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 = {p1, p2, p3, …, pr}, pi is a text string pattern; and we define text string to be matched as TEXT = {t1t2t3…tn}, ti 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 p1, p2, p3, …, pr
(3) If not exist Goto(0,x), then 0 → Goto(0,x), here x is an ASCII character
(4) The implementation of creating DFA for p = {a1a2a3…an} as follows:

(1) 0 → state; j=1
(2) While Goto(state,a) exists
    Goto(state,a) → state; j + 1 → j
(3) For every ai from aj to am, create a new state for it
(4) As to last state of pi = statepi, pi → Output(statepi)

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), here state 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(stateemo) is a pattern-set, it includes the patterns that should be matched at stateemo. And if Output(stateemo) is null, text string TEXT is not matched at
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.

1. Read in TEXT and Patterns
2. Create global AC preprocessing functions
3. Divide TEXT into nThreads parts
4. Send nThreads parts sub-TEXT to sub-search
5. Search 1st part
6. Search 2nd part
7. ...  
8. Search last part
9. Wait until all sub-search thread end

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

\[
 n_{\text{max}} = \max \{n_1, n_2, n_3, ..., n_r\} 
\]

(2) Calculate the \( n_{\text{MaxThreads}} \). \( n_{\text{MaxThreads}} \) is the maximum allowable number of threads. \( T_n \) is the length of TEXT.

\[
 n_{\text{MaxThreads}} = \left[ \frac{T_n}{n_{\text{max}}} \right] - 1 
\]

(3) Set \( n_{\text{Threads}} \). \( n_{\text{Threads}} \) is the number of running threads. \( n_{\text{Threads}} \) is no more than \( n_{\text{MaxThreads}} \).

(4) Calculate the \( t_{\text{OverlapLen}}, t_{\text{IndependentLen}} \) and \( t_{\text{IndLen}} \). \( t_{\text{Len}} \) is the length of sub-TEXT. Here, \( t_{\text{IndLen}} \) is the length of independent part \( t_{\text{IndLen}} \) for short). \( t_{\text{OverlapLen}} \) is the length of overlap part \( t_{\text{OlLen}} \) for short).

\[
 t_{\text{OlLen}} = n_{\text{max}} 
\]

\[
 t_{\text{IndLen}} = \left[ T_n - (n_{\text{Threads}} - 1) \times t_{\text{OlLen}} \right] / n_{\text{Threads}} 
\]

\[
 t_{\text{Len}} = t_{\text{OlLen}} + t_{\text{IndLen}} 
\]

(5) Calculate data interval for thread K.

\[
 \left[ T_{K \times t_{\text{IndLen}} + t_{\text{Len}}}, T_{K \times t_{\text{IndLen}} + t_{\text{Len}}} \times n_{\text{Len}} \right) (0 < K < n_{\text{Threads}} - 1) 
\]

\[
 \left[ T_{K \times t_{\text{IndLen}} + t_{\text{Len}}}, T_n \right) (K = n_{\text{Threads}} - 1) 
\]
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 T_i from T_start to T_end:
3         Begin
4             while (Goto (state) == null and state != 0):
5                 state = Failure(state)
6                 state = Goto (state, T_i)
7             if Output(state) != null and i>start+tOILen
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 1999 DARPA [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.

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_{par} + time_{sync}. time_{par} 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, 3 threads 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.

![Graph showing speedups for 2, 3, and 4 threads.]

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 presents a parallel multi-pattern matching algorithm, based on SPMD, which 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