Received January 31, 2020, accepted February 10, 2020, date of publication February 13, 2020, date of current version February 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2973741

Accelerating Forward Algorithm for Stochastic Automata on Graphics Processing Units

MUHAMMAD UMER SARWAR[1], MUHAMMAD KASHIF HANIF[1], RAMZAN TALIB[1], AND MUHAMMAD HARIS AZIZ[2]
[1]Department of Computer Science, Government College University, Faisalabad 38000, Pakistan
[2]Industrial Engineering Department, University of Engineering and Technology, Taxila 47050, Pakistan

Corresponding author: Muhammad Kashif Hanif (mkashifhanif@gcuf.edu.pk)

ABSTRACT A stochastic automaton is a non-deterministic automata with input and output behavior which works serially and synchronously. Stochastic automata is being used in different application areas. For large state space and sequence lengths, performance of stochastic automata is a major concern. For this purpose, graphics processing units can be employed to improve the performance. In this study, a parallel version of inference algorithm for stochastic automata is designed. The parallel version is mapped to graphics processing unit using the dynamic parallelism. The performance of parallel version is compared with different realizations and parameters. Parallel implementation of inference algorithm achieved approximately speedup factor of 50 for 256 states.

INDEX TERMS Stochastic automata, CUDA, GPU, forward algorithm, parallelization.

I. INTRODUCTION

Stochastic automata are probabilistic automata with input/output behavior. Stochastic automata emits an output symbol and moves into another state after reading input. Stochastic automata have been applied in language understanding [1], [2], modeling of soft real-time systems [3], [4], and machine learning. Typically, a stochastic automaton can have all transitions between states [5]. This can lead to high computational complexity for real world problems. There is need to improve the performance of stochastic automata algorithms.

Performance of stochastic automata algorithms can be enhanced with the help of modern high performance computing. Graphics Processing Units (GPUs) can efficiently solve complex problems. GPUs are many core and massively parallel architectures which can perform computation intensive tasks [6].

The *Compute Unified Device Architecture* (CUDA) programming model introduced by NVIDIA provides extension to the C language and supports the CPU/GPU execution. CUDA provides a hierarchy of thread groups, shared memories, and barrier synchronization [6]. The execution of a thread in CUDA is sequential. Many threads can be executed in parallel to process different parts of data [6], [7].

The associate editor coordinating the review of this manuscript and approving it for publication was Imran Ahmed.

The threads are grouped into a two-level hierarchy, i.e., grid and block. A grid is composed of one or more blocks and each block can have one or more threads. The efficiency of GPU programs depends on hardware configuration, the utilization of the allocated hardware, and the amount of parallelism exhibited by the problem [6]–[10].

In stochastic automata, the inference algorithm called Forward algorithm is a variant of Viterbi algorithm [11]. Forward algorithm is a dynamic programming algorithm to find the optimal sequence of states. For m states, the Forward algorithm finds the path with time complexity $O(m^2)n$. This work presents the formulation and experimental setup of parallel version of the Forward algorithm. The Forward algorithm is partitioned into data independent and dependent parts. The data independent part is implemented on the GPU to enhance the efficiency.

The remainder of the paper is structured in different sections. Section II provides a brief overview of the stochastic automata. Sections III and IV discuss the Forward algorithm for stochastic automata and different approaches to parallel this algorithm. Section V presents the results. Finally, section VI concludes the outcomes.

II. STOCHASTIC AUTOMATA

An automaton (Automata in plural) is a control mechanism or an abstract computing device which performs a pre-determined sequence of operations automatically.

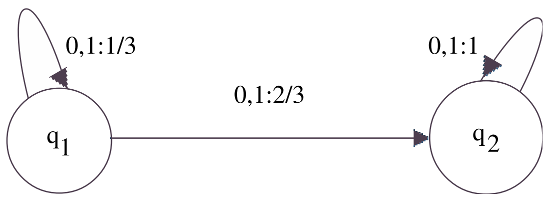


FIGURE 1. Stochastic automata diagram.

Finite automata is an automaton with a finite number of states. An automaton which change their state and give output according to probability is called stochastic automata. Stochastic automata has been introduced as systems which change their state and give some output according to some probability depending on the input and the actual state [11].

Shannon [12] and Von Neumann [13] introduced the theory of discrete stochastic systems by working on memory less communication channels and synthesis of reliable systems from unreliable components, respectively. The hidden Markov model (HMM) is a probabilistic network related to stochastic automata [5], [14], [15]. HMM addresses evaluation, decoding, and learning problems [1], [16].
The formal definition of stochastic automata is [17], [18]:

- Nonempty finite set of states Q,
- an alphabet of input symbols Γ,
- an alphabet of output symbols Ψ, and
- conditional probability distribution p on $\Psi \times Q$.

A stochastic automaton can be considered an abstract machine that takes on a well-defined state at each time step of computation. Conditional probability distribution $p(\cdot, \cdot|u, q)$ on $\Psi \times Q$ consists of non-negative numbers $p(v, q'|u, q)$ for all $q' \in Q$ and $v \in \Psi$ so that

$$\sum_{v\in\Psi}\sum_{q'\in Q} p(v, q'|u, q) = 1, \quad u \in \Gamma, q \in Q \tag{1}$$

For $|u| = |v|$,

$$p(vv', q'|uu', q) = \sum_{s'\in Q} p(v, s'|u, q) \cdot p(v', q'|u', s) \tag{2}$$

where all $u, u' \in \Gamma^*$, $v, v' \in \Psi^*$, and $q, q' \in Q$.

The implementation of stochastic automata can be constrained due to local maxima problem. This problem can be avoided by determining the allowed transitions for the given problem [5].

Example 1: Consider the stochastic automata $SA = (\{q_1, q_2\}, \{0\}, \{1\}, p)$ with conditional probability

$$p(1, q_1|0, q_1) = \frac{1}{3}, p(1, q_2|0, q_1) = \frac{2}{3}, p(1, q_2|0, q_2) = 1.$$

Figure 1 shows the stochastic automata graph. The sub-stochastic matrix is

$$P(u) = P(v|u) = \begin{pmatrix} \frac{1}{3} & \frac{2}{3} \\ 0 & 1 \end{pmatrix}$$

III. FORWARD ALGORITHM

Stochastic automata are abstract machines with input/output behavior. Consider the stochastic automata $SA = (Q, \Gamma, \Psi, P, \pi, f)$ with m-element set of states Q, m_1-element input alphabet Γ, and m_2-element output alphabet Ψ. The marginal distribution gives the probability of input sequence $U = u_1, u_2, \cdots, u_n \in \Gamma^n$ and output sequence $V = v_1, v_2, \cdots, v_n \in \Psi^n$.

$$p\ U, V(u, v) = \sum_{q\in Q^{n+1}} p\ U, Q, V(u, s, v) \tag{3}$$

The sum-product decomposition is

$$p\ U, V(u, v) = \frac{1}{m.m_1} \sum_{q_{n+1}\in Q} (\sum_{q_n\in Q} \theta_{v_n,q_{n+1};u_n,q_n} (\cdots(\sum_{q_2\in Q} \theta_{v_2,q_3;u_2,q_2}(\sum_{q_1\in Q} \theta_{v_1,q_2;u_1,q_1}))\cdots)) \tag{4}$$

A $n \times m$ matrix F can be used to calculate the probability p(u,v).

$$F[0, q] = \frac{1}{m.m_1}, \quad q \in Q, \tag{5}$$

$$F[i, q] = \sum_{q'\in Q} (\theta_{v_i,q;u_i,q'} \cdot F[i-1, q']), \quad q \in Q, 1 \le i \le n, \tag{6}$$

$$p(u, v) = \sum_{q\in Q} F[n, q]. \tag{7}$$

The aim is to determine the optimal state sequence $\bar{q} \in Q^{n+1}$.

$$\bar{q} = \text{argmax}_{q\in Q^{n+1}} \{p\ U, Q, V u, q, v\} \tag{8}$$

Algorithm 1 Forward(u, v, $\theta_{v_i,q';u_i,q}$)

```
Require: sequence $u \in \Gamma^n$, $v \in \Psi^n$, scores ($\theta_{v_i,q;u_i,q'}$)
Ensure: term $p(u, v)$
1: $F \leftarrow$ matrix[$0 \ldots n, 1 \ldots m$]
2: for $q \leftarrow 1$ to $m$ do
3:     $F[0, q] \leftarrow \frac{1}{m.m_1}$
4: end for
5: for $i \leftarrow 1$ to $n$ do
6:     for $q \leftarrow 1$ to $m$ do
7:         $F[i, q] \leftarrow 0$
8:         for $q' \leftarrow 1$ to $m$ do
9:             $F[i, q] \leftarrow sum(F[i, q], \theta_{v_i,q;u_i,q'} \cdot F[i-1, q'])$
10:        end for
11:    end for
12: end for
13: $p \leftarrow 0$
14: for $q \leftarrow 1$ to $m$ do
15:     $p \leftarrow sum(p, F[n, q])$
16: end for
```

Given the input sequence $U = u_1, u_2, \cdots, u_n$ and output sequence $V = v_1, v_2, \cdots, v_n$, the Forward algorithm finds the sequence of states that correspond to this input and output behavior [11]. This algorithm initializes the forward matrix F with fraction $\frac{1}{m.m_1}$. Matrix entries are calculated using the values of previous row. This algorithm finds the matrix entry $F[i, q]$ by adding the marginal probability of current state to the preceding value $F[i-1, q']$.

The computational time complexity of Forward algorithm is $O(m^2 n)$. The optimal state sequence $\bar{q}$ can be computed efficiently using tropicalization of sum-product decomposition [11]. For this, put $d(u, v) = -\log p\ U, V(u, v)$ and $d(u, q, v) = -\log p\ U, Q, V(u, q, v)$. The tropicalized term $d(u, v)$ can be computed by putting $s_{y,q';x,q} = -\log \theta_{v,q';u,q}$. The sums and products are replaced by tropical addition and tropical multiplication in sum-product decomposition [11].

$$d(u, v) = \bigoplus_{q_{n+1}\in Q} (\bigoplus_{q_n\in Q} s_{v_n,q_{n+1};u_n,q_n} \odot (\cdots \odot (\bigoplus_{q_2\in Q} s_{v_2,q_3;u_2,q_2} \odot (\bigoplus_{q_1\in Q} s_{v_1,q_2;u_1,q_1}))\cdots)) \quad (9)$$

By using the tropicalized sum-product decomposition, the term d(u,v) can be calculated.

$$F[0, q] = 0, \quad q \in Q, \quad (10)$$

$$F[i, q] = \bigoplus_{q'\in Q} (s_{v_i,q;u_i,q'} \odot F[i-1, q']), \quad q \in Q, 1 \le i \le n, \quad (11)$$

$$d(u, v) = \bigoplus_{q\in Q} F[n, q]. \quad (12)$$

Algorithm 2 Tropical_Forward($u, v, s_{v_i,q';u_i,q}$)

```
Require: sequence $u \in \Gamma^n$, $v \in \Psi^n$, scores $(s_{v_i,q;u_i,q'})$
Ensure: term $d(u, v)$
 1: $F \leftarrow$ matrix$[0 \ldots n, 1 \ldots m]$
 2: for $q \leftarrow 1$ to $m$ do
 3:     $F[0, q] \leftarrow 0$
 4: end for
 5: for $i \leftarrow 1$ to $n$ do
 6:     for $q \leftarrow 1$ to $m$ do
 7:         $F[i, q] \leftarrow \infty$
 8:         for $q' \leftarrow 1$ to $m$ do
 9:             $F[i, q] \leftarrow \min\{F[i, q], s_{v_i,q;u_i,q'} + F[i-1, q']\}$
10:         end for
11:     end for
12: end for
13: $d \leftarrow \infty$
14: for $q \leftarrow 1$ to $m$ do
15:     $d \leftarrow \min\{d, F[n, q]\}$
16: end for
```

Equation 10 to 12 gives the Tropical_Forward algorithm. Its input is given by the input sequence u, the output sequence v, and transition probability $s_{v_i,q';u_i,q}$. The first row of F is initialized with 0. The entry $F[i, q]$ has minimum value for the current state. Then, the tropicalized term $d(u, v)$ is computed. The high computational complexity limits the usage of stochastic automata. Optimal state sequence(s) can be obtained by backward algorithm [11].

Example 2: Consider the stochastic automaton $SA = (q_1, q_2, a, b, 0, 1, p)$ with conditional probabilities

p'	a, q_1	a, q_2	b, q_1	b, q_2
0	0.30	0.50	0.60	0.40
1	0.70	0.50	0.40	0.60

p''	q_1	q_2
q_1	0.60	0.30
q_2	0.40	0.70

Then probabilities are

$$\begin{aligned} p(0, q_1|a, q_1) &= 0.18, & p(0, q_2|a, q_1) &= 0.12,\\ p(1, q_1|a, q_1) &= 0.42, & p(1, q_2|a, q_1) &= 0.28,\\ p(0, q_1|a, q_2) &= 0.15, & p(0, q_2|a, q_2) &= 0.35,\\ p(1, q_1|a, q_2) &= 0.15, & p(1, q_2|a, q_2) &= 0.35,\\ p(0, q_1|b, q_1) &= 0.36, & p(0, q_2|b, q_1) &= 0.24,\\ p(1, q_1|b, q_1) &= 0.24, & p(1, q_2|b, q_1) &= 0.16,\\ p(0, q_1|b, q_2) &= 0.12, & p(0, q_2|b, q_2) &= 0.28,\\ p(1, q_1|b, q_2) &= 0.18, & p(1, q_2|b, q_2) &= 0.42. \end{aligned}$$

The tropicalized values obtained by taking natural logarithm are

$$\begin{aligned} s_{0,q_1|a,q_1} &= 1.71, & s_{0,q_2|a,q_1} &= 2.12,\\ s_{1,q_1|a,q_1} &= 0.87, & s_{1,q_2|a,q_1} &= 1.27,\\ s_{0,q_1|a,q_2} &= 1.90, & s_{0,q_2|a,q_2} &= 1.05,\\ s_{1,q_1|a,q_2} &= 1.90, & s_{1,q_2|a,q_2} &= 1.05,\\ s_{0,q_1|b,q_1} &= 1.02, & s_{0,q_2|b,q_1} &= 1.43,\\ s_{1,q_1|b,q_1} &= 1.43, & s_{1,q_2|b,q_1} &= 1.83,\\ s_{0,q_1|b,q_2} &= 2.12, & s_{0,q_2|b,q_2} &= 1.27,\\ s_{1,q_1|b,q_2} &= 1.71, & s_{1,q_2|b,q_2} &= 0.87. \end{aligned}$$

IV. ACCELERATING FORWARD ALGORITHM

For large sequence length and state space, the high complexity restricts the usage of the stochastic automata. The huge performance boosts can be attained by mapping the stochastic automata algorithms on GPU. Researchers have mapped HMM based applications to GPU and achieved order of magnitude speedup. They have applied task parallel [19]–[23], data parallel [24]–[27], and combination of task and data parallel [28]–[32] approaches for HMM. Similar approaches can be adopted to improve the performance of stochastic automata.

Different approaches can be used to enhance the performance of Tropical_Forward algorithm. One method is to compute the probabilities in advance and save in the matrix. The downside is this large matrix should be transferred into the GPU memory. Moreover, the maximum number of

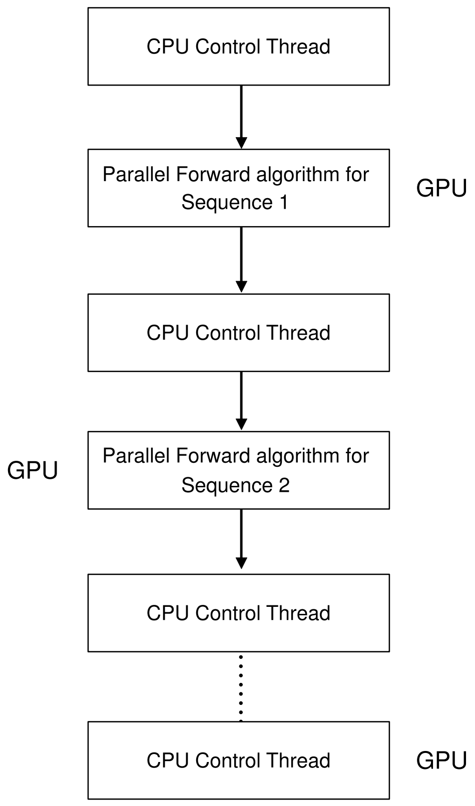

CPU Control Thread
Parallel Forward algorithm for Sequence 1
GPU
CPU Control Thread
GPU
Parallel Forward algorithm for Sequence 2
CPU Control Thread
CPU Control Thread
GPU

FIGURE 2. ***n*** **times GPU kernel invocation.**

sequences and states that can be processed is restricted due to small GPU memory size.

Another approach to accelerate the Tropical_Forward algorithm is to divide the algorithm into sequential CUDA kernel calls. In this manner, *n* kernels are launched for *n* sequences (Figure 2). Moreover, the number of blocks and threads for each kernel are selected according to problem size. Parallel implementation of this approach uses both coarse and fine grain granularity depending on the state space. However, each kernel launch requires to transfer the control to CPU. For large sequence lengths, multiple kernel launch and execution overheads can impact the performance.

Dynamic parallelism can be employed to reduce the GPU kernel launch overhead. Dynamic parallelism can minimize the need to transfer execution control and data between CPU and GPU [7]. This approach can be adopted to implement Tropical_Forward algorithm. This work launches multiple kernels in blocks using dynamic parallelism. Each block processes an input sequence (Figure 3). Parallel implementation uses both coarse and fine grain granularity. However, execution parameters configuration and allocation of resources is major concern for dynamic parallelism.

V. RESULTS AND DISCUSSION

In this section, performance results of the serial and parallel implementations of Forward algorithm for stochastic automata is presented. We have considered execution time and speed up for performance evaluation. The execution times is measured by taking average over twenty runs. Moreover, theoretical floating point operations (FLOPs) are not considered as performance measure. The reason is single floating point operation can be transformed to multiple operations during compilation. Moreover, we considered single precision arithmetic and different optimization flags for experimentation.

The computing environment used for implementation is an Intel Core i7 6700 CPU (3.40 GHz) the CUDA version 9.0 on an NVIDIA Titan XP graphics card. Different parameters and realizations were used to obtain the results. In order to achieve highest performance results, ECC mode was disabled. The tests are performed using different optimization flags like *maxrregcount*, *use_fast_math*. The kernel exploited different memory optimization techniques by considering different memory types. The test data is generated randomly. The performance is calculated using constant sequence size and variable state space and vice versa. The experimental results are examined for the state space up to 256 and maximum sequence size is 32,768.

Figures 4 and 5 shows the comparison of the parallel and serial versions of the Forward algorithm. First, results are obtained by altering the sequence length and fixing the number of states (Figure 4). The serial and parallel versions

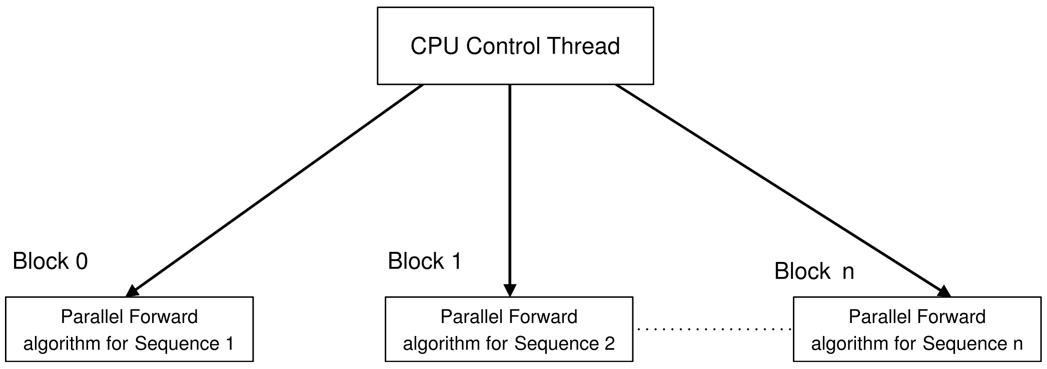

CPU Control Thread
Block 0
Block 1
Block n
Parallel Forward algorithm for Sequence 1
Parallel Forward algorithm for Sequence 2
Parallel Forward algorithm for Sequence n

FIGURE 3. **GPU kernel invocation using dynamic parallelism.**

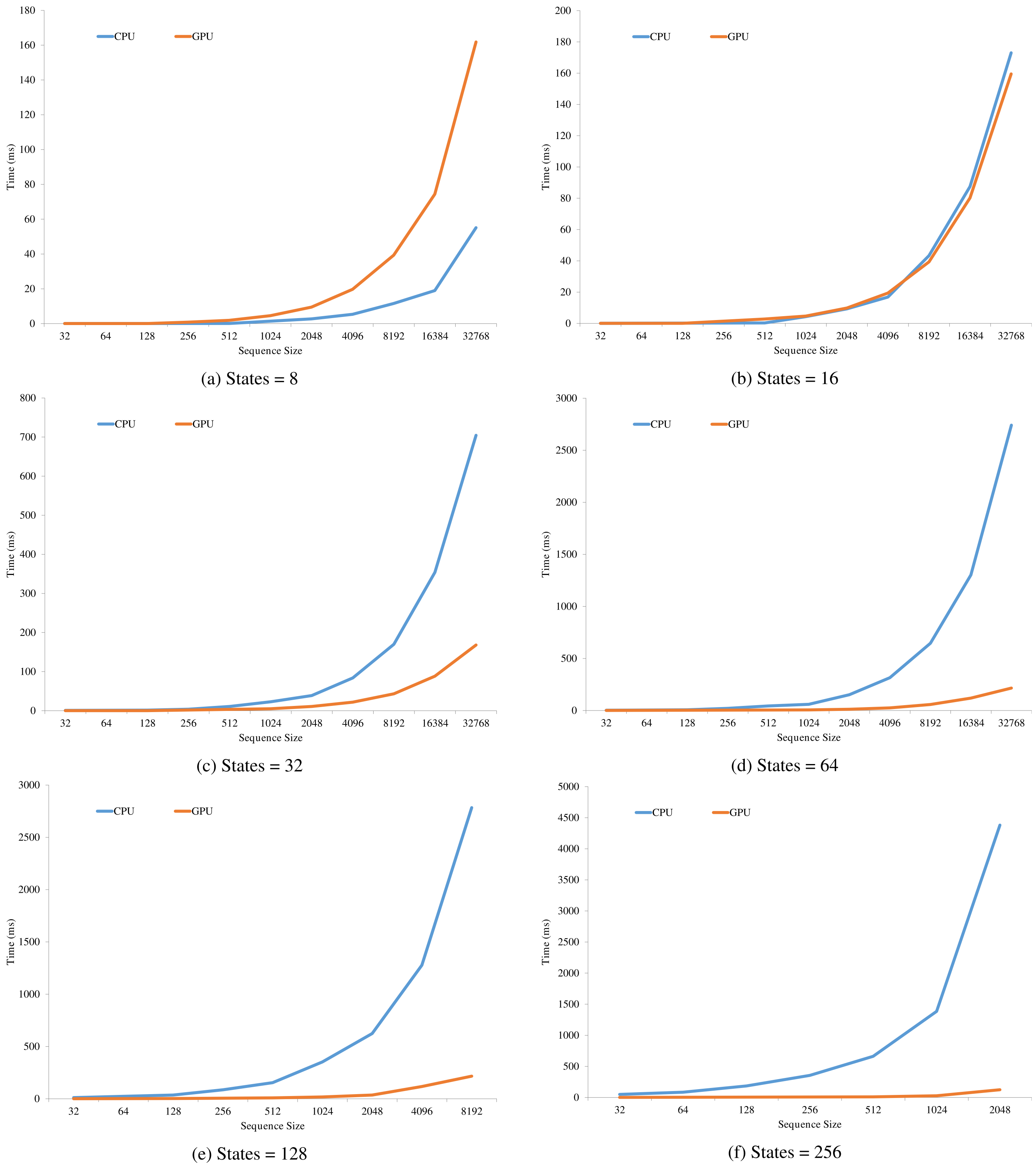
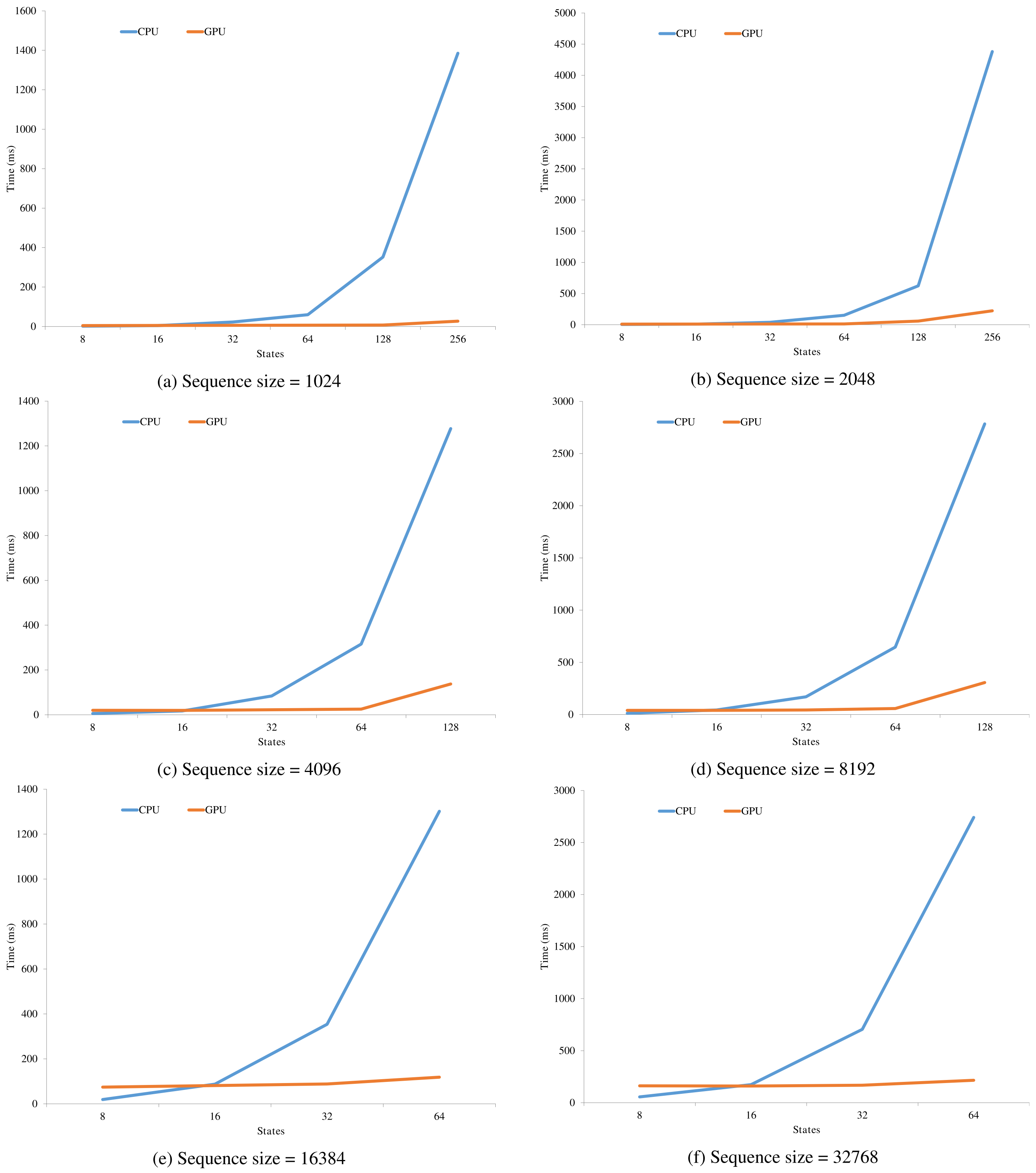


FIGURE 4. Runtime (ms) of the Forward algorithm using variable sequence size and constant number of states.

of Tropical_Forward algorithm have approximately the similar runtime for small number of states. The reason is threads within the block does not fully utilize the hardware. There exists processing overhead for context switching between the CPU and GPU. Parallel version of Tropical_Forward algorithm is not suitable for small state space. However, parallel version performs better than serial implementation by an order of magnitude for large state space. Moreover, the average execution time for parallel version is almost similar for all states (Table 1).



(a) Sequence size = 1024

(b) Sequence size = 2048

(c) Sequence size = 4096

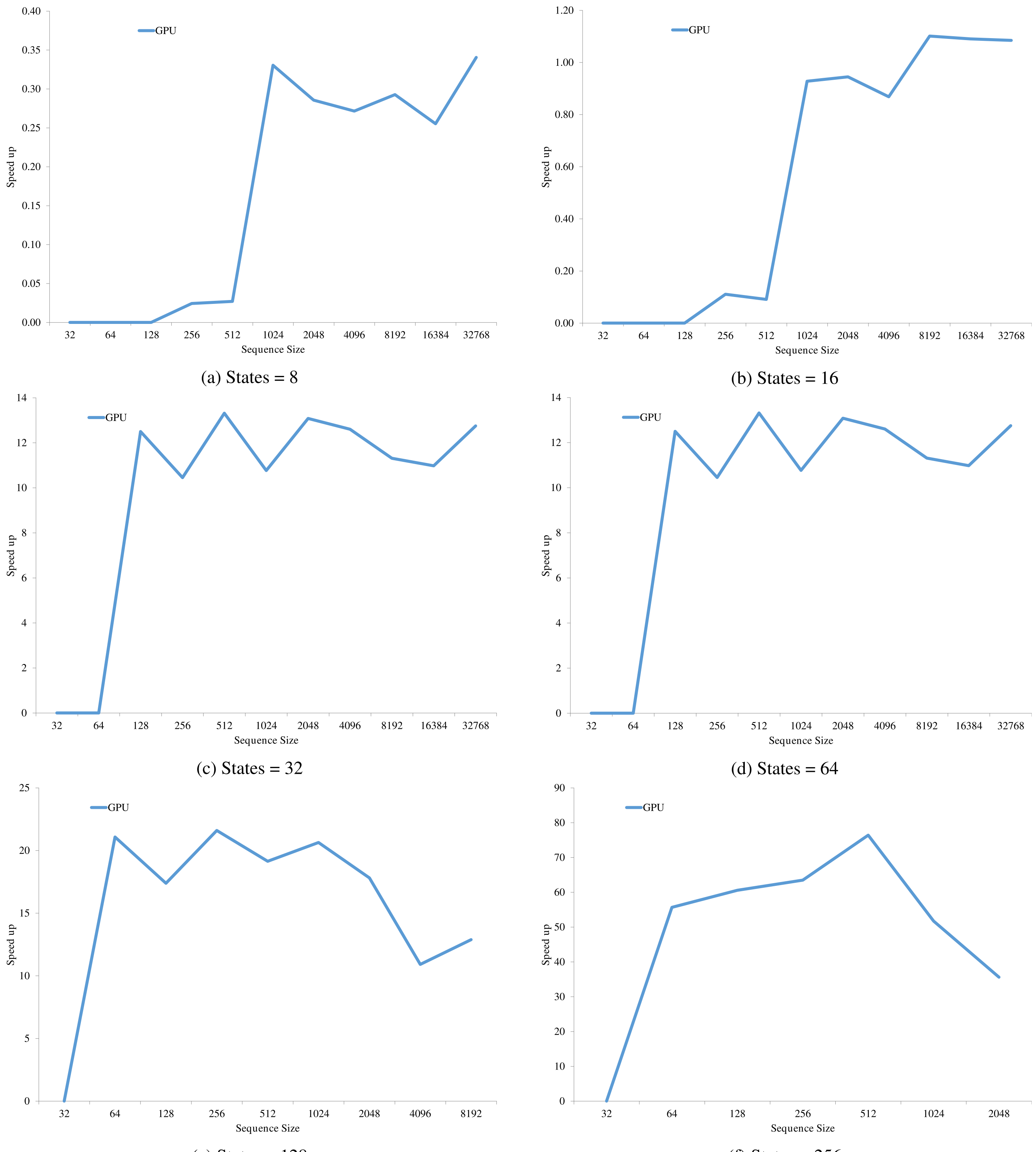
(d) Sequence size = 8192

(e) Sequence size = 16384

(f) Sequence size = 32768

FIGURE 5. Runtime (ms) of the Forward algorithm using variable number of states and constant sequence size.

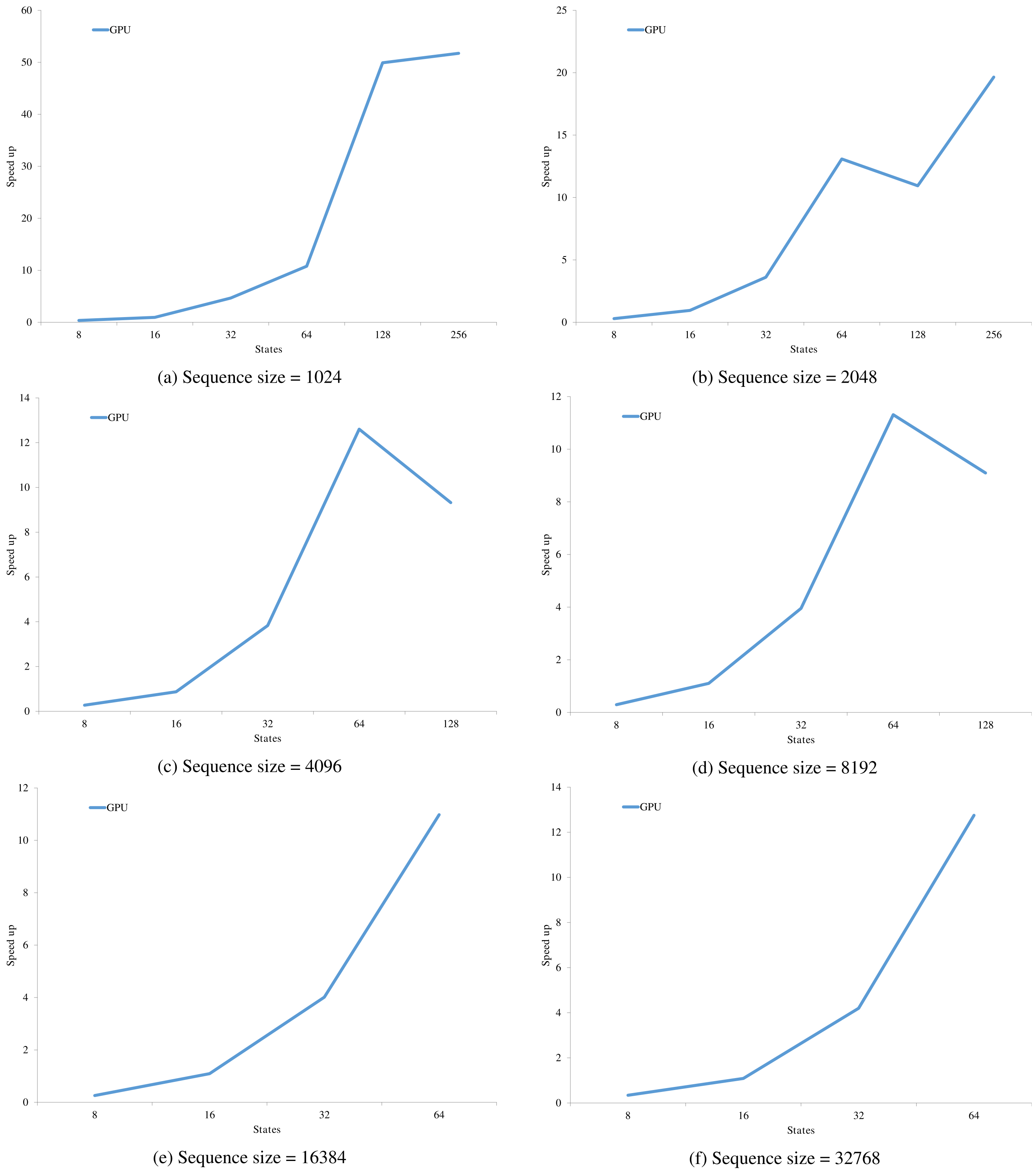
Figure 5 shows the performance by altering the states and fixing the sequence length. Parallel version performs much better than the serial version for large number of states. By increasing the number of states and launching multiple kernels using dynamic parallelism, the performance of parallel version is increased by an order of magnitude. The average runtime is directly proportional to the sequence length. This is valid for all approaches. Table 2 provides more detail.



(a) States = 8 (b) States = 16 (c) States = 32 (d) States = 64 (e) States = 128 (f) States = 256

FIGURE 6. **Speedup for Forward algorithm using fixed number of states and variable sequence length.**

Next, the speedups obtained is calculated by comparing with serial implementation (Figure 6 and 7). Figure 6 illustrates the speedup by altering sequence length and fixing the number of states. For small state space, the parallel version shows the non-monotonic behavior. The reason is small degree of parallelism exhibited by the small number of states.



(a) Sequence size = 1024

(b) Sequence size = 2048

(c) Sequence size = 4096

(d) Sequence size = 8192

(e) Sequence size = 16384

(f) Sequence size = 32768

FIGURE 7. Speedup for Forward algorithm using constant sequence size and variable number of states.

Large state space have better speed-up due to large degree of parallelism. The maximum average speedup is approximately 49 (Table 3).

Finally, the speedup by altering state space and fixing the sequence size is shown in Figure 7. For the large sequence size, the parallel version has performance degradation. The reason is large number of blocks are created. There are hardware limitations of parallel executions of the blocks. The maximum average speedup attained is approximately 21 (Table 4).

TABLE 1. Maximum and average runtime of the Forward algorithm using constant number of states.

states	CPU		GPU	
	Mean	Max	Mean	Max
8	8.655	55.100	28.348	161.800
16	30.408	172.988	28.809	159.498
32	126.452	704.370	31.031	167.901
64	480.798	2740.600	39.853	215.000
128	593.648	2783.400	44.456	216.000
256	1013.914	4380.000	24.080	123.000

TABLE 2. Maximum and average runtime of the Forward algorithm using constant sequence size.

Sequence size	CPU		GPU	
	Mean	Max	Mean	Max
1024	304.128	1385.000	8.898	26.780
2048	867.541	4380.000	53.591	223.000
4096	339.587	1277.000	44.590	137.000
8192	730.656	2783.400	96.939	306.000
16384	440.444	1301.544	90.324	118.560
32768	918.264	2740.600	176.050	215.000

TABLE 3. Maximum and average speedup of the Forward algorithm using constant number of states and and variable sequence size.

States	Speed up	
	Mean	Max
8	0.166	0.255
16	0.565	1.101
32	2.733	4.653
64	9.795	13.313
128	15.717	21.595
256	49.064	76.416

TABLE 4. Maximum and average speedup of the Forward algorithm using constant sequence size and variable number of states.

Sequence size	Speedup	
	Mean	Max
1024	19.715	21.718
2048	8.082	19.641
4096	5.377	12.600
8192	5.150	11.310
16384	4.084	10.978
32768	4.592	12.747

VI. CONCLUSION

Stochastic automata is a class of probabilistic automata with input/output behavior. The high computational complexity limits the usage of stochastic automata. This study presents a parallel version of Forward algorithm. The parallel version was designed using dynmaic parallelism. Forward algorithm achieves speed-up factor of approximately 49 for 256 states. This approach should be investigated for learning problem of Stochastic automata.

REFERENCES

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[2] G. Riccardi, R. Pieraccini, and E. Bocchieri, "Stochastic automata for language modeling," *Comput. Speech Lang.*, vol. 10, no. 4, pp. 265–293, Oct. 1996.

[3] P. R. D'Argenio and J.-P. Katoen, "A theory of stochastic systems. Part I: Stochastic automata," *Inf. Comput.*, vol. 203, no. 1, pp. 1–38, Nov. 2005.

[4] P. R. D'Argenio and J.-P. Katoen, "A theory of Stochastic systems. Part II: Process algebra," *Inf. Comput.*, vol. 203, no. 1, pp. 39–74, Nov. 2005.

[5] K.-H. Zimmermann, "Algebraic statistics," Inst. Comput. Technol., Hamburg Univ. Technol., Hamburg, Germany, 2009.

[6] *CUDA C Programming Guide*, NVIDIA Corp., Santa Clara, CA, USA, 2018.

[7] D. B. Kirk and W. H. Wen-Mei, *Programming Massively Parallel Processors: A Hands-On Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2016.

[8] S. Cuomo, V. De Angelis, G. Farina, L. Marcellino, and G. Toraldo, "A GPU-accelerated parallel K-means algorithm," *Comput. Electr. Eng.*, vol. 75, pp. 262–274, May 2019.

[9] Z. Ding, G. Mei, S. Cuomo, H. Tian, and N. Xu, "Accelerating multidimensional interpolation using moving least-squares on the GPU," *Concurrency Comput., Pract. Exper.*, vol. 30, no. 24, Dec. 2018, Art. no. e4904.

[10] Z. Ding, G. Mei, S. Cuomo, N. Xu, and H. Tian, "Performance evaluation of GPU-accelerated spatial interpolation using radial basis functions for building explicit surfaces," *Int. J. Parallel Programm.*, vol. 46, no. 5, pp. 963–991, Oct. 2018.

[11] K.-H. Zimmermann, "Inference and learning in stochastic automata," *Int. J. Pure Appl. Math.*, vol. 115, no. 3, pp. 621–639, 2017.

[12] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul./Oct. 1948.

[13] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Stud.*, vol. 34, pp. 43–98, Jan. 1956.

[14] D. Barber, *Bayesian Reasoning and Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[15] T. Koski and J. Noble, *Bayesian Networks: Introduction*, vol. 924. Hoboken, NJ, USA: Wiley, 2011.

[16] L. Rabiner and B. Juang, "An introduction to hidden Markov models," *IEEE ASSP Mag.*, vol. 3, no. 1, pp. 4–16, Jan. 1986.

[17] V. Claus, *Stochastische Automaten*. New York, NY, USA: Springer-Verlag, 2013.

[18] A. Salomaa and I. N. Sneddon, *Theory of Automata*. New York, NY, USA: Pergamon, 1969.

[19] C. Liu, "cuHMM: A CUDA implementation of hidden Markov model training and classification," The Chronicle of Higher Education, 2009, pp. 1–13.

[20] J. Li, S. Chen, and Y. Li, "The fast evaluation of hidden Markov models on GPU," in *Proc. IEEE Int. Conf. Intell. Comput. Intell. Syst.*, vol. 4, Nov. 2009, pp. 426–430.

[21] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, p. 777, Aug. 2004.

[22] D. R. Horn, M. Houston, and P. Hanrahan, "ClawHMMER: A streaming HMMer-search implementatio," in *Proc. ACM/IEEE SC Conf. (SC)*, Dec. 2005, p. 11.

[23] J. P. Walters, V. Balu, S. Kompalli, and V. Chaudhary, "Evaluating the use of GPUs in liver image segmentation and HMMER database searches," in *Proc. 2009 IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–12.

[24] Z. Du, Z. Yin, and D. A. Bader, "A tile-based parallel Viterbi algorithm for biological sequence alignment on GPU with CUDA," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–8.

[25] C.-S. Lin, W.-L. Liu, W.-T. Yeh, L.-W. Chang, W.-M.-W. Hwu, S.-J. Chen, and P.-A. Hsiung, "A tiling-scheme viterbi decoder in software defined radio for GPUs," in *Proc. 7th Int. Conf. Wireless Commun., Netw. Mobile Comput.*, Sep. 2011, pp. 1–4.

[26] Y. Lifshits, S. Mozes, O. Weimann, and M. Ziv-Ukelson, "Speeding up HMM decoding and training by exploiting sequence repetitions," *Algorithmica*, vol. 54, no. 3, pp. 379–399, 2009.

[27] S. Mozes, O. Weimann, and M. Ziv-Ukelson, "Speeding up HMM decoding and training by exploiting sequence repetitions," in *Proc. 18th Annu. Symp. Combinat. Pattern Matching (CPM)*, vol. 4580, in Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2007, pp. 4–15.

[28] N. Ganesan, R. D. Chamberlain, J. Buhler, and M. Taufer, "Accelerating HMMER on GPUs by implementing hybrid data and task parallelism," in *Proc. 1st ACM Int. Conf. Bioinf. Comput. Biol. (BCB)*, 2010, pp. 418–421.

[29] J. Nielsen and A. Sand, "Algorithms for a parallel implementation of hidden Markov models with a small state space," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 452–459.

[30] A. Sand, M. Kristiansen, C. N. Pedersen, and T. Mailund, "ZipHMMlib: A highly optimised HMM library exploiting repetitions in the input to speed up the forward algorithm," *BMC Bioinf.*, vol. 14, no. 1, p. 339, Dec. 2013.
[31] M. K. Hanif and K.-H. Zimmermann, "Accelerating Viterbi algorithm on graphics processing units," *Computing*, vol. 99, no. 11, pp. 1105–1123, Nov. 2017.
[32] M. K. Hanif, "Mapping dynamic programming algorithms on graphics processing units," Ph.D. dissertation, Inst. für Rechnertechnologie, Technische Univ. Hamburg, Hamburg, Germany, 2014.

MUHAMMAD UMER SARWAR is currently pursuing the Ph.D. degree. He is also an Assistant Professor with the Department of Computer Science, Government College University, Faisalabad. His research interests are database systems, text, and data mining.

MUHAMMAD KASHIF HANIF received the Ph.D. degree from the Hamburg University of Technology. He is currently an Assistant Professor with the Department of Computer Science, Government College University, Faisalabad. His research interests cover big data analytic, scientific computing, and bioinformatics.

RAMZAN TALIB received the Ph.D. degree from the University of Bayreuth, Germany. He is currently working as a Professor with the Department of Computer Science, Government College University, Faisalabad. His research interests include databases and information systems, data mining, data warehousing, business process management, and workflow management systems.

MUHAMMAD HARIS AZIZ received the Ph.D. degree in industrial and manufacturing engineering from the Asian Institute of Technology (AIT) Thailand. He is currently an Assistant Professor with the Industrial Engineering Department, University of Engineering and Technology (UET), Taxila, Pakistan. He is also a Fulbright Postdoctoral Scholar from the Department of System Science and Industrial Engineering (SSIE), Binghamton University (BU), Binghamton, NY, USA. His research interests include production planning and control, optimization of the operations of industrial systems in the context of sustainability, and circular economy using qualitative, mathematical and/or simulation tools. He has also interest in Engineering Education research, specifically collaborative project-based learning. His recent ventures include Cloud Manufacturing, Industry 4.0 and Data Analytics. He enjoys reading psychology, philosophy, and new thought writings.