Parallel Computing Lab Assignment

DFA Membership Test in CUDA

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)) in parallel.

But the above approach is not efficient in the sense that it results in load imbalance among the processors because the 1^{st} 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 1^{st} 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) =	(M*N)/(M+P-1)	if i == 0
	L(1)/M	if i ~= 0

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)
for i ← 0 to |P| – 1 do in parallel
    for j ← 0 to |Q| – 1 do
        L(i)[j] ← j ; // initialize vector Li
    Start ← StartPos(c(i))
    End ← EndPos(c(i))
    if i = 0 then // chunk c0
        for k ← Start to End do
            L(0)[0] ← δ(L(0)[0], Str[k])
    else // chunks c(1) . . . c(P–1)
        foreach j ∈ Q do
            for k ← Start to End do
                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,

```
                   0                                   for k = 0
StartPos(c(k)) =
                   floor(L(0)+(1/M)*(k-1)*L(0))        otherwise

                   N-1                                 for k = P-1
EndPos(c(k))   =
                   floor(L(0)+(1/M)*(k-1)*L(0))-1      otherwise
```

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)

CODE

Kernel code

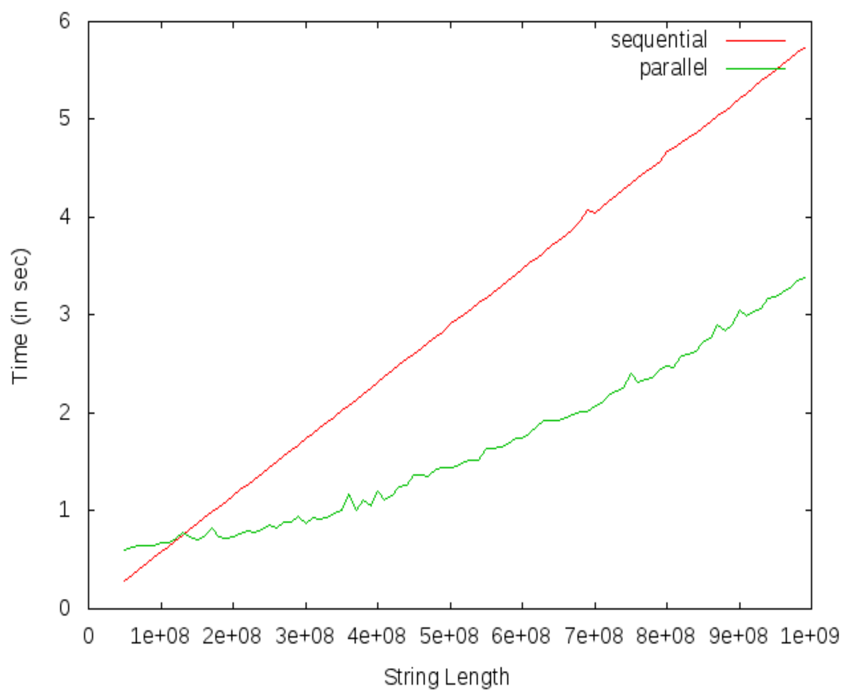
```
__global__ void specDFAMatching(int *delta, int *input, int *fStates, int q0, int len, int m, int n) {
    long long int idx = threadIdx.x + blockIdx.x*blockDim.x;
    long long int totalThreads = blockDim.x*gridDim.x;

    for(int i = 0; i < m; ++i) {
        fStates[i + idx*m] = i;
    }
    double L0 = (1.0*m*len)/(m+totalThreads-1);
    long long int start_=0, end_=len;
    if(idx!=0)
        start_ = (L0 + ((idx-1.0)*L0)/m);
    end_ = (L0 + (1.0*idx*L0)/m);
    if(end_ > len) {
        end_ = len;
    }

    if(start_ > end_ || end_ < 0 || start_ < 0)
        return;
    if(idx == 0) {
        for(long long int i = start_; i < end_; ++i) {
            fStates[idx*m] = delta[input[i] + fStates[idx*m]*n];
        }
    } else {
        for(int i = 0; i < m; ++i) {
            for(long long int j = start_; j < end_; ++j) {
                fStates[i+idx*m] = delta[input[j] + fStates[i+idx*m]*n];
            }
        }
    }
}
```

EXPERIMENTAL ANALYSIS AND CONCLUSION

Following graph represents experimental analysis with DFA of 4 states, # threads launched per block = 1024, # blocks launched = 32*1024



Hence, with an input string of size 10^9 a speedup of approximately 2 times was observed with parallel algorithm.

NOTE: Sorry, I din't get time to implement the reduction operation in my code. I'll do that soon after the endsems.