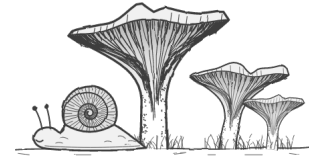


Associative Scan

Mikkel Paltorp (mpasc@dtu.dk), 30 July 2025

Department of Applied Mathematics and Computer Science, Technical University of Denmark



Recently a plethora of methods within machine learning have made use of state-space models (SSMs) to model sequences [1], [2], [3], [4]. While the early S4 model is computed as a convolution [1], a big reason for the success of later models such as S5 [2] and Mamba [3] is their usage of the associative scan operation to parallelize the state computation on the GPU. In my opinion this is a neat trick that was poorly explained in the original papers (although Appendix B in [4] does explain it for the scalar case), and I will therefore explain it in more detail in this blog post. The aim is to show that composing the transitions of a (linear) state-space model

$$\begin{aligned} \mathbf{h}_i &= \mathbf{A}_i \mathbf{h}_{i-1} + \mathbf{B}_i \mathbf{x}_i, \\ \mathbf{y}_i &= \mathbf{C}_i \mathbf{h}_i, \end{aligned} \tag{1}$$

is associative¹. In short this means that the states of a (linear) state-space model up until time step T can be computed in parallel using an associative scan operation. For details on the associative operation open the example box below.

Contents

Bibliography

¹ Associativity simply means that repeated applications of the operator (denoted by $\bullet$) can be grouped arbitrarily, i.e., that $a \bullet (b \bullet c) = (a \bullet b) \bullet c$. The order of the operands must still be kept, as the operator need not be commutative. Examples of associative operators are addition and multiplication, including matrix multiplication.

Example 1: Cumulative sum using associative scan

Example 1 (Cumulative sum using associative scan) The (parallel) associative scan, as the name suggests, is a way to apply an associative operator in parallel. The simplest such operator is addition, which is associative because we can group the additions however we like (i.e., $a + (b + c) = (a + b) + c$). In JAX the associative scan is implemented in the accurately named `lax.associative_scan` function, while Julia's `accumulate` function computes the same scan sequentially. If we supply the add operator the resulting computation will be equivalent to the cumulative sum, e.g. in Python

```
>>> lax.associative_scan(jnp.add, jnp.arange(0, 4))
Array([0, 1, 3, 6], dtype=int32)
```

or in Julia

```
julia> accumulate(+, 0:3)
4-element Vector{Int64}:
 0
 1
 3
 6
```

While the above computation is trivial and easily computed by looping from the first to last element in the vector, the main idea of the associative scan is that the operator can be applied pairwise in parallel. That means if enough processors are available the computation can be done in $\mathcal{O}(\log T)$ time rather than the trivial implementation of $\mathcal{O}(T)$.

The first step in showing the associativity of the state-space model is to define the transition of the state-space models using matrix multiplication (which is associative) by embedding the transition into a larger matrix $\mathbf{s}_k$ as follows

$$\mathbf{s}_k = \begin{bmatrix} \mathbf{A}_k & \mathbf{B}_k \mathbf{x}_k \\ \mathbf{0} & 1 \end{bmatrix}, \quad \mathbf{s}_0 = \begin{bmatrix} \mathbf{0} & \mathbf{h}_0 \\ \mathbf{0} & 1 \end{bmatrix} \quad (2)$$

Using the definition of $\mathbf{s}_k$ the state transition from state $k-1$ to state k can be computed using matrix multiplication as

$$\mathbf{s}_k \mathbf{s}_{k-1} = \begin{bmatrix} \mathbf{A}_k \mathbf{A}_{k-1} & \mathbf{A}_k (\mathbf{B}_{k-1} \mathbf{x}_{k-1}) + \mathbf{B}_k \mathbf{x}_k \\ \mathbf{0} & 1 \end{bmatrix}. \quad (3)$$

Using this we can compute the i th state of the state-space model as

$$\mathbf{h}_i = [\mathbf{I} \ 0] \left(\prod_{k=i}^0 \mathbf{s}_k \right) \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (4)$$

Given that the cumulative product can be computed using the associative scan operator the full dynamics can be computed as

$$\begin{aligned} \mathbf{p}_i &= \text{associative_scan}(\mathbf{s}_i, \text{init} = \mathbf{s}_0) \\ \mathbf{y}_i &= \mathbf{C}_i \mathbf{h}_i = [\mathbf{C}_i \quad 0] \mathbf{p}_i \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \end{aligned} \quad (5)$$

While the above works, it can be simplified slightly. As we are really only interested in the top row of blocks (the accumulated product $\mathbf{p}_i$ has $\mathbf{h}_i$ in its top-right block), we can instead define elements by just the top row, i.e., $\mathbf{s}_k = [\mathbf{A}_k \ \mathbf{B}_k \mathbf{x}_k]$, and then define the operator (denoted by $\bullet$) by how the top row propagates. For general pairs this reads

$$[\mathbf{A}'' \ \mathbf{b}''] \bullet [\mathbf{A}' \ \mathbf{b}'] = [\mathbf{A}'' \mathbf{A}' \ \mathbf{A}'' \mathbf{b}' + \mathbf{b}''], \quad (6)$$

which is the top block row of the product $\begin{bmatrix} \mathbf{A}'' & \mathbf{b}'' \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A}' & \mathbf{b}' \\ \mathbf{0} & 1 \end{bmatrix}$. The operator is therefore associative, because matrix multiplication is. The SSM states can then be computed as $\mathbf{p}_i = \mathbf{s}_i \bullet \mathbf{p}_{i-1}$ with $\mathbf{p}_0 = \mathbf{s}_0 = [0 \ \mathbf{h}_0]$, and $\mathbf{h}_i$ is the right block of $\mathbf{p}_i$.

As a final remark note that while the associative scan is parallelizable it performs matrix-matrix products of the form $\mathbf{A}_k \mathbf{A}_{k-1}$ which will be computationally prohibitive unless $\mathbf{A}_k$ has structure (e.g., diagonal or low-rank). This is one of the reasons why e.g. Mamba-2 utilizes a scaled identity as its $\mathbf{A}_k$ [4].

Bibliography

- [1] A. Gu, K. Goel, and C. Ré, “Efficiently Modeling Long Sequences with Structured State Spaces,” in *International Conference on Learning Representations (ICLR)*, 2022. [Online]. Available: <https://arxiv.org/abs/2111.00396>
- [2] J. T. H. Smith, A. Warrington, and S. W. Linderman, “Simplified State Space Layers for Sequence Modeling,” in *International Conference on Learning Representations (ICLR)*, 2023. [Online]. Available: <https://arxiv.org/abs/2208.04933>
- [3] A. Gu and T. Dao, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces.” [Online]. Available: <https://arxiv.org/abs/2312.00752>
- [4] T. Dao and A. Gu, “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality,” in *International Confer-*

ence on Machine Learning (ICML), 2024. [Online]. Available: <https://arxiv.org/abs/2405.21060>