DFA Membership Test in SIMT Environment
(Time complexity analysis only)

Dhruv Kohli
B.Tech Mathematics & Computing
Pre-final year
120123054

INTRODUCTION

Given a DFA D=(Q, q0, sigma, delta, F) where
- Q = finite set of states
- q0 = starting state
- sigma = finite set of input symbols
- delta = 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 the DFA D with S is an accepting/final state.

The sequential version of the algorithm for the membership test involves simulating the DFA with the given input string.

The parallelized version is a bit complex and is described in the next section.

DFA MEMBERSHIP TEST IN PARALLEL

Given a DFA D, let us denote the number of states by M, the starting state by 1, the number of processors available by P and the length of the input string S by N.

Following is a straightforward approach:
1) Dividing the given input string into P number of equal chunks {c(i), i=1,..p}
2) Simulating the DFA D with chunk c(i) with all the states as starting state (NOTE: We are required to simulate DFA D with state 1 only for chunk c(1))
3) Extracting the ending states for each state as starting state and for each chunk i.e. {e(i,j); where e(i,j) is the state reached on simulating the DFA D with starting state i on chunk c(j)}
4) Finally, since the actual starting state is 1, we find out
E = e(... e(e(e(e(1, 1), 2), 3), 4), ... p) to get the overall final state reached on simulating the DFA D with starting state 1 on input string S.

Note that step 4 can be performed using reduction in O(lg(P)) time in parallel.

But the above approach is not efficient in the sense that it results in load imbalance among the processors because the 1st processor does M times less work then all other processors and hence the overall time complexity is decided by the time taken by a processor othan than 1st processor which comes out to be **O(1) (step 1) + O(M*N/P) (step 2 and 3) + O(log()) (step 4).**

BETTER APPROACH:
Instead of equally partitioning the input string S into P equal chunks, we partition S into c(1), c(2), ... c(P) with lengths L(1), ... L(P) where
L(i) = L(1)/M i ~= 1 and
L(1)+L(2)+....+L(P)=N

solving above equations we get:

$$L(i) = \begin{cases} (M*N)/(M+P-1) & \text{if } i == 0 \\ L(1)/M & \text{if } i \sim= 0 \end{cases}$$

Using above lengths of chunk and step 2,3 and 4 of above mentioned approach, we'll be able to achieve load balancing and more efficiency.

DFA MEMBERSHIP TEST IN PARALLEL (ALGORITHM)

```
Basic speculative DFA matching
Input : δ, Q, Σ, P, Str = c(0)c(1) . . . c(P-1)
Output: vector L(i) for each chunk c(i)
1 for i ←0 to |P| – 1 do in parallel
2     for j ←0 to |Q| – 1 do
3          L(i)[j] ←j ; // initialize vector Li
4     Start ←StartPos(c(i))
5     End ←EndPos(c(i))
6     if i = 0 then // chunk c0
7          for k ←Start to End do
8               L(0)[0] ←δ(L(0)[0], Str[k])
9     else // chunks c(1) . . . c(P–1)
10         foreach j ∈Q do
11              for k ←Start to End do
12                   L(i)[j] ←δ(L(i)[j], Str[k])
```

SOURCE: A Speculative Parallel DFA Membership Test for Multicore, SIMD and Cloud Computing Environments;Yousun Ko · Minyoung Jung, Yo-Sub Han, Bernd Burgstaller.

Here,

$$\text{StartPos}(c(k)) = \begin{cases} 0 & \text{for } k = 0 \\ \text{floor}(L(0)+(1/M)*(k-1)*L(0)) & \text{otherwise} \end{cases}$$

$$\text{EndPos}(c(k)) = \begin{cases} N-1 & \text{for } k = P-1 \\ \text{floor}(L(0)+(1/M)*(k-1)*L(0))-1 & \text{otherwise} \end{cases}$$

TIME COMPLEXITY

Sequential algorithm takes O(N) time.
Parallel algorithm takes (O((M*N)/(M+P-1)) + O(lg(P))) ~ O((M*N)/(M+P-1)) time.

Speedup = O(1+(P-1)/M)

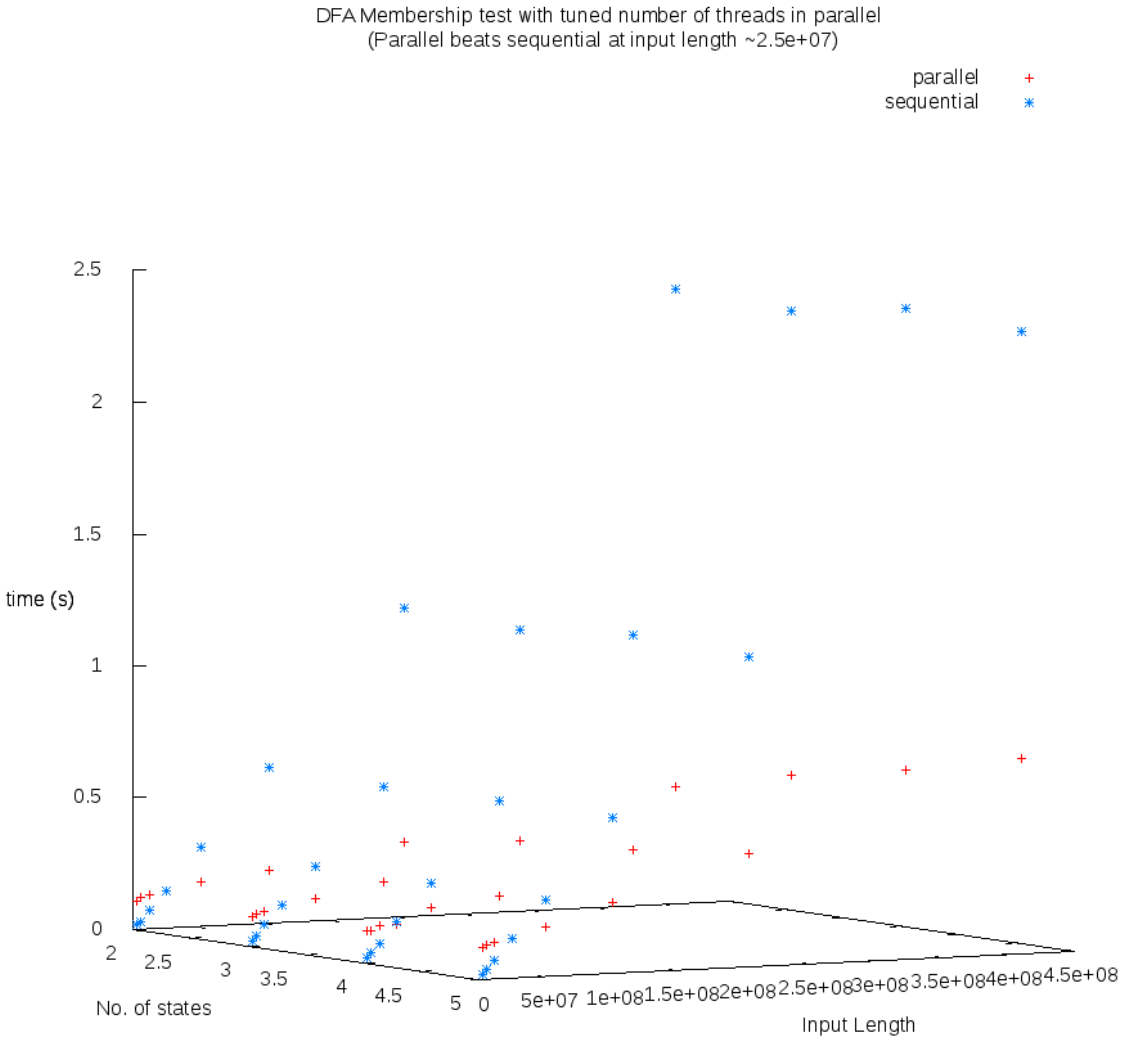
Here M=#states, N=Input length, P=#processors.

Code

```
__global__ void specDFAMatching(ull M, ull N, ull len, ull *delta, char *input, ull *fStates, ull q0, ull maxThreads) {
    ull idx=threadIdx.x+blockIdx.x*blockDim.x;
    if(idx>=maxThreads)
        return;
    ull i, j;
    if(idx==0) {
        ull fst=q0;
        ull l=0;
        ull r=(M)*(len/(M+maxThreads-1))-1;
        for(i=l; i<=r; ++i) {
            fst=delta[(ull)(input[i]-'0')+fst*N];
        }
        fStates[q0+idx*M]=fst;
    } else {
        ull l=(M)*(len/(M+maxThreads-1)) + (idx-1)*(len/(M+maxThreads-1));
        ull r=l+(len/(M+maxThreads-1))-1;
        for(j=0; j<M; ++j) {
            ull fst=j;
            for(i=l; i<=r; ++i) {
                fst=delta[(ull)(input[i]-'0')+fst*N];
            }
            fStates[j+idx*M]=fst;
        }
    }
}

__global__ void finalStateUsingRedn1(ull M, ull *fStates, ull q0, ull maxThreads, ull s) {
    ull idx=threadIdx.x+blockIdx.x*blockDim.x;
    if(idx+s>=maxThreads)
        return;
    if(idx%(2*s)==0) {
        for(ull i=0; i<M; ++i) {
            ull x=fStates[i+idx*M];
            fStates[i+idx*M]=fStates[x+(idx+s)*M];
        }
    }
}
```

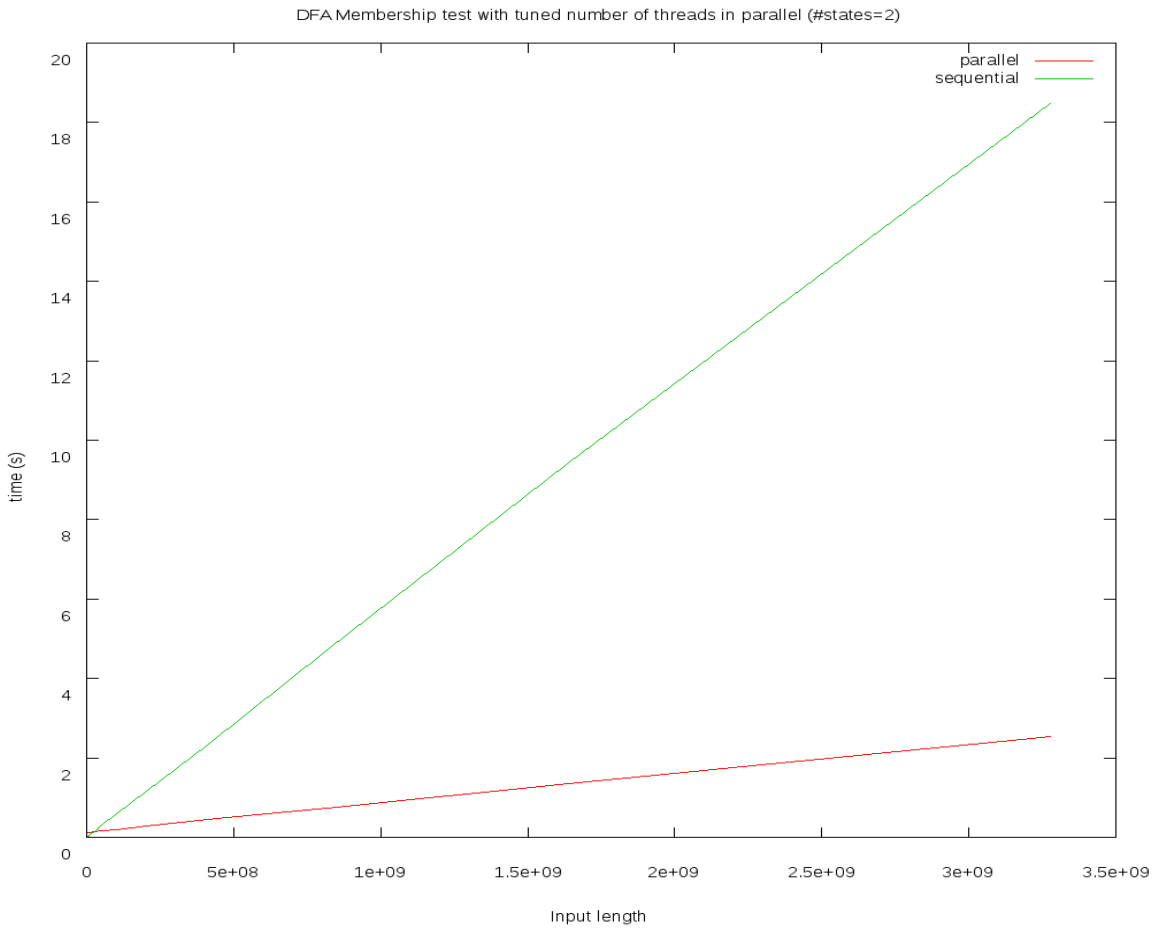
Experimental observations



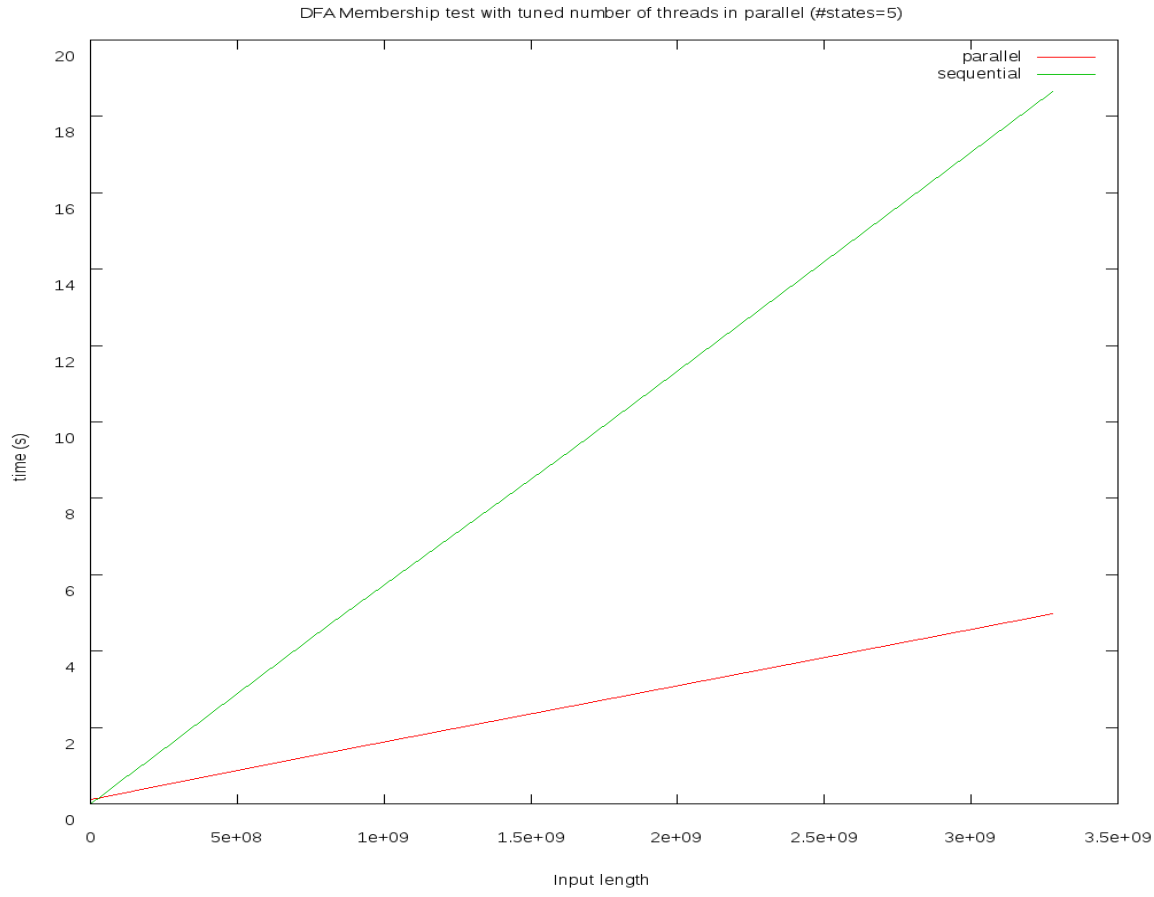
At an input length of ~2.56e+07 parallel version of DFA membership test (in CUDA) beats the sequential version. Also, as the number of states increases the exectution time of parallel version also increases. This will be more clear from next three plots where in 1st plot we plot seq vs parallel with #states = 2 and in 2nd with #states = 5 and then in 3rd we plot parallel execution time with different number of states. It can be seen that parallel execution time increases with # of states.

Note that the time complexity contains M(#states) in both numerator and denominator but N(input length) is too high making delta(M)*N >> delta(M)+P-1

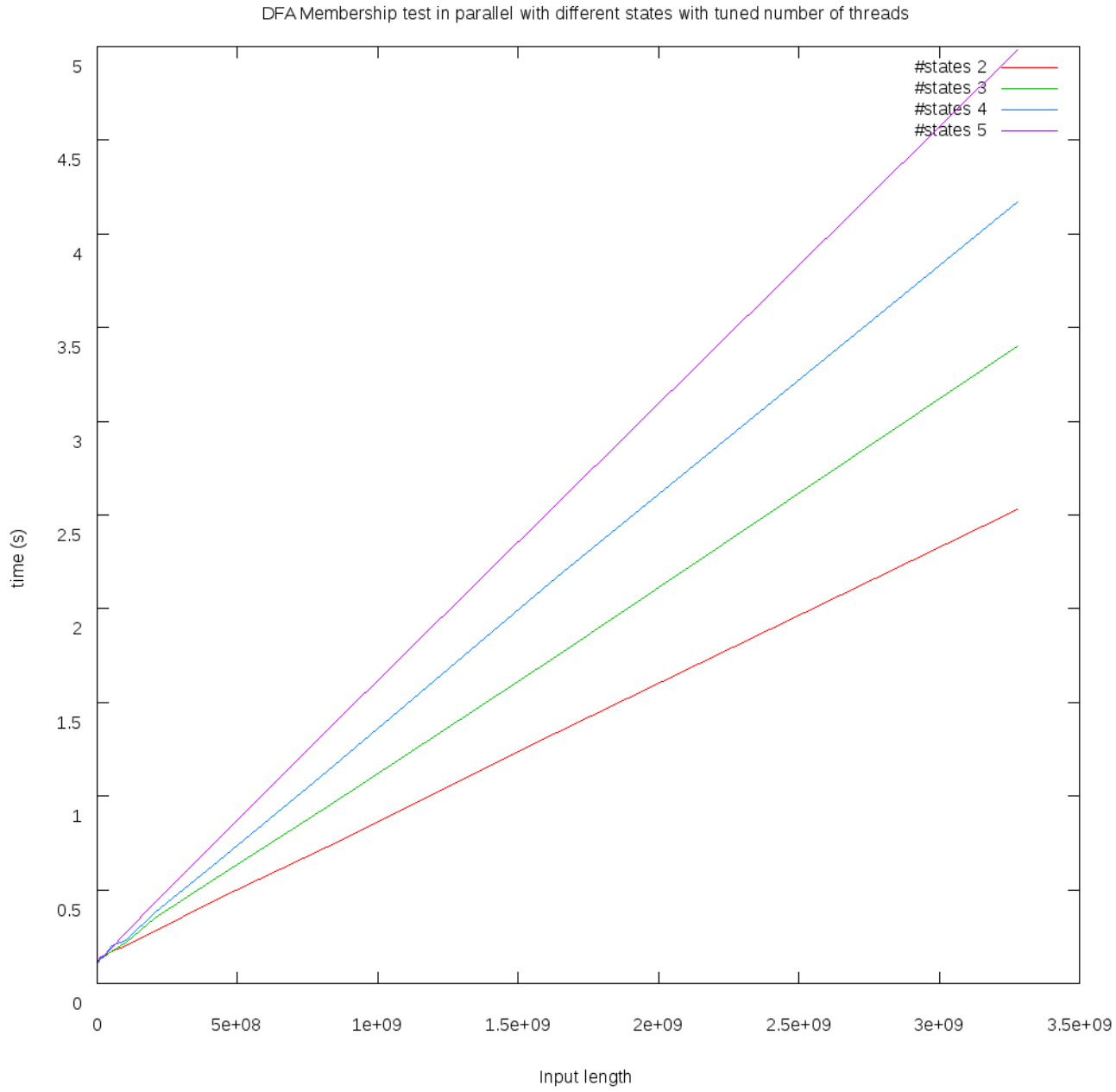
-#states = 2



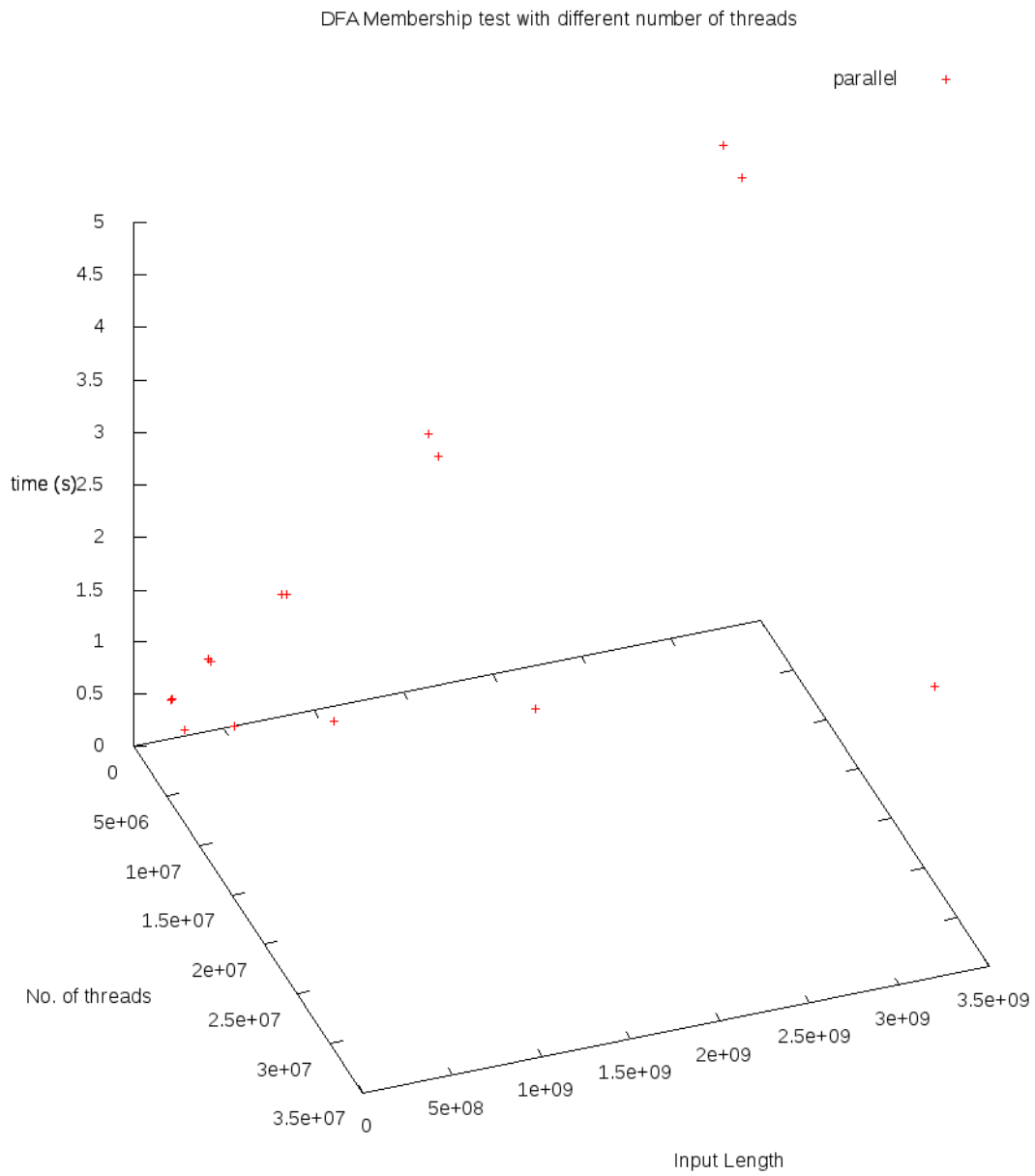
-#states = 5



-Varying number of states



-Varying number of threads: With same imput lenght and increased number of threads, parallel execution time gets reduced as expected (time complexity is inversely proportional to #processors)



Conclusion

-Our analysis consistent with the theoretical time complexity of both parallel and sequential version of DFA Membership test.

-We observed that in practice the parallel version beats sequential version when the input length goes above ~2.56e+07 characters.