Microprocessors and Microsystems 39 (2015) 122–134

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A hybrid multiple-character transition finite-automaton for string matching engine

Chien-Chi Chen, Sheng-De Wang [*]

National Taiwan University, 1, Roosevelt Road Section 4, Taipei 106, Taiwan

ARTICLE INFO

Article history:
Available online 17 February 2015

Keywords:
String matching
Deterministic and nondeterministic finite automaton
Aho-Corasick algorithm
Network intrusion detection system

ABSTRACT

The throughput of a string-matching engine can be multiplied up by inspecting multiple characters in parallel. However, the space that is required to implement a matching engine that can process multiple characters in every cycle grows dramatically with the number of characters to be processed in parallel. This paper presents a hybrid finite automaton (FA) that has deterministic and nondeterministic finite automaton (NFA and DFA) parts and is based on the Aho-Corasick algorithm, for inspecting multiple characters in parallel while maintaining favorable space utilization. In the presented approach, the number of multi-character transitions increases almost linearly with respect to the number of characters to be inspected in parallel. This paper also proposes a multi-stage architecture for implementing the hybrid FA. Since this multi-stage architecture has deterministic stages, configurable features can be introduced into it for processing various keyword sets by simply updating the configuration. The experimental results of the implementation of the multi-stage architecture on FPGAs for 8-character transitions reveal a 4.3 Gbps throughput with a 67 MHz clock, and the results obtained when the configurable architecture with two-stage pipelines was implemented in ASICs reveal a 7.9 Gbps throughput with a 123 MHz clock.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

String matching is used in many applications, such as in network intrusion detection systems (NIDS), and its efficiency dominates the performance of such applications. String matching typically includes exact string matching and regular expression matching; exact string matching is more efficient but less powerful than regular expression matching in searching for keywords in a text. The string matching algorithm of Aho and Corasick processes multiple keywords simultaneously, and locates all instances of these keywords in an n-character text with time complexity $O(n)$ [1]. Some applications like NIDS that must inspect a data stream online firstly use exact string matching to filter out suspicious data, and then use regular expression string matching to verify the filtered out data.

A hardware string-matching engine that is based on the AC-algorithm can effectively accelerate string matching in network applications [2–5]. However, network bandwidth is increasing as communication and integrated circuit technologies advance, so the performance of string-matching engines must also be improved to keep up with network throughput. Greatly increasing throughput in string matching depends on developing a hardware string-matching engine that can inspect multiple characters in parallel.

A string matching engine that is based on the AC-algorithm can be implemented in two ways: the first uses a deterministic finite automaton (DFA) and the second uses a nondeterministic finite automaton (NFA). In the DFA approach, a generalized architecture can be developed for processing various sets of keyword. This architecture is deterministic. Accordingly, the generalized architecture based on the DFA approach can be designed as a standalone device, which can be utilized for various keyword sets. However, the hardware efficiency of the DFA approach is poor, and the required space typically increases exponentially with the number of characters that are being inspected in parallel. The NFA approach is more efficient than the DFA approach in terms of hardware utilization, and the required space increases in proportional to the number of characters that are being inspected in parallel. Nevertheless, the hardware architecture varies with keyword sets in the NFA approach; accordingly, the NFA approach can be implemented only in programmable devices, such as FPGAs.

This paper proposes a hybrid approach, based on the AC-algorithm, that combines the NFA and DFA methods to provide the advantages of both hardware efficiency and a deterministic architecture. The proposed approach transforms an AC-trie, which

* Corresponding author at: Department of Electrical Engineering, BL-523 National Taiwan University 1, Roosevelt Road Section 4, Taipei 106, Taiwan. Tel.: +886 2 33663579.

E-mail address: sdwang@ntu.edu.tw (S.-D. Wang).

http://dx.doi.org/10.1016/j.micpro.2015.01.003
0141-9331/© 2015 Elsevier B.V. All rights reserved.

is a prefix tree that is based on the AC-algorithm, into a hybrid finite automaton, called AC-FA, that has NFA and DFA parts. The proposed hybrid AC-FA is converted into a multi-character hybrid AC-FA by iteratively performing concatenation operations, for the purpose of processing multiple characters in parallel. We have previously developed the algorithm for deriving multi-character transitions [6,7]. The states in an AC-trie with the same depth are grouped in the same level, and a level nearer to the root state is lower; the NFA part comprises the lower states. Since most states of an AC-trie are at lower levels, which belong to the NFA part, the growth of the number of transitions of a k-character AC-FA is almost linear with respect to k when the number of level in the NFA part is high enough.

This paper then develops a multi-stage architecture for implementing the proposed hybrid AC-FA. The transitions are grouped into stages based on their levels. The number of stages of the proposed multi-stage architecture is determined by the number of levels in the NFA part and the number of characters to be inspected in parallel. Since the number of stages can be determined as required, the proposed multi-stage architecture is further made into a configurable architecture that can be configured to process various keyword sets. The proposed configurable architecture can be utilized as a stand-alone device.

The proposed architecture is evaluated on FPGA and ASIC devices. The results of the evaluation reveal that the throughput and the hardware cost increase approximately linearly with respect to the number of characters to be inspected in parallel. In the implementation of k-character multi-stage architecture, for $k = 8$, the throughput is approximately 6.1 times and the hardware cost is approximately 2.7 times those for $k = 1$. The 8-character hybrid AC-FA with 300 keywords can be implemented with a 66.6 MHz clock and the achieved throughput is approximately 4.3 Gbps. The 8-character configurable architecture, including 512 rule units, can be implemented with a 54.18 MHz clock and the obtained throughput is approximately 3.5 Gbps. The proposed configurable architecture is also improved using the pipeline method. The 8-character configurable architecture with two-stage pipelines can be implemented at a clock rate of 123.44 MHz and the obtained throughput is approximately 7.9 Gbps, and the performance is roughly doubled that of the original configurable architecture. The main contributions of this paper are summarized as follows.

– The proposed multi-character hybrid AC-FA approach has the feature that the number of transitions increases almost linearly with respect to the number of characters to be inspected in parallel.
– A configurable architecture is developed, based on the proposed multi-stage architecture; it can process various keyword sets by simply updating the configuration data.

The rest of this paper is organized as follows. Section 2 reviews work on string matching. Section 3 briefly describes the AC-algorithm, and then develops the hybrid AC-FA and its implementation. Section 4 describes the construction of multi-character transitions and proposes a multi-stage architecture to implement the constructed multi-character transitions. Section 5 proposes the configurable architecture of the multi-character matching engine. Section 6 evaluates and discusses the proposed approach. Finally, Section 7 draws conclusions.

2. Related work

Many hardware-based approaches that are based on the AC-algorithm have been developed for accelerating string matching and these fall into two broad categories. One category focuses on improving the efficiency of hardware utilization because the AC-algorithm is a memory-exhausting algorithm, while the other focuses on improving the throughput of string matching, by increasing the clock rate of the hardware or by inspecting multiple characters simultaneously.

Owing to the progress and flexibility of the programmable devices such as FPGAs, developers can design and evaluate variant architectures according to the features of the AC-algorithm. However, the resources of programmable devices are limited, so hardware efficiency is important. To improve the memory efficiency, Tuck et al. proposed a bitmap-compression and path-compression approach to implement the AC-algorithm that effectively reduces the required memory and improves the performance of hardware implementation [8]. Zha and Sahni improved the bitmap-compression and path-compression approach such that it utilized much less memory [9]. Alicherry et al. implemented the AC-algorithm by integrating a ternary content addressable memory (TCAM) and an SRAM that utilizes the ternary matching of TCAM to match characters in negation expressions, subsequently reducing the space that is required for storing the state transitions [3]. Pao et al. and Lin and Liu developed pipeline architectures to implement an AC-trie that contains only goto functions of the AC-trie to reduce the space that is caused by extending failure functions [10,11]. Nan Hua et al. proposed another approach that was based on a block-oriented scheme, rather than the typical byte-oriented processing of patterns, to minimize the memory-usage [4]. The approach of Dimopoulos et al. partitions an entire AC-trie into numerous smaller tries to increase memory efficiency [12]. Nakahara improved the hardware efficiency of the implementation of regular expression matching by using the AC-algorithm to process shared patterns [2]. Becchi et al. presented a hybrid FA that combines the advantages of DFA and NFA to improve regular expression matching [13].

To multiply the throughput of string matching for a fixed same operating rate, some attempts have been made to develop string matching architectures that can inspect multiple characters in parallel. Such an architecture must account for both the complexity of hardware and the alignment problem, which arises in the k-character matching processes. Sugawara et al. developed a string matching method called suffix-based traversing (SBT), which extends the AC-algorithm to process multiple input characters in parallel and to reduce the size of the lookup table [14]. Alicherry et al. proposed a k-compressed AC-DFA to realize a k-character matching engine [3]. Some works have utilized multiple FSMs to achieve parallelism and solve the alignment problem; in these, each FSM processes a pattern, beginning at a different position, and then the matching results of the FSMs are combined using a specific logic [5,15,16]. Yamagaki et al. utilized additional states and transitions to solve the alignment problem [17].

3. Aho-Corasick algorithm and hybrid finite automaton

This section firstly describes the AC-algorithm. Then the original AC-trie is converted to a hybrid AC-FA which comprises an NFA part and a DFA part. Thereafter, the operations of the original AC-trie and the proposed hybrid AC-FA are explained and compared with each other. Finally, the proposed hybrid AC-FA is extended to a multi-character hybrid FA.

3.1. Aho-Corasick algorithm

The AC-trie that is shown in Fig. 1 is constructed with the keyword set {enhappy, happy, happen, happygo}, which is used as an example to explain the proposed approach. In the figure, the