July 2014

A NOVEL PATTERN MATCHING METHOD FOR MEMORY REDUCTION APPLICATIONS USING AC ALGORITHM

G.F. HARISH REDDY

School of Electronics, Vignan University, Vadlamudi, Guntur, AP, India, harishreddy.gf@gmail.com

S. INDIRA PRIYADARSINI

School of Electronics, Vignan University, Vadlamudi, Guntur, AP, India, si_priyadarsini@yahoo.com

J. PRATIBHA

School of Electronics, Vignan University, Vadlamudi, Guntur, AP, India, j.pratibha@gmail.com

Follow this and additional works at: <https://www.interscience.in/ijess>

Part of the [Electrical and Electronics Commons](#)

Recommended Citation

REDDY, G.F. HARISH; PRIYADARSINI, S. INDIRA; and PRATIBHA, J. (2014) "A NOVEL PATTERN MATCHING METHOD FOR MEMORY REDUCTION APPLICATIONS USING AC ALGORITHM," *International Journal of Electronics Signals and Systems*: Vol. 4 : Iss. 1 , Article 6.

DOI: 10.47893/IJESS.2014.1195

Available at: <https://www.interscience.in/ijess/vol4/iss1/6>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Electronics Signals and Systems by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

A NOVEL PATTERN MATCHING METHOD FOR MEMORY REDUCTION APPLICATIONS USING AC ALGORITHM

G.F. HARISH REDDY¹, S.INDIRA PRIYADARSINI², J.PRATIBHA³

¹PG Student (M.Tech VLSI), School of Electronics, Vignan University, Vadlamudi, Guntur, AP, India

^{2,3}Assistant Professor, School of Electronics, Vignan University, Vadlamudi, Guntur, AP, India

Abstract -- Advances in computer networks and storage subsystems continue to push the rate at which data streams must be processed between and within computer systems. Meanwhile, the content of such data streams is subjected to ever increasing scrutiny, as components at all levels mine the streams for patterns that can trigger time-sensitive action. The problem of discovering credit card numbers, currency values, or telephone numbers requires a more general specification mechanism. While there is a well developed theory for regular expressions and their implementation via Finite-State Machines (FSMs), the use of regular expressions for high-performance pattern matching is more difficult and is an area of ongoing research. In this Paper, a memory-efficient pattern matching algorithm which can significantly reduce the number of states and transitions by merging pseudo-equivalent states while maintaining correctness of string matching. In addition, the new algorithm is complementary to other memory reduction approaches and provides further reductions in memory needs.

Keywords -- Aho-Corasick (AC) algorithm, finite automata, pattern matching, Content Filtering, Pattern Matching, Network Intrusion Detection, Bloom Filters.

1. INTRODUCTION

It is becoming increasingly common for network devices to handle packets based on the contents of packet payloads. Example applications include intrusion detection, firewalls, and web proxies. These packet content inspection and filtering devices rely on a fast multi-pattern matching algorithm which is used to detect predefined keywords or signatures in the packets. Unfortunately, these signature sets are large (e.g., thousands) and complex, multi-pattern matching is often a performance bottleneck. Another problem is, to accelerate the speed, fast string matching is necessary. In the matching, a set of rules is statically giving].

The main purpose of a signature-based network intrusion detection system is to prevent malicious network attacks by identifying known attack patterns. Due to the increasing complexity of network traffic and the growing number of attacks, an intrusion detection system must be efficient, flexible and scalable.

The primary function of an intrusion detection system is to perform matching of attack string patterns. Because string matching is the most computative task in network intrusion detection (NIDS) systems, many hardware approaches are proposed to accelerate string matching. The hardware approaches may be classified into two main categories, the logic and the memory architectures.

The basic memory architecture works as follows. First, the (attack) string patterns are compiled to a finite-state machine (FSM) whose output is asserted when any

substring of input strings matches the string patterns. Then, the corresponding state transition table of the FSM is stored in memory.

Figure 1 DFA for matching “bcd” and “pcdg”.

For instance, Fig. 1 shows the state transition graph of the FSM to match two string patterns “bcd” and “pcdg”, where all transitions to state 0 are omitted. States 4 and 8 are the final states indicating the matching of string patterns “bcd” and “pcdg”, respectively. Fig. 2 presents a simple memory architecture to implement the FSM.

In the architecture, the memory address register consists of the current state and input character; the decoder converts the memory address to the corresponding memory location, which stores the next state and the match vector information.

Figure 2 Basic memory operation

A “0” in the match vector indicates that no “suspicious” pattern is matched; otherwise the value in the matched vector indicates which pattern is matched. For example in Fig. 2, suppose the current state is 7 and the input character is . The decoder will point to the memory location which stores the next state 8 and the match vector 2. Here, the match vector 2 indicates the pattern “pcdg” is matched.

In the past few years, several interesting algorithms and techniques have been proposed for multi-pattern matching in the context of network intrusion detection. The hardware-based techniques make use of commodity search technologies such as TCAM [1] or reconfigurable logic/FPGAs [6][1][3][7]. Some of the FPGA based techniques make use of the on-chip logic resources to compile patterns into parallel state-machines or combinatorial logic.

An approach presented in [5] uses FPGA logic with embedded memories to implement parallel Pattern Detection Modules (PDMs). PDMs can match arbitrarily long strings by segmenting them in smaller substrings and matching them sequentially.

A Bloom-filter based algorithm proposed in [8] makes use of a small amount of embedded-memory along with commodity offchip memory to scan a large number of strings at high speed. Using on-chip Bloom filters, a quick check is done on the payload strings to see if it is likely to match a string in the set. Upon a Bloom filter match, the presence of the string is verified by using a hash table in the off-chip memory. The authors argue that since the strings of interest are rarely found in the packets, the quick check in Bloom filter reduces more expensive memory accesses and improves the overall throughput greatly.

However, since the algorithm involves hashing over a maximum length pattern size text window, it does not scale for arbitrarily long strings (100s of bytes). It is reported that up to 16 bytes is a feasible pattern length for a high-speed implementation. As we will see, our algorithm combines the techniques in [8] with Aho-Corasick algorithm to get rid of the string length limitation. In this paper we propose merge FSM technique in additional with traditional AC Algorithm.

1. Proposed algorithm

In multi-string matching problem, we have a set of strings S and we would like to detect all the occurrences of any

of the strings in S in a text stream T . We will denote by $T[i..j]$ the character sequence from i th character to j th character of stream T . For a given set of strings, the Aho-Corasick algorithm constructs a finite automaton.

This finite automaton can be a Deterministic Finite Automaton (DFA) or a Non-deterministic Finite Automaton (NFA). For our purpose, we will focus on NFA version of the algorithm since that is the one we will improve upon. Otherwise, it makes a failure transition. In case of a failure transition the machine must reconsider the character causing the failure for the next transition and the same process is repeated recursively until the given character leads to a non-failure transition. The first fundamental problem Aho-Corasick algorithm suffers from is a high memory access requirement.

At least one memory access is needed to read the state node on each input character. Furthermore, the sequential failure transitions can cause more memory accesses. In the worst case, the average number of memory accesses required per input character is two. Therefore, given the high latency and slow speed of commodity memory chips, using them to implement Aho-Corasick algorithm can severely degrade the throughput of the system.

The second problem we observe in the regular Aho-Corasick algorithm is that it can not be readily parallelized. Hence, we are forced to consider only one character at a time from the text stream no matter how much logic and memory resources are available. The processing of one character per clock cycle of the system clock can create a bottleneck for high speed networks

Architectural Implementation

We now describe the hardware architecture we assume for implementing the state machine. The architecture consists of a TCAM, static RAM (SRAM) and a logic. Each TCAM entry represents a certain transition in the state machine, and has a corresponding memory block (structure) in the SRAM whose address can be computed from the TCAM index. We logically partition the TCAM entries into two fields: *current state* and *input*. If the state machine transitions from state $s1$ to state $s2$ on input a , then the TCAM contains an entry ($s1, a$) and the corresponding entry in SRAM contains $s2$. If the state $s2$ corresponds to one or more keywords, then the SRAM entry also contains pointers to those keywords.

Fig 3. Hardware Architecture

Fig.7 and 8 shows the results before and after integrating our algorithm to the AC algorithm that is the propose system versus the existing system. Columns one, two and three show the name of the rule set, the number of patterns, and the number of characters of the rule set. Columns four, five, and six show the number of state transitions, the number of states, and the memory size of

the AC algorithm. Columns seven, eight, and nine show the results of our approach. Column ten shows the memory reduction compared to the AC algorithm. As shown in Fig. 5, the memory requirement includes the size of the valid memory and the failure memory. Fig. 9 gives the Modelsim results of the Pattern matching cases like one pattern match case and two patterns match case.

Figure 7 Existing method Gate Count Synthesis Report

Figure 8 Reduction in the number of Flip flops (gates) in the proposed method

(a)

(b)

Figure 9 Simulation Result for the proposed AC Algorithm method

4. CONCLUSIONS

We have presented a high-speed and scalable pattern matching algorithm that uses multi-character transitions on finite state automata to increase the throughput, and also leverages a clever transition optimization technique to reduce the memory requirements. However this algorithm has a can efficiently work for string matching applications only. Since the number of states in *merg_FSM* can be drastically smaller than the original FSM, it results in a much smaller memory size. We also show that hardware needed to support the state-traversal mechanism is limited. The size and memory requirements are reduced. That is this algorithm can notify to the user if the string is completely matched as shown in the Fig.9. Additionally, packet inspection in the network is essential for various applications including QoS monitoring, bandwidth metering, stateful packet filtering etc. Our simulation results demonstrate that the proposed algorithm indeed scales well in practice to meet the current day requirements.

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments which were very helpful in improving the quality and presentation of this paper.

REFERENCES:

- [1] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, 1975.
- [2] <http://www.snort.org>. SNORT: Open-Source Network IDS/IPS.
- [3] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, 1975.
- [4] D. E. Knuth, J. H. M. Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, June 1977.
- [5] M. Crochemore and D. Perrin, "Two-way string matching," *J. ACM*, vol. 38, no. 3, pp. 650–674, 1991.
- [6] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to ddos attack detection and response," in *DISCEX*, 2003.
- [7] L. Spitzner, *Honeypots: Tracking Attackers*. Addison-Wesley, 2002.
- [8] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM J. Res. Dev.*, vol. 31, no. 2, pp. 249–260, 1987.
- [9] N. T. et.al., "Deterministic memory-efficient string matching algorithms for intrusion detection," *INFOCOM*, 2004. R. S. Boyer and J. S. Moore, "A fast string matching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, October 1977.