

Pure Attention

Deconstructing the Magic: A White-Box Engineering Approach

Sincari Sebastian-George
Andrei Cristian-David

Universitatea din Bucureşti - Facultatea de Matematică şi Informatică
Specializarea Informatică

Februarie, 2026

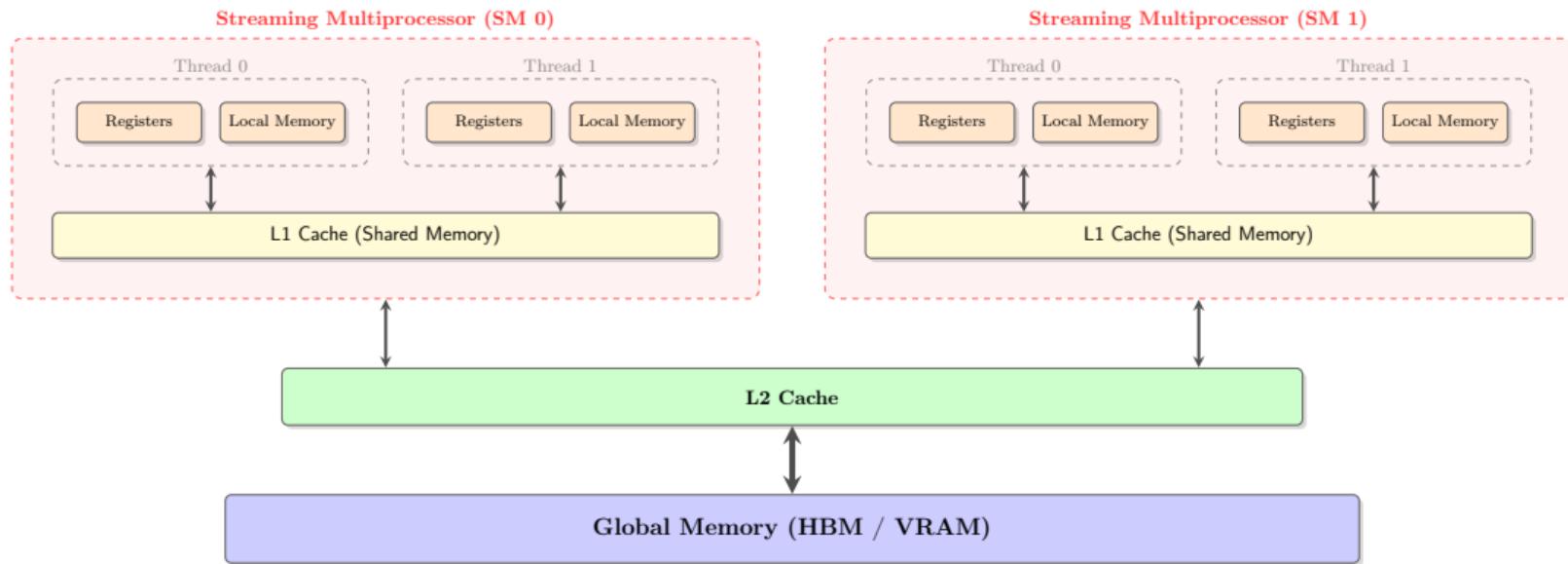
Problema Generală

- Librăriile moderne de Deep Learning transformă funcționalitățile în "Black Box".
- **Motivăție:** Demistificarea "magiei" din spatele antrenării (scop educațional).

Contribuție personală (PureAttention)

- Implementare *White-box* a mecanismelor specifice Deep Learning.
- Folosire **CUDA** pentru implementările low-level, **C++** pentru abstractizarea funcționalităților și **Python** pentru wrapping.
- Implementarea **Flash Attention V2**.
- Validare matematică vs. soluții analitice și PyTorch.
- Analiza performanței (GFLOPS) și a optimizărilor kernel.

Arhitectura GPU



Memory Coalescing

Row-Major
(Coalesced Access)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

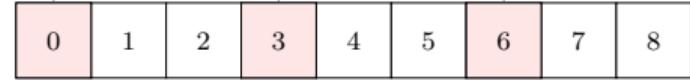
Physical
Memory:



One Memory Transaction

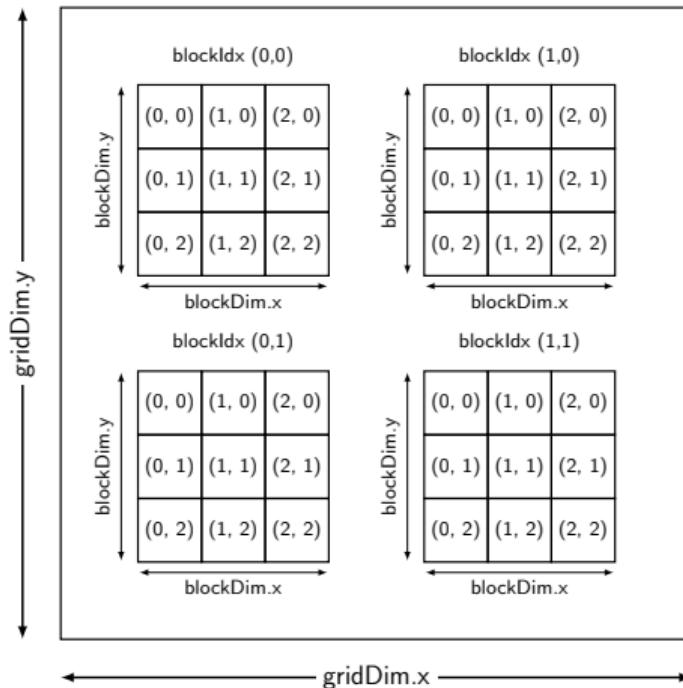
Column-Major
(Strided Access)

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)



Multiple Memory Transactions

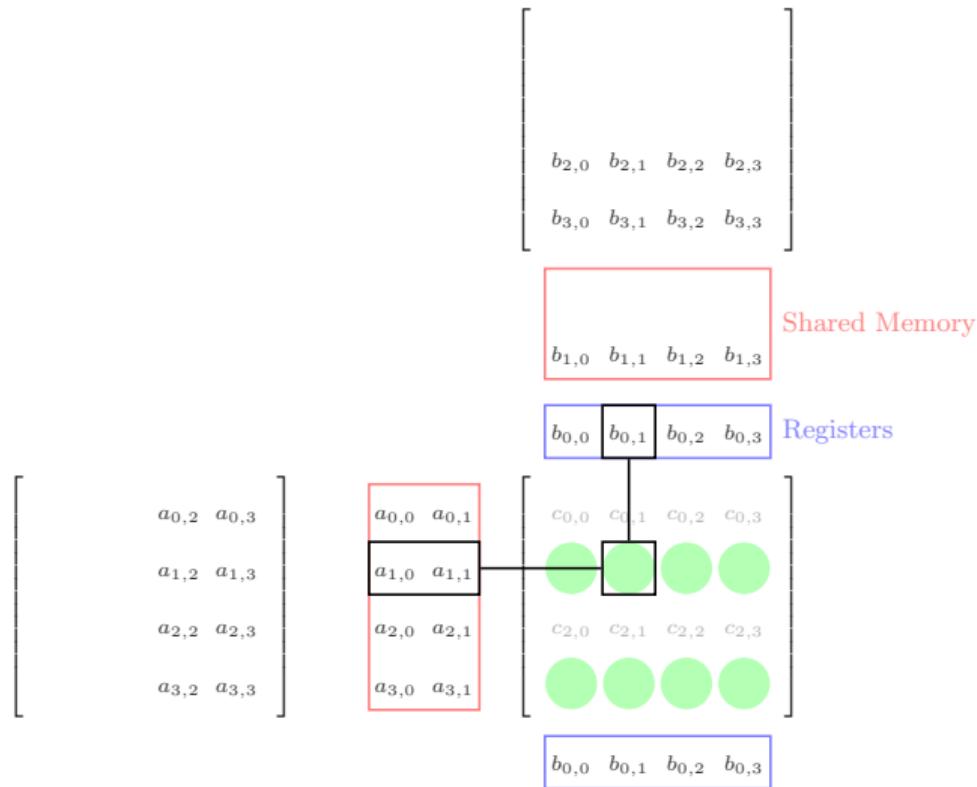
Thread Indexing



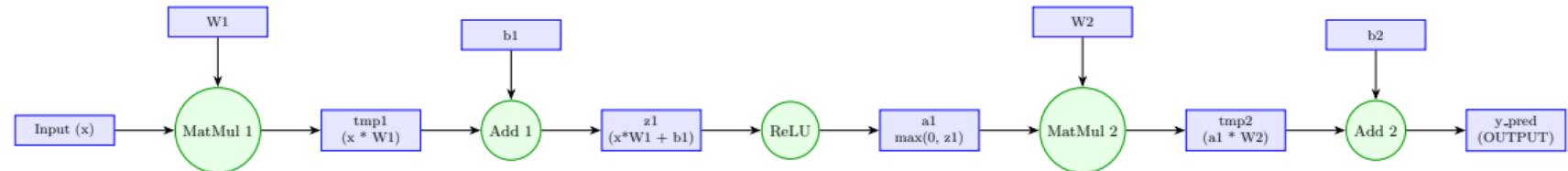
$$x = \text{blockIdx.x} \cdot \text{blockDim.x} + \text{threadIdx.x} \quad (1)$$

$$y = \text{blockIdx.y} \cdot \text{blockDim.y} + \text{threadIdx.y} \quad (2)$$

Tiled Shared Memory & Thread Coarsening

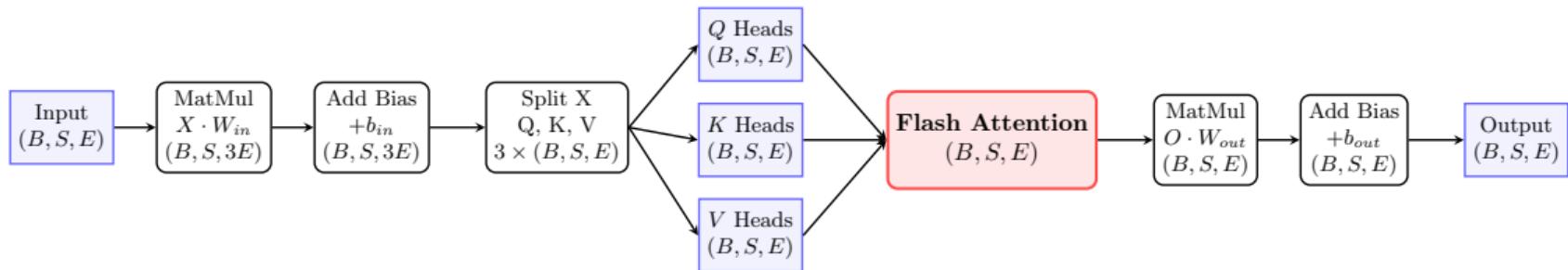


Arhitectura Sistemului & Funcționalități Implementate

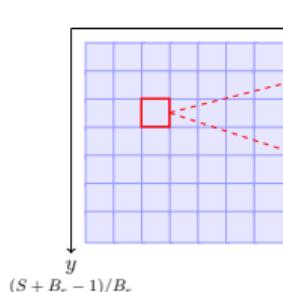


- **Funcție de Loss:** Mean Squared Error (MSE)
- **Optimizer:** Adam
- **Funcție de Activare:** ReLU
- **Initializare Weights:** He Initialization

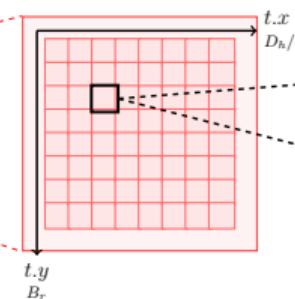
Flash Attention Pipeline



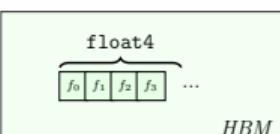
1. CUDA Grid
 $(GridDim)$



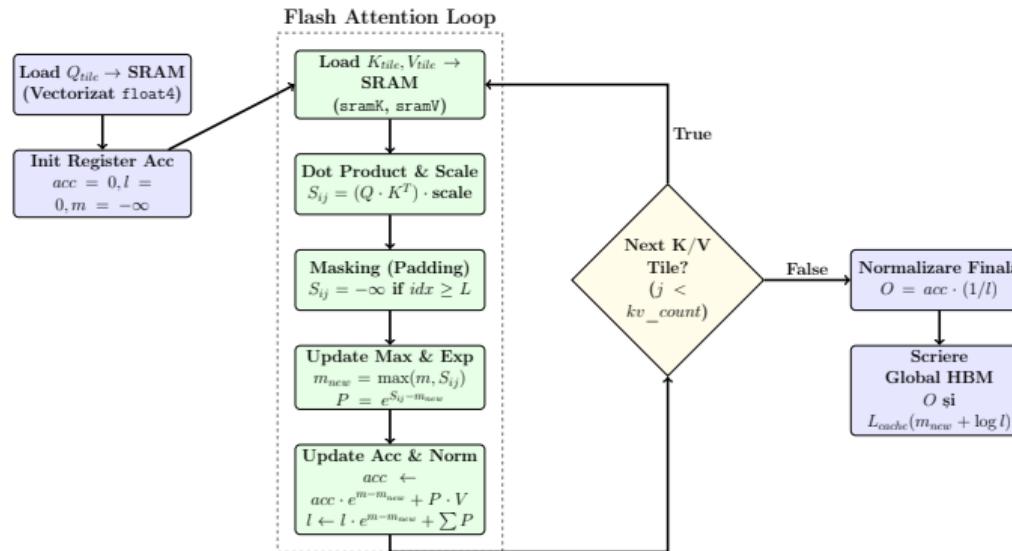
2. Thread Block
 $(BlockDim)$



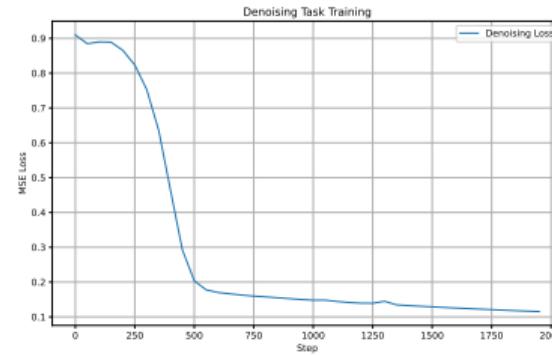
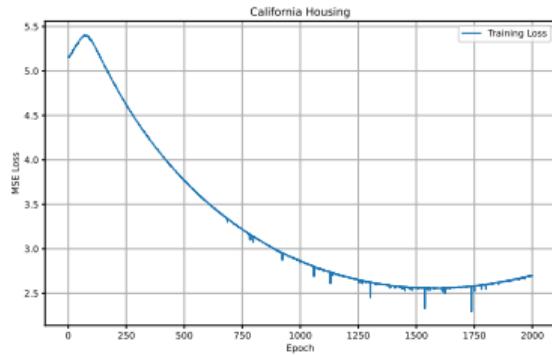
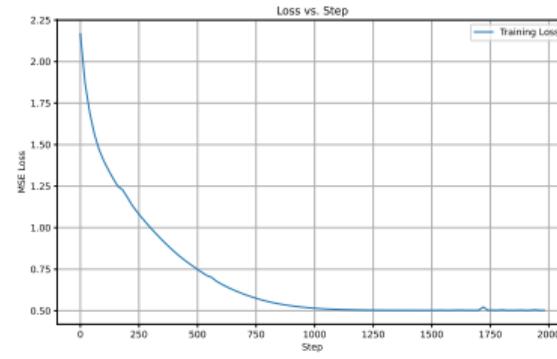
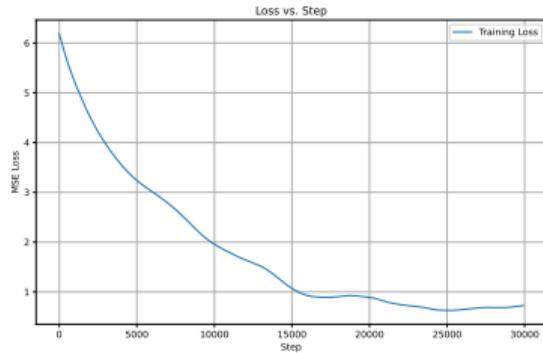
3. Memory Map
 (float4)



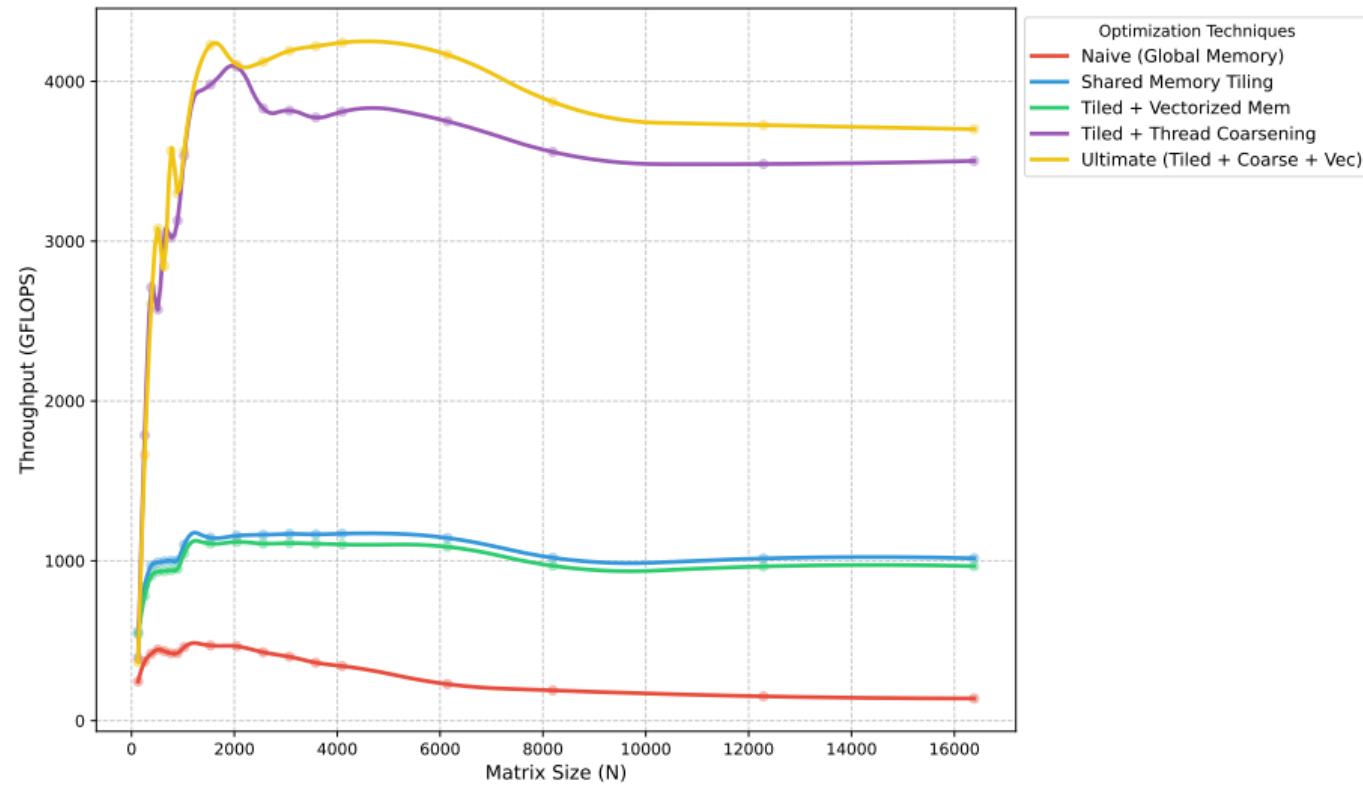
Flash Forward Execution



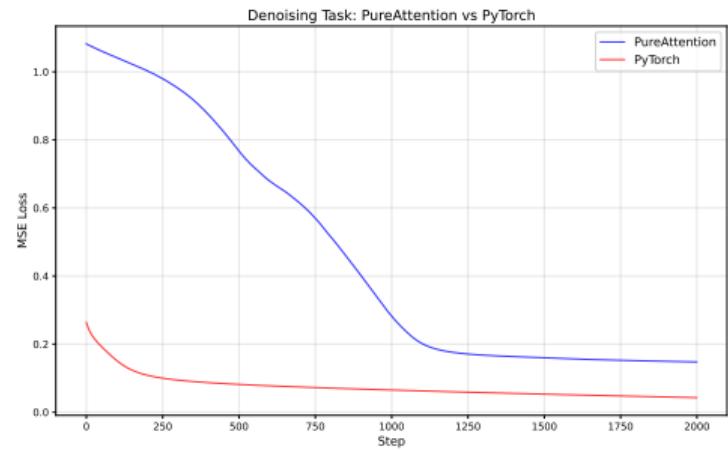
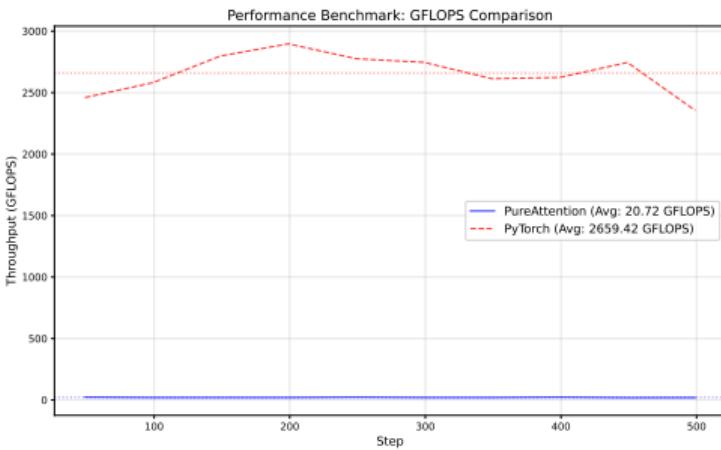
Validare Matematică și Convergență pe Serii de Timp



Performanță Kernel



Comparare cu PyTorch



- **Implementarea unui Caching Allocator:** Prin analiza de performanță am identificat alocarea dinamică a memoriei (apeluri repetitive de `cudaMalloc`) ca fiind un bottleneck major. O strategie de reutilizare a blocurilor de memorie ar reduce latențele semnificativ.
- **Suport pentru Tensor Cores:** Utilizarea instrucțiunilor `nvcuda::wmma` pentru a exploata unitățile matriciale specializate din arhitecturile Volta și Ampere, permitând accelerarea calculelor.
- **Kernel Fusion:** Combinarea operațiilor adiacente din graful de computație (ex: MatMul + Bias + Activation) într-un singur kernel pentru a reduce traficul de memorie globală și a crește intensitatea aritmetică.