A Parallel AC Algorithm Based on SPMD for Intrusion Detection System

Xiong Su
Department of Computer Science and Engineering
Harbin Institute of Technology
Harbin, China
hit_suxiong@163.com

Zhenzhou Ji
Department of Computer Science and Engineering
Harbin Institute of Technology
Harbin, China
jzz@pact518.hit.edu.cn

Xiaoyang Lian
Department of Computer Science and Engineering
Harbin Institute of Technology
Harbin, China
lianxiaoyang.happy@163.com

***Abstract*—AC algorithm, as a multi-pattern matching algorithm, plays an important role in the intrusion detection system. The efficiency of the pattern matching algorithm directly affects the overall efficiency of the intrusion detection system. But the number of the growing intrusion features and demanding for rapid detection put forward a new challenge to the efficiency of the pattern matching algorithm. In this paper, by analyzing the potential parallelism of the AC algorithm and using SPMD method, we design a parallel AC algorithm based on multi-core processors. The experiment shows that the parallel AC algorithm can greatly improve the efficiency of intrusion detection.**

Keywords-AC Algorithm; Parallel; SPMD; Intrusion Detection System; multi-core processors

I. INTRODUCTION

In the field of network security, intrusion detection system [1] is an important means of detecting network attacks. There are two common models [3] intrusion detection system. One is the anomaly detection model, another is misuse detection model. In the misuse detection model, we should establish the intrusion feature database firstly, and then compare the processed data-stream from the network interface with intrusion features from the database. If the intrusion feature is matched, the data-stream contains attacks. If not, the data-stream is safe.

The misuse detection model mainly uses pattern matching algorithm to make comparison with intrusion features in the database. And the comparison needs to be a real-time performance. So, it's import to improve the efficiency of the pattern matching algorithm. AC algorithm [2] [4], the BM algorithm [5] and WM algorithm [6] [7] are often used in the misuse detection model. But in the face of the environment of multi-core processors and cluster [8] [9], the algorithm's efficiency is still not good enough. The purpose of this paper is to put forward a parallel AC algorithm to improve the efficiency of the pattern matching algorithm.

II. CLASSICAL AC ALGORITHM

The classical AC algorithm includes preprocessing process and matching process two parts. In preprocessing process, we need to build Goto function, Failure function and Output function. In matching process, we need to build pattern matching function.

At first, we define pattern-set as *patterns* = {p_1, p_2, p_3, …, p_r}, p_i is a text string pattern; and we define text string to be matched as *TEXT* = {$t_1t_2t_3…t_n$}, t_i is an ASCII character.

A. Goto function

Goto function is a function that creates DFA for pattern-set *patterns*. The implementation of the function is as follows:

(1) Create state 0
(2) Create DFA for p_1, p_2, p_3, …, p_r
(3) If not exist Goto(0,x), then 0→Goto(0,x), here x is an ASCII character

The implementation of creating DFA for p_i = {$a_1a_2a_3…a_m$} as follows:

(1) 0→state; j=1
(2) While Goto(state,a_j) exists
    Goto(state,a_j) →state; j + 1→j
(3) For every a_i from a_j to a_m, create a new state for it
(4) As to last state of p_i = $state_{pi}$, p_i→Output($state_{pi}$)

B. Failure function

Failure function is a function that specifies which state should be next state when Goto function cannot specifies the next state. That is to say, the meaning of Failure(state1) = state2 is that if Goto(state1, x) not exist, we should turn to state2 for next step. And state2 has two principles: the first principle is that at position of state1 and state2 in DFA, they have the longest common suffix; the second principle is that they have no common suffix, and in this principle the state2 must be 0.

The implementation of the function is a breadth-first process. We define state 0 as tier 0. After that, we will build the Failure function of next tier as follows:

(1) EN queue states of tier1 and set 0→Failure($state_i$), here $state_i$ is in tier 1
(2) DE queue one state, we define it as r
(3) EN queue states that are next states of r and in next tier of r, and then according to the two principles, calculate the Fairlure(r)→s.
(4) Output(r)+Output(s)→Output(r)
(5) If queue is not empty, turn to (2); If not, quit

C. Output function

Output function is a function that determines whether a state is matched or not. Output($state_{out}$) is a pattern-set, it includes the patterns that should be matched at $state_{out}$. And if Output($state_{out}$) is null, text string *TEXT* is not matched at

$state_{out}$. The implementation of the function is contained in Goto function and Failure function.

D. *Search function*

Search function is a function that finds patterns included in pattern-set in text string *TEXT*. Later we will analyze and improve the function.

III. PARALLEL AC ALGORITHM

SPMD is a single program multiple data parallel implementation methods. In parallel AC algorithm, we use same sub-search program to process different part of *TEXT*.

The implementation of Parallel AC algorithm includes the design of main thread and sub-search thread.

A. *Main thread*

Fig. 1 shows the flow chart of main thread of parallel AC algorithm.

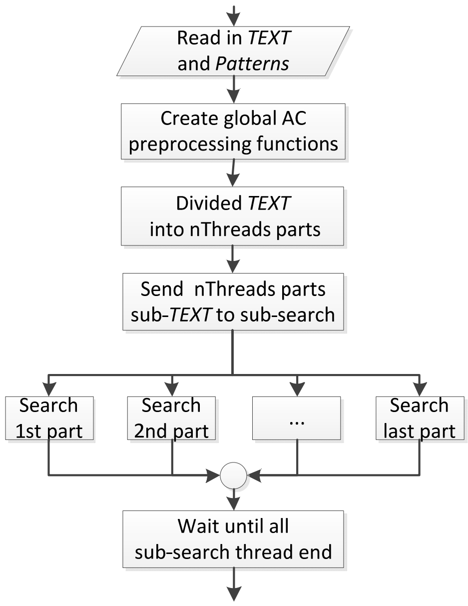

Figure 1. This is main thread flow chart of parallel AC algorithm which includes 5 main steps. And the step dividing *TEXT* is a most import step.

Main thread includes five steps. Step1 is Reading in *TEXT* and *Patterns*. Step2 is creating global AC preprocessing functions, and global preprocessing functions are shared by all threads. Step3 is dividing *TEXT*. Step4 is sending different data to different thread, and the function to create thread in windows operation system is *_beginthreadex*. Step5 is waiting for the end of all threads, the function is *waitforsingleobject*. And the threads we create are all searching threads.

In all steps of main thread, the most important step is how to divide *TEXT* into *nThreads*. In this paper, we use an overlap way to divide *TEXT* into *nThreads* parts. Every sub-search thread K (K from 0 to nThreads-1) handles its own sub-*TEXT*. The sub-*TEXT* includes two parts: the independent part and overlap part. The reason why we divide *TEXT* like this is to guarantee all positions in *TEXT* should be searched. At the same time, in order to improve the efficiency, the overlap part should be handled by only adjacent two threads and because of that we must set a maximum allowable number of threads. Fig.2 shows *TEXT* is divided into *nThreads* parts; here *nThreads* is 4.The division process of *TEXT* is as follows:

(1) Calculate the n_{max}. We define set N = $\{n_1, n_2, n_3, \ldots, n_r\}$. n_i is the length of pattern p_i.

$$n_{\max} = \max\{n_1, n_2, n_3, \ldots, n_r\} \quad (1)$$

(2) Calculate the *nMaxThreads*. *nMaxThreads* is the maximum allowable number of threads. T_n is the length of *TEXT*.

$$nMaxThreads = \left\lfloor \frac{T_n}{n_{\max}} \right\rfloor - 1 \quad (2)$$

(3) Set *nThreads*. *nThreads* is the number of running threads. *nThreads* is no more than *nMaxThreads*.

(4) Calculate the *tOverlapLen*, *tIndependentLen* and *tIndLen*. *tLen* is the length of sub-*TEXT*. Here, tIndependentLen is the length of independent part(*tIndLen* for short). *tOverlapLen* is the length of overlap part(*tOlLen* for short).

$$tOlLen = n_{\max} \quad (3)$$

$$tIndLen = \left\lfloor \frac{T_n - (nThreads - 1) * tOlLen}{nThreads} \right\rfloor \quad (4)$$

$$tLen = tOlLen + tIndLen \quad (5)$$

(5) Calculate data interval for thread K.

$$\begin{array}{ll} \left[T_{K*tIndLen}, T_{K*tIndLen+tLen}\right) & (0 < K < nThreads - 1) \\ \left[T_{K*tIndLen}, T_n\right) & (K = nThreads - 1) \end{array} \quad (6)$$

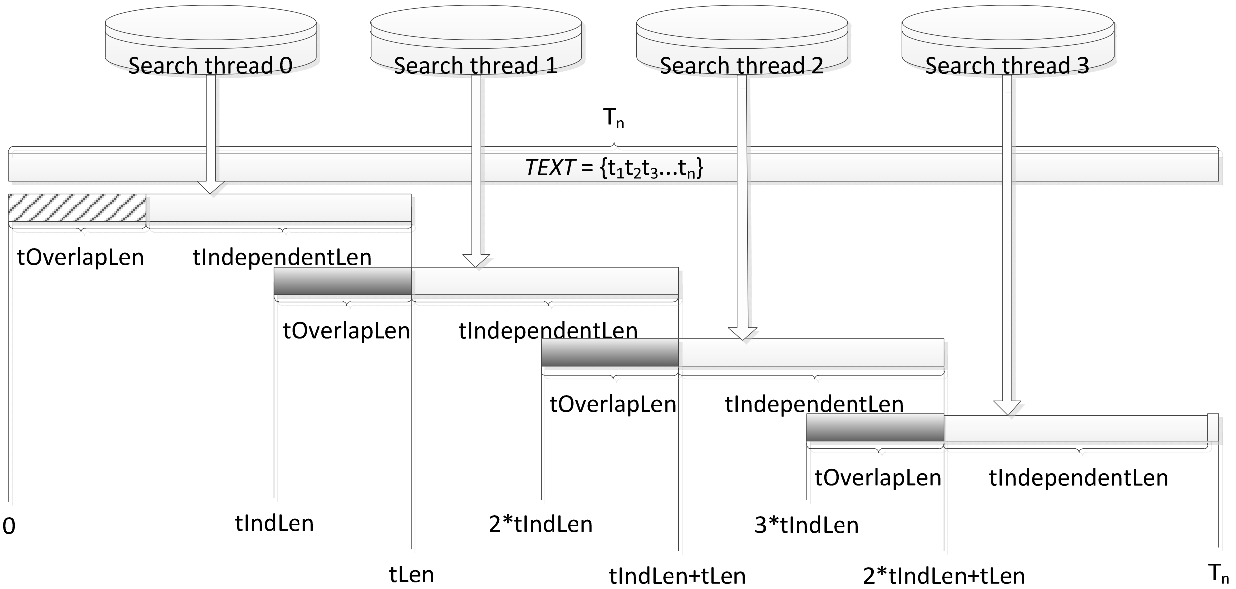

Figure 2. This is an example about how to divide *TEXT* into 4 parts, so that 4 threads should be create for parallel searching the *TEXT*

B. *Sub-search thread*

Every sub-search thread searches its own sub-*TEXT*. When searching independent part, it is the same as classical AC search function, which searches and outputs the pattern matched; when searching overlap part, the overlap part would be searched, but not output. The sub-search algorithm for thread 1 to nThreads-1 is as follows:

```
1 Begin
2    For Ti from Tstart to Tend:
3    Begin
4      while (Goto (state) == null and state != 0):
5            state = Failure(state)
6      state = Goto (state,Ti)
7      if Output(state) != null and i>start+tOlLen
8           store Output(state) and position i to Log file
9    End
10 End
```

Require special handling when K is thread 0. So, Statement (7) need to be changed as follows:

```
7    if Output(state) != null
```

IV. Experiments and Analysis

A. *Test platform and data sets*

Snort [1] - [3] is an open platform of Intrusion detection system. The default pattern matching algorithm is the classic AC algorithm. We use the parallel AC algorithm to replace it and recompile it. And then we make a comparison in terms of running time.

The experiment used a 1999DARPA [10] intrusion detection data set [7] [11]. The data set is the simulation the MIT Lincoln Lab collected for five consecutive weeks of real network data. We intercept the tcpdump (about 202M) of Monday of the fourth week to test the performance of parallel AC algorithm. Experimental environment: CPU, the Intel ® Core ™ 2 Quad 4-core processor; memory, 2G; Compilers, VC6.0.

B. *Analysis of Results*

Fig. 3 shows the result of the experiments.

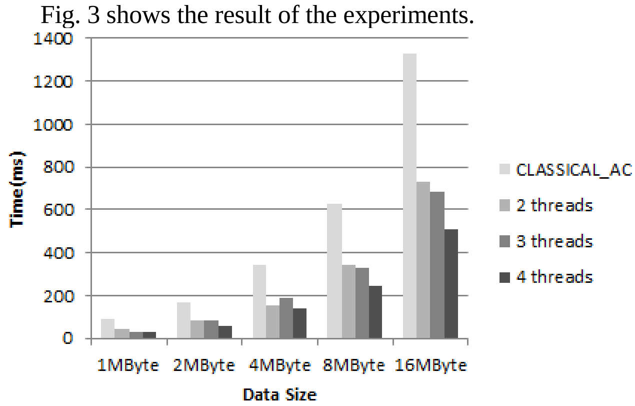

Figure 3. The picture shows results of the experiments, these are the original data collected from platform

With Parallel AC algorithm, the efficiency of snort has at least improved twice than classical AC algorithm. And the speedup becomes larger with the increase of the number of threads.

The ideal time complexity of the parallel AC algorithm is O (tLen). But the actual running time is O (tLen) + $time_{para}$ + $time_{sync}$. $time_{para}$ is parallel cost, including the cost of creating thread and the cost of destructing thread; $time_{sync}$ is synchronization cost, including cost of writing the log file. Therefore, the actual average speedups are 1.96, 2.16, 2.70 for 2 threads, 3threads and 4 threads, and the actual speedups are less than the ideal speedups 2, 3, 4. In addition, the gap between ideal speed and actual speedup is growing larger with increase of the number of threads. Fig. 4 shows the actual speedups.

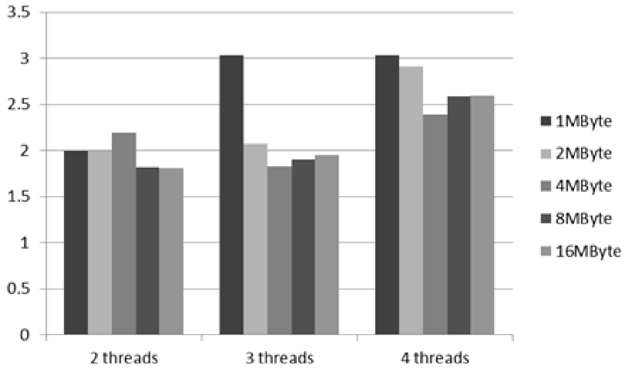

Figure 4. These are speedups for 2,3,4 threads. The speedup for 4 threads should be 4,but it's even no more than 3

V. CONCLUSION

Computer multi-core system has become a trend, and pattern matching algorithm which adapts to the environment of multi-core can effectively improve the efficiency of intrusion detection. This paper present a parallel multi-pattern matching algorithm, based on SPMD, can greatly improve the utilization of multi-core CPU and the efficiency of the intrusion detection. In addition, multi-threaded shared globally AC function, space complexity does not increase. Overall, compared to classical AC algorithm, the parallel AC algorithm has a larger optimization and can actually improve the efficiency of the intrusion detection.

ACKNOWLEDGMENT

This paper is supported by the National Natural Science Foundation of China, NO. 61173024.

REFERENCES

[1] ShuZheng Li, "The Rsearch of Fast Pattern Matching Algorithm Based on Snort System," JiLin University, 2009.4.

[2] Sun Qiang, Xin Yang, Chen Linshun, "Optimization and Application of the AC Multiple Patterns String Match Algorithm," *Sciencepaper Online*, vol. 6, no. 1, pp. 45-48, 2011.

[3] KONG Dong-lin, LUO Xiang-yang, DENG Qi-hao,LUO Jun-yong, "Reasearch on An Aho-Corasick Automaton Matching Algorithmbased Intrusion Detection System," *Microelectronics and Computer*, vol. 23, nol. 8, pp. 89-95, 2006.

[4] Aho A, Corasick M, "Efficient String Matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333-343, 1975.

[5] Boyer R S, Moore J S, "A Fast String Searching Algorithm," *Communication of the ACM*, vol. 20, no. 10, pp. 762-772, 1997.

[6] WU Sun, MANBER U, A Fast Algorithm for Multi-pattern Searching, Department of Computer Science, Chung-Cheng University, 1994.

[7] SONG Mingqiu, ZHANG Guoquan, DENG Guishi, "A New Multi-pattern Matching Algorithm of Intrusion Detection," *Computer Engineering*, vol. 32, no. 5,pp. 144-146, 2006.

[8] HongWei Yu, "Research and Implementation of Effective Intrusion Detection Based on Multi-core Processors," University of Electronic Science and Technology of China, 2009.

[9] LIU Yan-Heng, TIAN Da-Xin, YU Xue-Gang, WANG Jian, "Large-Scale Network Intrusion Detection Algorithm Based on Distributed Learning," *Journal of Software*, vol. 19, no. 4, pp. 993-1003, 2009.

[10] R.Lippmann, J.Haines, D.Fried, J.Korba, K.Das, " The 1999 DARPA off-line Intrusion Detection Evaluation," *Computer Networks*, vol. 34, pp. 579-595.

[11] XU Hong, QIN Zhi-guang, "An Improved AC Algorithm for Intrusion Detection," *Microelectronics and Computer*, vol. 27, no. 11, pp. 109-112, 2010