On The Temporal Parallelisation of The Viterbi Algorithm

Simo Särkkä
Dept. of Electrical Engineering and Automation
Aalto University
Espoo, Finland
simo.sarkka@aalto.fi

Ángel F. García-Fernández
Dept. of Electrical Engineering and Electronics, and ARIES
University of Liverpool, and Universidad Nebrija
Liverpool, UK, and Madrid, Spain
angel.garcia-fernandez@liverpool.ac.uk

***Abstract*—This paper presents an algorithm to parallelise the Viterbi algorithm along the temporal dimension to compute the maximum a posteriori (MAP) trajectory estimate of a hidden Markov model. We reformulate the MAP estimation problem as an optimal control problem. The proposed algorithm uses a parallelisation algorithm developed for optimal control problems that first performs a backward value function pass and then a forward trajectory recovery pass. The parallel Viterbi algorithm then corresponds to a specialised backward optimal control problem with a forward value function pass and backward MAP-trajectory recovery pass. The algorithm is empirically tested by running numerical simulations on a multi-core central processing unit (CPU) and a graphics processing unit (GPU).**

***Index Terms*—Viterbi algorithm, temporal parallelisation, maximum a posteriori estimation, hidden Markov model.**

I. Introduction

The Viterbi algorithm is a widely-used algorithm in signal processing with multiple applications, for example, in convolutional code decoding, target tracking, and speech recognition [1]–[6]. The Viterbi algorithm can be seen as a general algorithm for finding the maximum a posteriori (MAP) trajectory estimate in a hidden Markov model (HMM) of the form

$$x_k \sim p(x_k \mid x_{k-1}), \quad y_k \sim p(y_k \mid x_k), \tag{1}$$

with $x_0 \sim p(x_0)$, where x_k is the hidden state at time step k, and y_k its observation for $k = 1, \ldots, T$. The state x_k is discrete and belongs to the set $\{1, \ldots, D\}$, where D is the number of possible states. The observation y_k can belong to a discrete or a continuous space.

The MAP estimate of the state trajectory $x_{0:T}$ given the observations $y_{0:T}$ can be computed by

$$x^*_{0:T} = \arg\max_{x_{0:T}} p(x_0) \prod_{k=1}^{T} \left[p(y_k \mid x_k)\, p(x_k \mid x_{k-1}) \right]. \tag{2}$$

The classical Viterbi algorithm consists of forward and backward passes. On the forward pass it propagates value of the joint density in (2) at x_k, $W_k(x_k)$, given the optimal trajectory up to the previous time step, from $k = 0$ to $k = T$ via

$$\begin{aligned} W_k(x_k) &= \max_{x_{k-1}} \left\{ p(y_k \mid x_k)\, p(x_k \mid x_{k-1})\, W_{k-1}(x_{k-1}) \right\}, \\ W_0(x_0) &= p(x_0). \end{aligned} \tag{3}$$

The authors would like to thank Academy of Finland for funding.

The MAP trajectory can then be recovered [1], [3] by using a backward pass starting from

$$x^*_T = \arg\max_{x_T} W_T(x_T), \tag{4}$$

and proceeding backwards for $k = T-1, \ldots, 0$ by

$$x^*_k = \arg\max_{x_k} \left[p(x^*_{k+1} \mid x_k)\, W_k(x_k) \right]. \tag{5}$$

Another way to recover the optimal trajectory is to store the optimal mappings from one time step to the previous time step and run a backward pass after the forward pass is finished [2]. Yet another formulation of the Viterbi algorithm is to see it as a max-product algorithm [7], [8]. In that setting, we run two independent forward and backward passes across time that are then combined to produce the optimal trajectory. The time complexity of the Viterbi algorithm in the three forms is $O(T)$.

Two parallel versions of the Viterbi algorithm that achieve logarithmic time complexity $O(\log T)$ were proposed in [9]. These two algorithms are based on parallelising the classical (path-based) Viterbi and the max-product forms, respectively, by making a use of parallel scan algorithms, also called all-prefix-sums algorithms [10]. The max-product based parallel Viterbi formulation of [9] has the advantage that it is a memory and time efficient parallel algorithm, while the path-based formulation has the disadvantage that it requires storing of the partial paths of the solution which is memory-inefficient. In this paper the aim is to present a path-based parallel algorithm corresponding to [2] that does not have this disadvantage.

In order to arrive at the proposed algorithm, we use the point of view that MAP trajectory estimation and hence the Viterbi algorithm can also be seen as an instance of Bellman's dynamic programming algorithm [11] for a suitably defined optimal control problem [12], [13]. In a recent paper [14], we derived different forms for temporal parallelisation of Bellman's dynamic programming for optimal control. In this paper, the aim is to go back to the optimal control formulation of the Viterbi algorithm and specialise its parallel solutions to the trajectory estimation problem, which leads a novel parallelisation of the Viterbi algorithm. The new algorithm can be seen as the parallel version of the classical Viterbi with a backward pass for trajectory recovery using stored optimal mappings and it has a logarithmic time complexity $O(\log T)$.

II. The Viterbi algorithm as the solution to an optimal control problem

A. MAP trajectory estimation as an optimal control problem

We can write the MAP trajectory estimation problem in (2) as an instance of Bellman's solution to an optimal control problem [2], [12]. Taking the logarithm in (2), the MAP trajectory estimation problem minimises

$$C[x_{0:T}] = -\log p(x_0) - \sum_{k=1}^{T} \log[p(y_k \mid x_k)\, p(x_k \mid x_{k-1})].$$

If we time-reverse the system by defining $z_n = x_{T-n}$, then the above cost function becomes

$$C[z_{0:T}] = -\log p(z_T) - \sum_{n=1}^{T} \log[p(z_n \mid z_{n+1})\, p(y_{T-n} \mid z_n)].$$

We can then define the following terminal cost and the incremental costs at each time step:

$$\begin{aligned} \ell_T(z_T) &= -\log p(z_T), \\ \ell_n(z_n, z_{n+1}) &= -\log[p(z_n \mid z_{n+1})\, p(y_{T-n} \mid z_n)]. \end{aligned} \tag{6}$$

Then, the cost function is

$$C[z_{0:T}] = \ell_T(z_T) + \sum_{n=1}^{T} \ell_n(z_n, z_{n+1}). \tag{7}$$

Finally, we can define a dynamic model as

$$z_n = u_{n-1}, \tag{8}$$

which together with (7) defines an optimal control problem [13], where z_n and u_n are the state and control at time step n. It should be noted that the control problem is special because the state at time step n only depends on the control and not on previous state.

B. Bellman's solution is the Viterbi algorithm

In this section, we show that the Bellman's solution to the control problem defined by (7) and (8) corresponds to the Viterbi algorithm. The value function $\bar{V}_n(z_n)$ of the control problem can be computed with the (backward) recursion

$$\bar{V}_n(z_n) = \min_{u_n} \left\{ \ell_n(z_n, u_n) + \bar{V}_{n+1}(u_n) \right\}, \tag{9}$$

starting from $\bar{V}_T(z_T) = \ell_T(z_T)$. The value function $\bar{V}_n(z_n)$ is the cost of the trajectory if we make optimal decisions for the remaining steps up to T starting from state z_n. The minimization in (9) also defines the optimal decision function

$$\bar{u}_n^*(z_n) = \arg\min_{u_n} \left\{ \ell_n(z_n, u_n) + \bar{V}_{n+1}(u_n) \right\}, \tag{10}$$

which can be stored during the backward pass. The optimal trajectory can then be recovered by starting from $z_0^* = \arg\min \bar{V}_0(z_0)$ and proceeding forward by

$$z_n^* = \bar{u}_{n-1}^*(z_{n-1}^*). \tag{11}$$

This is the approach that will correspond to the novel parallel formulation that we consider in this paper.

An alternative approach would be to make an additional forward computation pass for the value functions and to combine the backward and forward value functions by minimizing their sum. That corresponds to the max-product formulation of the Viterbi algorithm [7]–[9].

C. Relationship with classical Viterbi

Let us now see how we can recover the classic Viterbi algorithm in (3)–(5) from the optimal control formulation. Plugging $z_{n+1} = u_n$ into (9) yields

$$\bar{V}_n(z_n) = \min_{z_{n+1}} \left\{ \ell_n(z_n, z_{n+1}) + \bar{V}_{n+1}(z_{n+1}) \right\}. \tag{12}$$

Using (6), $z_k = x_{T-k}$, and defining $V_k(x_k) = \bar{V}_{T-k}(x_{T-k})$, we obtain a forward pass

$$\begin{aligned} V_k(x_k) &= \min_{x_{k-1}} \big\{ -\log[p(y_k \mid x_k)\, p(x_k \mid x_{k-1})] \\ &\qquad + V_{k-1}(x_{k-1}) \big\}, \\ u_k^*(x_k) &= \arg\min_{x_{k-1}} \big\{ -\log[p(y_k \mid x_k)\, p(x_k \mid x_{k-1})] \\ &\qquad + V_{k-1}(x_{k-1}) \big\}, \end{aligned} \tag{13}$$

with $V_0(x_0) = -\log p(x_0)$. The trajectory recovery then starts from $x_T^* = \arg\min_{x_T} V_T(x_T)$ and it proceeds as

$$x_{k-1}^* = u_k^*(x_k^*). \tag{14}$$

By defining $W = \exp(-V)$ and using the property that $p(y_k \mid x_k)$ does not depend on x_{k-1}, this recursion can be written as the Viterbi recursions (3)–(5).

III. Direct Temporal Parallelisation of the Control Problem

The solution to the optimal control problem defined in Section II-A can be parallelised across time in two ways [14]. The first approach obtains the forward and backward value functions and then combines them. This approach is equivalent to the max-product approach to the Viterbi algorithm [9]. The second approach corresponds to a backward value function pass and a forward trajectory recovery pass, which corresponds to the new parallel Viterbi algorithm we propose in this paper.

A. All-prefix-sums operation

Parallelisation across the temporal domain is achieved via the all-prefix sums operation [10]. In general, given the elements $a_1, \ldots, a_T$ and an associative operator $\otimes$ on them, which may be a sum or a multiplication, or the combination of value functions [14], the all-prefix-sum operation computes

$$\begin{aligned} s_1 &= a_1, \\ s_2 &= a_1 \otimes a_2, \\ &\cdots \\ s_T &= a_1 \otimes a_3 \otimes \cdots \otimes a_T. \end{aligned} \tag{15}$$

The all-prefix-sums operation can be computed in $O(\log T)$ time by making use of parallel scans [10].

B. Conditional value functions and combination rule

Let us now take a look at the parallel solution to the optimal control problem defined in Section II-A. In optimal control, the elements $\bar{a}_1, \ldots, \bar{a}_T$ take the form of conditional value functions, which are defined as follows.

Definition 1: The conditional value function $\bar{V}_{k\to i}(z_k, z_i)$ is the cost of the optimal trajectory starting from z_k and ending at z_i [14]:

$$\bar{V}_{k\to i}(z_k, z_i) = \min_{u_{k:i-1}} \sum_{n=k}^{i-1} \ell_n(z_n, u_n), \tag{16}$$

subject to the dynamic constraint of the control problem

$$z_n = f_{n-1}(z_{n-1}, u_{n-1}) \quad \forall n \in \{k+1, ..., i\}. \tag{17}$$

If it is not possible to reach z_i from z_k with the dynamics (17), then $\bar{V}_{k\to i}(z_k, z_i) = \infty$.

Two consecutive value functions $\bar{V}_{k\to j}(z_k, z_j)$ and $\bar{V}_{j\to i}(z_j, z_i)$ can be combined with the associative operator $\otimes$, which is defined as follows [14]:

$$\begin{aligned} \bar{V}_{k\to i}(z_k, z_i) &= \bar{V}_{k\to j}(z_k, z_j) \otimes \bar{V}_{j\to i}(z_j, z_i) \\ &= \min_{z_j} \left\{ \bar{V}_{k\to j}(z_k, z_j) + \bar{V}_{j\to i}(z_j, z_i) \right\}, \end{aligned} \tag{18}$$

for $k < j < i \leq T$.

C. Definition of the elements

The elements of the all-prefix-sums operation that will enable us to solve the optimal control problem in $O(\log T)$ are defined in the following theorem [14].

Theorem 2: Let the k-th element $\bar{a}_k$ be

$$\bar{a}_k = \bar{V}_{k\to k+1}(z_k, z_{k+1}), \tag{19}$$

for $k = 0, \ldots, T$, where $\bar{V}_{T\to T+1}(z_T, z_{T+1}) \triangleq \bar{V}_T(z_T)$, then

$$\bar{a}_k \otimes \bar{a}_{k+1} \otimes \cdots \otimes \bar{a}_T = \bar{V}_k(z_k). \tag{20}$$

where the operator $\otimes$ is defined in (18).

D. Optimal trajectory recovery

This section reviews Method 1 of trajectory recovery in [14], as it results in the proposed parallel MAP trajectory estimation method. First, the optimal control law is obtained in parallel for all time steps using

$$\bar{u}_n(z_n) = \arg\min_{u_n} \left\{ \ell_n(z_n, u_n) + \bar{V}_{n+1}(f_n(z_n, u_n)) \right\}. \tag{21}$$

The optimal state z^*_{n+1} at time step $n+1$ given the optimal state z^*_n at time step k can then be found by

$$z^*_{n+1} = f_n(z^*_n, u_n(z^*_n)) = f^*_n(z^*_n). \tag{22}$$

Starting at time step $n = 0$, the optimal trajectory can then be recovered by the composition of these functions

$$z^*_n = \left(f^*_{n-1} \circ \ldots \circ f^*_1 \circ f^*_0\right)(z^*_0). \tag{23}$$

We can now implement (23) using all-prefix-sums and parallel scans. The element of the all-prefix-sums algorithm is the function of the state $f^*_a(\cdot)$. Then, the associative operator for trajectory recovery for two elements $a = f^*_a(\cdot)$ and $b = f^*_b(\cdot)$ is the composition of the functions:

$$a \otimes b \triangleq f^*_b \circ f^*_a. \tag{24}$$

IV. Improved Temporal Parallelisation of the Viterbi Algorithm

In this section we present a more efficient way to parallelise the Viterbi algorithm. In Section III, we presented the direct parallelisation in terms of its equivalent optimal control problem, which was written for a general dynamic function $z_n = f_{n-1}(z_{n-1}, u_{n-1})$. In this section, we explain how the expressions simplify when we use the specific dynamic model used in the Viterbi algorithm $z_n = u_{n-1}$ and also revert the time back to the original Viterbi direction.

A. Specialisation of the parallel solution to Viterbi

For the initialisation of the elements, see Theorem 2, we need to evaluate

$$\bar{V}_{k\to k+1}(z_k, z_{k+1}) = \min_{u_n} \ell_n(z_n, u_n), \quad z_{n+1} = u_n. \tag{25}$$

With the Viterbi dynamic model this expression simplifies to

$$\bar{V}_{k\to k+1}(z_k, z_{k+1}) = \ell_k(z_k, z_{k+1}). \tag{26}$$

If we reverse the time and use (6), we get that the initialization in terms of the Viterbi states reduces to

$$\begin{aligned} a_k &= V_{k\to k-1}(x_k, x_{k-1}) = -\log[p(y_k \mid x_k)\, p(x_k \mid x_{k-1})], \\ a_0 &= V_{0\to -1}(x_0, x_{-1}) = -\log p(x_0). \end{aligned} \tag{27}$$

Therefore, we can see that we do not need to solve an optimisation problem to initialise the elements, which saves computational time. Once the elements are initialised, we can apply the parallel scan algorithm to compute the value functions, using Theorem 2. The associative combination rule (18) still remains in the same form

$$\begin{aligned} V_{k\to i}(x_k, x_i) &= V_{k\to j}(x_k, x_j) \otimes V_{j\to i}(x_j, x_i) \\ &= \min_{x_j} \left\{ V_{k\to j}(x_k, x_j) + V_{j\to i}(x_j, x_i) \right\}, \end{aligned} \tag{28}$$

but now the parallel scan should be run forward (instead of backwards) as we now have

$$a_0 \otimes a_1 \otimes \cdots \otimes a_k = V_k(x_k). \tag{29}$$

The initialization of the function for trajectory recovery, using the optimal control, which is given by (22), becomes

$$\begin{aligned} \bar{f}^*_k(x_k) = \arg\min_{x_{k-1}} \big\{ &-\log[p(y_k \mid x_k)\, p(x_k \mid x_{k-1})] \\ &+ V_{k-1}(x_{k-1}) \big\}. \end{aligned}$$

Once we have initialised the functions for optimal trajectory recovery, we can use their composition to recover the optimal trajectory using parallel scans, see (23). However, now the parallel scan for the function decomposition should be done backwards in time because we have

$$x^*_k = (\bar{f}^*_{k+1} \circ \cdots \circ \bar{f}^*_{T-1} \circ \bar{f}^*_T)(x^*_T), \tag{30}$$

where $x^*_T = \arg\min_{x_T} V_T(x_T)$.

B. Memory usage and hybrid algorithm

It is relevant to note that the Viterbi algorithm requires the storage of a $D \times 1$ vector $W_k(x_k)$ at each time step k. Each step of the algorithm requires the combination of two $D \times 1$ vectors and a $D \times D$ matrix, see (3) and (13).

In contrast, the value function pass of the parallel version requires storing conditional value functions, each stored in a $D \times D$ matrix, and the combination rule requires processing two $D \times D$ matrices. Therefore, it has higher storage requirements, which may not be available for systems with large D. However, the function combination pass of the parallel version only requires combination of two $D \times 1$ vectors and therefore has much lower memory requirements.

In this setting, one option is to use a hybrid algorithm where we run the (sequential) Viterbi algorithm forward, and then recover the optimal trajectory backwards using parallel scans. With this approach the memory requirements remain moderate while the trajectory recovery pass is still computed in parallel.

V. Experimental Results

A. Gilbert–Elliot channel model experiment

In this experiment we test the proposed algorithm using the Gilbert–Elliot (GE) model which was used to evaluate the performance of the max-product-based parallel Viterbi algorithm in [9]. The GE model is a classic model for correction of error bursts in communications channels. The state of the system is encoded as the values $x_k \in \{1, 2, 3, 4\}$ and the measurements take values $y_k \in \{1, 2\}$. For details of the model, see [6], [9]. Example measurements, state sequences, and the result of Viterbi algorithm applied to the model are shown in Fig. 1. All the tested Viterbi algorithms produce essentially the same estimation result which is the one shown in the figure.

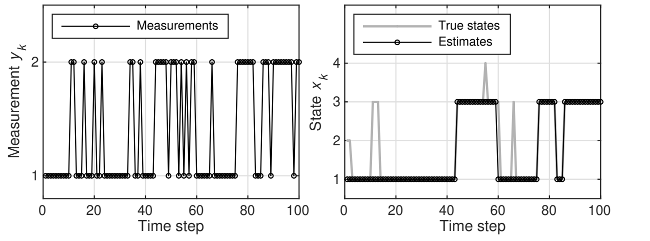

Fig. 1. Illustration of Gilbert–Elliot experiment. Left: simulated measurement sequence. Right: simulated state sequence and the Viterbi estimates.

We tested the following algorithms on a multi-core CPU and a graphics processing unit (GPU):

- *Sequential FW/BW*: A sequential Viterbi algorithm (the variant storing the optimal mappings).
- *Seq/par. FW/BW*: A hybrid algorithm discussed in Section IV-B which uses the sequential forward pass in combination with parallel backward pass.
- *Parallel FW/BW*: A parallel Viterbi algorithm which uses parallel forward and backward passes as described in Section IV-A.
- *Parallel MP*: A max-product-based Viterbi algorithm proposed in [9] (implemented in log-domain).

The CPU was a AMD EPYC 7643 48-core processor with 512GB of memory, and GPU was a NVIDIA A100-SXM GPU with 80GB of memory. The algorithms were implemented using TensorFlow.

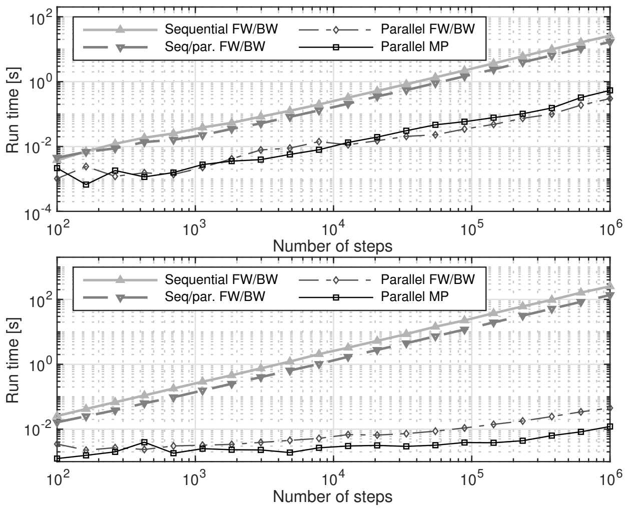

Fig. 2. Results of the experiment on the Gilbert–Elliot model. The CPU results are shown on top and the GPU results are at the bottom figure.

Fig. 2 shows the results of the experiment, where we ran the algorithms with increasing number of time steps 10^2–10^6 (repeated 10 times each). It can be seen that both of the parallel methods are significantly faster than the sequential and hybrid algorithms. In this case the max-product algorithm is slightly faster than the proposed forward-backward algorithm although in the CPU the order of the methods varies. The hybrid algorithm is consistently faster than the pure sequential algorithm both on CPU and GPU.

B. Text correction experiment

As the second experiment we consider automatic correction of errors in natural text. This experiment has a significantly higher number of states than the GE model in the previous section. The text excerpts are from the textbook [6] and they have been stripped and converted to only contain 26 alphabetical characters and the space symbol. We used the body text from the preface and Chapters 1–10 as the training data for estimation of the transition probabilities for 1st, 2nd, and 3rd order Markov models of the text. We then added 10% of independent random errors, each consisting in replacing a character with a random character, to the texts of the Chapters 11–17 and Appendix. The data is illustrated in Fig. 3.

The whole training data had the length of $68\,197$ characters and the test data $64\,620$ characters. The number of states in the 1st order Markov model was 27, in the 2nd order Markov model $27^2 = 729$, and in the 3rd order Markov model $27^3 = 19\,683$. Therefore the performance of the methods is heavily affected by their scaling with the number of states.

Table I shows the results of running the methods listed in Section V-A over the whole test set. The times were computed as the means from 10 runs although the run times were quite

Train data:
```
the aim of this book is to give a concise introduction to
non linear kalman filtering and smoothing particle filter
ing and smoothing and to the related parameter ...
```
Test data without errors:
```
although in many filtering problems gaussian approximatio
ns work well sometimes the filtering distributions can be
 for example multi modal or some of the state ...
```
Test data with errors:
```
altfoughzin mmny filteving projlems gauhsyan approximatbo
ns work well smmetimeb tpe bilzering jistrirutions can be
 fovxexampllllmulti modal or eome om the statc ...
```

Fig. 3. The data used in the text correction experiment. We estimated the transition probabilities of 1st, 2nd, and 3rd order Markov models from the training data (top). We then added 10% of random errors to the test text (middle) which resulted in the corrupted text (below). The Viterbi algorithm was then used to correct the errors.

TABLE I
RESULTS OF THE TEXT CORRECTION EXPERIMENT.

	CPU			GPU		
Method \ Order	1st	2nd	3rd	1st	2nd	3rd
Sequential FW/BW	1.9s	38.8s	-	16.9s	22.9s	1.6e3s
Seq/par. FW/BW	1.3s	36.4s	-	8.8s	15.3s	1.5e3s
Parallel FW/BW	2.5s	-	-	0.07s	-	-
Parallel MP	4.7s	-	-	0.08s	-	-

consistent over the runs. With the 1st order model (with 27 states), the speed differences of the methods are quite small on CPU, but on GPU the parallel methods are two orders of magnitude faster than the sequential ones. Furthermore, on the CPU the hybrid algorithm is slightly faster than the pure sequential method whereas on GPU the hybrid algorithm is almost two times faster than the sequential one. On CPU the parallel forward-backward method is faster than the max-product based parallel method. On GPU this is also the case, but the difference is much smaller.

With the 2nd order model with 729 states is no longer computable due to excessive memory use and the amount of computations needed on the forward pass of the parallel algorithm (cf. Sec. IV-B). However, the sequential and hybrid methods are still applicable and their performance is quite much equal on CPU. On GPU the hybrid algorithm is significantly faster. The 3rd order model takes an excessive amount of time on CPU and therefore we could only run sequential and hybrid methods on GPU for it. In this case the performances of the sequential and hybrid methods are quite much equal.

For completeness, we also show the results for the 1st order Markov model with increasing amount of test data. The results are shown in Fig. 4. The results confirm the conclusions drawn from Table I. However, it can be seen that the forward-backward method only starts to be faster than the max-product method with larger numbers of steps and with smaller numbers of steps the max-product method is slightly faster.

VI. Conclusion and discussion

We have presented a new method to parallelise the Viterbi algorithm in temporal direction. The method was derived via reformulating the Viterbi algorithm as an instance of an optimal control problem and parallelising it with a method adapted from [14]. The proposed method was experimentally tested on multi-core CPU and GPU, and the result show that it is competitive with and even better than the previously proposed max-product based parallelisation method [9]. The results also show that a hybrid algorithm combining sequential forward pass with parallel backward pass of the proposed method can outperform the sequential method in large state spaces where the fully parallel methods are not applicable.

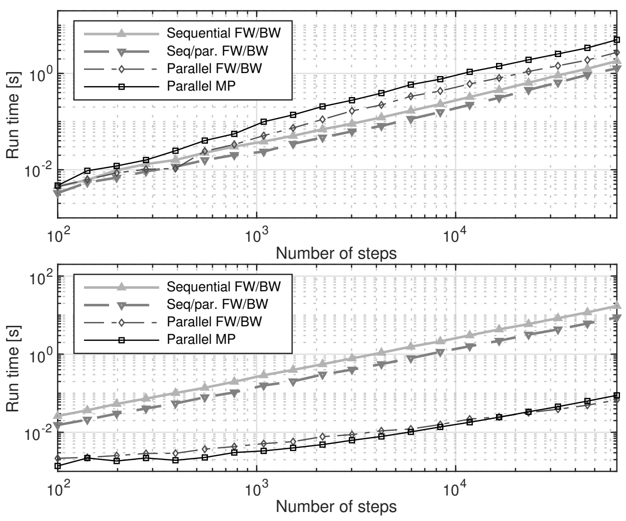

Fig. 4. Results for text experiment with 1st order Markov model (27 states). The CPU results are on top and the GPU results at the bottom.

References

[1] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, 1967.

[2] R. E. Larson and J. Peschon, "A dynamic programming approach to trajectory estimation," *IEEE Transactions on Automatic Control*, vol. 11, no. 3, pp. 537–540, 1966.

[3] G. D. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, March 1973.

[4] T. Yamada, S. Nakamura, and K. Shikano, "Distant-talking speech recognition based on a 3-D Viterbi search using a microphone array," *IEEE Trans. Speech and Audio Processing*, vol. 10, no. 2, pp. 48–56, 2002.

[5] Y. You and T. J. Oechtering, "Hidden Markov model based data-driven calibration of non-dispersive infrared gas sensor," in *28th European Signal Processing Conference*, 2021, pp. 1717–1721.

[6] S. Särkkä and L. Svensson, *Bayesian Filtering and Smoothing*, 2nd ed. Cambridge University Press, 2023.

[7] H. Wymeersch, *Iterative Receiver Design*. Cam. Univ. Press, 2007.

[8] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[9] S. Hassan, S. Särkkä, and A. F. García-Fernández, "Temporal parallelization of inference in hidden Markov models," *IEEE Transactions on Signal Processing*, vol. 69, pp. 4875–4887, 2021.

[10] G. E. Blelloch, "Scans as primitive parallel operations," *IEEE Transactions on Computers*, vol. 38, no. 11, pp. 1526–1538, 1989.

[11] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[12] J. Omura, "On the Viterbi decoding algorithm," *IEEE transactions on information theory*, vol. 15, no. 1, pp. 177–179, 1969.

[13] F. L. Lewis and V. L. Syrmos, *Optimal Control*, 2nd ed. Wiley, 1995.

[14] S. Särkkä and A. F. García-Fernández, "Temporal parallelization of dynamic programming and linear quadratic control," *IEEE Transactions on Automatic Control*, vol. 68, pp. 851–866, 2023.