MA 451

Parallel Computing

Department of Mathematics

Indian Institute of Technology, Guwahati

Deterministic Finite Automata Membership Test in SIMT Environment

Dhruv Kohli

(Roll No. 120123054)

supervised by

Prof. Kalpesh Kapoor

March 16, 2015

1 Introduction

A Deterministic Finite Automaton (DFA) is defined as $D = (Q, q_0, \sigma, \delta, F)$ where

- Q = Finite set of states
- q_0 = Initial state
- σ = Finite set of symbols
- δ = Transition function
- F = Set of final states (accepting states)

A string S is said to be a member of the language generated by the DFA D if the state reached after simulating S in D lies in the set F i.e. the state reached is a final (also known as accepting) state. The task of checking whether S is a member of the language generated by D is called *membership test*. The sequential version of the algorithm for the membership test involves simulating the DFA D with the given input string. However, an efficient parallel algorithm for the membership test is not intuitive and is explained in next section.

Briefly, section 2 discusses about the parallel algorithm for DFA membership test and the time complexity of the sequential and parallel algorithm, section 3 states the experimental results with section 4 concluding our work.

2 Algorithm and Time Complexity Analysis

Given a DFA D (defined as in section 1), let us denote the number of states by M, the starting state by 1, the set of states by $\{1, 2, ..., M\}$, the number of available processors by P and the length of the input string S by N. A straightforward parallel algorithm A_0 for the membership test involves the following steps:

1. Division of input string S into P chunks of equal size (say, c_i, $i \in \{1, 2, \ldots, P\}$).
2. Simulation of D on all chunks with each state as starting state and computation of the ending states i.e. $e(i, j)$ which represents the state reached on simulating D on chunk c_j with i as starting state.
3. Computation of $E = e(...e(e(e(e(1, 1), 2), 3), 4), ..., P)$ to get the overall ending state which is attained when D is simulated on S with 1 as the starting state.

However, easy to implement, the above approach is not efficient in terms of the time complexity. The partition of the input string into P equal chunks where each processor operates on one of the chunk results in load imbalance among the processors in step 2 of the algorithm. This is because the first processor which is *aware* of the *true* starting state does M times lesser work than all other processors. Therefore, the worst-case time complexity of the algorithm is decided by a processor other than p_1. More specifically, p_1 is required to iterate over c_1 which is of size $\frac{N}{P}$ only once, while the other processors p_i, $i \in \{2, \ldots, P\}$ are required to iterate over chunks

$c_i,\ i \in \{2, \dots, P\}$, M times where M is the number of states. Therefore, the work done by p_1 is proportional to $\frac{N}{P}$ while the work done by $p_i,\ i \neq 1$ is proportional to $\frac{MN}{P}$ and the total work done by the algorithm A_0 is $O(1)$ in step 1, $O(\frac{MN}{P})$ in step 2 and $O(\log(P))$ in step 3 (using reduction operation in parallel). Thus, the algorithm A_0 has a time complexity of $O(\frac{MN}{P})$. However, we can achieve a lower time complexity using algorithm A_1 as described below.

Instead of partitioning the input string S into P equal chunks, algorithm A_1 partitions S into chunks $c_i,\ i \in \{1, 2, \dots, P\}$ of lengths $L_i,\ i \in \{1, 2, \dots, P\}$ where,

$$L_i = \frac{L_1}{M}, \forall i \in \{2, 3, ..., P\} \tag{1}$$

$$\sum_{i=1}^{P} L_i = N \tag{2}$$

The solution of above equation is:

$$L_i = \begin{cases} \frac{MN}{M+P-1}, & i = 1 \\ L_i = \frac{L_1}{M}, & i \neq 1 \end{cases} \tag{3}$$

The algorithm A_1 performs the partitioning of the input string S into chunks $c_i,\ i \in \{1, 2, \dots, P\}$ of lengths $L_i,\ i \in \{1, 2, \dots, P\}$ where L_i is given by equations 3. Then, A_1 executes the same steps 2 and 3 of the algorithm A_0. This specialized partitioning of the input string ensures that the load is balanced among the processors while executing step 2, thus achieving better time complexity. In step 2, the processor p_1 does work proportional to $O(L_1) = O(\frac{MN}{M+P-1})$ while the processor $p_i,\ i \neq 1$ does work proportional to $O(M\frac{L_1}{M}) = O(L_1) = O(\frac{MN}{M+P-1})$. Therefore, the total work done by algorithm A_1 is $O(1)$ in step 1, $O(\frac{MN}{M+P-1})$ in step 2 and $O(\log(P))$ in step 3. Thus, the algorithm A_1 has a time complexity of $O(\frac{MN}{M+P-1})$ which is better than that of A_0. The speed up obtained by algorithm A_1 compared to the sequential algorithm is $O(1 + \frac{(P-1)}{M})$. Table 1 summarizes the time complexities of the sequential and the two parallel algorithms discussed.

Algorithm	**Time Complexity**
Sequential	$O(N)$
Parallel A_0	$O(\frac{MN}{P})$
Parallel A_1	$O(\frac{MN}{M+P-1})$

Table 1: **Time complexities of different algorithms for DFA Membership Test**. N is the length of the input string, M is the number of states and P is the number of available processors.

A more formal pseudocode of the step 2 of the algorithm A_1 is described in [1]. My implementation of the algorithm A_1 in CUDA is available at this link.

3 Experiments and Results

We perform two categories of experiments - Measuring running time of sequential algorithm and parallel A_1 algorithm as the input string length N and number of states M of DFA are varied and measuring the running time of parallel algorithm A_1 as the input string length N and number of processors P are varied. Although a thread is different from a process, we consider the number of threads launched for the execution of the kernel in our CUDA program to be a substitute for the number of available processors. Also, note that these experiments are performed on a Tesla K40.

Figrure 1 shows the plot of running time of sequential and parallel A_1 algorithm as a function of input string length N and number of states M of DFA. Note that the running time of sequential algorithm is greater than that of the parallel algorithm A_1 when the input length is greater than approximately $2.5e + 07$ characters. For input strings of length lesser than $2.5e + 07$ characters, the parallel algorithm has greater running time due to the overhead caused by launching threads.

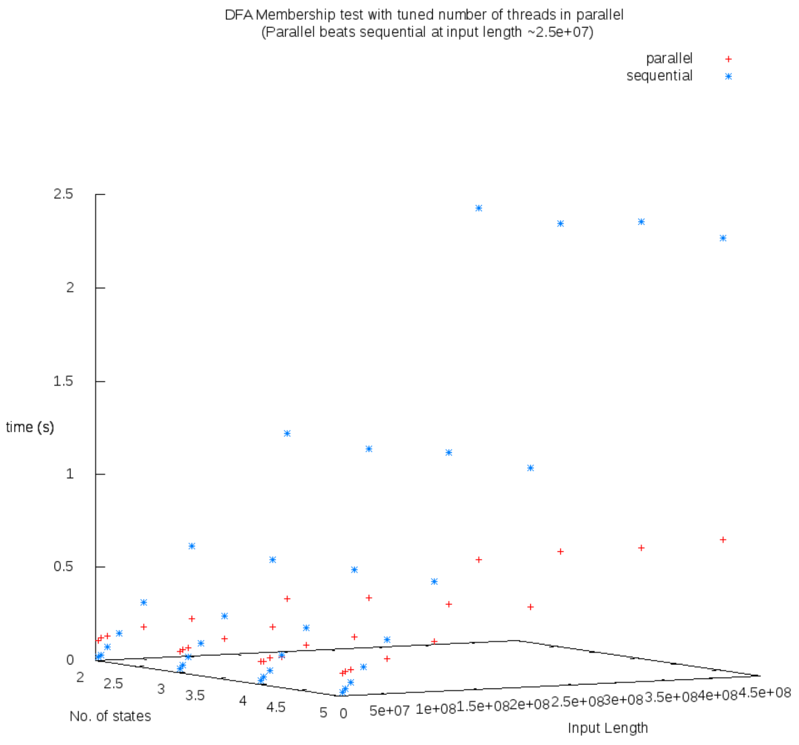

Figure 1: The running time of sequential and parallel A_1 algorithm as a function of input string length N and number of states M of DFA.

Figure 2 shows the plot of running time of parallel A_1 algorithm as a function of input string length N with varied number of states M. Note that the running time of the parallel algorithm A_1 increases as the number of states increases which is consistent with the derived time complexity of $O(\frac{MN}{M+P-1})$ of the parallel algorithm A_1.

Figure 3 shows the plot of running time of parallel A_1 algorithm as a function of input string length N and number of threads P. As expected, the running time of the parallel algorithm decreases as the number of threads increases.

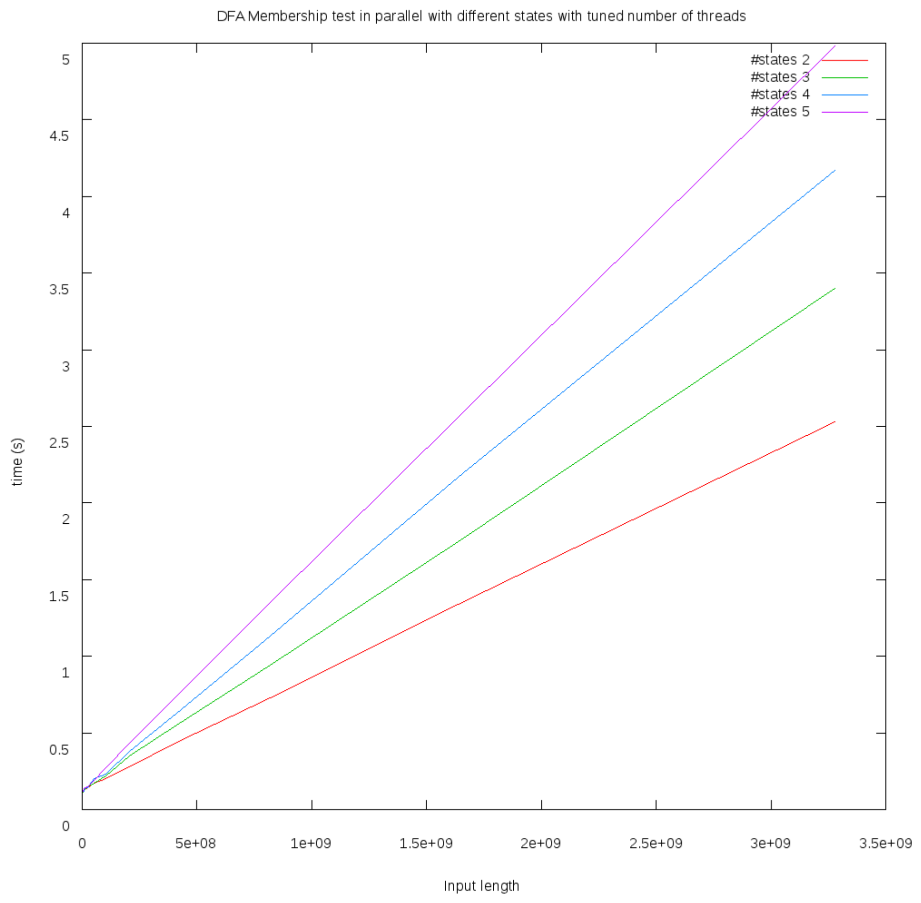

Figure 2: The running time of parallel A_1 algorithm as a function of input string length N with varied number of states M.

4 Conclusion

We, theoretically, showed that a parallel algorithm with a better time complexity can be obtained by a specialized partitioning of the input string so as to balance load among the processers and therefore, minimizing latency. We showed empirical results which are consistent with the time complexity of parallel algorithm A_1 derived theoretically. We also empirically showed that, on a Tesla K40, for input strings of length approximately greater than approximately $2.5e + 07$ characters, our implementation of the parallel algorithm A_1 has lower running time than sequential algorithm.

References

[1] Yousun Ko, Minyoung Jung, Yo-Sub Han, and Bernd Burgstaller. A speculative parallel dfa membership test for multicore, simd and cloud computing environments. *International Journal of Parallel Programming*, 42(3):456–489, 2014.

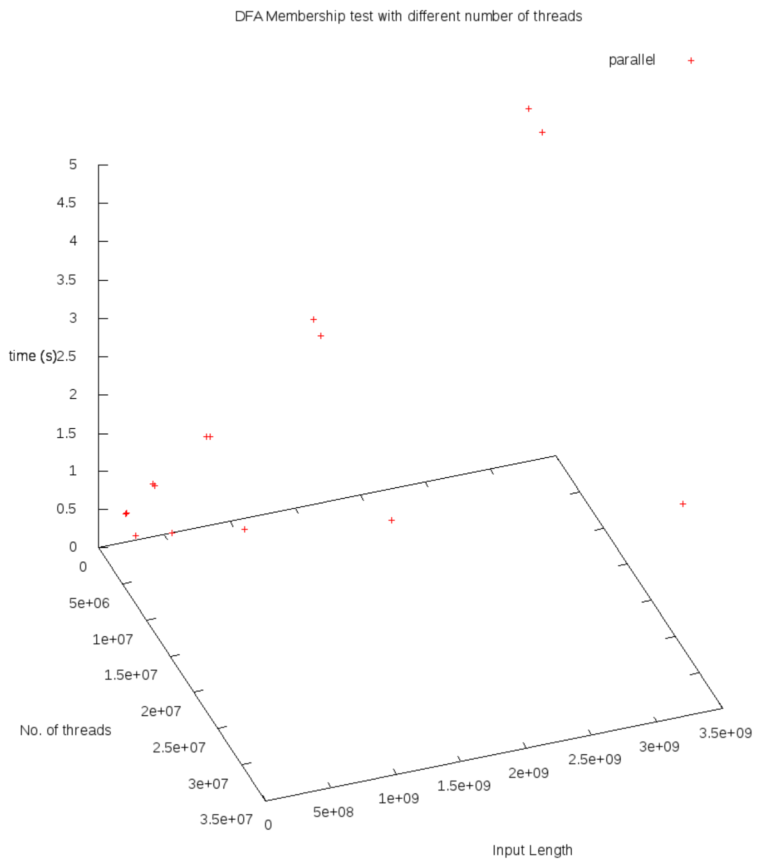

Figure 3: The running time of parallel A_1 algorithm as a function of input string length N and number of threads P.