Temporal Parallelization of Inference in Hidden Markov Models

Syeda Sakira Hassan, *Member, IEEE,* Simo Särkkä, *Senior Member, IEEE,* and Ángel F. García-Fernández

***Abstract*—This paper presents algorithms for the parallelization of inference in hidden Markov models (HMMs). In particular, we propose a parallel forward-backward type of filtering and smoothing algorithm as well as a parallel Viterbi-type maximum-a-posteriori (MAP) algorithm. We define associative elements and operators to pose these inference problems as all-prefix-sums computations and parallelize them using the parallel-scan algorithm. The advantage of the proposed algorithms is that they are computationally efficient in HMM inference problems with long time horizons. We empirically compare the performance of the proposed methods to classical methods on a highly parallel graphics processing unit (GPU).**

***Index Terms*—Parallel forward-backward algorithm, parallel sum-product algorithm, parallel max-product algorithm, parallel Viterbi algorithm.**

I. Introduction

HIDDEN Markov models (HMMs) have gained a lot of attention due to their simplicity and a broad range of applications [1]–[5]. Successful real-world application areas of HMMs include speech recognition, convolutional code decoding, target tracking and localization, facial expression recognition, gene prediction, gesture recognition, musical composition, and bioinformatics [1], [6]–[11]. An HMM is a statistical model that provides a simple and flexible framework that can be used to express the conditional independence and joint distributions using graph-like structures. An HMM can be thought of as a specific form of a probabilistic graphical model consisting of two components: a structural component that defines the *edges* and a parametric component that encodes *potentials* associated with the edges in the graph. A graphical representation of an HMM is shown in Fig. 1. An HMM is a doubly stochastic process, where the underlying stochastic process (light gray-colored nodes) can be only observed through another stochastic process (dark gray-colored nodes) [1], [12]. A primary task in graphical models is to perform inference, that is, to compute marginals or the most likely values of the unobserved modes given the observed modes.

If the HMM model has D states and the length of the sequence is T, then there are D^T possible state sequences. For the forward-backward algorithm, whose objective is to find the marginal distributions, and for the Viterbi algorithm, whose objective is to find the most likely sequence (the Viterbi path), the time complexity is $O(D^2T)$ [1], [6], [13]. Some methods to speed up these algorithms via parallelization have appeared in literature. For instance, in order to speed up the inference task, Lifshits et al. [14] used a compression scheme, which utilizes the repetitive patterns in the observed sequences. Sand et al. [15] developed a parallel forward algorithm using single instruction multiple data (SIMD) processors and multiple cores. Nielsen and Sand [16] presented a parallel reduction on the forward algorithm. Chatterjee and Russell [17] used the temporal abstraction concept from dynamic programming to speed up the Viterbi algorithm. Using accelerated hardware, a tile-based Viterbi algorithm was proposed by Zhihui et al. [18], where the matrix multiplication was done in parallel. Maleki et al. [19] proposed an optimized method that is able to solve a particular class of dynamic programming problems by tropical semirings. They showed that it can be used to optimize the Viterbi algorithm. Nevertheless, parallelization has not been fully investigated in HMM inference tasks. In this paper, we develop novel parallel versions of the forward-backward algorithm and the Viterbi algorithm that have $O(\log T)$ span complexities.

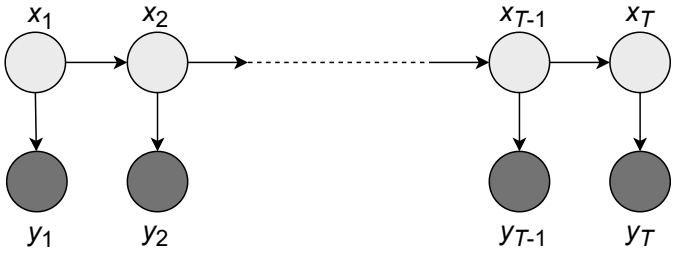

Fig. 1. An HMM diagram. Observed nodes are shaded in dark gray, whereas unobserved nodes are shaded in light gray.

Parallel algorithms can take advantage of the computational power of specialized hardware accelerators, such as general purpose *graphics processing units* (GPUs) [20], *neural processing units* (NPUs) [21], or *tensor processing units* (TPUs) [22]. These accelerators allow us to perform the parallel computation on a large amount of data, making computation faster and cheaper and, therefore, economically viable. Among these accelerators, GPUs are the most widely used alternatives. GPU architectures enable us to harness the massive parallelization of general-purpose algorithms [23]. Table I summarizes relevant works on the HMM inference tasks, which were implemented on GPUs. However, these works were optimized particularly for speech recognition tasks and did not explore the full capabilities of parallelism. In

S. Hassan and S. Särkkä are with the Department of Electrical Engineering and Automation, Aalto University, 02150 Espoo, Finland (emails: syeda.s.hassan@aalto.fi, simo.sarkka@aalto.fi).

Ángel F. García-Fernández is with the Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool L69 3GJ, United Kingdom (email: angel.garcia-fernandez@liverpool.ac.uk). He is also with the ARIES Research Centre, Universidad Antonio de Nebrija, Madrid, Spain.

order to utilize parallelism, sequential algorithms need to be reformulated in terms of primitive operations that enable us to perform the parallel execution on parallel platforms. For example, the parallel-scan algorithm [24], [25] can be used to run computations in parallel provided that they can be formulated in terms of binary associative operators.

TABLE I
PREVIOUS WORKS ON THE HMM INFERENCE TASK USING GPUS. THE NOTATION '-' MEANS THAT THE VALUE WAS NOT MENTIONED IN THE REFERENCED ARTICLE.

Algorithm	States	Observations	Speed improvement
Forward-backward [26]	8	200	3.5x
Forward [27]	512	3-10	880x
Baum-Welch [27]	512	3-10	180x
Viterbi [28]	-	2000-3000	3x
Forward [29]	4000	1000	4x
Baum-Welch [29]	4000	1000	65x

Recently, the parallel-scan algorithm has been used in parallel Bayesian filtering and smoothing algorithms to speed up the sequential computations via parallelization [30], [31]. Although this framework is applicable to HMMs as well, the difference to our proposed framework is that here we formulate the inference problem in terms of potentials. This results in a different backward pass to compute the posterior marginals which corresponds to a two-filter smoother formulation, whereas the formulation in Ref. [30], [31] is a Rauch–Tung–Striebel type of smoother [32]. In this article, we also present a parallel Viterbi-type maximum-a-posteriori (MAP) algorithm which has not been explored before.

The main contribution of this paper is to present a parallel framework to perform the HMM inference tasks efficiently by using the parallel-scan algorithm. In particular, we formulate the sequential operations of sum-product and max-product algorithms as binary associative operations. This formulation allows us to construct parallel versions of the forward-backward algorithm and the Viterbi algorithm with $O(\log T)$ span complexities. For the latter algorithm, we propose two alternative parallelization approaches: a path-based and a forward-backward based formulation. We also empirically evaluate the computational speed advantage of these methods on a GPU platform.

The structure of the paper is the following. In Section II, we define the HMM inference problems using a probabilistic graph. Then, in Section III, we review the classical sum-product and forward-backward algorithms, introduce the elements and associative operations for parallel operations, and formulate the parallel-scan algorithm on these elements and operators. Next, in Section IV, we show that we can apply the parallel-scan algorithm to the max-product algorithm, which results in a parallel version of the Viterbi algorithm. In Section V, we discuss extensions and generalizations of the methodology, and in Section VI, we present experimental results for both parallel sum-product and parallel max-product based algorithms. Section VII concludes the article.

II. PROBLEM FORMULATION

Assume that we have T discrete random variables $\mathbf{x} = (x_1, \ldots, x_T)$ which correspond to nodes in a probabilistic graph and N potential functions $\psi_1, \ldots, \psi_N$ which define the cliques of the graph [33]–[35]. We also assume each variable x_t takes values in the set $\mathcal{X} = \{1, \ldots, D\}$, where D represents the number of states. Each potential is a function of a subset of x_t's, defined by multi-indices $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_T$ with elements $\boldsymbol{\alpha}_t = (\alpha_{t,1}, \ldots, \alpha_{t,|\boldsymbol{\alpha}_t|})$. We denote the subset as $\mathbf{x}_{\boldsymbol{\alpha}_t}$. The joint distribution of the random variables has the representation

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} \psi_t\left(\mathbf{x}_{\boldsymbol{\alpha}_t}\right), \tag{1}$$

where $Z = \sum_{\mathbf{x}} \prod_{t=1}^{T} \psi_t\left(\mathbf{x}_{\boldsymbol{\alpha}_t}\right)$ is the normalization constant, also known as the partition function [33]. A typical inference task is the computation of all the marginals

$$p(x_k) = \frac{1}{Z} \sum_{\mathbf{x} \setminus x_k} \prod_{t=1}^{T} \psi_t\left(\mathbf{x}_{\boldsymbol{\alpha}_t}\right), \tag{2}$$

where $\mathbf{x} \setminus x_k = (x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_T)$, that is, the summation is performed over all the variables except x_k. Another inference task is the computation of the similar quantity for the maximum

$$p^*(x_k) = \max_{\mathbf{x} \setminus x_k} \prod_{t=1}^{T} \psi_t\left(\mathbf{x}_{\boldsymbol{\alpha}_t}\right), \tag{3}$$

where the maximum is computed with respect to all variables in $\mathbf{x}$ except x_k.

Let us now consider inference problems in a hidden Markov model (HMM) of the form

$$x_k \sim p(x_k \mid x_{k-1}), \tag{4a}$$
$$y_k \sim p(y_k \mid x_k), \tag{4b}$$

where $\sim$ stands for "is distributed as". Here, we assume that the sequence $x_1, \ldots, x_T$ is Markovian with transition probabilities $p(x_k \mid x_{k-1})$, and the observations y_k are conditionally independent given x_k with likelihoods $p(y_k \mid x_k)$. Furthermore, we have a prior $x_1 \sim p(x_1)$. We are interested in computing the smoothing posterior distributions $p(x_k \mid y_1, \ldots, y_T)$ for all $k = 1, \ldots, T$ as well as in computing the MAP estimate (i.e., the Viterbi path) of $p(\mathbf{x})$. These can be computed using sequential algorithms in a linear computational time [13], [32], [36]. However, our aim is to reduce this computational time by using parallelization in the temporal domain.

We can express the inference problem in Eq. (1) by defining

$$\psi_1(x_1) = p(y_1 \mid x_1)\, p(x_1), \tag{5a}$$
$$\psi_k(x_{k-1}, x_k) = p(y_k \mid x_k)\, p(x_k \mid x_{k-1}), \quad \text{for } k > 1. \tag{5b}$$

With these definitions the joint distribution in Eq. (1) takes the form

$$p(\mathbf{x}) = \frac{1}{Z} \psi_1(x_1) \prod_{t=2}^{T} \psi_t(x_{t-1}, x_t). \tag{6}$$

In this Markovian case, each potential is a function of the neighboring nodes x_{k-1}, x_k, that is, we have $\boldsymbol{\alpha}_k = (k-1, k)$ for $k > 1$ and $\boldsymbol{\alpha}_1 = (1)$.

Typically, the inference task in the HMM is either to compute the marginals $p(x_k \mid y_1, \ldots, y_T)$ or to find the

MAP sequence $x^*_{1:T}$, the Viterbi path. In the potential function formulation, the smoothing distribution is given by Eq. (2) and the maximum of Eq. (3) provides the Viterbi path at step k. Both of these correspond to the following kinds of general operations on Eq. (6):

$$F(x_k) = \frac{1}{Z}\,\mathrm{OP}_k\left(\psi_1(x_1)\prod_{t=2}^{T}\psi_t(x_{t-1},x_t)\right), k=1,\ldots,T, \tag{7}$$

where $\mathrm{OP}_k(.)$ is a sequence of operations applied to all elements but x_k, such as $\sum_{\mathbf{x}\setminus x_k}$ or $\max_{\mathbf{x}\setminus x_k}$, resulting in a function of x_k. One way to compute these operations in Eq. (7) is to use sum-product or max-product algorithms [37]–[39], for summation and maximization operations, respectively. However, in the following section we show that, since the summation and the maximum operations are associative, we can use parallel-scan algorithm for parallelizing these computations. The same principle would also apply to any other binary associative operation, but here we specifically concentrate on these two operations.

III. Parallel sum-product algorithm for HMMs

In this section, we present a parallel formulation of the sum-product algorithm. In Section III-A, we review the classical sum-product algorithm. In Section III-B, we revisit the parallel-scan algorithm. In Section III-C, we show the decomposition of the sum-product algorithm in terms of the binary associative operations. Finally, in Section III-D, we propose the parallel sum-product algorithm with $O(\log T)$ span complexity.

A. Classical sum-product algorithm

The sum-product algorithm can be used to find the marginal distributions of all variables in a graph [40], [41]. We first define the forward potential as

$$\psi^f_{1,k}(x_k) = \sum_{x_{1:k-1}}\psi_1(x_1)\prod_{t=2}^{k}\psi_{t-1,t}(x_{t-1},x_t) \tag{8}$$

and the backward potential as

$$\psi^b_{k,T}(x_k) = \sum_{x_{k+1:T}}\prod_{t=k}^{T}\psi_{t,t+1}(x_t,x_{t+1}). \tag{9}$$

Here, we have defined $\psi_{T,T+1}(x_T,x_{T+1}) = 1$ and denoted $\sum_{x_{1:k}} = \sum_{x_1}\ldots\sum_{x_k}$. We can now express the marginal distribution $p(x_k)$, which corresponds to summation operation in Eq. (7), as a normalized product of the forward and backward potentials

$$p(x_k) = \frac{1}{Z_k}\,\psi^f_{1,k}(x_k)\,\psi^b_{k,T}(x_k), \tag{10}$$

where $Z_k = \sum_{x_k}\psi^f_{1,k}(x_k)\,\psi^b_{k,T}(x_k)$. If we want to compute all the marginals $p(x_1)$, $p(x_2),\ldots,p(x_T)$, then we need to compute all the terms $\psi^f_{1,k}(x_k)$ and $\psi^b_{k,T}(x_k)$ and combine them. It turns out that we can compute these forward and backward potentials in $O(D^2\,T)$ steps using Algorithm 1, which is an instance of a sum-product algorithm.

Algorithm 1 The classical sum-product algorithm for computing the forward and backward potentials.

Input: The potentials $\psi_k(\cdot)$ for $k=1,\ldots,T$.
Output: The forward and backward potentials $\psi^f_{1,k}(x_k)$ and $\psi^b_{k,T}(x_k)$ for $k=1,\ldots,T$.

// Forward pass:
$\psi^f_{1,1}(x_1) = \psi_1(x_1)$ ▷ Initialization
for $k \leftarrow 2$ **to** T **do** ▷ Sequentially
  $\psi^f_{1,k}(x_k) = \sum_{x_{k-1}}\psi^f_{1,k-1}(x_{k-1})\,\psi_{k-1,k}(x_{k-1},x_k)$
end for
// Backward pass:
$\psi^b_{T,T}(x_T) = 1$ ▷ Initialization
for $k \leftarrow T-1$ **to** 1 **do** ▷ Sequentially
  $\psi^b_{k,T}(x_k) = \sum_{x_{k+1}}\psi_{k,k+1}(x_k,x_{k+1})\,\psi^b_{k+1,T}(x_{k+1})$
end for

It should be noted that the belief propagation algorithm [37], operating on a Bayesian network, corresponds to the sum-product algorithm in a factor graph with similar factorization. The forward algorithm of the HMM model is equivalent to filtering, whereas the backward algorithm corresponds to the backward pass in two-filter smoothing [32].

B. Parallel-scan algorithm

In this section, we revisit the parallel-scan algorithm. The parallel-scan algorithm [24], [42] is a general framework to compute the generalized all-prefix-sums of associative operators in parallel. It was originally designed for computing all-prefix-sums [42] using the summation operator. Later, it was generalized to other associative operators [24], [25] and became a fundamental building block for solving many sequential algorithms in parallel including sorting, linear programming, and graph algorithms.

The generalized all-prefix-sums operation for an operator $\otimes$ is defined as follows.

Definition 1. *For a sequence of T elements $(a_1,a_2,\ldots,a_T)$ and a binary associative operator $\otimes$, the all-prefix-sums operation computes the following sequence of length T:*

$$(a_1, a_1\otimes a_2, \cdots, a_1\otimes a_2\otimes\ldots\otimes a_T). \tag{11}$$

Similarly to the above, we can define the reversed all-prefix-sums as follows.

Definition 2. *For a sequence of T elements $(a_1,a_2,\ldots,a_T)$ and a binary associative operator $\otimes$, the reversed all-prefix-sums operation computes the following sequence of length T:*

$$(a_1\otimes a_2\otimes\cdots\otimes a_T,\ldots,a_{T-1}\otimes a_T, a_T). \tag{12}$$

The all-prefix-sums operation can be computed through the parallel-scan algorithm [24], [25] in $O(\log T)$ span complexity. This algorithm computes only the non-reversed all-prefix-sums. However, it is also possible to compute the reversed all-prefix-sums. This can be achieved by reversing the inputs before performing the parallel-scan algorithm and reversing

the outputs after the operation is performed. In addition to these, we also need to reverse the operation inside the algorithm. A pseudocode for the parallel-scan is provided by Algorithm 2.

Algorithm 2 The parallel-scan algorithm for in-place transformation of the sequence (a_k) into its all-prefix-sums in $O(\log T)$ span complexity. Note that the algorithm in this form assumes that T is a power of 2, but it can easily be generalized to an arbitrary T.

Input: The elements a_k for $k = 1, \ldots, T$ and the operator $\otimes$.
Output: The prefix sums in a_k for $k = 1, \ldots, T$.

// Save the input:
for $i \leftarrow 1$ **to** T **do** ▷ Compute in parallel
  $b_i \leftarrow a_i$
end for
// Up sweep:
for $d \leftarrow 0$ **to** $\log_2 T - 1$ **do**
  for $i \leftarrow 0$ **to** $T - 1$ **by** 2^{d+1} **do** ▷ Compute in parallel
    $j \leftarrow i + 2^d$
    $k \leftarrow i + 2^{d+1}$
    $a_k \leftarrow a_j \otimes a_k$
  end for
end for
$a_T \leftarrow 0$ ▷ Here, 0 is actually the neutral element for $\otimes$
// Down sweep:
for $d \leftarrow \log_2 T - 1$ **to** 0 **do**
  for $i \leftarrow 0$ **to** $T - 1$ **by** 2^{d+1} **do** ▷ Compute in parallel
    $j \leftarrow i + 2^d$
    $k \leftarrow i + 2^{d+1}$
    $t \leftarrow a_j$
    $a_j \leftarrow a_k$
    $a_k \leftarrow a_k \otimes t$
  end for
end for
// Final pass:
for $i \leftarrow 1$ **to** T **do** ▷ Compute in parallel
  $a_i \leftarrow a_i \otimes b_i$
end for

The fundamental idea of the parallel-scan algorithm is to reorder the computations by using the associative property of the operator so that the resulting independent subproblems can be computed in parallel. The parallel-scan algorithm consists of two parts: *up-sweep* and *down-sweep*. The algorithm can be thought of as two traversals in a balanced binary tree. The first pass is the up-sweep that starts from leaves and ends at the root. The second pass is the down-sweep in reverse direction, which starts from the root and ends at leaves. Then, an additional pass is used to form the final result. Because a binary tree with T leaves has the depth of $\log T$, the span complexity of the algorithm is $O(\log T)$. Our aim is to define the associative operators and elements corresponding to the sum-product and max-product algorithms and use the parallel-scan framework to parallelize the computations.

C. Sum-product algorithm in terms of associative operations

We can formulate the sum-product algorithm in a more abstract form by defining a general element $a_{i:k}$ and considering a binary associative operator $\otimes$ such that [25]

$$a_{i:k} = a_{i:j} \otimes a_{j:k}, \text{ for } i < j < k. \tag{13}$$

As it is shown in the following, the computation of the forward and backward terms reduces to computing $a_{0:k}$ and $a_{k:T+1}$.

Definition 3. *We define an element $a_{i:k}$ recursively as follows. We have*

$$\begin{aligned} a_{0:1} &= \psi_1(x_1), \\ a_{k-1:k} &= \psi_k(x_{k-1}, x_k), \\ a_{T:T+1} &= 1. \end{aligned} \tag{14}$$

For notational convenience and to enhance readability, we also define

$$\begin{aligned} \psi_{0,1}(x_0, x_1) &\triangleq \psi_1(x_1), \\ \psi_{k-1,k}(x_{k-1}, x_k) &\triangleq \psi_k(x_{k-1}, x_k), \\ \psi_{T,T+1}(x_T, x_{T+1}) &\triangleq 1. \end{aligned} \tag{15}$$

Now, given two elements $a_{i:j}$ and $a_{j:k}$, the binary associative operator $\otimes$ for the forward-backward algorithm in an HMM for $0 \leq i < j < k$ is

$$a_{i:j} \otimes a_{j:k} = \sum_{x_j} \psi_{i,j}(x_i, x_j)\psi_{j,k}(x_j, x_k), \tag{16}$$

which also implies the following representation for a general element:

$$a_{i:k} = \psi_{i,k}(x_i, x_k). \tag{17}$$

Lemma 1. *The operator $\otimes$ is associative.*

Proof. See Appendix A for the proof. □

Now, we present two theorems that allow us to formulate the forward-backward algorithm in terms of the associative operations.

Theorem 1. *The forward potential is given as*

$$a_{0:k} = \psi^f_{1,k}(x_k), \quad k > 0.$$

Proof. Since the operator is associative, it is enough to prove by induction that

$$\begin{aligned} a_{0:k} &= a_{0:1} \otimes a_{1:2} \otimes \cdots \\ &\quad \otimes a_{k-2:k-1} \otimes a_{k-1:k} = \psi^f_{1,k}(x_k). \end{aligned} \tag{18}$$

Eq. (18) holds for $k = 1$ by definition of $a_{0:1}$ in Eq. (14). That is, $a_{0:1} = \psi_1(x_1)$. Then, we assume that

$$a_{0:k-1} = a_{0:1} \otimes a_{1:2} \otimes \cdots \otimes a_{k-2:k-1} = \psi^f_{1,k-1}(x_{k-1}) \tag{19}$$

holds. We need to prove that Eq. (18) holds for any $k > 1$. By applying the binary operator $\otimes$ with $a_{k-1:k}$ to the left-hand side of Eq. (19), we get

$$\begin{aligned}
&a_{0:k-1} \otimes a_{k-1:k} \\
&= \sum_{x_{k-1}} \psi^f_{1,k-1}(x_{k-1})\, \psi_{k-1,k}(x_{k-1}, x_k) \\
&= \sum_{x_{k-1}} \Bigg[\left(\sum_{x_{1:k-2}} \psi_1(x_1) \prod_{t=2}^{k-1} \psi_{t-1,t}(x_{t-1}, x_t) \right) \\
&\qquad \times \psi_{k-1,k}(x_{k-1}, x_k) \Bigg] \\
&= \sum_{x_{1:k-1}} \psi_1(x_1) \prod_{t=2}^{k} \psi_{t-1,t}(x_{t-1}, x_t) \\
&= \psi^f_{1,k}(x_k).
\end{aligned}$$

This concludes the proof. □

Theorem 2. *The backward potential is given as*

$$a_{k:T+1} = \psi^b_{k,T}(x_k), \quad k > 0.$$

Proof. We prove by induction that

$$\begin{aligned}
a_{k:T+1} &= a_{k:k+1} \otimes a_{k+1:k+2} \otimes \cdots \\
&\quad \otimes a_{T-1:T} \otimes a_{T:T+1} = \psi^b_{k,T}(x_k)
\end{aligned} \tag{20}$$

holds. Eq. (20) holds for $k = T$ by definition of $a_{T:T+1} = 1$ in Eq. (14). Then, we assume that

$$\begin{aligned}
a_{k+1:T+1} &= a_{k+1:k+2} \otimes a_{k+2:k+3} \otimes \cdots \otimes a_{T:T+1} \\
&= \psi^b_{k+1,T}(x_{k+1})
\end{aligned} \tag{21}$$

holds. We need to prove that Eq. (20) holds for any $k < T$. We start by applying the binary operator $\otimes$ with $a_{k:k+1}$ to the left-hand side of Eq. (21) to yield

$$\begin{aligned}
&a_{k:k+1} \otimes a_{k+1:T+1} \\
&= \sum_{x_{k+1}} \psi_{k,k+1}(x_k, x_{k+1})\, \psi^b_{k+1,T}(x_{k+1}) \\
&= \sum_{x_{k+1}} \Bigg[\psi_{k,k+1}(x_k, x_{k+1}) \\
&\qquad \times \left(\sum_{x_{k+2:T}} \prod_{t=k+1}^{T} \psi_{t,t+1}(x_t, x_{t+1}) \right) \Bigg] \\
&= \sum_{x_{k+1:T}} \prod_{t=k}^{T} \psi_{t,t+1}(x_t, x_{t+1}) \\
&= \psi^b_{k,T}(x_k).
\end{aligned}$$

This concludes the proof. □

Now, we can express the marginal in Eq. (10) using $a_{0:k}$ and $a_{k:T+1}$ as

$$p(x_k) = \frac{1}{Z_k}\, a_{0:k}\, a_{k:T+1}. \tag{22}$$

In the proofs above we have implicitly used the sum-product algorithm in Algorithm 1 to derive the results, which can be done in $O(TD^2)$ steps. However, because the operator is associative, we can reorder the computations by recombining the operations which is the key to parallelization. This is discussed next.

D. Parallelization of the sum-product algorithm

In this section, the aim is to enable the parallel computation of the forward and backward potentials. It follows from the previous section that the computation of the forward potentials corresponds to the computation of the all-prefix-sums operation for the associative operator $\otimes$ and elements $a_{i:i+1}$. In fact, we have that

$$\begin{aligned}
&a_{0:1} = \psi^f_{1,1}(x_1), \\
&a_{0:2} = a_{0:1} \otimes a_{1:2} = \psi^f_{1,2}(x_2), \\
&\quad \vdots \\
&a_{0:T} = a_{0:1} \otimes \cdots \otimes a_{T-1:T} = \psi^f_{1,T}(x_T).
\end{aligned} \tag{23}$$

Similarly, computing the backward potentials, which correspond to elements $\{a_{k:T}\}$ can be seen as the reversed all-prefix-sums operation.

Therefore, we can use the parallel-scan algorithm to compute the forward and backward potentials along with the marginal distributions in parallel. The pseudocode of the resulting algorithm is summarized in Algorithm 3.

Algorithm 3 The parallel sum-product algorithm.

Input: The potentials $\psi_k(\cdot)$, $k = 1, \ldots, T$ and the operator $\otimes$, see Definition 3.

Output: The marginals $p(x_k)$ for $k = 1, \ldots, T$.

1: **for** $k \leftarrow 1$ **to** T **do** ▷ In parallel
2: Initialize $a_{k-1:k}$.
3: **end for**
4: Run parallel-scan to get $a_{0:k} = \psi^f_{1,k}(x_k)$, $k = 1, \ldots, T$.
5: **for** $k \leftarrow 1$ **to** T **do** ▷ In parallel
6: Initialize $a_{k:k+1}$.
7: **end for**
8: Run reversed parallel-scan to get $a_{k:T+1} = \psi^b_{k,T}(x_k)$, $k = 1, \ldots, T$.
9: **for** $k \leftarrow 1$ **to** T **do** ▷ In parallel
10: Compute marginals $p(x_k)$ using Eq. (22).
11: **end for**

As all the steps in Algorithm 3 are either fully parallelizable, or parallelizable by means of the parallel-scan algorithm, then span and work complexities can be summarized as follows.

Proposition 1. *The parallel sum-product algorithm (Algorithm 3) has a span complexity $O(\log T)$ and a work complexity $O(T)$.*

Proof. The initializations for the elements for both the forward and backward passes as well as the marginal computations are fully parallelizable and, hence, have span complexities $O(1)$ and work complexities $O(T)$. The parallel-scan algorithm passes have span complexities $O(\log T)$ and work complexities $O(T)$. Therefore, the total span complexity is $O(\log T)$ and the total work complexity is $O(T)$. □

It is useful to remark that the computational complexity depends also on the number of states D. However, it depends on the details of how much we can parallelize inside the initializations and operator applications, which is what determines the dependence of the span complexity on D.

IV. Parallel Viterbi and max-product algorithms

In this section, we present the parallel formulation of the Viterbi and max-product algorithms. In Section IV-A, we review the classical Viterbi algorithm. In Section IV-B, we propose a parallel Viterbi algorithm based on optimal paths. In Section IV-C, we propose another alternative parallel Viterbi algorithm based on the max-product formulation.

A. Classical Viterbi algorithm

The *Viterbi algorithm* is a classical algorithm based on dynamic programming principle that computes the most likely sequence of states [2], [13], [43], also known as the MAP estimate of the hidden states. We aim to compute the estimate $x^*_{1:T}$ by maximizing the posterior distribution

$$p(x_{1:T} \mid y_{1:T}) = \frac{p(y_{1:T}, x_{1:T})}{p(y_{1:T})} \propto p(y_{1:T}, x_{1:T}). \tag{24}$$

This is equivalent to maximizing the joint probability distribution

$$p(y_{1:T}, x_{1:T}) = p(x_1)\, p(y_1 \mid x_1) \prod_{t=2}^{T} p(y_t \mid x_t)\, p(x_t \mid x_{t-1}), \tag{25}$$

which results in

$$x^*_{1:T} = \arg\max_{x_{1:T}} p(y_{1:T}, x_{1:T}). \tag{26}$$

In terms of potentials, obtaining $x^*_{1:T}$ is equivalent to the maximization of Eq. (6).

The classical Viterbi algorithm [2], [13] operates as follows. Assume that we have $V_{k-1}(x_{k-1})$ which denotes the probability of the maximum probability path $x^*_{1:k-1}$ that ends at x_{k-1}. We also assume the corresponding optimal state is x^*_{k-1}, which is a function of x_k, and hence, denoted as $u_{k-1}(x_k)$. Then, $V_k(x_k)$ and $u_{k-1}(x_k)$ are given by

$$\begin{aligned} V_k(x_k) &= \max_{x_{k-1}} [p(y_k \mid x_k)\, p(x_k \mid x_{k-1})\, V_{k-1}(x_{k-1})], \\ u_{k-1}(x_k) &= \arg\max_{x_{k-1}} [p(y_k \mid x_k)\, p(x_k \mid x_{k-1})\, V_{k-1}(x_{k-1})], \end{aligned} \tag{27}$$

with the initial condition

$$V_1(x_1) = p(x_1)\, p(y_1 \mid x_1). \tag{28}$$

At the final step, we just take

$$x^*_T = \arg\max_{x_T} V_T(x_T), \tag{29}$$

and then we compute in backwards to recover the Viterbi path $x^*_{1:T}$ as

$$x^*_{k-1} = u_{k-1}(x^*_k). \tag{30}$$

In terms of potentials, the pseudocode for Viterbi algorithm can be written as in Algorithm 4. As the computational complexity of the forward pass is $O(D^2T)$ and that of the backward pass is $O(T)$, the total computational complexity of the Viterbi algorithm is $O(D^2T)$.

Algorithm 4 The classical Viterbi algorithm.

Input: The potentials $\psi_k(\cdot)$ for $k = 1, \ldots, T$.
Output: The Viterbi path $x^*_{1:T}$.

// Forward pass:
$V_1(x_1) = \psi_1(x_1)$
for $k \leftarrow 2$ **to** T **do** ▷ Sequentially
  $V_k(x_k) = \max_{x_{k-1}} [\psi_k(x_{k-1}, x_k)\, V_{k-1}(x_{k-1})]$
  $u_{k-1}(x_k) = \arg\max_{x_{k-1}} [\psi_k(x_{k-1}, x_k)\, V_{k-1}(x_{k-1})]$
end for
// Backward pass:
$x^*_T = \arg\max_{x_T} V_T(x_T)$
for $k \leftarrow T$ **to** 2 **do**
  $x^*_{k-1} = u_{k-1}(x^*_k)$
end for

Please note that, in this paper, we assume that the MAP estimate is unique for simplicity of exposition. However, the results can be extended to account for multiple solutions.

B. Path-based parallelization

In order to design a parallel version of the Viterbi algorithm, we first need to find an element $\tilde{a}$ and the binary associative operator $\vee$ for the Viterbi algorithm. Let $\tilde{a}_{i:j}$ consists of a pair of elements associated with probability and the path from state x_i to x_j such that

$$\tilde{a}_{i:j} = \begin{pmatrix} A_{i:j}(x_i, x_j) \\ \hat{X}_{i:j}(x_i, x_j) \end{pmatrix}. \tag{31}$$

Here, $A_{i:j}(x_i, x_j)$ is the maximum probability of the path and $\hat{X}_{i:j}(x_i, x_j)$ is a column vector containing the most probable path sequence starting at x_i and ending at x_j. Now, we define the associative operator $\vee$.

Definition 4. *For two elements*

$$\tilde{a}_{i:j} = \begin{pmatrix} A_{i:j}(x_i, x_j) \\ \hat{X}_{i:j}(x_i, x_j) \end{pmatrix}, \quad \tilde{a}_{j:k} = \begin{pmatrix} A_{j:k}(x_j, x_k) \\ \hat{X}_{j:k}(x_j, x_k) \end{pmatrix}, \tag{32}$$

the binary associative operator $\vee$ for the MAP estimate in an HMM is defined as

$$\tilde{a}_{i:k} = \tilde{a}_{i:j} \vee \tilde{a}_{j:k}, \tag{33}$$

where

$$\begin{aligned} \tilde{a}_{i:k} &= \begin{pmatrix} A_{i:k}(x_i, x_k) \\ \hat{X}_{i:k}(x_i, x_k) \end{pmatrix} \\ &= \begin{pmatrix} \max_{x_j} A_{i:j}(x_i, x_j) A_{j:k}(x_j, x_k) \\ \left(\hat{X}_{i:j}(x_i, \hat{x}_j(x_i, x_k)), \hat{x}_j(x_i, x_k), \hat{X}_{j:k}(\hat{x}_j(x_i, x_k), x_k)\right) \end{pmatrix} \end{aligned} \tag{34}$$

and

$$\hat{x}_j(x_i, x_k) = \arg\max_{x_j} A_{i:j}(x_i, x_j) A_{j:k}(x_j, x_k) \tag{35}$$

with

$$\begin{aligned} A_{k-1:k}(x_{k-1}, x_k) &= \psi_{k-1,k}(x_{k-1}, x_k), \\ \hat{X}_{k-1:k}(x_{k-1}, x_k) &= \emptyset, \\ A_{0:1}(., x_1) &= \psi_1(x_1), \\ \hat{X}_{0:1}(., x_1) &= \emptyset. \end{aligned} \tag{36}$$

Lemma 2. *The operator $\vee$ is associative.*

Proof. See Appendix B for the proof. □

Theorem 3. *We have*

$$\begin{aligned} A_{i:j}(x_i, x_j) &= \max_{x_{i+1:j-1}} \prod_{k=i+1}^{j} \psi_{k-1,k}(x_{k-1}, x_k), \\ \hat{X}_{i:j}(x_i, x_j) &= \arg\max_{x_{i+1:j-1}} \prod_{k=i+1}^{j} \psi_{k-1,k}(x_{k-1}, x_k), \end{aligned} \tag{37}$$

where $A_{i:j}$ corresponds to the probability of the MAP estimate starting from x_i and ending at x_j, and $\hat{X}_{i:j}$ represents the sequence of the corresponding paths.

Proof. See Appendix C. □

Putting $i = 0$ and $j = T + 1$ in Theorem 3 above leads to the following result.

Corollary 1. *We have*

$$\tilde{a}_{0:T+1} = \begin{pmatrix} \psi_1(x_1^*) \prod_{t=2}^{T} \psi_{t-1,t}(x_{t-1}^*, x_t^*) \\ x_{1:T}^* \end{pmatrix},$$

where $x_{1:T}^*$ is the MAP estimate given by Eq. (26)*.*

It is now possible to form a parallel algorithm for the elements $\tilde{a}_{i:j}$ with the associative operator $\vee$ by leveraging the parallel-scan algorithm for computing the quantity in Corollary 1. However, each element $\tilde{a}_{i:j}$ contains a path of length $j - i - 2$ for each state pair and, therefore, the memory requirements to store the state sequences are high. Thus, in the next section, we propose an alternative approach with lower memory requirements.

C. Max-product formulation

Until now, we have used the Viterbi algorithm in forward direction. However, the MAP estimate can also be computed by using the max-product algorithm (see cf. [33]). Let $\tilde{\psi}_k^f(x_k)$ denote the maximum probability of the optimal path ending at x_k, and $\tilde{\psi}_k^b(x_k)$ denote the maximum probability of starting at x_k. This implies that

$$\begin{aligned} \tilde{\psi}_k^f(x_k) &= A_{0:k}(x_0, x_k), \\ \tilde{\psi}_k^b(x_k) &= A_{k:T+1}(x_k, x_{T+1}), \end{aligned} \tag{38}$$

where the dependence on x_0 and x_{T+1} is only a notational expression (cf. Eq. (15)). From Definition 4, we get the following recursions for these quantities.

Lemma 3. *The maximum forward and backward probabilities admit the recursions*

$$\begin{aligned} \tilde{\psi}_k^f(x_k) &= \max_{x_{k-1}} \psi_{k-1:k}(x_{k-1}, x_k) \tilde{\psi}_{k-1}^f(x_{k-1}), \\ \tilde{\psi}_k^b(x_k) &= \max_{x_{k+1}} \psi_{k:k+1}(x_k, x_{k+1}) \tilde{\psi}_{k+1}^b(x_{k+1}), \end{aligned} \tag{39}$$

with initial conditions $\tilde{\psi}_1^f(x_1) = \psi_1(x_1)$ and $\tilde{\psi}_T^b(x_T) = 1$.

The MAP estimate x_k^* can be computed by maximizing the product $\tilde{\psi}_k^f(x_k)\tilde{\psi}_k^b(x_k)$. This is summarized in the following theorem.

Theorem 4. *Given the maximum forward potentials $\tilde{\psi}_k^f(x_k)$ and the maximum backward potentials $\tilde{\psi}_k^b(x_k)$, the MAP estimate at time step k is determined by*

$$x_k^* = \arg\max_{x_k} \tilde{\psi}_k^f(x_k) \tilde{\psi}_k^b(x_k) \tag{40}$$

for $k = 1, \ldots, T$.

Proof. See Appendix D. □

Theorem 4 follows from the generalized MAP estimate for an arbitrary graph discussed in Ref. [33, ch. 13]. However, we compute the elements of these forward and backward potentials in parallel.

Let us now define an element $\bar{a}_{i:j}$ that consists of the upper part of $\tilde{a}_{i:j}$ where the path is left out. These elements can be computed without actually storing the paths $x_{i:j}^*$ which provides a computational advantage.

Definition 5. *For two elements $\bar{a}_{i:j} = A_{i:j}(x_i, x_j)$ and $\bar{a}_{j:k} = A_{j:k}(x_j, x_k)$, the binary operator $\vee$ can be defined as*

$$\bar{a}_{i:k} = \bar{a}_{i:j} \vee \bar{a}_{j:k}, \tag{41}$$

where

$$\bar{a}_{i:k} = \max_{x_j} A_{i:j}(x_i, x_j) A_{j:k}(x_j, x_k). \tag{42}$$

The element $\bar{a}_{i:j}$ and the operator also inherit the associative property of the element $\tilde{a}_{i:j}$ and, therefore, we also have $\bar{a}_{i:k} = A_{i:k}(x_i, x_k)$.

We can now compute the maximum forward and backward potentials in terms of these associative operators and elements, which is summarized in the following.

Proposition 2. *The maximum forward potential can be computed as*

$$\bar{a}_{0:k} = \tilde{\psi}_k^f(x_k), \quad k > 0.$$

Proof. This follows from Theorem 3. □

Proposition 3. *The maximum backward potential can be computed as*

$$\bar{a}_{k:T+1} = \tilde{\psi}_k^b(x_k), \quad k > 0.$$

Proof. This follows from Theorem 3. □

Similarly to the sum-product algorithm in Section III-D, we can now parallelize the max-product algorithm. The steps are summarized in Algorithm 5. The span and work complexities of the algorithm are summarized in the following proposition.

Algorithm 5 The parallel max-product algorithm.

Input: The potentials $\psi_k(\cdot)$, $k = 1, \ldots, T$ and the operator $\vee$, see Definition 5.
Output: The Viterbi path $x^*_{1:T}$.

for $k \leftarrow 1$ **to** T **do** ▷ In parallel
  Initialize $\bar{a}_{k-1:k}$.
end for
Run parallel-scan to get $\bar{a}_{0:k} = \tilde{\psi}^f_{1,k}(x_k)$, $k = 1, \ldots, T$.
for $k \leftarrow 1$ **to** T **do** ▷ In parallel
  Initialize $\bar{a}_{k:k+1}$.
end for
Run reversed parallel-scan to get $\bar{a}_{k:T+1} = \tilde{\psi}^b_{k,T}(x_k)$, $k = 1, \ldots, T$.
for $k \leftarrow 1$ **to** T **do** ▷ In parallel
  Compute the optimal state x^*_k using Eq. (40).
end for

Proposition 4. *The span complexity of the parallel max-product algorithm (Algorithm 5) is $O(\log T)$ and the work complexity is $O(T)$.*

Proof. The initializations and final state combinations have span complexities $O(1)$ and work complexities $O(T)$, whereas the parallel-scans have span complexities $O(\log T)$ and work complexities $O(T)$. Hence the result follows. □

V. Extensions

In this section, we discuss extensions of the parallel-scan framework for the inference in HMMs.

A. Generic associative operations

We defined the parallel-scan framework in terms of sum-product and max-product for HMMs. We can easily extend this framework to operations of the form expressed by Eq. (7) for arbitrary associative operators. In particular, we can also consider continuous-state Markov processes; in this case, the operator becomes integration and we get similar algorithms to the ones described in [30], except that the smoother will be a two-filter smoother instead of a Rauch–Tung–Striebel smoother. In particular, for linear Gaussian systems, we get a parallel version of the two-filter Kalman smoother.

B. Block-wise operations

In this article, we have restricted our consideration to pair-wise Markovian elements, that is, we assign a single observation and state to a single element. However, it is possible to perform binary association operations in parallel on a block level, where we assign a single computational element to a set of consecutive observations and states [30]. In other words, we can define a block of consecutive l observations and states as a single element in the parallel framework. This single element processes a block of measurements before combining the results with other elements. This kind of block-processing can be advantageous when the number of computational cores is limited.

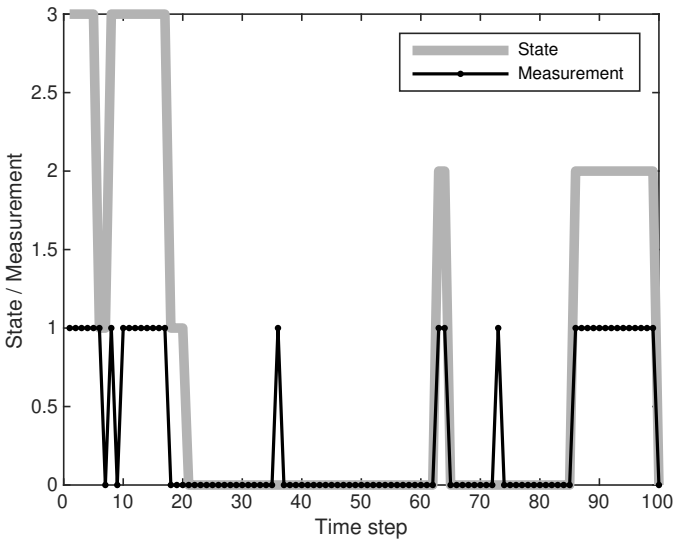

Fig. 2. An example of states and measurements from the Gilbert–Elliot hidden Markov model with the number of time steps $T = 100$.

C. Parameter estimation

Another task of the HMM inference is to estimate parameters of the model. One possible solution is to use the *Baum-Welch algorithm* (BWA) [44], which is a special case of *expectation–maximization* (EM) algorithm [45]. In expectation step, BWA uses the forward-backward algorithm, which can be parallelized using the methods proposed in this article.

VI. Experimental results

For our experiment, we consider the Gilbert–Elliott (GE) channel model [2], which is a classical model used in the transmission of signals in digital communication channels. The model describes the burst error patterns in communication channels and simulates the error performance of the communication link. This model consists of two hidden Markov states b_k and s_k, where b_k represents the binary input signal to be transmitted and s_k represents the channel regime binary signal. It is assumed that the input signal b_k may be flipped by an independent error, which can be modeled as $y_k = b_k \oplus v_k$. Here, y_k is the measurement signal, which is observable. Furthermore, $v_k \in \{0, 1\}$ is a Bernoulli sequence, and $\oplus$ is the exclusive-or operation. The exclusive-or operation ensures that $y_k = b_k$ if $v_k = 0$, and $y_k \neq b_k$ if $v_k = 1$.

The regime input signal s_k is modeled as a two-state Markov chain that represents high and low error conditions of the channel. More specifically, if an error occurs ($v_k = 1$), the probability of error has either a small value (q_0) or a large value (q_1). Moreover, we denote the probability of transition from a high error state ($s_{k-1} = 1$) to a low error state ($s_k = 0$) as p_0. Similarly, we present the probability of transition from a low error state ($s_{k-1} = 0$) to a high error state ($s_k = 1$) as p_1. We also denote the state switch probability of b_k as p_2.

In order to recover the hidden states, we use the joint model $x_k = (s_k, b_k)$ which is a 4-state Markov chain ($D = 4$) consisting of the states $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$. These states are encoded as $x_k \in \{0, 1, 2, 3\}$. The transition matrix Π

$$\Pi = \begin{pmatrix} (1-p_0)(1-p_2) & p_0(1-p_2) & (1-p_0)p_2 & p_0 p_2 \\ p_1(1-p_2) & (1-p_1)(1-p_2) & p_1 p_2 & (1-p_1)p_2 \\ (1-p_0)p_2 & p_0 p_2 & (1-p_0)(1-p_2) & p_0(1-p_2) \\ p_1 p_2 & (1-p_1)p_2 & p_1(1-p_2) & (1-p_1)(1-p_2) \end{pmatrix}, \quad O = \begin{pmatrix} (1-q_0) & q_0 \\ (1-q_1) & q_1 \\ q_0 & (1-q_0) \\ q_1 & (1-q_1) \end{pmatrix}. \quad (43)$$

Here,

$$\Pi = p(x_k \mid x_{k-1})$$

and

$$O = p(y_k \mid x_k).$$

---

and the observation model O, which encode the information in $p(x_k \mid x_{k-1})$ and $p(y_k \mid x_k)$, respectively, are given in Eq. (43). An example of the states and the corresponding measurements from the GE model is shown in Fig. 2.

To evaluate the performance of our proposed methods, we simulated the states and measurements with varying lengths of time series ranging from $T = 10^2$ to $T = 10^5$ and averaged the run times (10 repetitions for sequential methods and 100 for parallel ones). It is worth noting that we evaluate the methods only in terms of computational time (not error performance), because the parallel and sequential methods are algebraically equivalent and, therefore, there is no difference in their error performance.

In the experimental setup, we used the open-source TensorFlow 2.4 software library [46]. The library natively implements the vectorization and associative scan primitives that can be used to implement the methodology for both CPUs and GPUs. In this experiment, we set the values of parameters of the GE model given in Eq. (43) as follows: $p_0 = 0.03, p_1 = 0.1, p_2 = 0.05, q_0 = 0.01, q_1 = 0.1$. The initial prior for the state is uniform, $p(x_1) = 0.25$ for $x_1 \in \{0, 1, 2, 3\}$. We ran the sequential and parallel Bayesian smoothers (BS-Seq, BS-Par) [32], sequential and parallel sum-product based smoothers (SP-Seq, SP-Par) from Section III, and sequential and parallel max-product based MAP estimators (MP-Seq, MP-Par) from Section IV on both CPU and GPU. We also ran the classical Viterbi algorithm (see Algorithm 4) for comparison. The experiment was carried both on a CPU, AMD Ryzen™ Threadripper™ 3960X with 24 Cores and 3.8 GHz, and a GPU, NVIDIA® Ampere® GeForce RTX 3090 (GA102) with 10496 cores.

Average run times are shown in Figs. 3 and 4. It is clear that the parallel algorithms are computationally faster than the sequential versions both on the CPU and the GPU. Although the difference is much more pronounced on the GPU, even on the CPU, we can see a benefit of parallelization. Nevertheless, the number of computational cores is orders of magnitude smaller than in the GPU (24 vs. 10496 cores). Among the compared methods, the max-product-based parallel method is the fastest, sum-product-based parallel method is the second, and the parallel Bayesian smoother is the third, on both the CPU and the GPU. A similar order can be seen among the sequential method results and the classical Viterbi is placed between the other sequential methods.

Average run times of the parallel methods on the GPU

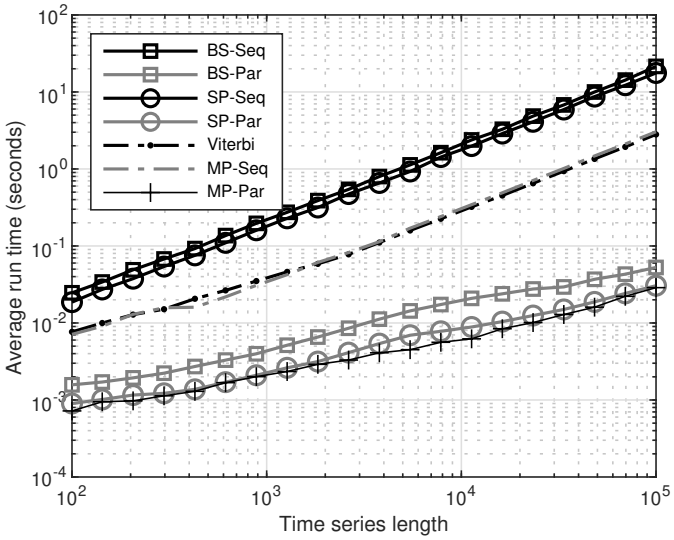

Fig. 3. Average computation times on the CPU for sequential and parallel Bayesian smoothers (BS-Seq, BS-Par), sequential and parallel sum-product algorithms (SP-Seq, SP-Par), sequential and parallel max-product algorithms (MP-Seq, MP-Par), and the classical Viterbi algorithm (Viterbi).

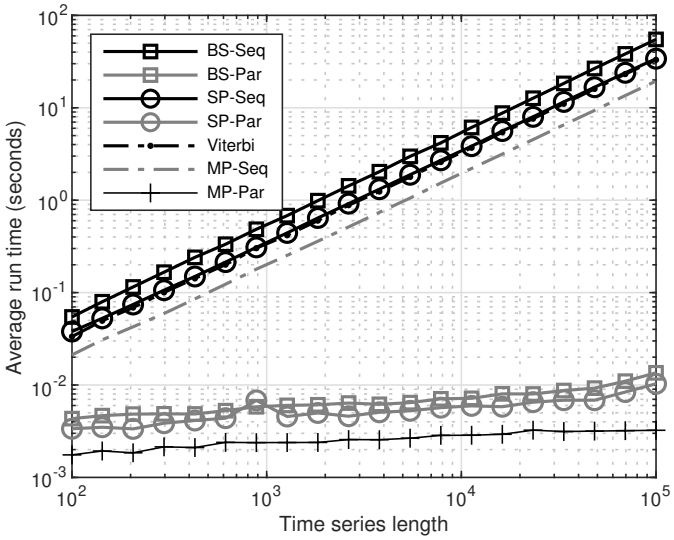

Fig. 4. Average computation times on the GPU for sequential and parallel Bayesian smoothers (BS-Seq, BS-Par), sequential and parallel sum-product algorithms (SP-Seq, SP-Par), sequential and parallel max-product algorithms (MP-Seq, MP-Par), and the classical Viterbi algorithm (Viterbi).

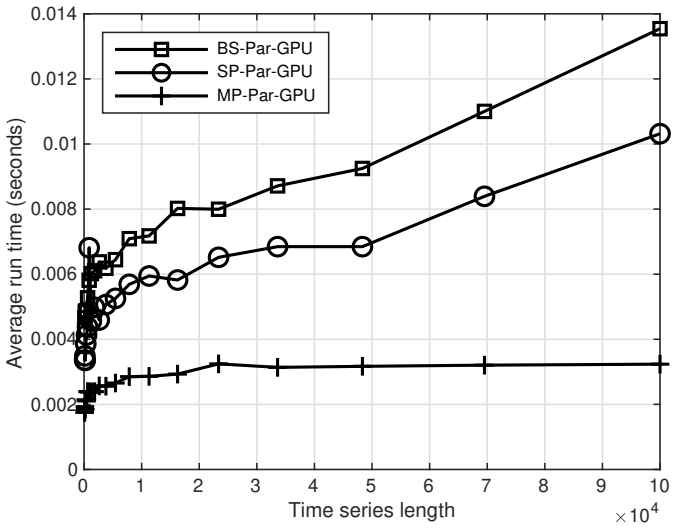

Fig. 5. Average computation times on the GPU for parallel Bayesian smoothers (BS-Par-GPU), parallel sum-product algorithm (SP-Par-GPU), and parallel max-product algorithm (MP-Par-GPU).

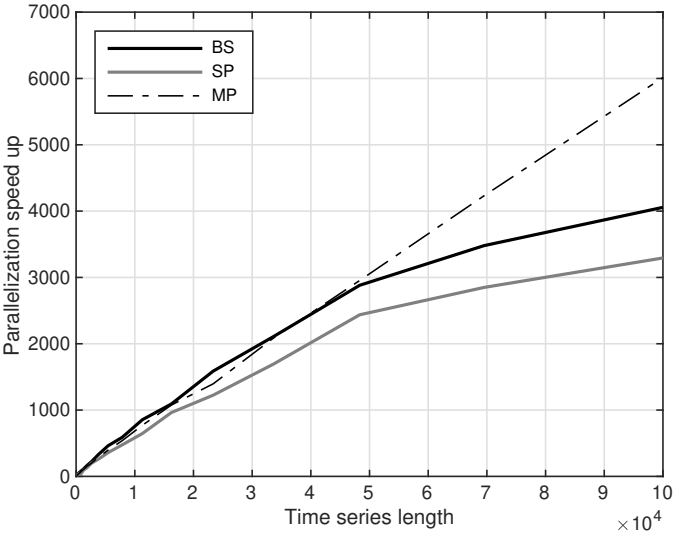

Fig. 6. Ratio of the average run times of the sequential methods (BS) and the corresponding parallel methods (SP, MP) on the GPU.

(on a linear scale) are separately shown in Fig. 5. From the figure, it can be seen that the computational times initially grow logarithmically as is predicted by the theory and then, for the Bayesian smoother and the sum-product method retain back to linear when the time series length becomes longer than $\sim 5 \times 10^4$. However, the max-product method remains sub-linear even beyond the time series length of 10^5, which is likely due to less computational operations needed. It is worth mentioning that the mean absolute error between Bayesian smoothers and sum-product based smoothers is insignificant ($\leq 10^{-16}$). This difference is due to numerical inaccuracies. The same conclusion can be drawn for max-product based MAP estimators and the Viterbi algorithm.

Finally, Fig. 6 shows the ratio of the average run times of the sequential methods and the corresponding parallel methods on the GPU. It can be seen that with time series length of $\sim 5 \times 10^4$, the speed-up is already between 2000–3000 and with time series of length 10^5, the speed-up in the max-product method is $\sim$ 6000. On the other hand, with the Bayesian smoother and sum-product methods, the speed-up is $\sim$ 3000–4000.

VII. Conclusion

In this paper, we have proposed a parallel formulation of HMM inference. We have considered the computation of both the marginal smoothing distributions as well as the MAP estimate. The proposed formulation enables efficient parallel computation is HMMs by reducing the time complexity from linear to logarithmic. The algorithms are based on reformulating the HMM inference problems in terms of associative operators which can be parallelized using the parallel-scan algorithm. We also showed the practical advantage of our proposed methods with experiments in multi-core CPU and GPU.

Appendix

A. Proof of Lemma 1

In this section, we prove the associative property of the operator $\otimes$ stated in Lemma 1. For this, we need to prove that for three general elements $a_{i:j}$, $a_{j:k}$, $a_{k:l}$, the following statement holds

$$\begin{aligned}&(a_{i:j} \otimes a_{j:k}) \otimes a_{k:l} = a_{i:j} \otimes (a_{j:k} \otimes a_{k:l}), \\ &\quad \text{where } 0 \leq i < j < k < l.\end{aligned} \tag{44}$$

We proceed to perform the calculations to the left-hand side of Eq. (44) to check that they yield the same result as in the right-hand side, that is,

$$\begin{aligned}&(a_{i:j} \otimes a_{j:k}) \otimes a_{k:l} \\ &= \sum_{x_k} \left(\sum_{x_j} \psi_{i,j}(x_i, x_j)\, \psi_{j,k}(x_j, x_k) \right) \psi_{k,l}(x_k, x_l) \\ &= \sum_{x_j, x_k} \psi_{i,j}(x_i, x_j)\, \psi_{j,k}(x_j, x_k)\, \psi_{k,l}(x_k, x_l) \\ &= \sum_{x_j} \psi_{i,j}(x_i, x_j) \left(\sum_{x_k} \psi_{j,k}(x_j, x_k)\, \psi_{k,l}(x_k, x_l) \right) \\ &= a_{i:j} \otimes (a_{j:k} \otimes a_{k:l}),\end{aligned}$$

which gives the result.

B. Proof of Lemma 2

In this section, we prove the associative property of the operator $\vee$ as stated in Lemma 2. We need to prove that for three general elements $\tilde{a}_{i:j}$, $\tilde{a}_{j:k}$, $\tilde{a}_{k:l}$, the following statement holds

$$\begin{aligned}&(\tilde{a}_{i:j} \vee \tilde{a}_{j:k}) \vee \tilde{a}_{k:l} = \tilde{a}_{i:j} \vee (\tilde{a}_{j:k} \vee \tilde{a}_{k:l}), \\ &\quad \text{where } 0 \leq i < j < k < l.\end{aligned} \tag{45}$$

From Definition 4, we can write

$$\tilde{a}_{i:j} \vee \tilde{a}_{j:k} = \begin{pmatrix} A_{i:k}(x_i, x_k) \\ \hat{X}_{i:k}(x_i, x_k) \end{pmatrix}, \tag{46}$$

where

$$A_{i:k}(x_i, x_k) = \max_{x_j} A_{i:j}(x_i, x_j) A_{j:k}(x_j, x_k), \tag{47a}$$

$$\hat{X}_{i:k}(x_i, x_k) = \big(\hat{X}_{i:j}(x_i, \hat{x}_j(x_i, x_j)), \hat{x}_j(x_i, x_j), \hat{X}_{j:k}(\hat{x}_j(x_i, x_j), x_k))\big), \tag{47b}$$

and

$$\hat{x}_j(x_i, x_j) = \arg\max_{x_j} A_{i:j}(x_i, x_j) A_{j:k}(x_j, x_k). \tag{48}$$

Now, combining the maximum probability of MAP estimates for the element $\tilde{a}_{k:l}$, we can write Eq. (47a) as

$$\begin{aligned} A_{i:l}(x_i, x_l) &= \max_{x_k} A_{i:k}(x_i, x_k) A_{k:l}(x_k, x_l) \\ &= \max_{x_k} \Bigg[\left\{ \max_{x_j} A_{i:j}(x_i, x_j) A_{j:k}(x_j, x_k) \right\} \\ &\quad \times A_{k:l}(x_k, x_l) \Bigg] \\ &= \max_{x_j} \Bigg[A_{i:j}(x_i, x_j) \\ &\quad \times \left\{ \max_{x_k} A_{j:k}(x_j, x_k) A_{k:l}(x_k, x_l) \right\} \Bigg] \\ &= \max_{x_j} A_{i:j}(x_i, x_j) A_{j:l}(x_j, x_l). \end{aligned} \tag{49}$$

Now, combining the Viterbi path of the element $\tilde{a}_{k:l}$, we can write Eq. (47b) as

$$\begin{aligned} \hat{X}_{i:l}(x_i, x_l) &= \arg\max_{x_k} A_{i:k}(x_i, x_k) A_{k:l}(x_k, x_l) \\ &= \arg\max_{x_k} \Bigg[\left\{ \arg\max_{x_j} A_{i:j}(x_i, x_j) A_{j:k}(x_j, x_k) \right\} \\ &\quad \times A_{k:l}(x_k, x_l) \Bigg] \\ &= \arg\max_{x_j} \Bigg[A_{i:j}(x_i, x_j) \\ &\quad \times \left\{ \arg\max_{x_k} A_{j:k}(x_j, x_k) A_{k:l}(x_k, x_l) \right\} \Bigg] \\ &= \arg\max_{x_j} A_{i:j}(x_i, x_j) A_{j:l}(x_j, x_l). \end{aligned} \tag{50}$$

From Eq. (49) and Eq. (50), it follows that the operator $\vee$ is associative.

C. Proof of Theorem 3

In this section, we prove Theorem 3 by induction. We first note that the theorem is true for $i = k$ and $j = k + 1$. Furthermore, if the assertion is true for $i+1 < j$, we proceed to show that it holds for any i. By using Eq. (34) we get

$$\begin{aligned} &A_{i:j}(x_i, x_j) \\ &= \max_{x_{i+1}} A_{i:i+1}(x_i, x_{i+1}) A_{i+1:j}(x_{i+1}, x_j) \\ &= \max_{x_{i+1}} \psi_{i,i+1}(x_i, x_{i+1}) \max_{x_{i+2:j-1}} \prod_{t=i+2}^{j} \psi_{t-1,t}(x_{t-1}, x_t) \\ &= \max_{x_{i+1:j-1}} \prod_{t=i+1}^{j} \psi_{t-1,t}(x_{t-1}, x_t). \end{aligned} \tag{51}$$

We similarly get

$$\begin{aligned} &\hat{X}_{i:j}(x_i, x_j) \\ &= \Big(\hat{X}_{i:i+1}(x_i, \hat{x}_{i+1}(x_i, x_j)), \hat{x}_{i+1}(x_i, x_j), \\ &\quad \hat{X}_{i+1:j}(\hat{x}_{i+1}(x_i, x_j), x_j) \Big) \\ &= \Big(\hat{x}_{i+1}(x_i, x_j), \hat{X}_{i+1:j}(\hat{x}_{i+1}(x_i, x_j), x_j) \Big) \\ &= \Bigg(\arg\max_{x_{i+1}} \psi_{i,i+1}(x_i, x_{i+1}) \max_{x_{i+2:j-1}} \prod_{t=i+2}^{j} \psi_{t-1,t}(x_{t-1}, x_t), \\ &\quad \max_{x_{i+1}} \arg\max_{x_{i+2:j-1}} \prod_{t=i+2}^{j} \psi_{t-1,t}(x_{t-1}, x_t) \Bigg) \\ &= \arg\max_{x_{i+1:j-1}} \prod_{t=i+1}^{j} \psi_{t-1,t}(x_{t-1}, x_t), \end{aligned} \tag{52}$$

which concludes the proof.

D. Proof of Theorem 4

In this section, we prove Theorem 4. That is, given the maximum forward probability $\tilde{\psi}_k^f(x_k)$ of path ending at x_k, and the maximum backward probability $\tilde{\psi}_k^b(x_k)$ of path starting at x_k, the MAP estimate at time step k is given by Eq. (40).

Due to the associative property of the operator, we can write

$$\tilde{a}_{0:T+1} = \tilde{a}_{0:k} \vee \tilde{a}_{k:T+1} = \begin{pmatrix} \psi_1(x_1^*) \prod_{t=2}^{T} \psi_t(x_{t-1}^*, x_t^*) \\ x_{1:T}^* \end{pmatrix}, \tag{53}$$

which is given by

$$\begin{aligned} &\tilde{a}_{0:k} \vee \tilde{a}_{k:T+1} \\ &= \begin{pmatrix} \max_{x_k} A_{0:k}(x_0, x_k) A_{k:T+1}(x_k, x_{T+1}) \\ \Big(\hat{X}_{0:k}(x_0, \hat{x}_k(x_0, x_{T+1})), \hat{x}_k(x_0, x_{T+1}), \\ \qquad \hat{X}_{k:T+1}(\hat{x}_k(x_0, x_{T+1}), x_{T+1}) \Big) \end{pmatrix}. \end{aligned} \tag{54}$$

Here, the dependence on x_0 and x_{T+1} is only a notational expression and the term

$$\begin{aligned} &\hat{x}_k(x_0, x_{T+1}) \\ &= \arg\max_{x_k} A_{0:k}(x_0, x_k) A_{k:T+1}(x_k, x_{T+1}) \\ &= \arg\max_{x_k} \tilde{\psi}^f(x_k)\, \tilde{\psi}^b(x_k) \end{aligned} \tag{55}$$

has to be the kth element on the optimal path in order to match the right-hand side of Eq. (53).

ACKNOWLEDGMENT

The authors would like to thank Academy of Finland for funding. The authors would also like to thank Adrien Corenflos for helping with the experiments.

REFERENCES

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

[2] O. Cappé, E. Moulines, and T. Rydén, *Inference in hidden Markov models*. Springer, 2006.

[3] Z. Song and A. Dogandžić, "A max-product EM algorithm for reconstructing Markov-tree sparse signals from compressive samples," *IEEE Transactions on Signal Processing*, vol. 61, no. 23, pp. 5917–5931, 2013.

[4] Y. Ueng, K. Liao, H. Chou, and C. Yang, "A high-throughput trellis-based layered decoding architecture for non-binary LDPC codes using max-log-QSPA," *IEEE Transactions on Signal Processing*, vol. 61, no. 11, pp. 2940–2951, 2013.

[5] B. Mor, S. Garhwal, and A. Kumar, "A systematic review of hidden Markov models and their applications," *Archives of Computational Methods in Engineering*, 2020.

[6] B. H. Juang and L. R. Rabiner, "Hidden Markov models for speech recognition," *Technometrics*, vol. 33, no. 3, pp. 251–272, 1991.

[7] A. A. Salah, M. Bicego, L. Akarun, E. Grosso, and M. Tistarelli, "Hidden Markov model-based face recognition using selective attention," in *Human Vision and Electronic Imaging XII*, vol. 6492, International Society for Optics and Photonics. SPIE, 2007, pp. 404–412.

[8] A. Krogh, B. Larsson, G. Von Heijne, and E. L. Sonnhammer, "Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes," *Journal of molecular biology*, vol. 305, no. 3, pp. 567–580, 2001.

[9] L. Pan, M. W. Marcellin, W. E. Ryan, and B. Vasic, "Viterbi detection for compressively sampled FHSS-GFSK signals," *IEEE Transactions on Signal Processing*, vol. 63, no. 22, pp. 5965–5975, 2015.

[10] Y. Yang, S. Chen, M. A. Maddah-Ali, P. Grover, S. Kar, and J. Kovačević, "Fast temporal path localization on graphs via multiscale Viterbi decoding," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5588–5603, 2018.

[11] T. Long, L. Zheng, X. Chen, Y. Li, and T. Zeng, "Improved probabilistic multi-hypothesis tracker for multiple target tracking with switching attribute states," *IEEE Transactions on Signal Processing*, vol. 59, no. 12, pp. 5721–5733, 2011.

[12] P. Di Viesti, G. M. Vitetta, and E. Sirignano, "Double Bayesian smoothing as message passing," *IEEE Transactions on Signal Processing*, vol. 67, no. 21, pp. 5495–5510, 2019.

[13] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[14] 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.

[15] A. Sand, C. N. Pedersen, T. Mailund, and A. T. Brask, "HMMlib: A C++ library for general hidden Markov models exploiting modern CPUs," in *2010 Ninth International Workshop on Parallel and Distributed Methods in Verification, and Second International Workshop on High Performance Computational Systems Biology*. IEEE, 2010, pp. 126–134.

[16] J. Nielsen and A. Sand, "Algorithms for a parallel implementation of hidden Markov models with a small state space," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. IEEE, 2011, pp. 452–459.

[17] S. Chatterjee and S. Russell, "A temporally abstracted Viterbi algorithm," in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, 2011, p. 96–104.

[18] Zhihui Du, Zhaoming Yin, and D. A. Bader, "A tile-based parallel Viterbi algorithm for biological sequence alignment on GPU with CUDA," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010, pp. 1–8.

[19] S. Maleki, M. Musuvathi, and T. Mytkowicz, "Parallelizing dynamic programming through rank convergence," *ACM SIGPLAN Notices*, vol. 49, no. 8, pp. 219–232, 2014.

[20] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[21] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 449–460.

[22] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[23] D. Luebke, "CUDA: Scalable parallel programming for high-performance scientific computing," in *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 2008, pp. 836–838.

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

[25] ——, "Prefix sums and their applications," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-90-190, 1990.

[26] J. Li, S. Chen, and Y. Li, "The fast evaluation of hidden Markov models on GPU," in *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 4. IEEE, 2009, pp. 426–430.

[27] C. Liu, "cuHMM: a CUDA implementation of hidden Markov model training and classification," Johns Hopkins University, Tech. Rep., 2009.

[28] D. Zhang, R. Zhao, L. Han, T. Wang, and J. Qu, "An implementation of Viterbi algorithm on GPU," in *2009 First International Conference on Information Science and Engineering*. IEEE, 2009, pp. 121–124.

[29] S. Hymel, "Massively parallel hidden Markov models for wireless applications," Ph.D. dissertation, Virginia Tech, 2011.

[30] S. Särkkä and Á. F. García-Fernández, "Temporal parallelization of Bayesian smoothers," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 299–306, 2021.

[31] F. Yaghoobi, A. Corenflos, S. Hassan, and S. Särkkä, "Parallel iterated extended and sigma-point Kalman smoothers," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 5350–5354.

[32] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

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

[34] S. Rangan, A. K. Fletcher, V. K. Goyal, E. Byrne, and P. Schniter, "Hybrid approximate message passing," *IEEE Transactions on Signal Processing*, vol. 65, no. 17, pp. 4577–4592, 2017.

[35] Y. Weiss, "Correctness of local probability propagation in graphical models with loops," *Neural computation*, vol. 12, no. 1, pp. 1–41, 2000.

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

[37] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[38] G. Shafer and P. Shenoy, "Probability propagation," *Annals of Mathematics and Artificial Intelligence*, vol. 2, p. 327–351, 1990.

[39] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[40] R. G. Cowell, P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter, *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer, 2006.

[41] B. J. Frey, J. F. Brendan, and B. J. Frey, *Graphical models for machine learning and digital communication*. MIT press, 1998.

[42] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the Association for Computing Machinery (JACM)*, vol. 27, no. 4, pp. 831–838, 1980.

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

[44] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The annals of mathematical statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[45] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[46] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

Syeda Sakira Hassan received her Master of Science (Tech.) degree (with distinction) in information technology, and Doctor of Science (Tech.) degree in computing and electrical engineering from Tampere University, Tampere, Finland, in 2013 and 2019, respectively. From 2019 to 2020 she worked in Basware Oyj, Finland as a Data Scientist. Currently, Dr. Hassan is a Postdoctoral Researcher in the Sensor Informatics and Medical Technology research group at Department of Electrical Engineering and Automation (EEA), Aalto University. She is also working as an AI Scientist with Silo.AI, Finland. Her research interests are machine learning, probabilistic modeling, dynamic programming, and optimal control theory.

Simo Särkkä received his Master of Science (Tech.) degree (with distinction) in engineering physics and mathematics, and Doctor of Science (Tech.) degree (with distinction) in electrical and communications engineering from Helsinki University of Technology, Espoo, Finland, in 2000 and 2006, respectively. Currently, Dr. Särkkä is an Associate Professor with Aalto University, Technical Advisor of IndoorAtlas Ltd., and an Adjunct Professor with Tampere University of Technology and Lappeenranta University of Technology. His research interests are in multisensor data processing systems with applications in location sensing, health and medical technology, machine learning, inverse problems, and brain imaging. He has authored or coauthored over 150 peer-reviewed scientific articles and his books "Bayesian Filtering and Smoothing" and "Applied Stochastic Differential Equations" along with the Chinese translation of the former were recently published via the Cambridge University Press. He is a Senior Member of IEEE and serving as an Associate Editor of IEEE Signal Processing Letters.

Ángel F. García-Fernández received the telecommunication engineering degree (with honours) and the Ph.D. degree from Universidad Politécnica de Madrid, Madrid, Spain, in 2007 and 2011, respectively. He is currently a Lecturer in the Department of Electrical Engineering and Electronics at the University of Liverpool, Liverpool, UK. His main research activities and interests are in the area of Bayesian inference, with emphasis on nonlinear dynamic systems and multiple target tracking. He has received paper awards at the International Conference on Information Fusion in 2017 and 2019.