Assignment 9

Inderpreet Singh Chera

Word acceptance by a finite state machine

Let S be an alphabet and L be a language defined over S. Suppose M is a minimal deterministic complete finite state machine that recognizes L. A parallel program was written to do the same using two types of reduction:

1. Trivial Reduction
2. Binary Reduction

System Settings

All tests were done on **Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz** processor. This computer is **dual core** where each core has **2 threads**. Also during each experiment it was insured that **no other applications** were running in the background, so that we don't have any biased readings.

Implementation

Implementation of the required problem was inspired by the paper "**Implementation of Deterministic Finite Automata on Parallel Computers**". Author assumes each processor has transition table, final states, initial state, part of string, and total number of processors running beforehand.

Now, the problem is that each processor doesn't know which is the starting state of the part of the input text they have got. So, each processor computes the final state for all the states in the DFA on the part of the input text they have. Now, we are left with just finding the correct final state for the given start state for the complete string. To do so, we have the following two types of reduction.

Trivial Reduction

Now, as each thread doesn't know the start state for the part of input text they got other than P_0 processor. So, we can just pass the initial state one by one to each processor from P_0. This is done by trivially iterating through the processes and updating the final state. Pseudocode for the same as given below. Here L is the vector storing the final state for each initial state for each processor. R was used in the paper for pattern matching. It can be ignored for our problem statement

Algorithm 3.8 (Sequential reduction for Algorithm 3.7)
Input: All variables and results of Algorithm 3.7 stored in shared memory and temporary variables $\mathcal{L}_{temp}$, $\mathcal{R}_{temp}$
Output: Reduced results stored in shared memory
Method: Only one processor performs this reduction, reads data from shared memory and stores result in variables $\mathcal{R}[0]$ and $\mathcal{L}[0]$

$\mathcal{L}_{temp} \leftarrow q_0$
$\mathcal{R}_{temp} \leftarrow 0$

for $k \leftarrow 0, 1, \ldots, |P| - 1$ **do**
  $\mathcal{R}_{temp} \leftarrow \mathcal{R}_{temp} + \mathcal{R}[k][\mathcal{L}_{temp}]$
  $\mathcal{L}_{temp} \leftarrow \mathcal{L}[k][\mathcal{L}_{temp}]$
endfor

$\mathcal{R}[0][q_0] \leftarrow \mathcal{R}_{temp}$
$\mathcal{L}[0][q_0] \leftarrow \mathcal{L}_{temp}$

Binary Reduction

Another way to do the reduction is to get the final state assuming each state as initial state and then finally for the given initial state get the final state. We can combine the final state from two processors P_i and P_j as:

$$\mathcal{L}_{P_iP_j} = \begin{bmatrix} \mathcal{L}_{P_j}[\mathcal{L}_{P_i}[0]] \\ \mathcal{L}_{P_j}[\mathcal{L}_{P_i}[1]] \\ \vdots \\ \mathcal{L}_{P_j}[\mathcal{L}_{P_i}[|Q|-1]] \end{bmatrix}.$$

Now, this combining can be done in log(P) steps to get the final states from all the states as the initial state on the whole input string. Pseudo Code for the same is as below. L, and R same as described above.

Algorithm 3.9 (Binary reduction for Algorithm 3.7)
Input: All variables and results of Algorithm 3.7 stored in shared memory and temporary variables $\mathcal{L}_{temp}, \mathcal{R}_{temp}$
Output: Reduced results stored in shared memory
Method: All processors perform this reduction, read data from shared memory, write partial results and store final result in variables $\mathcal{R}[0]$ and $\mathcal{L}[0]$

for all $P_0, P_1, \ldots, P_{|P|-1}$ **do in parallel**
  for $m \leftarrow 1, 2, \ldots, \lceil \log |P| \rceil$ **do**
    if $(P_i \mod 2^m) = 0$ **and** $(P_i + 2^{m-1}) < |P|$ **then**
      for $x \leftarrow 0 \ldots |Q| - 1$ **do**
        $\mathcal{R}[P_i][x] \leftarrow \mathcal{R}[P_i][x] + \mathcal{R}[P_i + 2^{m-1}][\mathcal{L}_{P_i}[x]]$
        $\mathcal{L}[P_i][x] \leftarrow \mathcal{L}[P_i + 2^{m-1}][\mathcal{L}[P_i][x]]$

      endfor
    endif

  endfor

endfor

Speedup

Trivial Reduction:

Time to run the algorithm with trivial reduction is:

$$T_p = O(\frac{|Q|*n}{|P|} + log\ |P| + |P|)$$

Where Q is number of states of automation M, n is the string length, P is number of processes

Binary Reduction:

Time to run the algorithm with binary reduction is:

$$T_p = O(\frac{|Q|*n}{|P|} + log\ |P| + |Q|log\ |P|)$$

Where Q is number of states of automation M, n is the string length, P is number of processes.

Hence Speedup (in both cases):

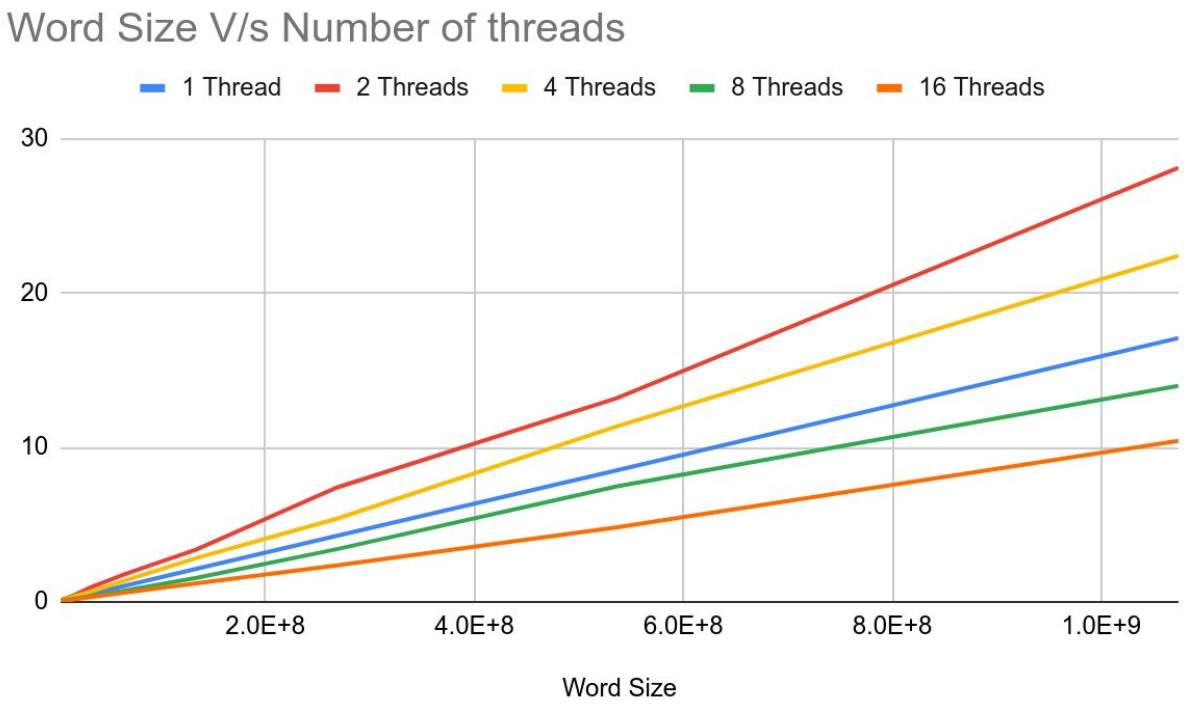
$$S\ (n, |P|) = O(\frac{n}{\frac{|Q|*n}{|P|}}) = O(\frac{|P|}{|Q|})$$

Observations

Time(Speedup) vs Number of Threads using Trivial Reduction with 2 state DFA:

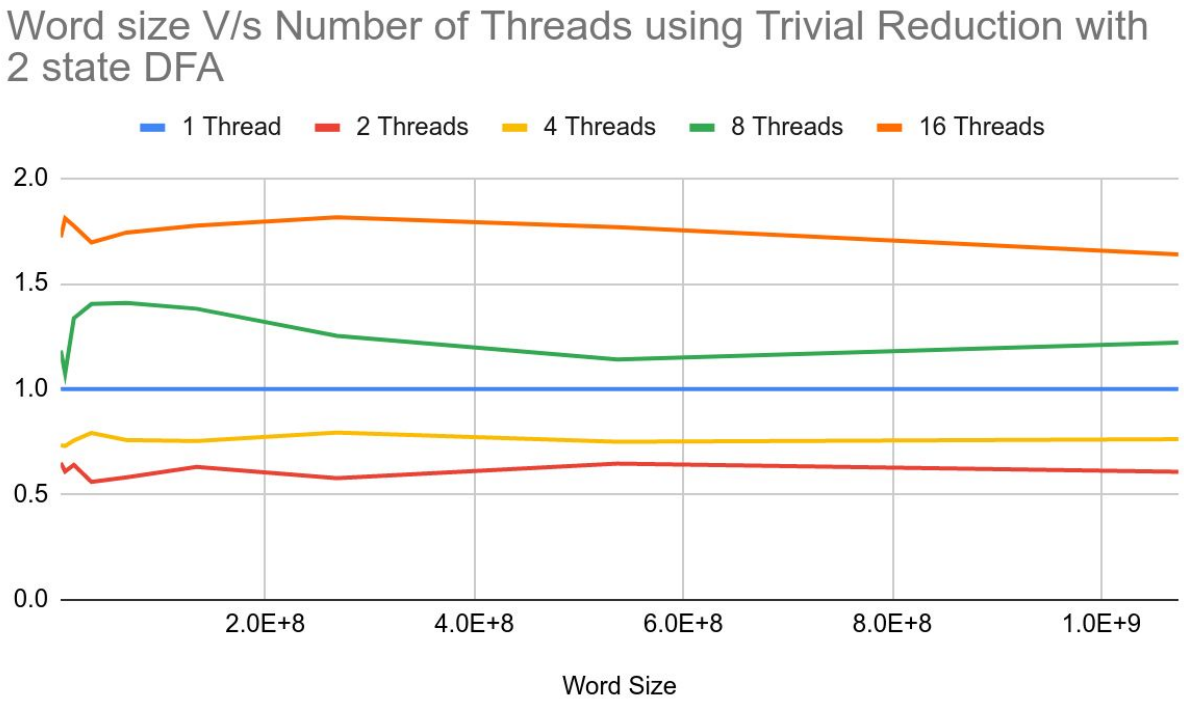
Time V/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0667342	0.102658	0.090998	0.0564075	0.0388016
8388608	0.133279	0.218787	0.182675	0.124177	0.0736345
16777216	0.269627	0.421098	0.356622	0.201836	0.151989
33554432	0.534183	0.954595	0.674769	0.380709	0.315029
67108864	1.06878	1.83932	1.41009	0.75874	0.613463
134217728	2.14476	3.40098	2.84678	1.55334	1.20848
268435456	4.27415	7.40514	5.38662	3.41426	2.35609
536870912	8.53386	13.2045	11.3803	7.4877	4.82608
1073741824	17.0792	28.1154	22.4097	13.9983	10.4248



SpeedUp V/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	1	0.650063317	0.7333589749	1.183073173	1.719882685
8388608	1	0.6091723914	0.7295962775	1.073298598	1.810007537
16777216	1	0.6402951332	0.7560582353	1.335871698	1.773990223
33554432	1	0.5595912403	0.7916531435	1.403126798	1.695662939
67108864	1	0.5810734402	0.7579516201	1.408624825	1.742207762
134217728	1	0.6306299949	0.7533985766	1.380740855	1.774758374
268435456	1	0.5771869269	0.7934753148	1.251852524	1.814086049
536870912	1	0.6462842213	0.7498800559	1.139717136	1.768279846
1073741824	1	0.6074677935	0.7621342544	1.220091011	1.638323997



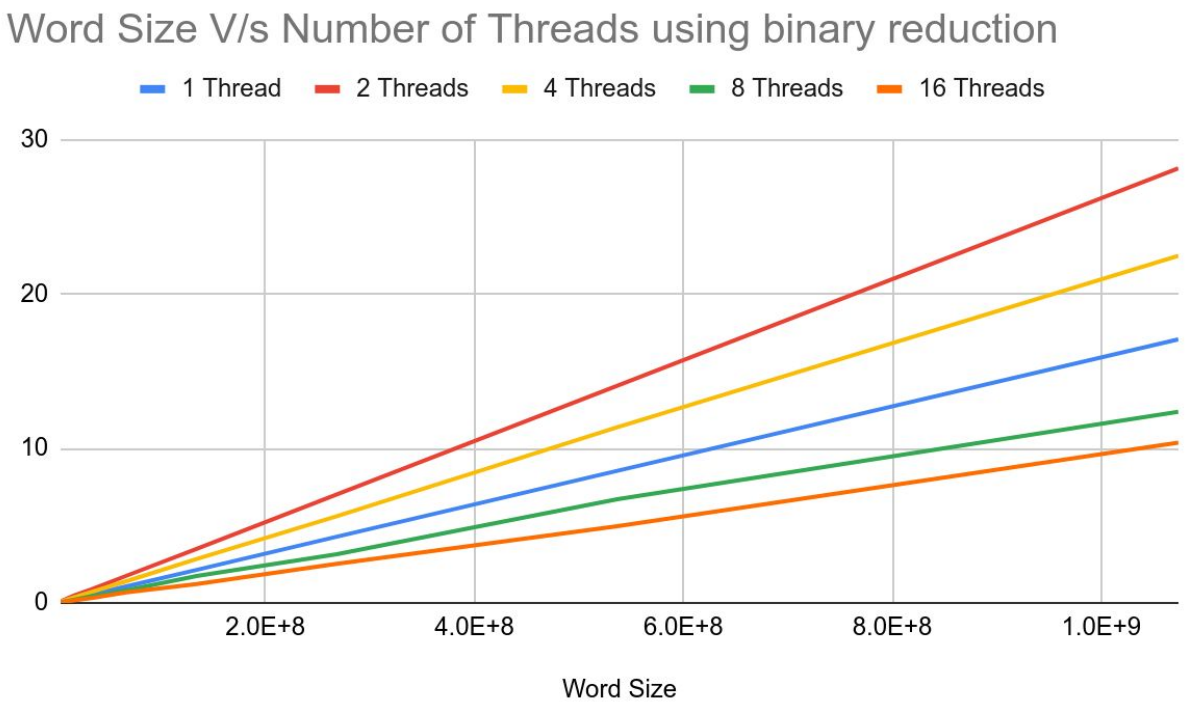
Observations:

The speedup curve is not as expected. Theoretically according to the speedup equation for the same number of states in DFA speedup should have increased, but for 2 and 4 number of threads, this doesn't happen. Although after that we see significant performance increase.

Time(Speedup) vs Number of Threads using Binary Reduction with 2 state DFA:

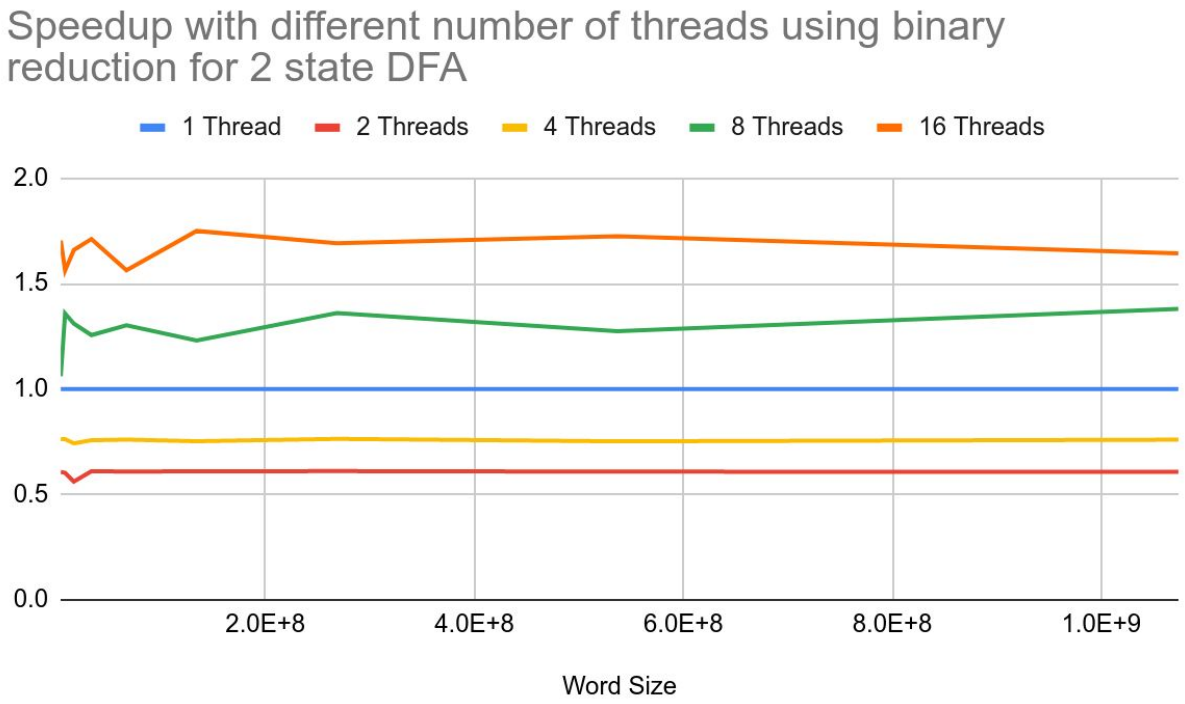
Time V/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0669148	0.110197	0.0877716	0.0630864	0.0392617
8388608	0.133272	0.221255	0.174909	0.0981263	0.0852892
16777216	0.267073	0.476018	0.359671	0.203855	0.160921
33554432	0.534042	0.875125	0.70533	0.425262	0.312008
67108864	1.06967	1.75742	1.40727	0.821666	0.684296
134217728	2.13666	3.50399	2.83929	1.73611	1.22103
268435456	4.28698	7.02055	5.61457	3.15164	2.53399
536870912	8.55543	14.065	11.3752	6.71609	4.96208
1073741824	17.0747	28.1407	22.4767	12.3742	10.3855



Speedup v/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	1	0.6072288719	0.762374162	1.060685029	1.704327627
8388608	1	0.6023457097	0.7619505	1.358167994	1.562589402
16777216	1	0.5610565147	0.7425480509	1.31011258	1.659652873
33554432	1	0.6102465362	0.7571519714	1.255795251	1.711629189
67108864	1	0.6086592846	0.7601028943	1.301830671	1.56316857
134217728	1	0.6097791375	0.7525332037	1.230716948	1.749883295
268435456	1	0.6106330701	0.7635455609	1.360237844	1.691790417
536870912	1	0.608277995	0.7521124903	1.27387066	1.724162045
1073741824	1	0.6067617366	0.759662228	1.379862941	1.644090318



Observations:

We face similar problems as in the trivial reduction case. For threads 2 and 4 speedup is less than 1 but after that it has improved significantly.

Also there is no significant difference in time on using binary or trivial reduction probably because the factor affecting the reduction is too less when only 2 states are used.

Results for 3 state DFA also followed similar trends as of 2 state DFA.

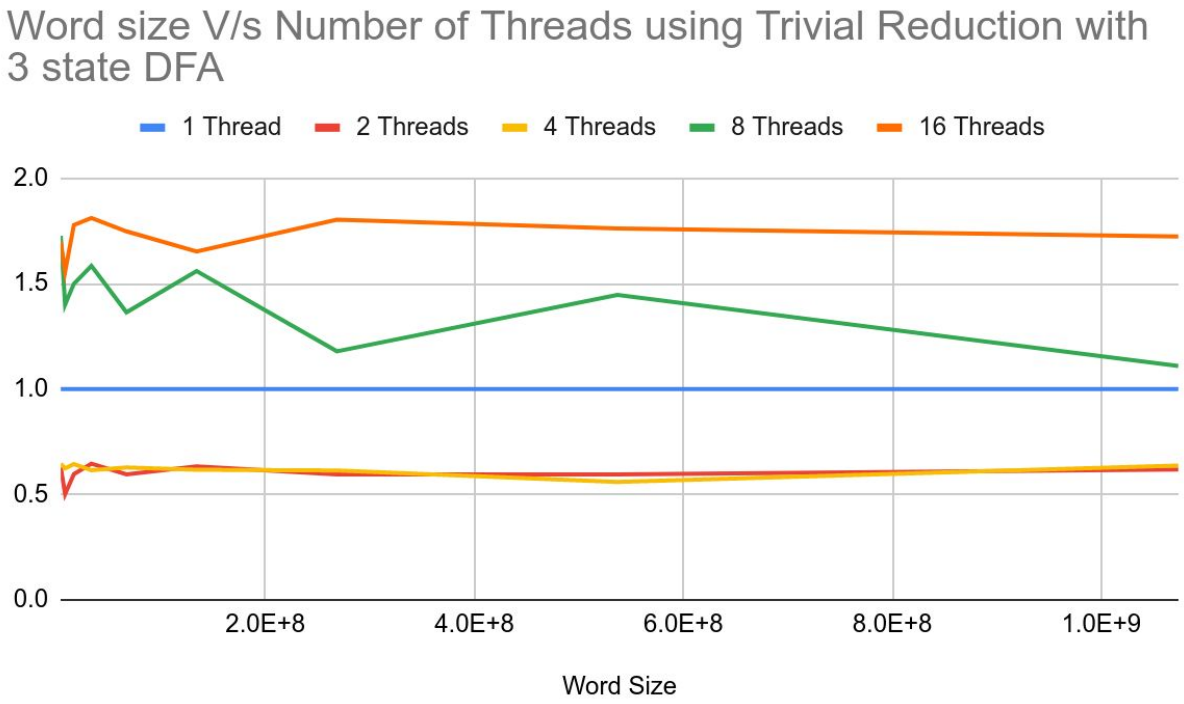
Using trivial reduction:

Time vs word Size:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0951585	0.151554	0.147417	0.055099	0.056004
8388608	0.190208	0.379032	0.304946	0.135802	0.122222
16777216	0.381379	0.639194	0.592206	0.254395	0.214505
33554432	0.766801	1.18654	1.24695	0.483966	0.423335
67108864	1.52584	2.56414	2.42965	1.11851	0.872975
134217728	3.05314	4.81849	4.9464	1.95701	1.84773

268435456	6.10252	10.2595	9.94302	5.17535	3.38452
536870912	12.1877	20.474	21.7893	8.42452	6.91881
1073741824	24.371	39.3971	38.2472	21.9614	14.1401

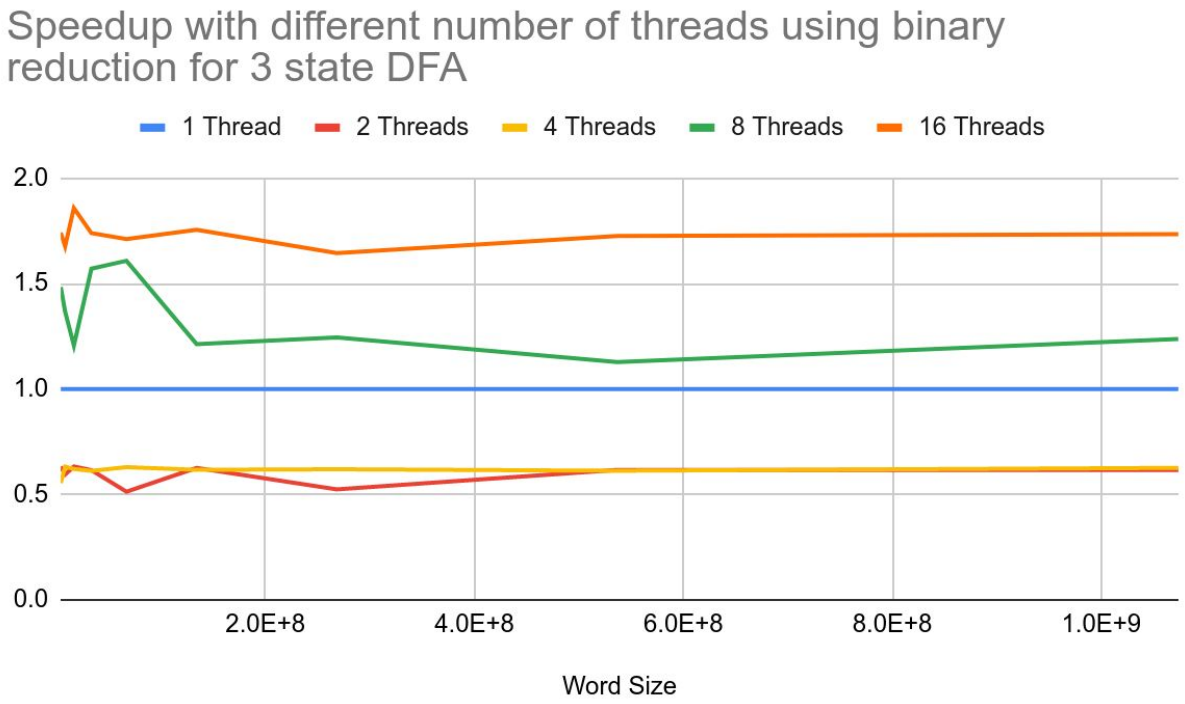
Speedup vs word size graph:



Using Binary reduction:

Time v/s Word Size:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0951045	0.150483	0.171628	0.0640809	0.0546069
8388608	0.190261	0.319899	0.301043	0.139113	0.113427
16777216	0.380873	0.602084	0.613455	0.315535	0.204989
33554432	0.76243	1.23924	1.24606	0.485308	0.438351
67108864	1.52881	2.97994	2.42847	0.950576	0.89327
134217728	3.04785	4.86998	4.94126	2.51196	1.73655
268435456	6.10102	11.6401	9.84731	4.89902	3.71005
536870912	12.1917	19.7765	19.9063	10.8117	7.06347
1073741824	24.388	39.5911	38.9483	19.7141	14.0624



Observations:

Same trends as two state DFA were found here too.