Parallelization of String Matching Algorithm with Compaction of DFA

Apurva Joshi and Tanvi Shah

Abstract String matching algorithms are widely acknowledged due to its use in many areas such as digital forensics, intrusion detection system, plagiarism checking, bioinformatics. For improving the efficiency of the string matching, speed of matching the strings must be elevated. Hence, an approach has been proposed which would significantly reduce the time for matching the strings. Ternary content addressable memory (TCAM) has been used by many for reducing the time requirement. But TCAM has many disadvantages such as high cost, very high power dissipation, problem due to pipelining. Small-scale applications may not be able to bear all the disadvantages associated with TCAM. Hence, an approach has been proposed which would overcome all the disadvantages associated with TCAM. Modern CPUs have multicore facility. These multiple cores have been exploited to provide parallelism. Parallelism greatly helps to increase the speed of matching the string. Apart from this, reducing the memory requirement for string matching algorithm is also necessary. When reduction in memory requirement and parallelization are applied simultaneously, it provides improved results. High response time would be obtained by using this approach.

Keywords Aho–Corasick algorithm (AC algorithm) · Finite automaton (FA) · Parallelization · String matching

1 Introduction

String matching algorithms are the ones which find whether the given string is present within the larger string. These algorithms are at the heart of many important applications such as intrusion detection system (IDS), bioinformatics, digital

A. Joshi (✉) · T. Shah
Department of Computer Engineering, VJTI, Mumbai, Maharashtra, India
e-mail: apurva.joshi91@gmail.com

T. Shah
e-mail: tanvishahvcet06@gmail.com

P.K. Sa et al. (eds.), *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, Advances in Intelligent Systems and Computing 518,
DOI 10.1007/978-981-10-3373-5_33

forensics, plagiarism checking. When using string matching in IDS, the security factor and time factor along with accuracy must be considered, and when using in plagiarism checking and bioinformatics, correctness is important. Due to high criticality of applications using string matching, we can conclude that string matching applications are very critical and it is necessary to improve its efficiency.

The Aho–Corasick algorithm [1] is a widely used algorithm, used for the purpose of string matching. There are many factors which have led to wide spread use of this algorithm. AC has deterministic performance, and this is because each symbol scan results in a state transition. This does not depend on some sort of specific input, due to which it is not vulnerable to various attacks. Apart from this, this algorithm has a property that more than one string can be matched in a single pass. As an example, in the word share, there are three keywords, viz. share, hare and are. By using AC algorithm, all the three keywords are obtained in a single pass. Due to these characteristics of AC algorithm, it is widely adopted among all the string matching algorithms. AC algorithm is basically divided in major two stages, viz. (I) construction of finite machine (FA) which would act as a string matching machine and (II) traversal of string using the FA. For the purpose of construction of FA, goto, failure transitions and output function are calculated. goto function helps us create a simple FA. Failure function is the longest suffix of the string which is also the prefix of some node. The goal of the failure function is to restrict the algorithm from transiting to any state more than once. Failure functions would redirect us to correct transition in case the word is present, and if the word is not present, it would give a failure. Output function gives the output of a particular state, i.e. the keyword obtained on reaching a particular state. After the creation of FA with the failure states, the next stage is to inspect the string letter by traversing the FA.

Traditional approach for storage of FA requires storing one rule per transition. This leads to much wastage of memory. For the compaction of FA, [2] proposes such an algorithm which makes it possible for us to store one rule per state which in turn reduces the memory requirement to great extent. This algorithm suggests that the states should be encoded in a way such that all the transitions to a single state should be represented by a single prefix. Due to this, the problem boils down to longest prefix match first. The algorithm mainly consists of following stages, viz. (i) state grouping which finds longest common suffix for each state, (ii) common suffix tree which involves creation of tree by encoding each state with smaller number of bits and (iii) state and node encoding which creates a table containing current state, symbol and next state. Using this approach, compaction of FA has been made.

Parallelism is the process in which we divide the given task into chunks and perform those tasks simultaneously. This helps us reduce response time considerably. Parallelism has reached in various nooks and corners of technical fields. The technique of parallelism has been applied even in the field of string matching, but a hardware approach has been proposed which has its own disadvantages. To overcome those disadvantages, a software approach has been proposed which would exploit the multicore architecture of the CPU and give much faster results than its

serial counterpart. In this approach, we assign one thread to each letter and make the traversal. Each thread would traverse the FA individually. The threads assigned to the letters which do not start from starting state are terminated immediately. Due to this, there are not many threads running at the same time and there is not much overhead on the system.

2 Related Work

Many intensive efforts have been made for improving the efficiency of AC algorithm. Methods for compaction of DFA have been proposed. Also, the use of TCAM and IP lookup chips has been proposed. These algorithms have greatly contributed in making the string matching more efficient and time saving. For the real-time applications like IDS, use of ternary content addressable memory is very useful which has very good speed. But this comes with its own disadvantages.

Many efforts have been made to improve the efficiency of string matching keeping the AC algorithm as base. [3] suggests the elimination of failure transitions from AC algorithm. [4] suggests a technique of compression which in turn reduces the memory bandwidth required for processing the string. When processing a string which is of length N, this approach would require minimum 2N traversals. [5–8] propose a mechanism for reducing the space required to store the DFA. Some use Chinese remainder theorem, while some make use of more compact data structure for storage of the larger strings. [8] proposes use of NFA, i.e. nondeterministic finite automaton instead of using a DFA. [9, 10] have suggested methods for reducing the time required to match the string with the larger string by proposing the use of TCAM, i.e. ternary content addressable memory. The most considerable contribution is done by [2] which suggest compact DFA. The DFA used by AC algorithm uses one rule per transition as a result; there are many rules per state. This causes high requirement of memory for storage of DFA. Compact DFA suggests use of only one rule per state. As a result, the storage requirement would reduce, which would ultimately lead to improved throughput. These rules are stored as a triplet of [11]:

$$\text{Current State Field} \rightarrow \text{Symbol Field} \rightarrow \text{Next State Field}$$

3 Implementation

As referred above, the basic intention here is to increase the efficiency of the system, i.e. bring improvement in response time. For that purpose, following steps are followed: An Aho–Corasick machine is created. The technology used for implementation of the AC machine is CPP. goto function, failure function and

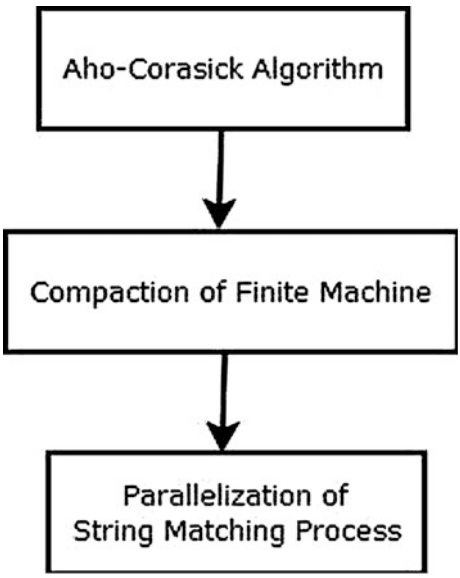

Fig. 1 Flow chart for the proposed system

construction of output function for FA are done using CPP. This leads to the completion of the phase of construction of FA. After that compaction of the FA, which is created is done. Each state is considered separately, and longest prefix for individual state is calculated which follows construction of common suffix tree based on the results of first step and then follows node encoding in which the actual encoding of each and every node is done. This process gives us a rule set which is much compact than its AC algorithm counterpart (Fig. 1).

This compact DFA structure has a table which contains current state, symbol and next state. Current state gives us the state where the transition has reached currently, symbol is the letter which causes transition from one state to another, and next state gives us the state where the transition would reach if the symbol found is correct. This compact data structure is much lesser in size and thus can be stored in faster memory. Current state for start state is always encoded by ***** and the states following the start state. For the purpose of parallelization, OpenMP with MPI has been used. OpenMP is a memory multiprocessing API which can be used with C, CPP, Fortran, etc. This API takes advantage of many cores available on the CPU. MPI helps to make distributed computing possible. It helps us to make use of several cores of different computers for the purpose of computations. Nowadays, every CPU is multicore. Using this API, one thread can be assigned to each core and program could be made to run in parallel. Each thread is assigned a single letter of the larger string. In the first pass, it checks whether the letter matches with the symbol led created in the above table, having current state as *****. If it does, the thread proceeds with its transitions, else it terminates then and there. This approach ensures the property of AC algorithm that all substrings could be identified in a single pass is maintained. As an example, in the word share, there is a substring

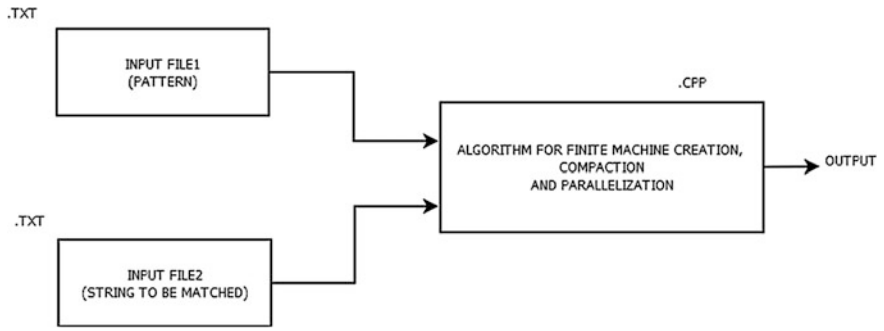

Fig. 2 Architecture for the proposed system

share, hare and are. All these three words could be identified in a single pass with AC algorithm. Along with preserving the property, it also helps us reduce the response time. A small improvement would be to use shared memory for storing the variables instead of using global memory. Due to this, the overhead involved in accessing the global variables will decrease greatly.

Figure 2 shows the architecture. All the patterns are already stored in a text file. These patterns are passed as an input to the program which implements AC algorithm. The program takes these patterns one line at a time and creates a FA and applies compaction on it. After that, the larger string, which should be searched into, is passed as an input to the same program. The program extracts words in parallel (as many threads are assigned through the program) and performs string matching. This explains the total working and architecture of the proposed work.

4 Results

Evaluation of the technique proposed in this paper has been done by checking it on a pattern set which contains about 50,000 different patterns. In the experiment, about 50,000 plus strings to be matched are taken and the response time is obtained. Here, the response time is the addition of time required for construction of FA and time for traversal of FA for finding the matches. The response time required is very less and satisfies the real-time requirements. In comparison with its serial counterpart, the response time reduces effectively. Suppose there are n threads which have been assigned to the system to perform matching, then the decrease in response time of the system is multiple of n.

It is proved fact that the smaller the memory size, higher is the speed of the system. When we perform compaction of FA, the FA could be stored in more compact memory than the amount of memory needed when it was not compacted. Hence, we could conclude that the speed of the system increases significantly. Apart from this, as we are performing task of matching in parallel, the response

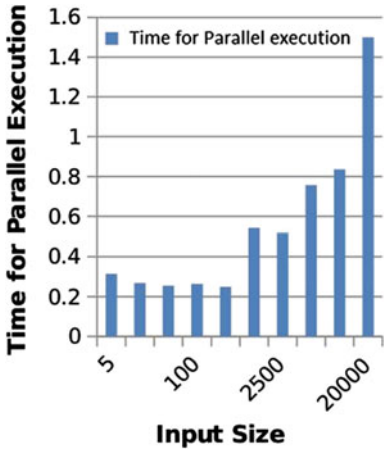

Fig. 3 Graph representing response time of parallel execution

time would decrease further. There is no overhead involved in matching the string in parallel; as a result, the response time decreases without involvement of inner costs. There is no pipelining involved nor there is any stalling of the process of parallelization.

For parallel processing, Fig. 3 shows us the results on string matching. For serial processing, each 100th increase in the input size matters and the response time increases additively for such increase in input size. Figure 4 shows us the results of serial processing on string matching.

Figure 5 Shows Graph representing comparison in response time of serial and parallel execution. Below figure shows the comparison between the parallel

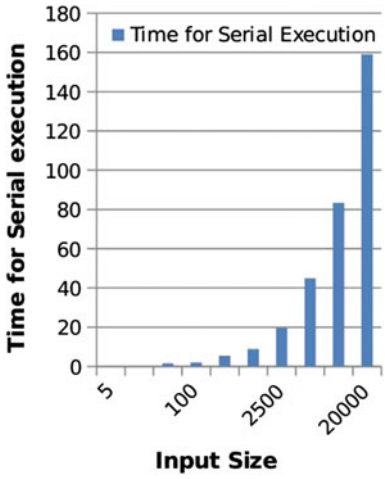

Fig. 4 Graph representing response time of serial execution

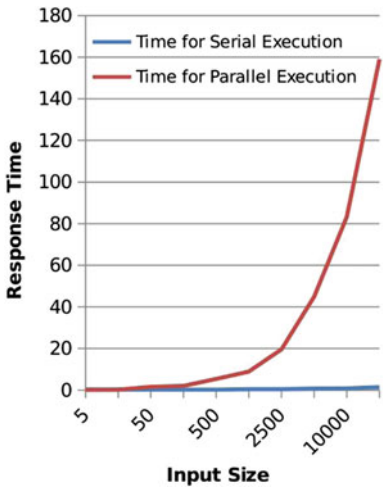

Fig. 5 Graph representing comparison in response time of serial and parallel execution

algorithm and its serial counterpart. For small inputs, serial processing takes less time as compared to parallel. This is because of the overhead involved in breaking up the input data set in subsets so that one thread is assigned to each subset. Also, overhead involved in assigning thread to each subset is countable. As the input size increases, the disparity between the response times required for serial and parallel processing increases. For the input set of 20,000, the disparity is highest. Serial processing requires 159.099 s as against 1.4994 s of parallel processing.

5 Conclusion

This paper shows how parallelization and compaction of DFA helps in reducing the response time. Parallelization is obtained without any stalling or any requirement to wait until some other activity gets completed. This helps us to obtain an improvement in response time. How the use of parallelization helps in improvement of string matching algorithms efficiency has been illustrated in this paper. For processing a pattern set as large as 50,000, 100,000, only 3.67 s is taken. Compaction leads to further reduction in time. Lesser the size of memory, more it is fast. Hence, reducing the rules for string matching process also leads to improved results. The approach proposed could be used in all the applications which cannot bear the highly costly TCAM chips, and the application which require power saving facilities could also go with this approach.

References

1. V. Aho and Margaret J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, communications of the ACM June 1975, Volume 18 Number 6.
2. Anat Bremler-Barr, Member, IEEE, David Hay, Member, IEEE, and Yaron Koral, Student Member, IEEE, CompactDFA: Scalable Pattern Matching Using Longest Pre x Match Solutions, IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 22, NO. 2, APRIL 2014.
3. Sang Kyun Yun Dept. of Comput. Telecommun. Eng., Yonsei Univ., Wonju, South Korea, An Efficient TCAM-Based Implementation of Multipattern Matching using Covered State Encoding, IEEE TRANSACTIONS ON COMPUTERS, VOL. 61, NO. 2, FEBRUARY 2012.
4. M. Becchi and P. Crowley, An improved algorithm to accelerate regular expression evaluation, in Proc. ACM/IEEE ANCS, 2007, pp. 145–154.
5. A. Bremler-Barr, D. Hay, and Y. Koral, CompactDFA: Generic state machine compression for scalable pattern matching, in Proc. IEEE INFOCOM, 2010, pp. 657–667.
6. Tzu-Fang Sheu Inst. of Commun. Eng., Nat. Tsing-Hua Univ., Hsinchu Nen-Fu Huang; Hsiao-Ping Lee, A Time- and Memory- Efficient String Matching Algorithm for Intrusion Detection Systems, Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE.
7. S. Kumar, J. Turner, P. Crowley, and M. Mitzenmacher, HEXA: Compact data structures for faster packet processing, in Proc. IEEE ICNP, 2007, pp. 246–255.
8. R. Sidhu and V.K. Prasanna, Fast Regular Expression Matching Using FP-GAs, Proc. Ninth Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM 01), pp. 227–238, 2001.
9. Y. Weinsberg, S. Tzur-David, D. Dolev, and T. Anker, High performance string matching algorithm for a network intrusion prevention system (NIPS), in Proc. IEEE HPSR, 2006, pp. 147–153.
10. J. van Lunteren, High-performance pattern-matching for intrusion detection, in Proc. IEEE INFOCOM, Apr. 2006, pp. 113.
11. Cheng-Hung Lin, Member, IEEE, Chen-Hsiung Liu, Lung-Sheng Chien, and Shih-Chieh Chang, Member, IEEE, Accelerating Pattern Matching Using a Novel Parallel Algorithm on GPUs, IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 10, OCTOBER 2013.