A Algorithm for Multi-step Selective-Scan-Update kernel

Algorithm 1 Fused multi-step State-Update kernel. This Algorithm is simplified with the only input being x, A, B, C, Δ, h in reference to the implementation of [10].

Notation $Z^* \leftarrow Z$: load Z into shared memory for computation. $Z \leftarrow Z^*$: Storing Z out to High-Bandwidth-Memory (HBM). Any variable with $*$ means that this variable is in shared memory, everything else is in HBM.

Shapes B - Batch, L - Sequence Length, H - Num heads, P - State Dimension , D - Head Dimension, G - Num groups

Input: $h_0 : (B, H, D, P)$ - initial state, $x : (B, L, H, D)$ $A : (H, D, P)$, $B : (B, L, G, P)$, $C : (B, L, G, P)$, $\Delta : (B, L, H, D)$

Output: $y : (B, L, H, D)$ - Output, $\hat{h} : (B, H, D, P)$ - updated state

1: $h^* \leftarrow h_0$
2: $i \leftarrow 0$
3: **while** $i < L$ **do**
4: $\quad x^* \leftarrow x[:, i, :, :]$
5: $\quad \Delta^* \leftarrow \Delta[:, i, :, :]$
6: $\quad B^* \leftarrow B[:, i, :, :]$
7: $\quad C^* \leftarrow C[:, i, :, :]$
8: $\quad A^* \leftarrow A$
9: $\quad \bar{A}^* \leftarrow \exp(\Delta^* \times A^*)$
10: $\quad \bar{B}^* \leftarrow \Delta^* \times B^*$
11: $\quad h^* \leftarrow h^* \times \bar{A}^* + \bar{B}^* \times x^*$ ▷ State is not stored out to HBM at this point
12: $\quad y^* \leftarrow C^* \times h^*$
13: $\quad y[:, i, :, :] \leftarrow y^*$
14: **end while**
15: $\hat{h} \leftarrow h^*$

B Algorithm for Activation Replay

Line 1, 3, 5, 8 are performed by the original Mamba-2 block to get the output and update the states. The added operations are the extra $Conv1d\_update$ and SSM_update at line 2 and 5 to attain the updated states using the correct cached activations. Line 4 and 7 saves the activation of the current iteration for the use of next iteration.

Algorithm 2 Abstracted Mamba-2 block with Activation Replay. The two functions are simplified abstraction of what is actually used in the implementation.

Notation $A[a:b]$: a-th position to b-th position of A in the sequence length dimension
function $Conv1d\_update(x, B, C, M)$:
  return (x', B', C', M') ▷ Performs 1d convolution on x, B, C, and update conv state M
function $SSM\_update(x, B, C, dt, S)$:
  return (y, S') ▷ Performs SSM update with x, B, C, dt to get output y and update SSM state S
Input: E : token embedding, S : SSM state, M : Conv state, c : number of correct tokens
  $(x_{cache}^{pre\_conv}, B_{cache}^{pre\_conv}, C_{cache}^{pre\_conv})$: Cached activation for Conv states from last forward pass
  $(x_{cache}^{post\_conv}, B_{cache}^{post\_conv}, dt_{cache})$: Cached activation for SSM states from last forward pass
Output: $\hat{E}$: New token embedding, $\hat{S}$: SSM state after the i-th correct token, $\hat{M}$: Conv state after the i-th correct token

$x^{pre\_conv}, B^{pre\_conv}, C^{pre\_conv}, dt \leftarrow in\_projection(E)$
$(\_, \_, \_, \hat{M}) \leftarrow Conv1d\_update(x_{cache}^{pre\_conv}[0:c], B_{cache}^{pre\_conv}[0:c], C_{cache}^{pre\_conv}[0:c], M)$ ▷ Getting updated Conv state
$(x^{post\_conv}, B^{post\_conv}, C^{post\_conv}, \_) \leftarrow Conv1d\_update(x^{pre\_conv}, B^{pre\_conv}, C^{pre\_conv}, \hat{M})$ ▷ Actual Conv1d update
$(x_{cache}^{pre\_conv}, B_{cache}^{pre\_conv}, C_{cache}^{pre\_conv}) \leftarrow (x^{pre\_conv}, B^{pre\_conv}, C^{pre\_conv})$ ▷ Saving activations for next forward pass
$(\_, \hat{S}) \leftarrow SSM\_update(x_{cache}^{post\_conv}[0:c], B_{cache}^{post\_conv}[0:c], dt_{cache}[0:c], S)$ ▷ Getting the updated state using the correctly predicted tokens
$(y, \_) \leftarrow SSM\_update(x^{post\_conv}, B^{post\_conv}, C^{post\_conv}, dt, \hat{S})$ ▷ Actual SSM update
$(x_{cache}^{post\_conv}, B_{cache}^{post\_conv}, dt_{cache}) \leftarrow (x^{post\_conv}, B^{post\_conv}, dt)$ ▷ Saving activations for next forward pass
$\hat{E} \leftarrow out\_projection(y)$