動、獲得獎項、重要國際合作、研究成果國際影響力及其他協助產業技術發展之具體效益事項等，請以文字敘述填列。)	本計畫的成果分別發表於美國 IEEE GLOBECOM 2010 與國內第廿一屆 VLSI Design/CAD symposium，並獲得最佳論文候選。

	成果項目	量化	名稱或內容性質簡述
科教處計畫加填項目	測驗工具(含質性與量性)	0	
	課程/模組	0	
	電腦及網路系統或工具	0	
	教材	0	
	舉辦之活動/競賽	0	
	研討會/工作坊	0	
	電子報、網站	0	
	計畫成果推廣之參與（閱聽）人數	0	

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

1. 請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

■達成目標

□未達成目標（請說明，以 100 字為限）

□實驗失敗

□因故實驗中斷

□其他原因

說明：

2. 研究成果在學術期刊發表或申請專利等情形：

論文：■已發表 □未發表之文稿 □撰寫中 □無

專利：□已獲得 □申請中 ■無

技轉：□已技轉 □洽談中 ■無

其他：(以 100 字為限)

論文分別發表於 IEEE GLOBAL COMMUNICATIONS CONFERENCE (IEEE GLOBECOM 2010) 與 21st VLSI Design/CAD Symposium

3. 請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）(以 500 字為限)

本計畫的主要目的提出以多核心圖形處理器(GPU)加速正規表示法比對之演算法與架構設計。本計畫所提出之新穎的平行演算法有效利用 GPU 多核心架構有效加速字串比對，相較實現 Aho-Corasick 演算法於 CPU 的方法，加速約 4000 倍，相較其他硬體架構，亦有 2-3 倍的效能改善。此外，我們所提出的演算法亦可降低傳統 Aho-Corasick 演算法的記憶體需求。研究成果可運用於網路入侵偵測系統或防毒軟體。