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 $(P_i \bmod 2^m) = 0$ **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:

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

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):

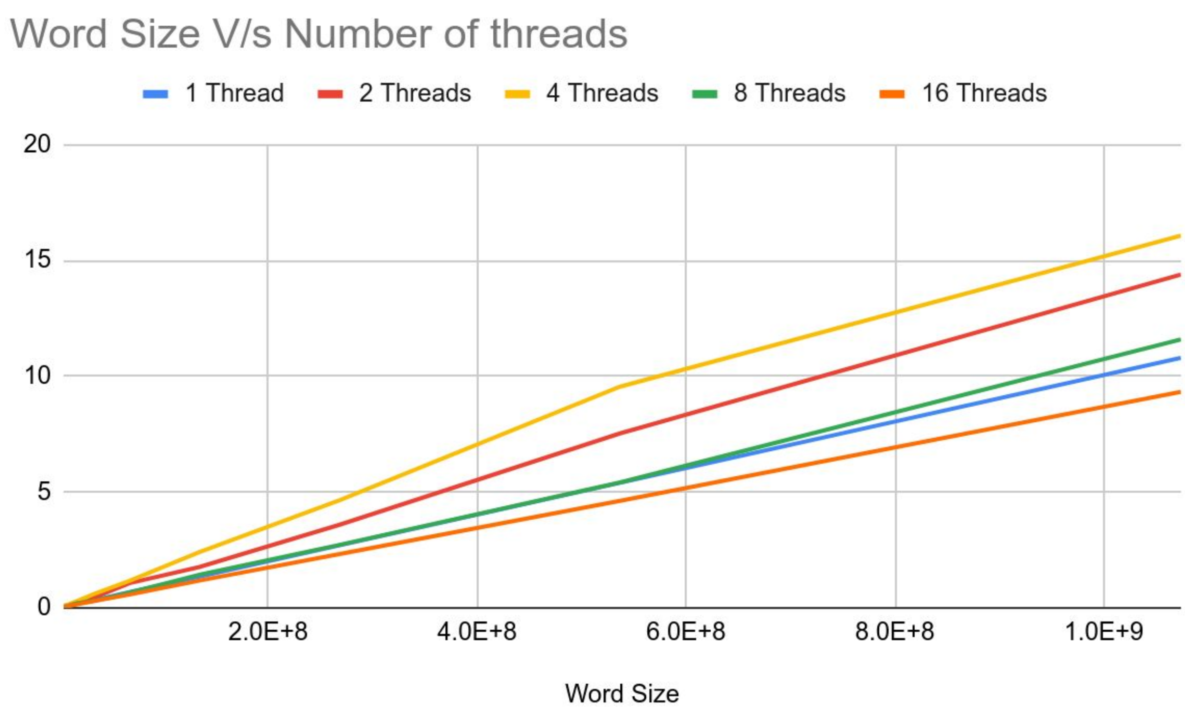
$$S\ (n, |P|) = O(\frac{n}{\frac{|Q|*n}{|P|}}) = O(\frac{|P|}{|Q|})$$

Observations

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

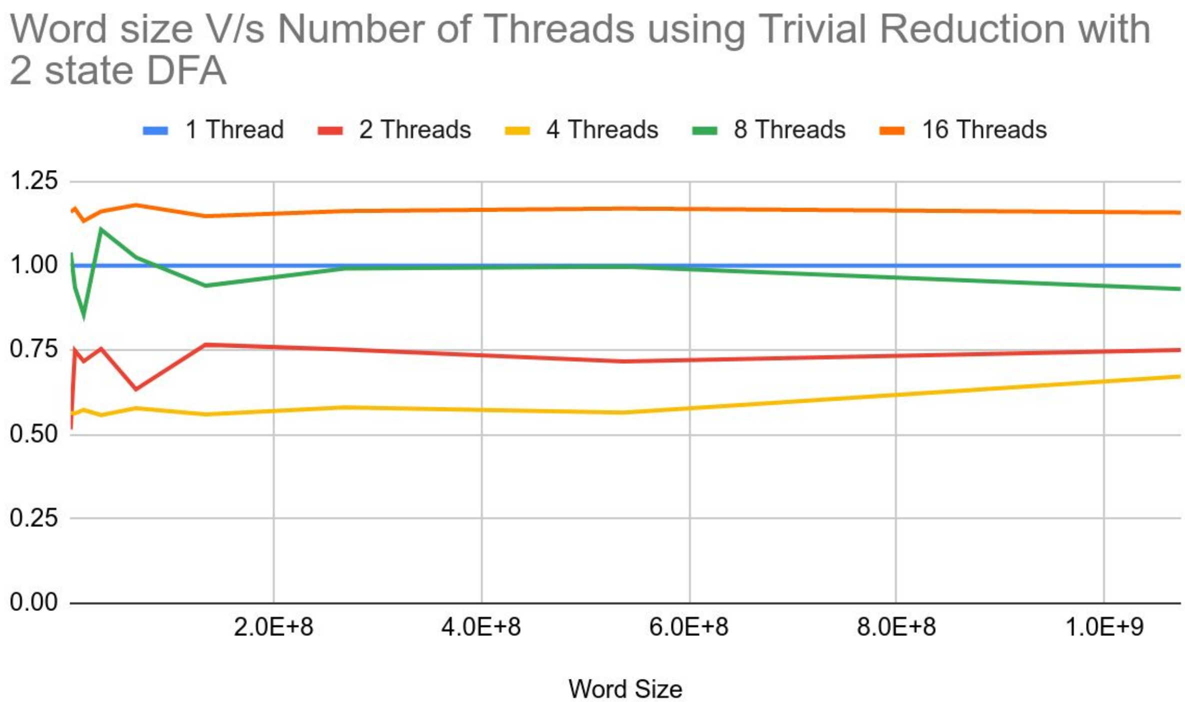
Time V/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0421552	0.0818303	0.0747662	0.040559	0.0363601
8388608	0.0842987	0.112657	0.149871	0.0900889	0.0721318
16777216	0.16908	0.235859	0.295212	0.197533	0.149282
33554432	0.337943	0.448372	0.606043	0.305346	0.291119
67108864	0.675515	1.0659	1.16825	0.659372	0.57265
134217728	1.35148	1.76424	2.41401	1.43666	1.1786
268435456	2.69931	3.59195	4.65141	2.72263	2.32315
536870912	5.40049	7.5337	9.55247	5.4154	4.61826
1073741824	10.8007	14.3936	16.0726	11.5971	9.33172



SpeedUp V/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	1	0.5151539222	0.5638269699	1.039355014	1.15938075
8388608	1	0.7482775149	0.5624750619	0.935727931	1.168675952
16777216	1	0.7168689768	0.5727409455	0.855958245	1.132621481
33554432	1	0.7537112041	0.5576221489	1.106754305	1.160841443
67108864	1	0.6337508209	0.578228119	1.024482386	1.179629791
134217728	1	0.7660409015	0.5598485508	0.9407097017	1.146682505
268435456	1	0.7514887457	0.5803208059	0.9914347524	1.161918085
536870912	1	0.7168443129	0.5653501136	0.9972467408	1.169377644
1073741824	1	0.7503821143	0.6719945746	0.9313276595	1.157417925



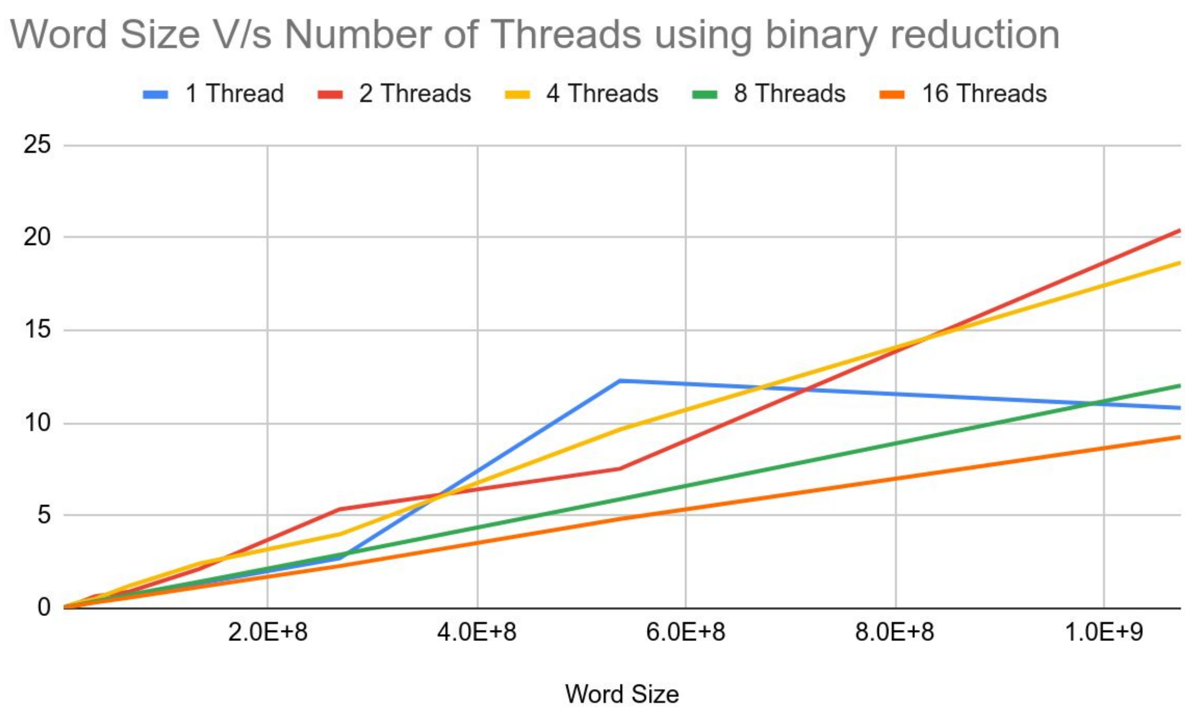
Observations:

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:

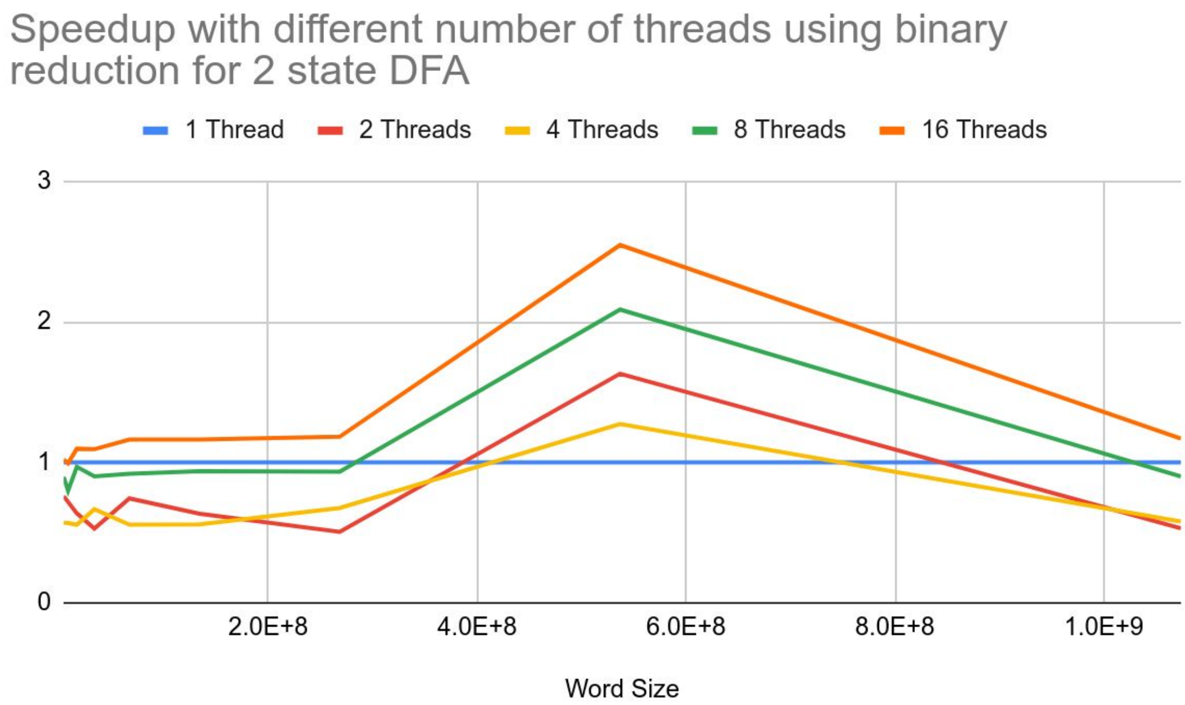
Time V/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0428337	0.0564308	0.074854	0.0478003	0.0419639
8388608	0.0843133	0.117406	0.148668	0.105592	0.0848873
16777216	0.169177	0.265506	0.304037	0.174853	0.154274
33554432	0.337487	0.63844	0.506437	0.374908	0.30869
67108864	0.67492	0.907932	1.21378	0.734716	0.580921
134217728	1.35113	2.12935	2.41556	1.44243	1.16203
268435456	2.69893	5.34375	3.99763	2.89429	2.28305
536870912	12.2841	7.53419	9.65637	5.88596	4.82503
1073741824	10.8136	20.4151	18.6628	12.0281	9.25221



Speedup v/s Number of Threads:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	1	0.7590482502	0.572229941	0.8960968864	1.020727339
8388608	1	0.7181345076	0.5671247343	0.7984818926	0.9932380933
16777216	1	0.6371871069	0.5564355654	0.9675384466	1.096600853
33554432	1	0.5286119291	0.6663948329	0.900186179	1.093287764
67108864	1	0.7433596349	0.5560480482	0.9186134506	1.161810298
134217728	1	0.6345269683	0.559344417	0.9367040342	1.16273246
268435456	1	0.505062924	0.675132516	0.9325015807	1.18215983
536870912	1	1.630447334	1.272123997	2.087017241	2.545911632
1073741824	1	0.5296863596	0.5794200227	0.8990281092	1.168758599



Observations:

Here, at 2^{29} we see that speedup for each thread is greater than 1. 16 threads always have speed up greater than 1. Rest of them fluctuate between less than 1 and greater than 1.

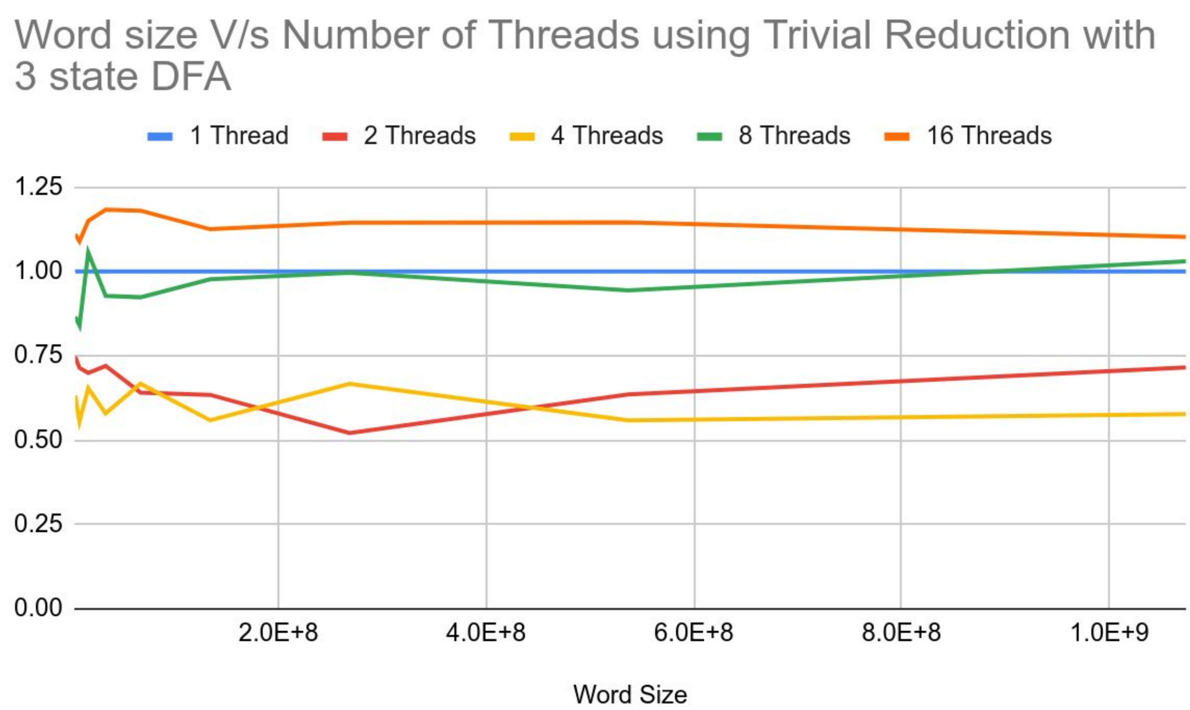
Results for 3 state DFA also followed similar trends as of 2 state DFA.

Using trivial reduction:

Time vs word Size:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0422315	0.0565265	0.0668053	0.0487475	0.0380444
8388608	0.0842927	0.117974	0.151659	0.100309	0.0773455
16777216	0.168942	0.241586	0.258241	0.159959	0.14693
33554432	0.339279	0.471072	0.585674	0.365598	0.286804
67108864	0.675413	1.053	1.01157	0.731164	0.572136
134217728	1.35031	2.12842	2.4153	1.38222	1.19993
268435456	2.69734	5.17573	4.04154	2.70692	2.35637
536870912	5.4011	8.50016	9.65983	5.7187	4.71307
1073741824	10.7962	15.0771	18.7059	10.4807	9.78991

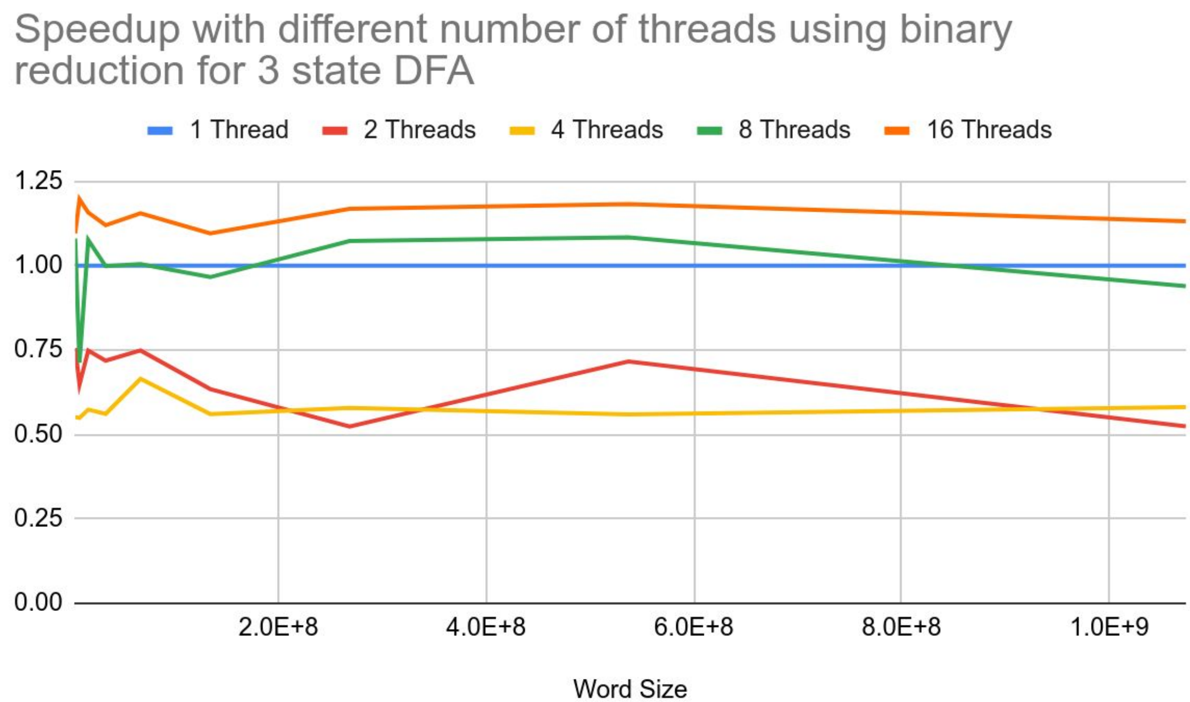
Speedup vs word size graph:



Using Binary reduction:

Time v/s Word Size:

Word Size	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
4194304	0.0424962	0.056186	0.0770571	0.0393312	0.0387948
8388608	0.0843077	0.130021	0.153547	0.118295	0.0704546
16777216	0.168989	0.22574	0.294551	0.157041	0.146007
33554432	0.337903	0.470098	0.601943	0.338397	0.301671
67108864	0.674925	0.900937	1.01504	0.671906	0.584285
134217728	1.35048	2.12777	2.40873	1.39771	1.23271
268435456	2.69857	5.15159	4.66327	2.51405	2.30852
536870912	5.39907	7.5397	9.65285	4.98042	4.5669
1073741824	10.7942	20.5927	18.5775	11.4872	9.53863



Observations:

Same trends as two state DFA were found here too.