The Implementation of a Network Traffic Detection Model Based on GPU and CPU Heterogeneous Platform

Peng Xie[1], Yu Wu[1] & Kefei Cheng[1]

[1]College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, 400065, China

Keywords: AC algorithm, Parallel Computing, Video Memory Optimization, Network Traffic Detection.

Abstract. The technology of network traffic detection is very important in the area of network managing, network monitoring and network security. The network traffics are sharply increasing along with the growing of the Internet. Because of the hardware limitation, the traditional traffic detection method can not meet the requirement of detecting gigabit high-speed network in real-time. A network traffic detection model based on GPU and CPU heterogeneous platform is proposed. This detection model makes some simplification and changes of the AC algorithm. It can take the advantage of the excellent parallel computing capability of GPU to speed up the string matching module which costs the most of the time in the detection. Experiments show that the detecting speed of GPU and CPU heterogeneous model is much faster than that of the traditional model.

Introduction

In the area of network managing, network monitoring and network security, it is very important to detect the specific network traffic we are interested in. In the network traffic detection system, timeliness, accuracy and the size of the amount of the information handled are the most important indicator to judge the system's performance[1].

With the Internet growing explosively and the popularity of the mobile terminals, the network traffic is increasing rapidly. The backbone network traffic has already reach 10G/s level. Facing such huge size of information, the traditional traffic detection system cat not guarantee the detecting accuracy while meeting the requirement of real-time detection.

In the network traffic detection, the string matching module costs the most computing resources. For example, it can cost more than 50% computing resources in the Intrusion Detection System[2].

There are lots of classical string matching algorithms: AC algorithm[3], BM algorithm[4], SUNDAY algorithm[5], Wu-Manber algorithm[6] and so on. The AC algorithm discussed in this paper is proposed by Alfred V.Aho and Margaret J.Corasick in 1975. It is a classical multi-string matching algorithm based on a finite state machine. The time AC algorithm costs is only related to the length of the STRING, not to the length and number of the PATTERNs which the AC state machine is built from. Its time complexity is O(n), n means the length of the STRING.

After years of researches and developments, AC algorithm has been improved in many aspects. First, because each state of AC state machine often has few effective next transition state, so lots of the elements of the matrix corresponding to the AC machine are 0. It wastes a lot of memories. To reduce the memory cost as much as possible, Norton[7] proposed a method based on sparse vector compression to compress each row of the AC machine. Considering each column of the machine, Xu Hong proposed a double-compression AC algorithm[8]. Then, Yang Chao[9] proposed bidirectional AC algorithm which scan the STRING from left and right simultaneously. Next, combining the AC algorithm and BM algorithm, Commentz-Walter[10] proposed AC-BM algorithm. What's more, Wan Yin-guo[11] proposed an improved AC-BM algorithm which only use the bad character jumping strategy in order to reduce the time complexity of AC-BM algorithm.

The improvement of AC algorithm above are all based on the CPU architecture. Because the traditional CPU only has few computing cores, although these algorithms have greatly improved the matching speed in the software level, they can not meet the requirement of detecting the 10G/s

network in real-time. Using multi-core computer or special hardware may solve these problems, but it need more investment, more development time and more hardware development experiences[12].

At the same time, GPU is developed rapidly. Compared with traditional CPU, GPU has more computing cores and higher graphic memory bandwidth. Current researches are focus on building CPU and GPU heterogeneous platform to take advantage of GPU to accelerate the computing-intensive tasks[13]. Peng Jiangfeng has achieved GAC algorithm[14] on GPU and got high speedup. Cheng Hung[15] proposed a new state machine which is more fit for GPU's properties. Tran optimized the AC algorithm on GPU and tested it carefully[16]. Xin yin[17] analyse two communication strategies between CPU and GPU. A network traffic detection model based on GPU and CPU heterogeneous platform is proposed in this paper. This detection model achieves the GAC algorithm and makes some simplification and changes. At last, the experimental results show that this heterogeneous model gets better speedup than the traditional model.

Genetic algorithm Traffic Detection Model Based on CPU and GPU Heterogeneous Model

In this heterogeneous model, CPU is responsible for the overall scheduling, building the AC state machine, collecting network traffic, text preprocessing and outputting the matching results. GPU is only responsible for accelerating the string matching process.

In the actual detection period, we continuous collect the network traffic in real-time, analyse the network protocol and extract the application layer payload from the network packet using libpcap libraries.

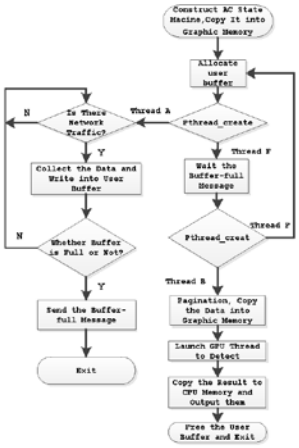

Fig. 1: the Flowchart of Heterogeneous Model

The model are shown as Fig.1. The model use multithread to work cooperatively when processing. At the beginning, thread F uses the PATTERNs to build and optimize the AC state machine. And then, copy it to the GPU. Next, F allocates the user buffer *buf1*. After that, thread F creates thread A to collect the network traffic and F blocks until A sends it a *buffer-full* message.

Thread A collects the network traffic continuously, removes the protocol header from the packet and extracts the application layer payload. Then, thread A filters some useless payload information and puts the strings need to be scan into the user buffer *buf1*.

When the user buffer is full, thread A sends a *buffer-full* message to F and exits. Thread F creates thread B after receiving the *buffer-full* message. And then, F reallocates user buffer *buf2* and recreates thread A to collect network traffic again.

Thread B preprocesses the user buffer in CPU first, and then, it copies the buffer into GPU and begins detecting. At the same time, B is also responsible for freeing the user buffer after detecting. If there are some matching results, B copies them back to CPU and outputs them. At last, B exits.

Implementation and Simplification of AC Algorithm on GPU

Implementation of GAC algorithm. The matching time of AC algorithm is only related to the

length of the STRING. We can take advantage of this property to accelerate the algorithm using GPU. In this paper, GAC algorithm divides the STRING into lots of pages. Therefore, the length of STRING is reduced effectively on the logical level.

GPU has powerful concurrent ability and high memory bandwidth. What's more, programming on GPU becomes easier since the CUDA programming model is proposed[18]. But there are still some limitations. On one hand, we can't use too complicated data structure. On the other hand, GPU are suit for computing-intensive tasks. If the tasks has too complex logicality, the GPU's parallel capability will be reduced sharply[13]. So, when transplanting the AC algorithm to GPU, we need to modify it to make it as simple as possible.

In this paper, the data structure of AC state machine on GPU is as follow. First, the base data structure of the AC state machine is allocated as static continuous integer array *tranList*. During the building process of the state machine, the number of actual state *actualStateNum* can be calculated. Then, a continuous space whose size is *(actualStateNum+1)*256*sizeof(int)* is allocated. Each state use *256*sizeof(int) Bytes* continuous space, which is respectively corresponding to the 256 ASCII code. So, the value stored in *tranList(C*256+ch)* is the return value of *Goto(C, ch)* function. Then, we put the indicator of the next jump state into the corresponding position of *tranList* to eliminate the failure function. After that, the total matching process is just a jumping process from one state to another in *tranList*.

Second, during the jumping process, when reaching the next state, the state machine needs to check whether there are matches. An integer entry index which can identify different pattern strings is added to the pattern list. the *matchList* stores the index value of the output of each state other than the pattern pointer. Using this method, the copying of pattern list can be avoided and we will no longer care about the complicated pointer on GPU.

Now, the AC algorithm can be transplanted from CPU to GPU. But the experiments show that it takes the most of time when each GPU thread reads STRING from the global memory. This is because the program don't meet the coalesce accessing requirement. While lots of threads are running at the same memory access instructions simultaneously, the accessed space are not continuous or aligned. Therefore, only one or few threads can access the memory. In the worst situation, the whole threads must access the memory in order. The parallel efficiency of GPU is seriously reduced.

Coalesce Accessing. For devices of compute capability 2.x, if each thread access no more than 4 Bytes space, the accesses to the continuous memory space of the threads of a warp will coalesce into only one memory access. The memory access performance will be improved efficiently through this method[19].

On the other hand, since the on chip shared memory of GPU has wide bandwidth and less delay. The data can be copied from global memory to shared memory through coalesce accessing first, and then, data in shared memory are actually used during the matching process. The parallel efficiency of GPU can be significantly increased in this way.

What's more, under specific conditions, multiple threads are allowed to access the shared memory simultaneously. The shared memory is composed by bank which is smaller storage unit. The conflicts will occur while multiple threads access the same bank at the same time. To reduce the conflicts of shared memory, Peng Jiangfeng and Tran design a very complicated strategy, which put the data belong to the same page into one bank. As a result, during the matching time, different thread accesses different bank automatically. In one block, the coalesce accessing are shown as Fig.2 .

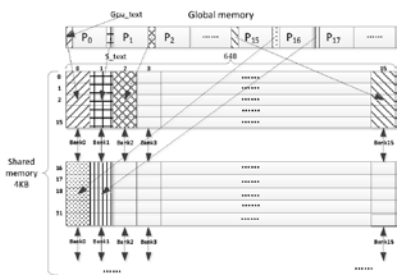

Fig. 2: Complicated Coalesce

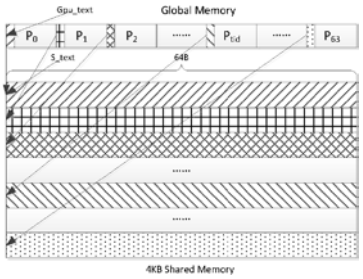

Fig. 3: Simple Coalesce

In this strategy, 64 GPU thread are launched, and 64 pages are processed while 64 Bytes data of a page are detected each time. 4KB shared memory are allocated, each thread uses 64 Bytes. The 64 threads are divided into 4 groups, 16 threads of one group are responsible for copying the 64B data of one page into the shared memory. 64 thread can deal with 4 pages each time. After 16 loops, the corresponding 64 Bytes data of 64 pages will be copied into the 4KB shared memory allocated previously. And then, this 4KB data are really used. Then, the 64 threads copy the next 64 Bytes data and do the matching process until all the data of every page has been scanned.

Experiments show that in this strategy, the shared memory confliction can be reduced during the matching period. But, when copying the data from global memory to shared memory, multi-thread copying the data of the same page must access the same page in shared memory, as a result, they must access the same bank. So, actually, the conflicts can not be avoided.

In this paper, to make some simplification, the shared memory is used as a two dimensional array. Each thread uses the continuous 64 Bytes space which hold 16 banks other than one bank. In this way, although still occurring during the matching period, the conflicts can be reduced effectively during the data copying period. It is the most important that this design makes the index simple to understand and implement. This simple coalesce accessing is shown as Fig. 3.

Table 1 and Table 2 show that using this simple coalesce accessing can really reduce the confliction during copying data period and has not much influence of the performance.

Table 1: Compare Data Copying Time

Text Size	*Complicated Index*	*Simple Index*	*Speedup*
100MB	0.004969	0.004620	1.08
200MB	0.009595	0.008892	1.08
400MB	0.018858	0.017499	1.08

Table 2: Compare Total Time

Text Size	*Complicated Index*	*Simple Index*	*Speedup*
100MB	0.010992	0.011300	0.973
200MB	0.021179	0.021749	0.974
400MB	0.041894	0.043053	0.973

In the real implementation, to meet the coalesce accessing requirement, the key problem is how to design the corresponding index of the shared memory and global memory. The pseudo-code are as follow:

```
for(k = bid;k < blockProcessNum; k += GRID)
    for(i = 0;i < page_size / 64;i++)
        for(j = 0;j < 16;j++)
            int s_index,gpu_index;
            s_text[s_index] = gpu_text[gpu_index];
```

The *s_text* and *gpu_text* are the beginning position of shared memory and global memory. The global memory index *gpu_index* is:

$$K * BLOCK * page\ size / 4 + (tid >> 4) * page\ size / 4 + (tid\&15) + j * page\ size + I * 16$$

*K*BLOCK*page_size/4* is the beginning position of each block. *j*4*page_size/4* means the inner loop deal with 4 pages each time and moves to the next 4 pages with the growth of *j*. *(tid>>4)*page_size/4* divides the 64 thread into 4 groups in which 16 threads are responsible for one page. *i*64/4* is the corresponding 64 Bytes of each page and moves to the next 64 Bytes with the growth of i. *(tid&15)* represents the 64 Bytes the thread actually accessed.

The shared memory index *s_index* is:

$$4 * j * 64 / 4 + ((tid >> 4) * 64 / 4) + (tid\&15)$$

4 j*64/4* means moving 256 Bytes each time. *((tid>>4)* 64/4)* divided the 64 threads into 4 groups, each group deal with one page. *(tid&15)* represent the 64 Bytes waiting for matching in every page. While doing really matching on the shared memory, the beginning position of each 64 Bytes using by different thread is calculated as *tid*64*.

Change the Location of State Machine. The initial GAC algorithm stores the AC state machine

in the texture memory. The texture memory is designed for the graphic applications which have a lot of two-dimensional spatial locality. In a certain computer application, it means that the position which a thread is accessing is very close to that of another thread. Therefore it is an optimization for the application which has lots of spatial locality. Under the situation where the number of PATTERNs are small or the length of PATTERNs is short, the state machine has lots of special locality so that the accessing process of the thread can be accelerated by using the texture memory. But in the practical applications, there are often lots of PATTERNs and the length of them may be very long. Therefore, the spatial locality of the state machine is often small. The cost of using texture memory is bigger than the performance improvement it brings. In the experiment, 200 snort rules are used as the complicated PATTERNs to build the state machine in the practical application. Table 3 compared the matching time of these two strategy.

Table 3: Compare Matching Time of Different Storage Strategy

Text Size	*Texture Matching Time*	*Global Memory Matching Time*	*Speedup*
100MB	0.143	0.017	8.4
300MB	0.430	0.063	6.8
600MB	0.850	0.127	6.7

It is shown that, in the practical detection, if there are lots of PATTERNs or the length of PATTERNs is very long. Putting the state machine into the global memory straight-forward can get a better speedup. So in this paper, the state machine is put into the global memory.

Experiments and Conclusions

Table 4: Test Speedup in Experiments

Buffer Size	*Tranditional Model*	*Heterogeneous Model*	*Speedup*
100MB	1.010	0.027	37.4
250MB	4.723	0.083	56.9
500MB	8.330	0.197	42.3

Table 4 compared the matching time between traditional CPU model and the GPU and CPU heterogeneous model under different buffer size.

The experiment results show that, compared with the traditional CPU model, the CPU and GPU heterogeneous model can get 45.5 times speedup on average. The average matching speed can reach 14368 Mb/s, which meet the requirement of detecting the 10G/s network in real-time.

In summary, the string matching speed of network detection system can be greatly improved using this CPU and GPU heterogeneous model. In the future, with the continued increasing of network traffic, in order to improve the degree of the parallelism, the research can focus on using multi-GPU to build GPU clusters. And this model can also be used as a prototype for doing the research about virus scanning or intrusion detection system on GPU.

Acknowledgements

This paper is supported by Chongqing Science & Technology Commission (No. cstc2012jcsf-jfzhX0011) and Chongqing Municipal Education Commission (No. KJ1402002). The author would like to thank both of them.

References

[1] T. Dingche, The research of the parallel string matching algorithm based on gpu. Master's thesis, Xiangtan University, 2009.

[2] M. Fisk and G.Varghese, An analysis offast string matching applied to content-based forwarding and intrusion detection. Technical Report CS2001-0670 (updated version), University of California-San Diego, Tech. Rep., 2002.

[3] A. V. Aho and M. J. Corasick, Efficient string matching: an aid to bibliographic search. Communications of the ACM, vol. 18, no. 6, pp. 333–340, 1975.

[4] R. S. Boyer and J. S. Moore, A fast string searching algorithm. Communications of the ACM, vol. 20, no. 10, pp. 762–772, 1977.

[5] D. M. Sunday, A very fast substring search algorithm. Communications of the ACM, vol. 33, no. 8, pp. 132–142, 1990.

[6] S.Wu, U. Manber et al., A fast algorithm for multi-pattern searching. Technical Report TR-94-17, University of Arizona, Tech. Rep., 1994.

[7] M. Norton, Optimizing pattern matching for intrusion detection. Sourcefire, Inc., Columbia, MD, 2004.

[8] Q. Z. Xu Hong, An improved ac algorithm for intrusion detecting. Microelectronics and Computer, no. 11, pp. 109–112, 2010.

[9] Y. Chao, Two-way ac algorithm and its application to intrusion detection system. Computer Systems and Applications, vol. 20, no. 3, pp. 222–225, 2011.

[10] B. Commentz-Walter, A string matching algorithm fast on the average. Springer, 1979.

[11] Q. Z.Wan Guogen, Improved ac-bm algorithm for matching multiple strings. Journal of University of Electronic Science and Technology of China, vol. 35, no. 4, pp. 531–533, 2006.

[12] H. Jiang, the algorithm and implementation of string matching in the intrusion detection system. Ph.D. dissertation, Huazhong University of Science and Technology, 2008.

[13] Z. Y. Zhang Shu, GPU high performance computing about CUDA. China WaterPower Press, 2009.

[14] P. Jiangfeng, the research and implementation of string matching algorithm based on cpu and gpu heterogeneous platform. Master's thesis, South China University of Technology, 2011.

[15] C.-H. Lin, S.-Y. Tsai, C.-H. Liu, S.-C. Chang, and J.-M. Shyu, Accelerating string matching using multi-threaded algorithm on gpu. in Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE. IEEE, 2010, pp. 1–5.

[16] N.-P. Tran, M. Lee, S. Hong, and J. Bae, Performance optimization of aho-corasick algorithm on a gpu. in Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on. IEEE, 2013, pp. 1143–1152.

[17] X. Zha and S. Sahni, Gpu-to-gpu and host-to-host multipattern string matching on a gpu. Computers, IEEE Transactions on, vol. 62, no. 6, pp. 1156–1169, 2013.

[18] E. K. Jason Sanders, an Introduction to General-Purpose GPU Programming. China Machine Press, 2011.

[19] N. Corporation, Cuda c best practices guide. 2012.