NYU

Lecture 11: Transformer & DNN Training Accelerators

Notes

- Final Presentation on Dec 16, and Dec 17
- Lab 3 grade has been posted

NYU SAI LAB

Recap

- Hardware accelerator: Overview
- Convolutional operation conversion
- Systolic array
- Convolutional Neural Network System
- Popular accelerator design

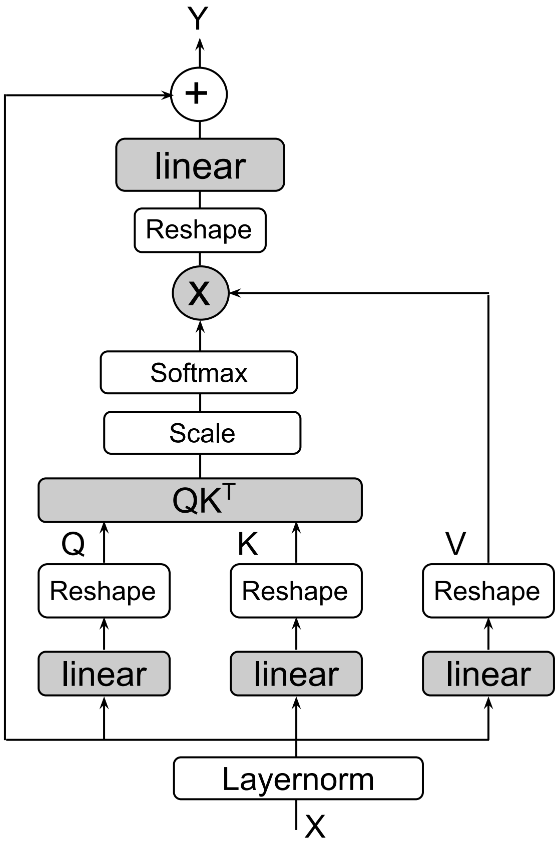
Topics

- Hardware design for Operations other than Matrix Multiplication
- Hardware architecture for backward propagation design.
- Training Accelerator Design: FAST

NYU SAI LAB

Self-Attention Block

- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
  - QK^T→(B, L✖E) ✖ (B,E✖L) → (B, L✖L)
- Scale and normalize the score using softmax.
  - Softmax(QK^T) → (B, L✖L)
- Multiply each value vector by the softmax score.
  - Softmax(QK^T) ✖ V
  - (B, L✖L) ✖ (B, L✖E) → (B, L✖E)
- Pass the result to the linear layer, sum with the input.

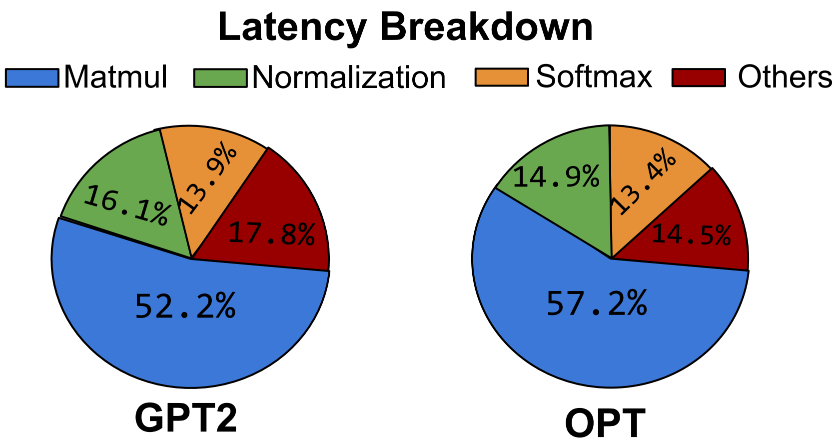


NYU SAI LAB

Operations Other than Multiplications

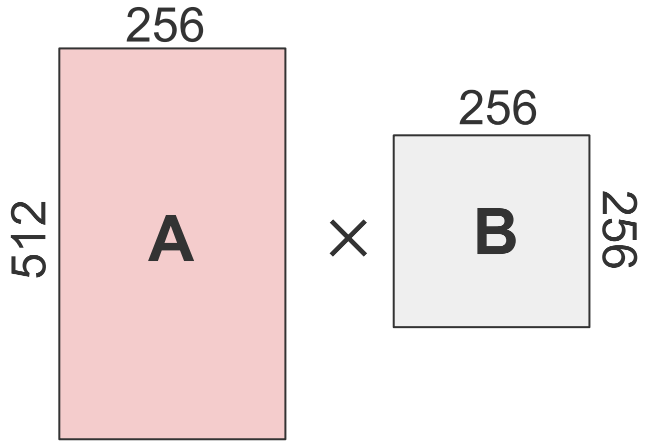
- Transposition
- Nonlinear operations
  - Softmax
  - LayerNorm
  - GeLU

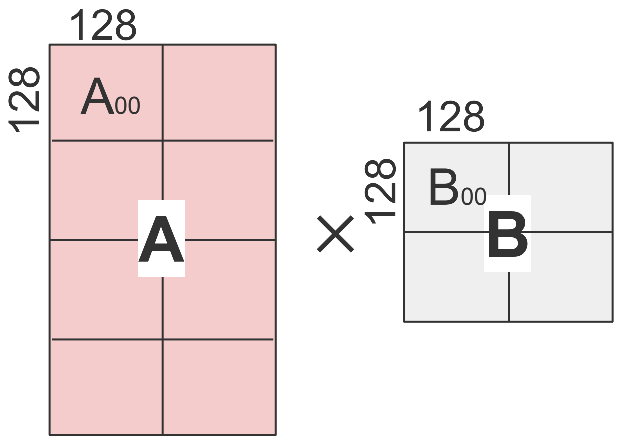
Breakdown on Computational Cost



- Matmul still contributes to majority of the overall latency.
- Nonlinear operations are not negligible.
- Also other operations (e.g., transposition) also contributes to a great portion of the overall latency.

Matrix Multiplication

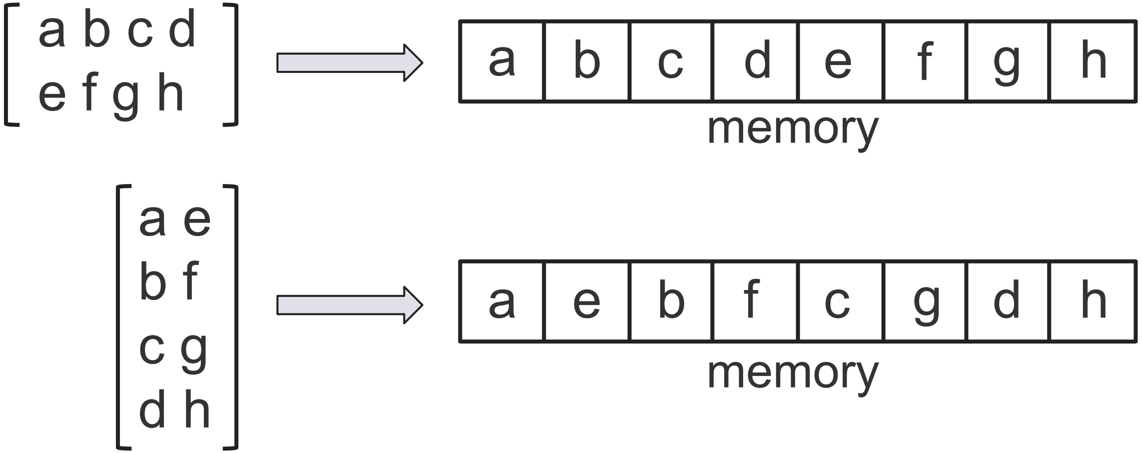




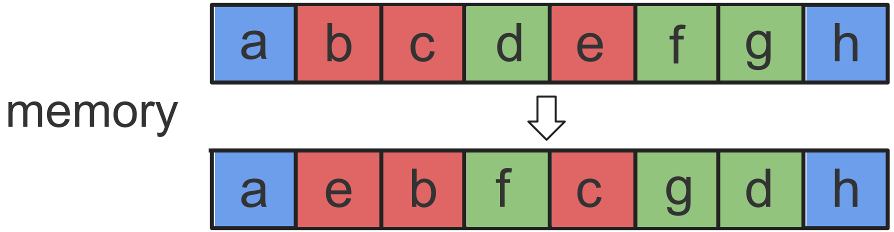
- The large matrix operands are first partitioned into tiles that can fit the size of the compute core.

In-Place Matrix Transposition

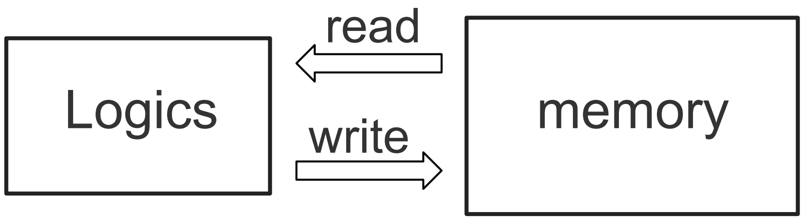
- In-place matrix transposition refers to the process of transposing a matrix directly within its existing memory space, requiring only a minimal amount of extra storage.



In-Place Matrix Transposition

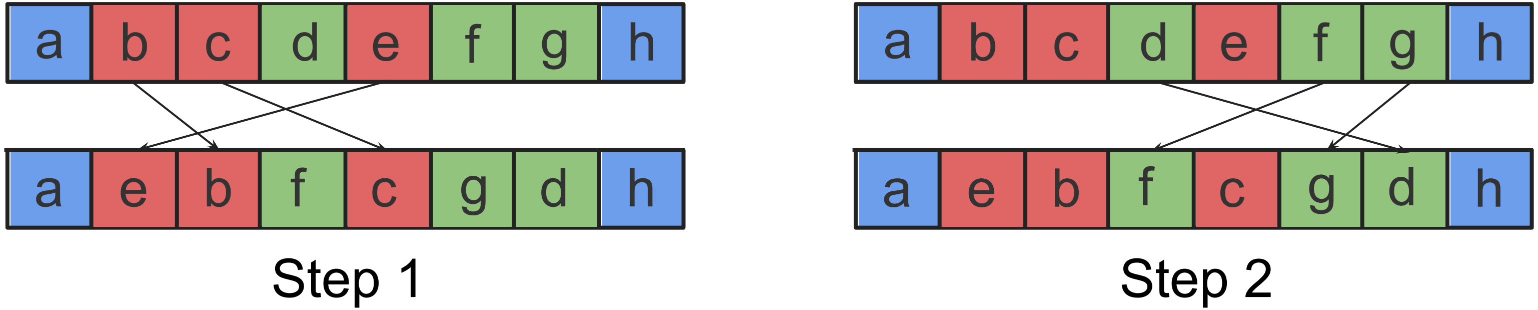


(b,c,e) → (e,b,c)
(d,f,g) → (f,g,d)



- Need to read multiple entries from the memory, permute them and write them back.
- This operation should be performed efficiently with minimal memory access cost.

In-Place Matrix Transposition



- The search for optimal swapping patterns that minimize permutations is a well-established problem in mathematics.

Norman Brenner, "Algorithm 467: matrix transposition in place," ACM Transactions on Mathematical Software 16 (11), p. 692-694 (1973). doi:10.1145/355611.362542

Implementation of Nonlinear Operations: Softmax

- Softmax operations are heavily adopted in the transformer.

$$s_i = \frac{e^{z_i}}{\Sigma_{j=0}^{j=N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

- For positive z with INT representation, we can approximate the values of e^z using the following derivations:

$$e^z = 2^{z\ log_2 e} = 2^{u+v} \qquad log_2 e \approx 1.0111_2$$

$$z\ log_2 e \approx z + (z >> 2) + (z >> 3) + (z >> 4)$$

- To compute 2^{u+v}, we can perform shift and multiplication:

$$e^{z'} = 2^{u+v} \approx 2^u (1 + v/2)$$

u and v are the integer and fractional part of the exponent, v/2 is the mantissa, u is the exponent

NYU SAI LAB

Taylor Approximation

- A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function f(x) about a point x=a is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots.$$

- For small v, e^v can be approximated as:

$$e^v \approx 1 + \frac{v}{2} \qquad 2^v \approx 1 + \frac{v}{2} \qquad log(1+x) \approx x$$

NYU SAI LAB

Xia, Tianhua, and Sai Qian Zhang. "Softmax Acceleration with Adaptive Numeric Format for both Training and Inference." *arXiv preprint arXiv:2311.13290* (2023).

Taylor Approximation

- A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function f(x) about a point x=a is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots.$$

- For small v, e^v can be approximated as:

$$e^v \approx 1 + \frac{v}{2} \qquad 2^v \approx 1 + \frac{v}{2} \qquad log(1+x) \approx x$$

NYU SAI LAB

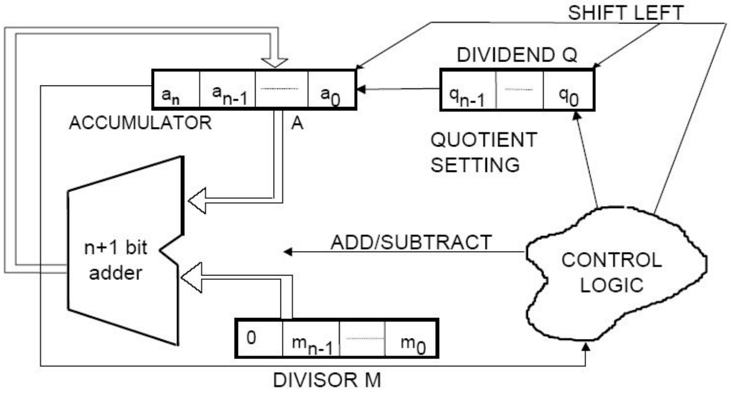
Xia, Tianhua, and Sai Qian Zhang. "Softmax Acceleration with Adaptive Numeric Format for both Training and Inference." *arXiv preprint arXiv:2311.13290* (2023).

Division

- To implement division operation with FP format, we can always apply the following derivations:

$$\frac{a}{b} = 2^{e_a}(1+m_a)/2^{e_b}(1+m_b) = 2^{e_a - e_b + log_2(1+m_a) - log_2(1+m_b)}$$
$$\approx 2^{e_a - e_b + m_a - m_b} \approx 2^{e_a - e_b}(1 + \frac{m_a + m_b}{2})$$

- For INT division, we can also implement the hardware divisor.



NYU SAI LAB

Implementation of Nonlinear Operations: LayerNorm

- For the input vector z, the normalization operation requires to computes its mean and variance, then the intermediate results are scaled with some predefined values.

$$s = \alpha \frac{z - \mu_z}{\sigma_z} + \beta \qquad \mu_z = \frac{\sum_i z_i}{N} \qquad \sigma_z = \sqrt{\frac{\sum_i (z_i - \mu_z)^2}{N}}$$

- Most of the operations are supported, the inverse of square root can be computed as follows:

$$y = \frac{1}{\sqrt{x}} \qquad \log_2(y) = -\frac{1}{2}\log_2(x)$$

NYU SAI LAB

Implementation of Nonlinear Operations: LayerNorm

- Most of the operations are supported, the inverse of square root can be computed as follows:

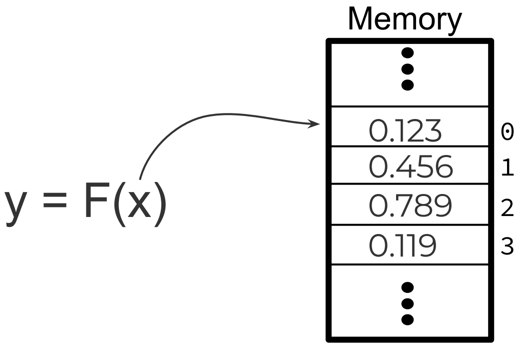
$$y = \frac{1}{\sqrt{x}} \qquad \log_2(y) = -\frac{1}{2}\log_2(x)$$

$$x = 2^{E_x - Q}(1 + M_x/2^L) \quad \log_2 x = E_x - Q + \log_2(1 + M_x/2^L)$$
$$\approx E_x - Q + M_x/2^L + \sigma_x$$

- Q is the bias, Ex and Mx are the binary representations of the exponent and mantissa, respectively.

Table Lookup

- For other complicated nonlinear functions, we can always precompute the results and store them in the buffer.



- However, this will inevitably lead to additional memory access cost and footprint.

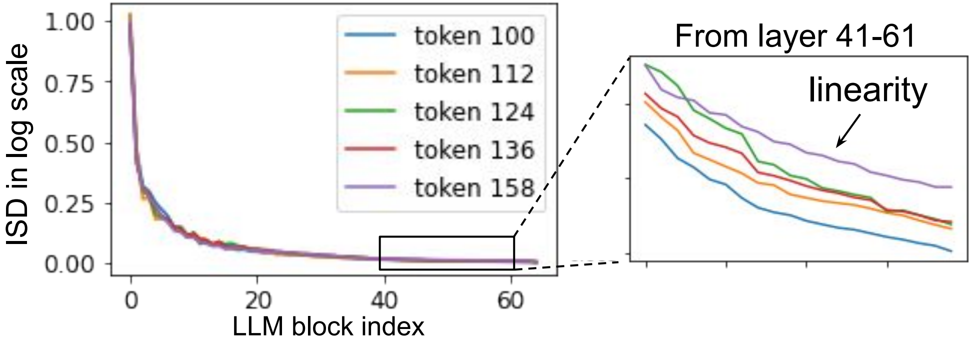
NYU SAI LAB

HAAN: LayerNorm Accelerator

Layer Normalization:

$$s = \alpha \frac{z - \mu_z}{\sigma_z} + \beta$$

Computing the inverse of standard deviation of costly



- Exploit correlation in input statistics across layers.
- Skip redundant computations and estimate normalization statistics.

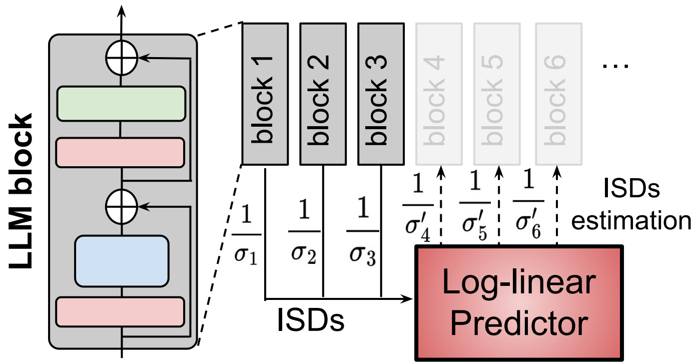
Peng, Tianfan, et al. "HAAN: A Holistic Approach for Accelerating Normalization Operations in Large Language Models." *arXiv preprint arXiv:2502.11832* (2025).

HAAN: LayerNorm Accelerator

Layer Normalization:

$$s = \alpha \frac{z - \mu_z}{\sigma_z} + \beta$$

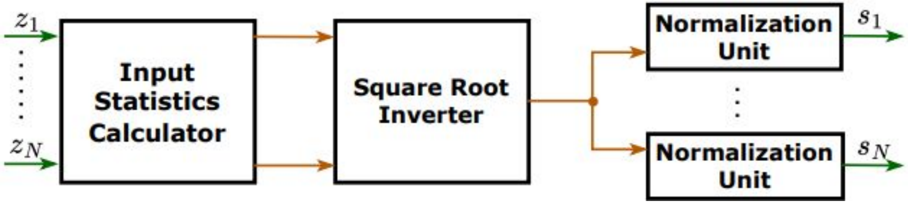
Computing the inverse of standard deviation of costly

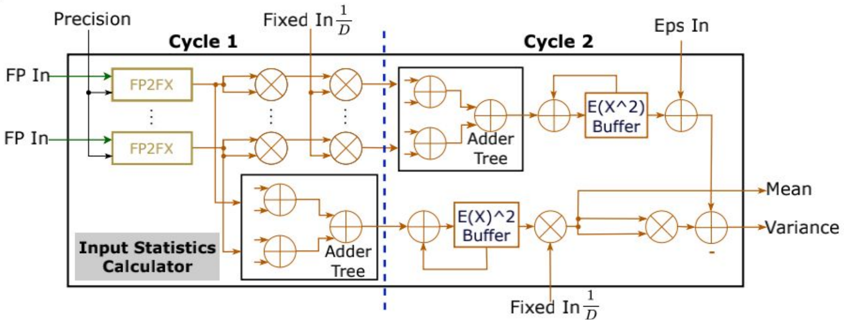


- Exploit correlation in input statistics across layers.
- Skip redundant computations and estimate normalization statistics.

NYU SAI LAB

HAAN: LayerNorm Accelerator

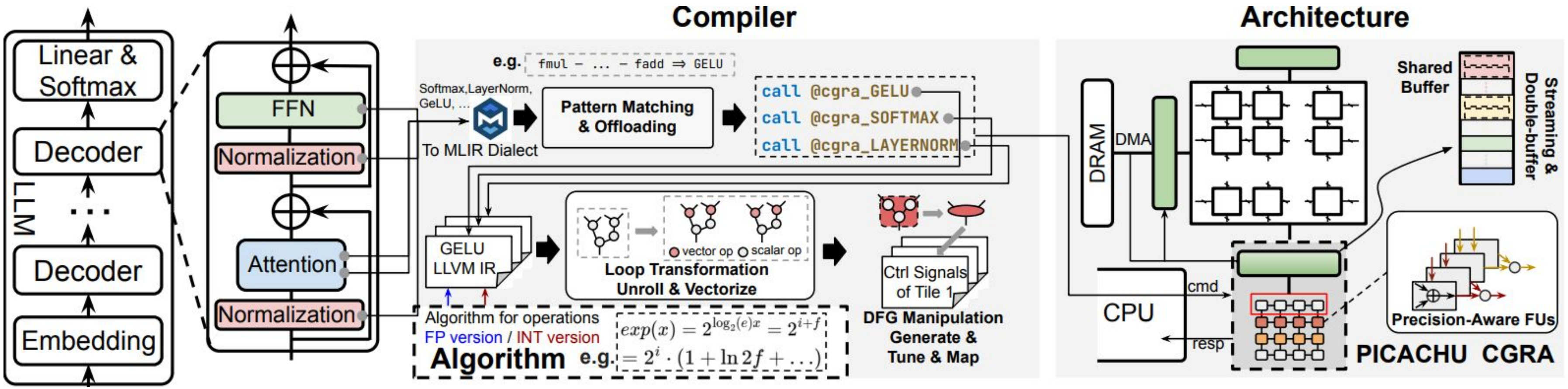




- **Overall Architecture**
  - Input Statistics Calculator.
  - Square Root Inverter.
  - Normalization Unit.
- **Input Statistics Calculator**
  - Compute mean and variance.
  - Parallel processing to reduce latency.
- **Square Root Inverter**
  - Approximate inverse square root using Newton's method.
  - Support for layer skipping.

NYU SAI LAB

PICACHU



- PICACHU is a plug-in coarse-grained reconfigurable accelerator tailored to efficiently handle nonlinear operations by using custom algorithms and a dedicated compiler toolchain.

Qin, Jiajun, et al. "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2025.

PICACHU

Categories	Nonlinear Operations	Mathematical Operator	Representative LLMs
Activation Function	Softmax$(x_i) := \frac{\exp(x_i)}{\sum_{j=1}^{k}\exp(x_j)} = \frac{\exp(x_i - u)}{\sum_{j=1}^{k}\exp(x_j - u)}$; $u = \max_{j=1} x_j$	Division, Exponential	All
	$\mathrm{ReLU}(x) := \max(0, x)$	Maximum	OPT [145], T5 [90]
	$\mathrm{GeLU}(x) := 0.5x\left(1 + \mathrm{Tanh}(\sqrt{2/\pi}(x + 0.044715x^3))\right)$; $\mathrm{Tanh}(x) = (\exp(x) + \exp(-x))/(\exp(x) - \exp(-x))$	Division, Exponential	GPT [14, 84, 87, 88], BLOOM [57], Falcon [83], PanGu-α [144], Jurassic-1 [64], Gopher [89]
	$\mathrm{GeGLU}(x) := \mathrm{GeLU}(xW + b) \oplus (xV + c)$	Division, Exponential	LaMDA [110], GLM-130B [143]
	$\mathrm{SwiGLU}(x) := \mathrm{SiLU}(xW + b) \oplus (xV + c)$; $\mathrm{SiLU}(x) = x \cdot \mathrm{sigmoid}(x) = x \cdot \frac{1}{1+\exp(-x)}$	Division, Exponential	PaLM [17], LLaMA [113, 114], Qwen [7], DeepSeek [11], InternLM [15], Yi [135]
Normalization Function	LayerNorm$(x_i) := \frac{x_i - \mu}{\sigma}$; $\mu = \frac{1}{C}\sum_{i=1}^{C} x_i$, $\sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i - \mu)^2 + \epsilon}$	Inverted Square Root	GPT [14, 84, 87, 88], BLOOM [57], BERT [20], OPT [145], PanGu-α [144], Jurassic-1 [64]
	RMSNorm$(x_i) := \frac{x_i}{\sigma}$; $\sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i)^2 + \epsilon}$	Inverted Square Root	LLaMA [113, 114], T5 [90], Mistral [43], Qwen [7], DeepSeek [11], Gopher [89]
Positional Embedding	RoPE $\begin{pmatrix} x_{2i-1} \\ x_{2i} \end{pmatrix} = \begin{pmatrix} x_{2i-1}\cos(m\theta_i) - x_{2i}\sin(m\theta_i) \\ x_{2i-1}\sin(m\theta_i) + x_{2i}\cos(m\theta_i) \end{pmatrix}$; $\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]$	Sine, Cosine	GPTNeo-20B [13], LLaMA [113, 114], PaLM [17], GLM-130B [143], Qwen [7], DeepSeek [11]

- All nonlinear operations within LLM can be broken down into various mathematical operators.

NYU SAI LAB

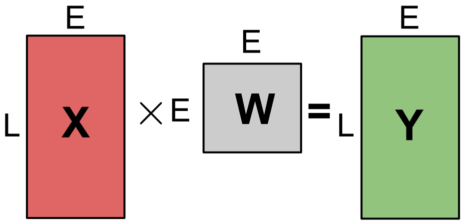
Qin, Jiajun, et al. "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2025.

Topics

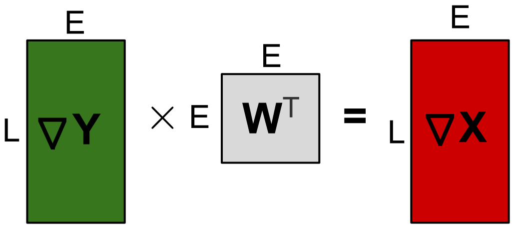
- Hardware design for Operations other than Matrix Multiplication
- Hardware architecture for backward propagation design
- Training Accelerator Design: FAST

NYU SAI LAB

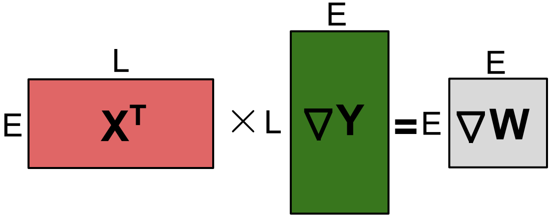
In-place Transposed Matrix Multiplication



Forward propagation

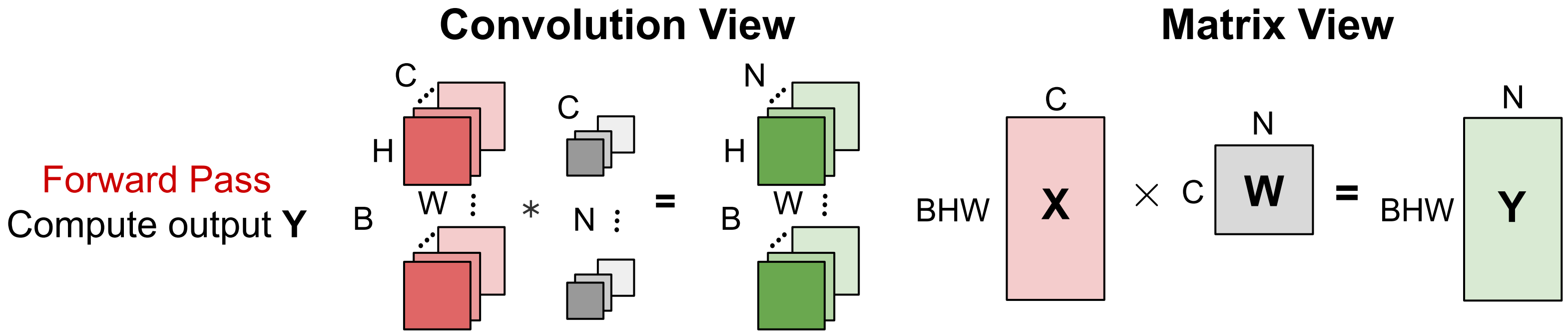


Backward propagation: weight gradient computation



Backward propagation: input gradient computation

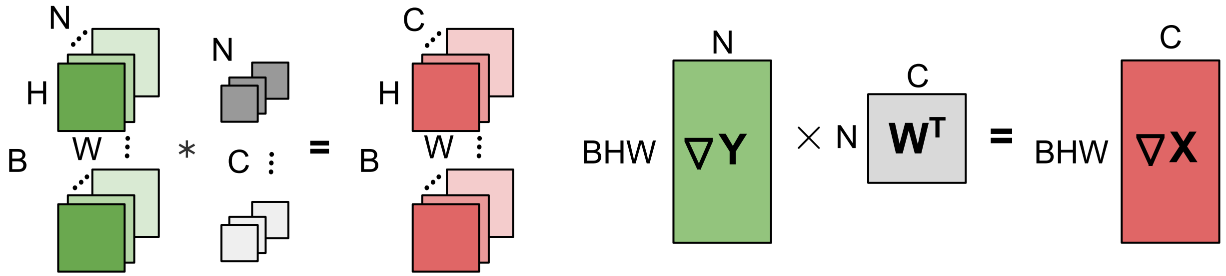
Forward Pass for Convolutional Layer



- Assume a weight kernel size of 1×1.

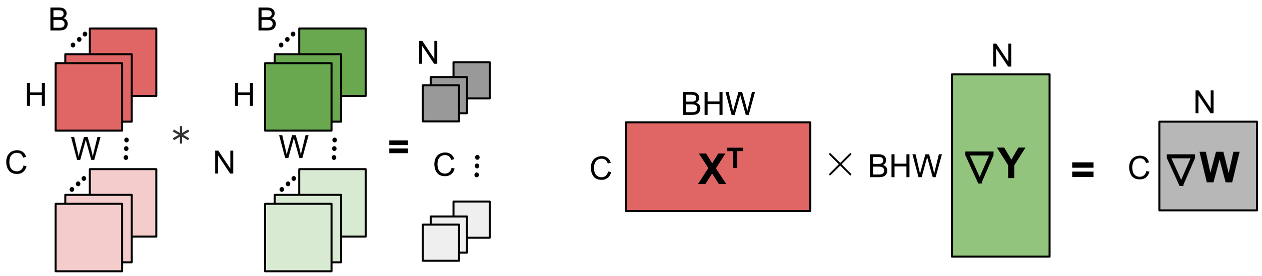
Backward Pass for Convolutional Layer

Backward Pass
Compute Activation gradients $\nabla \mathbf{X}$



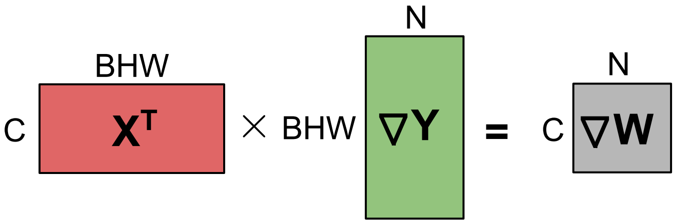
Backward Pass for Convolutional Layer

Backward Pass
Compute weight gradients ∇**W**



In-place Transposed Matrix Multiplication

- In the training of neural networks, we need to perform transposed matrix multiplication
- Instead of using a separate hardware for matrix transposition, transposed matrix multiplication can be performed using a systolic array.



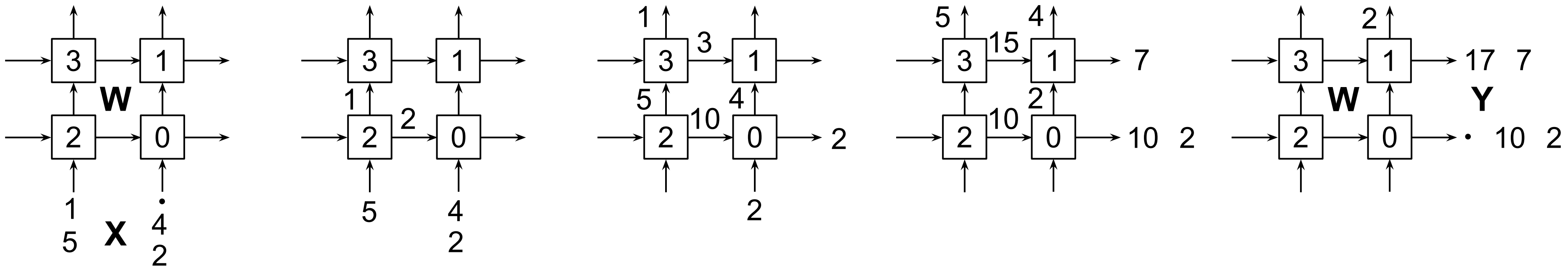
NYU SAI LAB

In-place Transposed Matrix Multiplication

$$\begin{bmatrix} 1 & 4 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 7 \\ 10 & 17 \end{bmatrix}$$

X **W** **Y**

- Weight stationary, input from bottom, accumulation from left to right



- The weights are preloaded into the systolic array, while the input matrix is streamed into the array from bottom to top.
- The output is produced at the right.

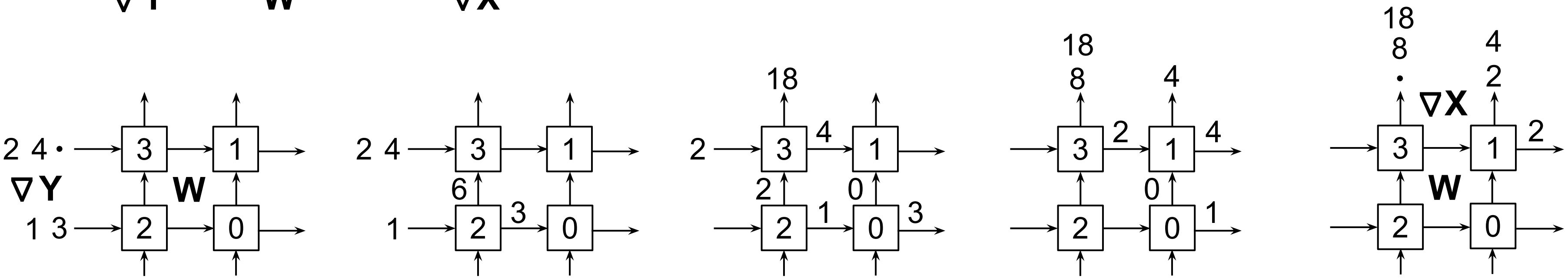
NYU SAI LAB

In-place Transposed Matrix Multiplication

$$\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 18 & 4 \\ 8 & 2 \end{bmatrix}$$

$\nabla Y \quad W^T \quad \nabla X$

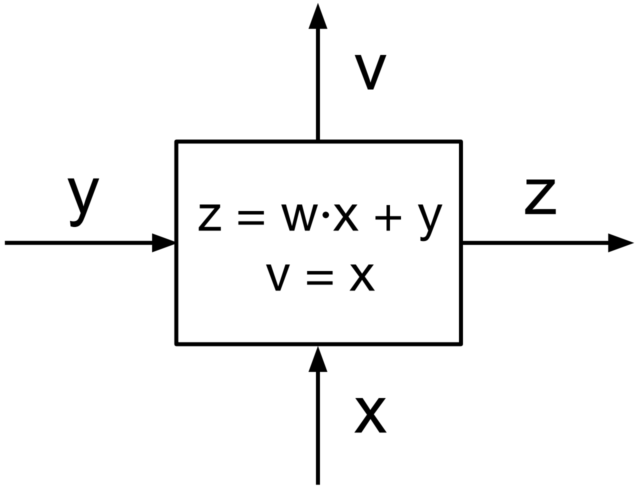
- Weight stationary, input from left, accumulation upwards



- To compute the input gradient, the data gradient is fed into the systolic array from the left, and the output is produced at the top.

NYU SAI LAB

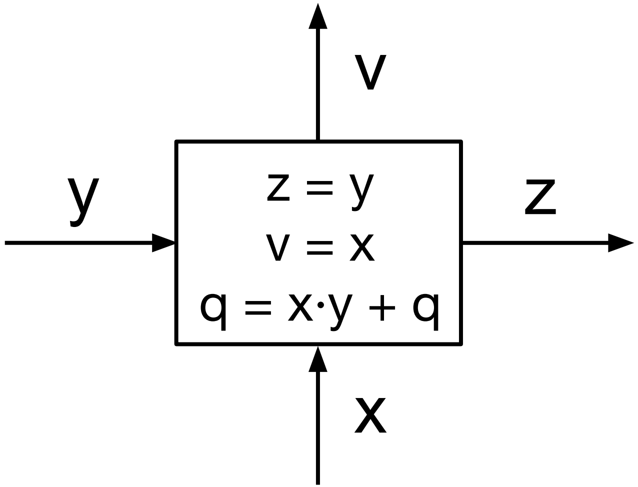
Systolic Array: Weight-Stationary Version



- Takes data (x and y) as input
- w stays in the systolic cell
- Performs a multiply-accumulate operation

NYU SAI LAB

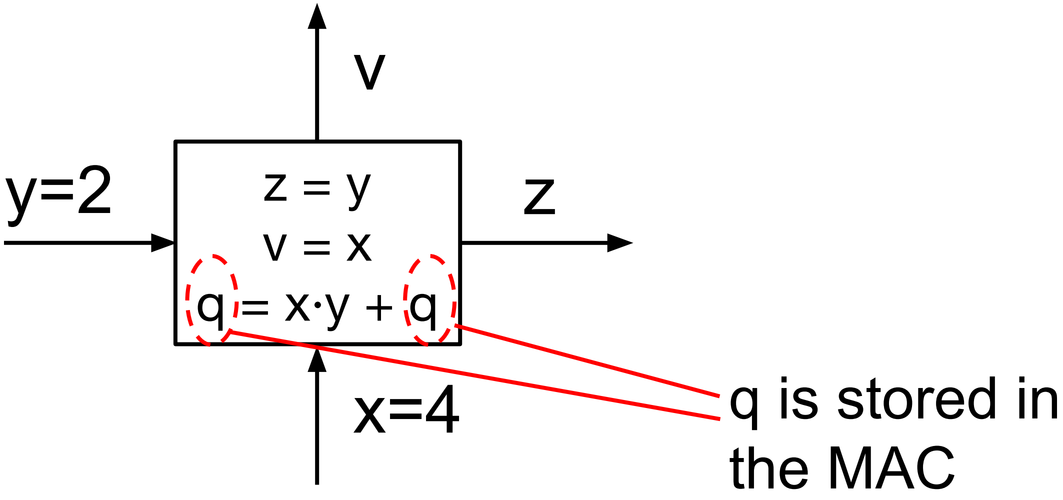
Systolic Array: Accumulation-Stationary Version



- Takes data (x and y) as input
- Accumulated result q stays in the systolic cell
- Performs a multiply-accumulate operation

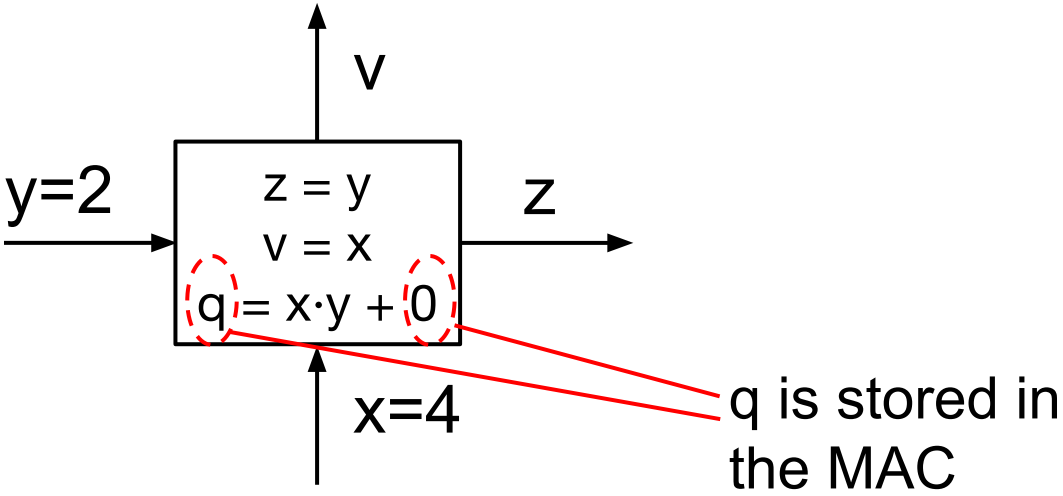
NYU SAI LAB

Systolic Cell



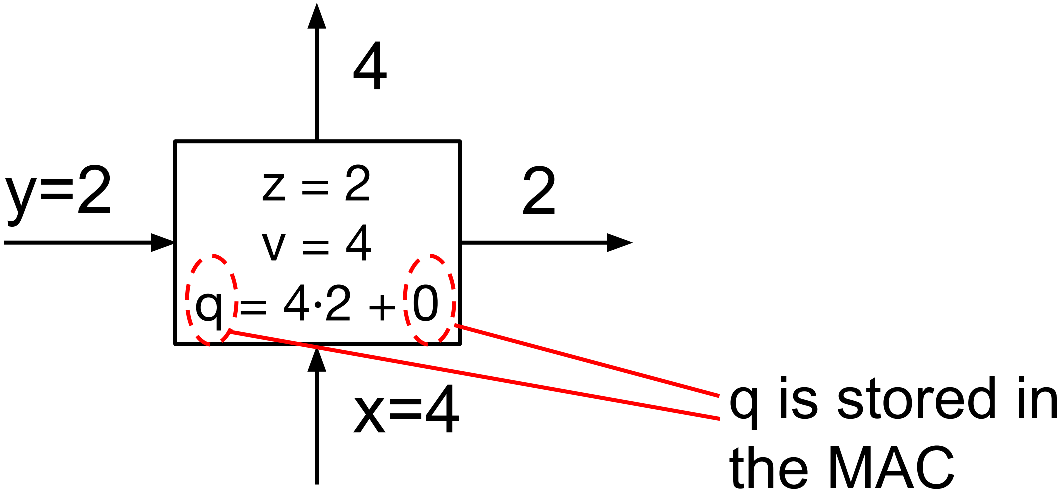
NYU SAI LAB

Systolic Cell

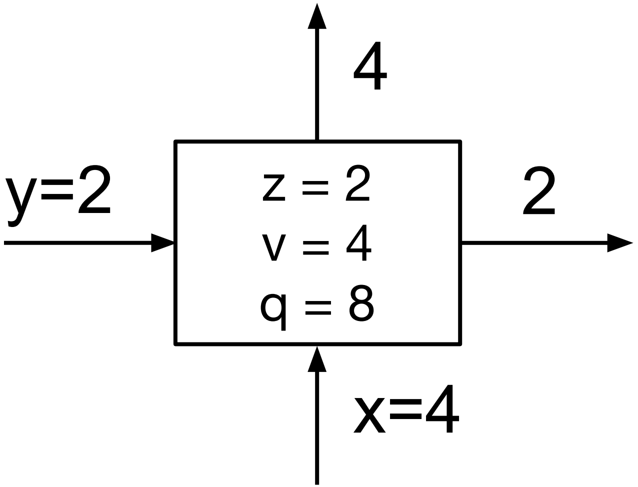


NYU SAI LAB

Systolic Cell



Systolic Cell

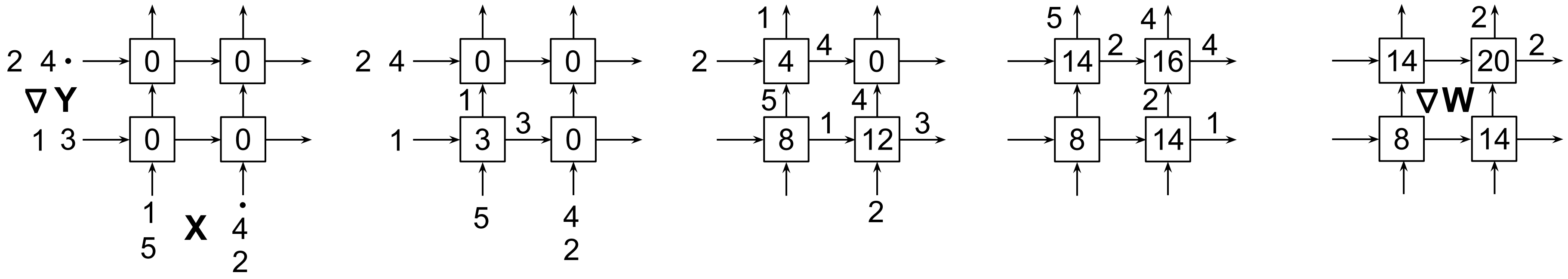


NYU SAI LAB

In-place Transposed Matrix Multiplication

$$\underset{X^T}{\begin{bmatrix} 1 & 5 \\ 4 & 2 \end{bmatrix}} \underset{\nabla Y}{\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}} = \underset{\nabla W}{\begin{bmatrix} 8 & 14 \\ 14 & 20 \end{bmatrix}}$$

- Input from left and bottom, accumulation stationary.



- To compute the weight gradient, the data gradient is input from the left side of the systolic array, while the input activations are fed from the bottom. The resulting weight gradients are accumulated and stored within the systolic cells.