Parallelizing the **SCYFI** algorithm for GPUs using the Julia Programming Language

Internship of Computer Science B.Sc., University Heidelberg

Kai Rothe

advised by Daniel Durstewitz and Filip Sadlo

Piece-wise Linear Recurrent Neural Networks

off-diagonal matrix

$\boldsymbol{W} \in \mathbb{R}^{M \times M}$ of coupling weights

bias term $\boldsymbol{h} \in \mathbb{R}^M$

$\boldsymbol{z}_t \in \mathbb{R}^M$

$$\boldsymbol{z}_{t+1} = F_{\boldsymbol{\theta}}(\boldsymbol{z}_t) = \boldsymbol{A}\,\boldsymbol{z}_t + \boldsymbol{W}\,\phi(\boldsymbol{z}_t) + \boldsymbol{h}$$

diagonal matrix $\boldsymbol{A} \in \mathbb{R}^{M \times M}$

of auto-regression weights

element-wise applied

rectified linear unit (ReLU) function

$$\phi(\boldsymbol{z}_t) = \max(\boldsymbol{z}_t, 0)$$

Searching for Fixed Points and Limit Cycles

$\{\boldsymbol{z}_1^*, \ldots, \boldsymbol{z}_k^*\}$ of order k

$$\boldsymbol{D}_{ij}(\boldsymbol{z}_t) := \begin{cases} 1, & \text{if } i = j \text{ and } \boldsymbol{z}_{t,i} > 0 \\ 0, & \text{else} \end{cases}$$

$$\boldsymbol{z}_{t+1} = F_{\boldsymbol{\theta}}(\boldsymbol{z}_t) = \boldsymbol{A}\,\boldsymbol{z}_t + \boldsymbol{W}\boldsymbol{D}(\boldsymbol{z}_t)\,\boldsymbol{z}_t + \boldsymbol{h} \;= \underbrace{(\boldsymbol{A} + \boldsymbol{W}\boldsymbol{D}(\boldsymbol{z}_t))}_{\boldsymbol{W}(\boldsymbol{z}_t)}\,\boldsymbol{z}_t + \boldsymbol{h}$$

$$\begin{aligned} \boldsymbol{z}_k^* &= F_{\boldsymbol{\theta}}(\boldsymbol{z}_{k-1}^*) = \boldsymbol{W}(\boldsymbol{z}_{k-1}^*)\,\boldsymbol{z}_{k-1}^* + \boldsymbol{h} \\ &= \boldsymbol{W}(\boldsymbol{z}_{k-1}^*)\left[\boldsymbol{W}(\boldsymbol{z}_{k-2}^*)\,\boldsymbol{z}_{k-2}^* + \boldsymbol{h}\right] + \boldsymbol{h} \\ &= \ldots \\ &= \left(\prod_{r=0}^{k-1} \boldsymbol{W}(\boldsymbol{z}_{k-r}^*)\right)\boldsymbol{z}_k^* + \left[\left(\sum_{j=2}^{k-1}\prod_{r=0}^{k-j} \boldsymbol{W}(\boldsymbol{z}_{k-r}^*)\right) + \mathbb{1}\right]\boldsymbol{h}, \end{aligned}$$

Searching for Fixed Points and Limit Cycles

$\{\boldsymbol{z}_1^*, \ldots, \boldsymbol{z}_k^*\}$ of order k

$$\boldsymbol{D}_{ij}(\boldsymbol{z}_t) := \begin{cases} 1, & \text{if } i = j \text{ and } \boldsymbol{z}_{t,i} > 0 \\ 0, & \text{else} \end{cases}$$

$$\boldsymbol{z}_{t+1} = F_{\boldsymbol{\theta}}(\boldsymbol{z}_t) = \boldsymbol{A}\,\boldsymbol{z}_t + \boldsymbol{W}\boldsymbol{D}(\boldsymbol{z}_t)\,\boldsymbol{z}_t + \boldsymbol{h} \;= \underbrace{(\boldsymbol{A} + \boldsymbol{W}\boldsymbol{D}(\boldsymbol{z}_t))}_{\boldsymbol{W}(\boldsymbol{z}_t)}\,\boldsymbol{z}_t + \boldsymbol{h}$$

$$\boldsymbol{z}_k^* = \left(\mathbb{1} - \prod_{r=0}^{k-1} \boldsymbol{W}(\boldsymbol{z}_{k-r}^*)\right)^{-1} \left[\left(\sum_{j=2}^{k-1} \prod_{r=0}^{k-j} \boldsymbol{W}(\boldsymbol{z}_{k-r}^*)\right) + \mathbb{1}\right] \boldsymbol{h}$$

SCYFI by Eisenman et. al., 2023

1. Randomly guess indicator matrices $\{\boldsymbol{D}_l\}_{l=1}^{k}$

2. Solve

$$\boldsymbol{z}_k^* = \left(\mathbb{1} - \prod_{r=0}^{k-1} \boldsymbol{W}(\boldsymbol{z}_{k-r}^*)\right)^{-1} \left[\left(\sum_{j=2}^{k-1} \prod_{r=0}^{k-j} \boldsymbol{W}(\boldsymbol{z}_{k-r}^*)\right) + \mathbb{1}\right] \boldsymbol{h} \quad (1)$$

$$\boldsymbol{z}_l^* = F^l(\boldsymbol{z}_k^*) \text{ for } l \in [1, \dots, k-1]$$

3. Check for consistency: $\boldsymbol{D}(z_l^*) = \boldsymbol{D}_l$?

$$\boldsymbol{D}_{ij}(\boldsymbol{z}_t) := \begin{cases} 1, & \text{if } i = j \text{ and } \boldsymbol{z}_{t,i} > 0 \\ 0, & \text{else} \end{cases} \quad (2)$$

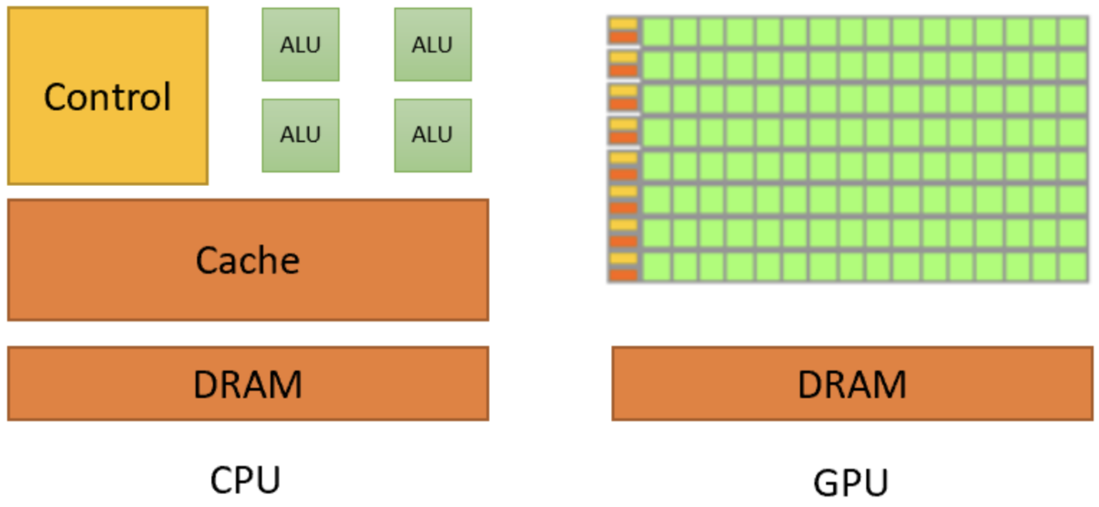
4. Else repeat with $\boldsymbol{D}_l = \boldsymbol{D}(z_l^*)$

CPU Preperation

- profile to find bottlenecks, e.g. flame graphs [4]

- store only diagonals of indicator matrices
- preallocation, in-place & element-wise computations within loops
- simplify control flow, store results of time-consuming computations (avoid repetition), etc.

GPU Computing with CUDA



[3]

1. Copy data from host (CPU) memory to device (GPU) memory
2. Load and execute GPU program
3. Copy result from device back to host

GPU Parallelization

- *CuArrays* of CUDA.jl support matrix multiplication and inversion, only rounding operations and eigenvalues on CPU [2]

- avoid CPU fallback
- minimize host-device data transfer: pre-allocation and offloading computations even if standalone slower on GPU

- future improvement (ca. x 1.5): custom SCYFI GPU kernel

Algorithm 2 GPU implementation of SCYFI

Input: PLRNN parameters $\boldsymbol{A}$, $\boldsymbol{W}$, $\boldsymbol{h}$

Parameters:

$N_{out}(k)$: max. number of random initialisations for searching k-cycles;

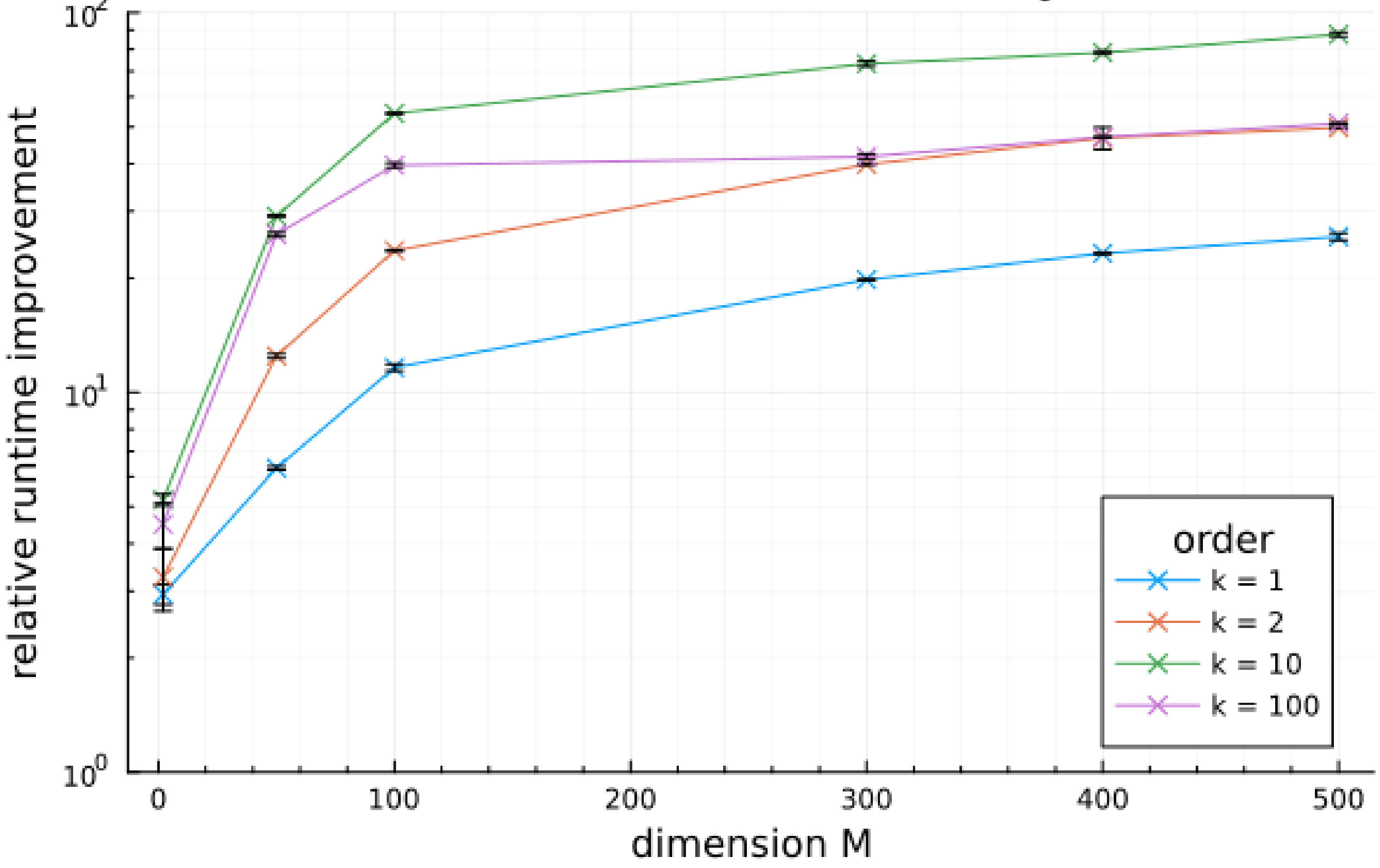
$N_{in}(k)$: max. number of iterations for searching k-cycles

Output: $\mathcal{L}$: tuple of all sets $\mathcal{L}_k$ of all discovered cycles of order $k \in [1, \dots, K_{max}]$

1: Preallocate $\mathcal{L} \rightarrow (\mathcal{L}_1, \dots, \mathcal{L}_{K_{max}})$ as tuple of $\mathcal{L}_k = \{\}$ for all $k \in [1, \dots, K_{max}]$
2: Transfer $\boldsymbol{A}$, $\boldsymbol{W}$, $\boldsymbol{h}$ as CuArrays to GPU
3: Preallocate other $M \times M$ CuArrays (corresponding to factors in (1)) on GPU
4: **for** $k \in [1, \dots, K_{max}]$ **do**
5:     Preallocate diagonals of indicator matrices $\boldsymbol{D}_{init}$, $\boldsymbol{D}^*$ as $M \times k$ CuArray on GPU
6:     Preallocate trajectory $\boldsymbol{Z}$ as $M \times k$ CuArray on GPU
7:     $i \rightarrow 0$
8:     **while** $i < N_{out}(k)$ **do** on the GPU..
9:         Fill $\boldsymbol{D}_{init}$ with random binary values
10:         $c \rightarrow 0$
11:         **while** $c < N_{in}(k)$ **do**
12:             Solve (1) inplace for cycle candidate $\boldsymbol{Z}$ using $\boldsymbol{D}_{init}$ and preallocated arrays
13:             Determine $\boldsymbol{D}^*$ from $\boldsymbol{Z}$ using (2) inplace
14:             **if** not self-consistent, i.e. $\boldsymbol{D}_{init} \neq \boldsymbol{D}^*$ **then**
15:                 $\boldsymbol{D}_{init} \rightarrow \boldsymbol{D}^*$ inplace
16:             **else** on the CPU..
17:                 **if** $\boldsymbol{Z}$ not found yet, i.e. no column $\boldsymbol{Z}_j \in \mathcal{L}_i$ for any $i, j \in [1, \dots, k]$ **then**
18:                     Transfer the set $\{\boldsymbol{Z}_j\}_{j=1}^k$ of all columns of $\boldsymbol{Z}$ on CPU and add to $\mathcal{L}_k$
19:                 **end if**
20:                 **go to** line 8, setting $i \rightarrow c \rightarrow 0$
21:             **end if**
22:             $c \rightarrow c + 1$
23:         **end while**
24:         $i \rightarrow i + 1$
25:     **end while**
26: **end for; return** $\mathcal{L}$

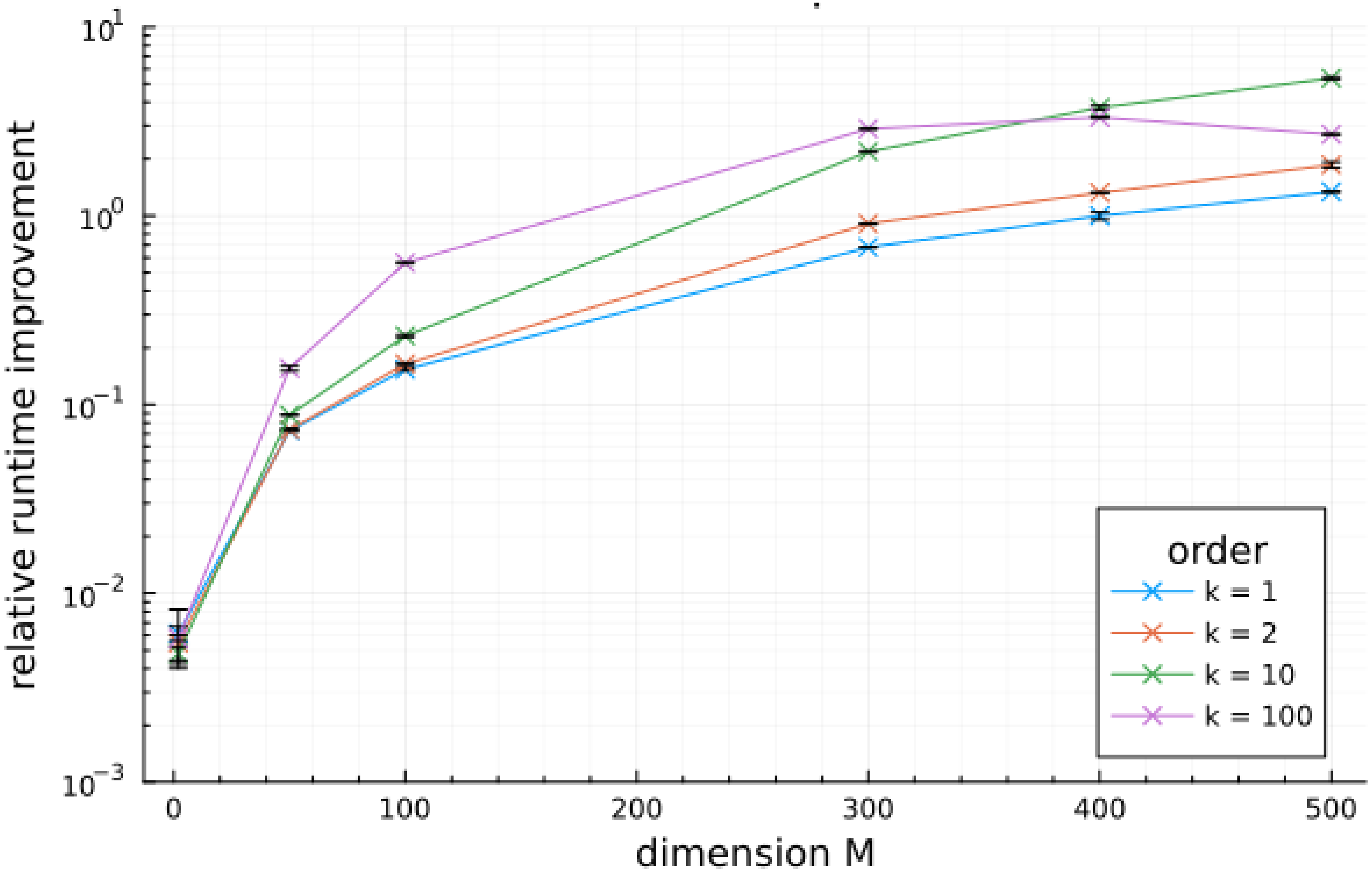
CPU Version

$$\text{CPU improvement} = \frac{\text{runtime of improved CPU version}}{\text{runtime of original version}}$$

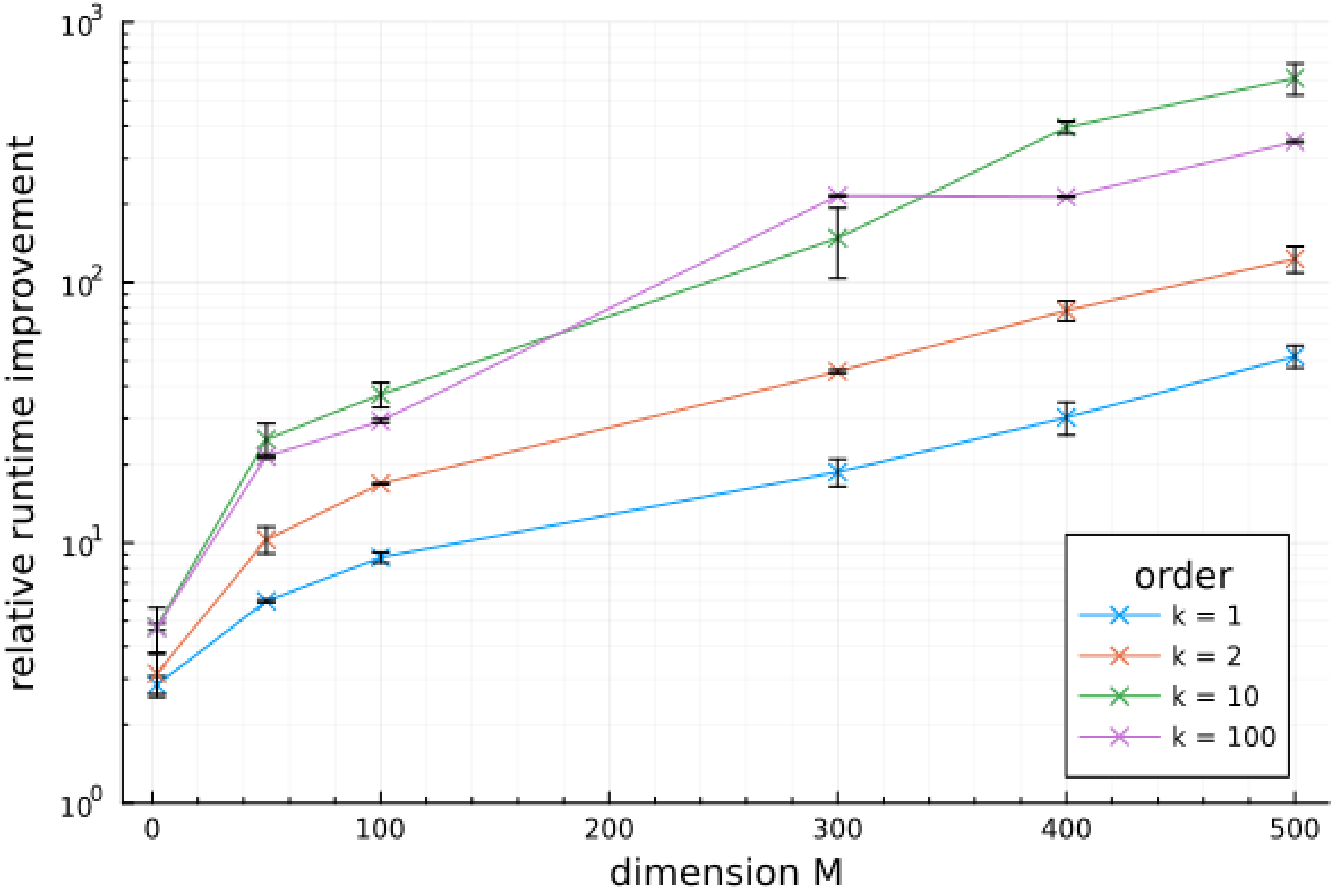


GPU Version

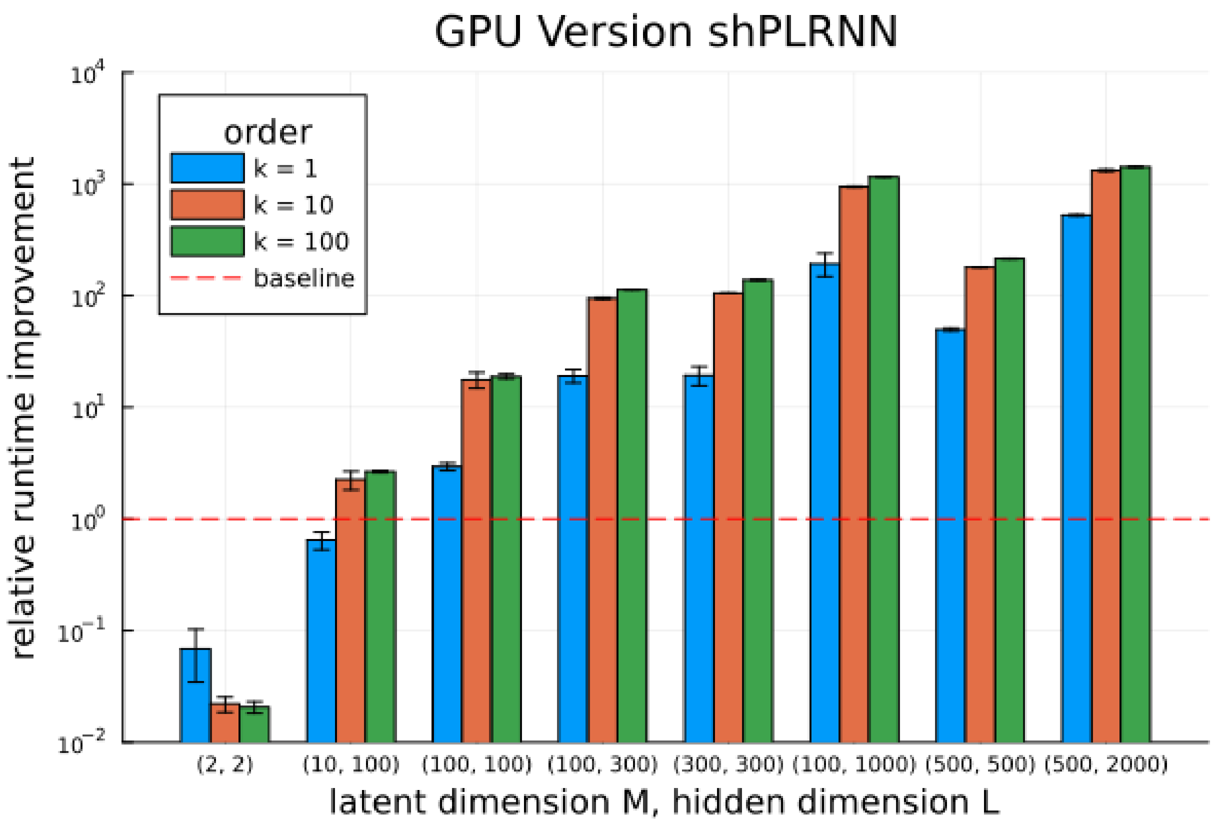
$$\text{CPU vs. GPU improvement} = \frac{\text{runtime of improved GPU version}}{\text{runtime of improved CPU version}}$$

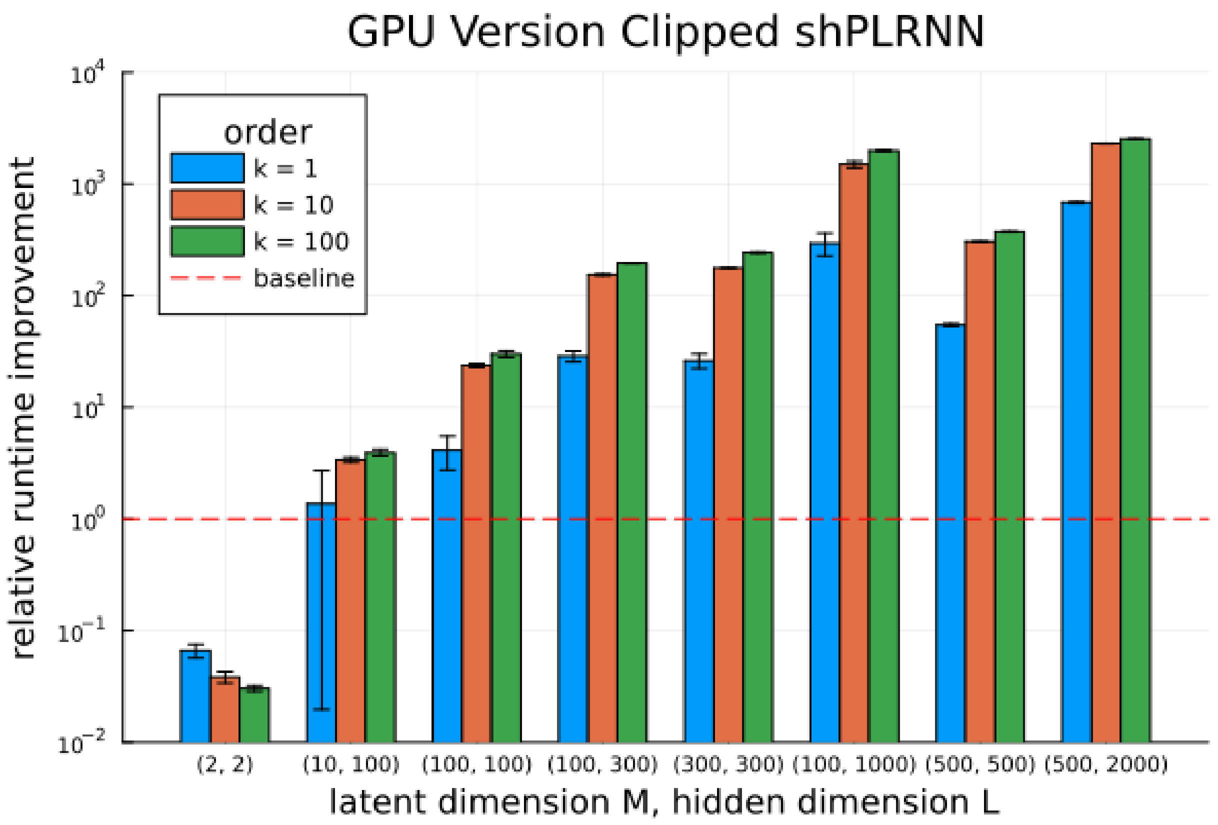


Hybrid Version



(Clipped) Shallow PLRNN Version





Summary & More

- Core performance tricks that worked for me:
    - profile first to find actual bottlenecks and after every change
    - prepare CPU program by using improved data structures, preallocation and in-place computations
    - offload matrix multiplication and inversion to GPU but minimize data transfer between host and device
- More Versions:
    - shallow and clipped shallow PLRNN
    - multiple CPU threads + GPUs
    - initialize subregions visited during data reconstruction

Selected References

- [1] Lukas Eisenmann et al. “Bifurcations and loss jumps in RNN training”. In: Advances in Neural Information Processing Systems. Vol. 36. 2023.
- [2] CUDA.jl Documentation. url: https://cuda.juliagpu.org/stable/. (accessed: 2024-02-16).
- [3] Pradeep Gupta. The CUDA Refresher Series. url: https://developer.nvidia.com/blog/tag/cuda-refresher/. (accessed: 2024-02-16).
- [4] Brendan Gregg. “The Flame Graph: This visualization of software execution is a new necessity for performance profiling and debugging.” In: Queue 14.2 (Mar. 2016), url: https://doi.org/10.1145/2927299.2927301.
- [5] BenchmarkTools.jl Documentation. url: https://juliaci.github.io/BenchmarkTools.jl (accessed: 2024-02-16).

Questions?