ETH zürich

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

FMA Is (Almost) All You Need: Sharing FMA Resources for Linear and Nonlinear Function Acceleration in Attention-Based Networks

Arpan Suravi Prasad[*], Andrea Belano[‡†], Davide Rossi[‡†], Francesco Conti[‡], Luca Benini[*‡]

[*]*Integrated Systems Laboratory, ETH Zurich;*
[‡]*DEI, University of Bologna;*
[†]*Chips-IT, Italy*

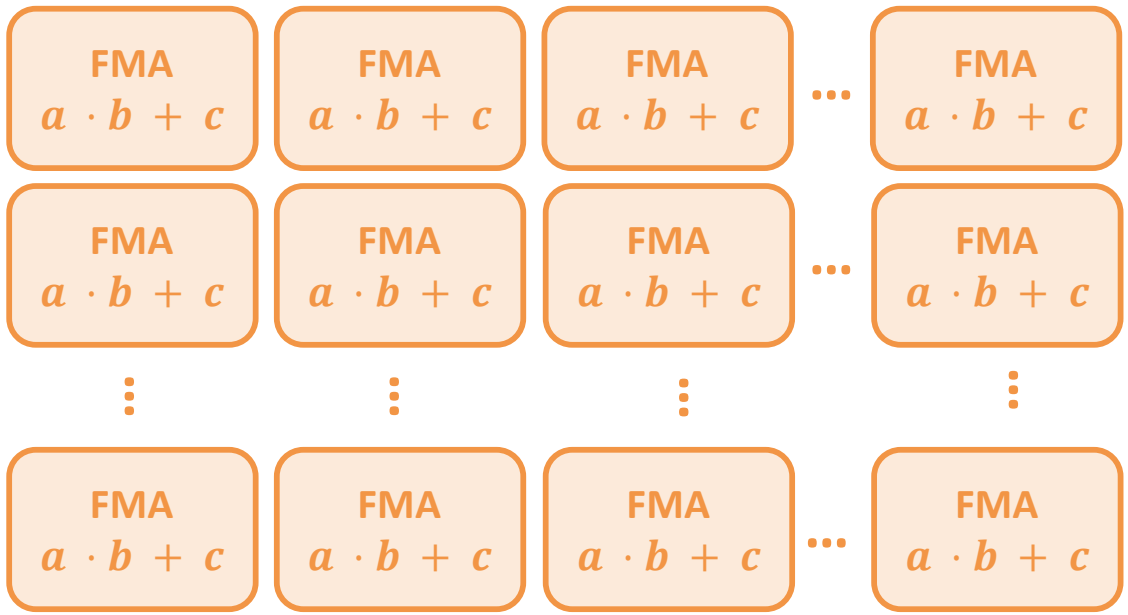
pulp-platform.org
@pulp_platform
company/pulp-platform
youtube.com/pulp_platform

PULP Platform
Open Source Hardware, the way it should be!

Most AI Accelerators are built from arrays of FMAs

2D Array of Processing Elements (PEs)

Each PE executes: $acc \leftarrow a \cdot b + acc$ (fused multiply-add)



Z = W × X + Y, computed by chained FMAs.
Thousands of PEs run in parallel to deliver TOPs throughput

GEMM is highly accelerated by matrix engines

Non-linearities don't fit the FMA model

- exp(x), GeLU, SiLU, 1/√x, 1/x → none reduce to $a \cdot b + c$
- Special dedicated units → high area overhead and synchronization overhead

Can the abundant FMA hardware be ***repurposed*** to implement non-linearities with **low area overhead**?

Classes of Nonlinear Operations

Nonlinear Activations

- Elementwise operation

$$Y(x) = Nonlinear(x)$$

- SiLU GELU etc.

$$\mathrm{SiLU}(x) = \frac{x}{1 + e^{-x}}$$

- Low arithmetic intensity → best used as post-processing to optimize memory access.

Softmax

- Commonly used in attention
- Non-elementwise operation

$$\mathrm{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$$

- Includes exp division

Layer Normalization

- Non-elementwise operation

$$\mathrm{Layernorm}(x_i) = \frac{x_i - \mu}{\sqrt{(\sigma^2 + \epsilon)}}$$

- Includes online statistics computation mean, variance
- Also includes division $\sqrt{}$

Approximate the nonlinear function using **Piecewise Polynomial Approximation (PWPA)**

Pointwise Functions: SiLU, GELU, exp

Nonlinear Activations

- Elementwise operation

$$Y(x) = Nonlinear(x)$$

- SiLU GELU etc.

$$\mathrm{SiLU}(x) = \frac{x}{1 + e^{-x}}$$

- Low arithmetic intensity → best used as post-processing to optimize memory access.

Softmax

- Commonly used in attention
- Non-elementwise operation

$$\mathrm{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$$

- Includes exp division

Layer Normalization

- Non-elementwise operation

$$\mathrm{Layernorm}(x_i) = \frac{x_i - \mu}{\sqrt{(\sigma^2 + \epsilon)}}$$

- Includes online statistics computation mean, variance
- Also includes division $\sqrt{}$

Approximate the nonlinear function using **Piecewise Polynomial Approximation (PWPA)**

Approximate $1/b$ and $1/\sqrt{(b)}$ with **domain-reduced PWPA**, then compute a/b as $a \cdot PWPA(b)$

Non-Elementwise: Softmax & Layer Norm

Nonlinear Activations

- Elementwise operation

$$Y(x) = Nonlinear(x)$$

- SiLU GELU etc.

$$\mathrm{SiLU}(x) = \frac{x}{1 + e^{-x}}$$

- Low arithmetic intensity → best used as post-processing to optimize memory access.

Softmax

- Commonly used in attention
- Non-elementwise operation

$$\mathrm{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$$

- Includes exp division

Layer Normalization

- Non-elementwise operation

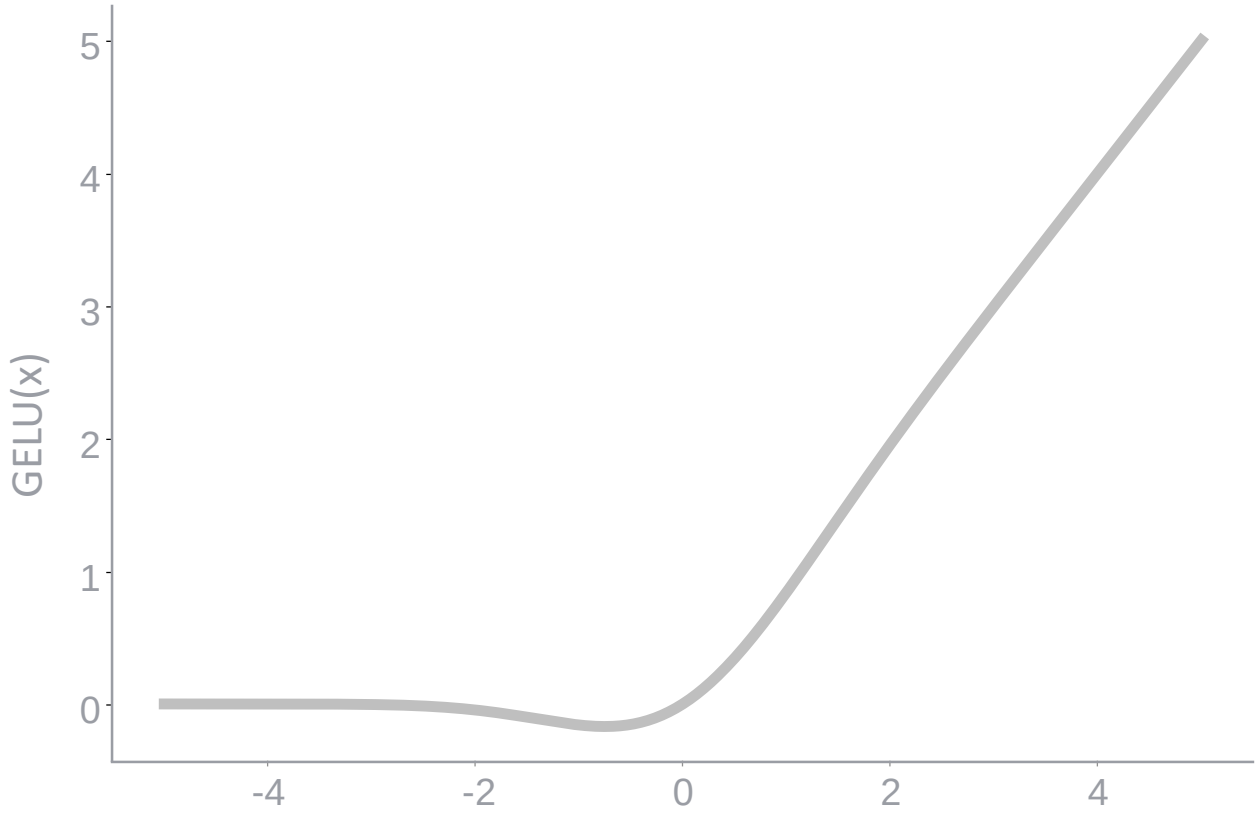
$$\mathrm{Layernorm}(x_i) = \frac{x_i - \mu}{\sqrt{(\sigma^2 + \epsilon)}}$$

- Includes online statistics computation mean, variance
- Also includes division $\sqrt{}$

Approximate the nonlinear function using **Piecewise Polynomial Approximation (PWPA)**

Approximate $1/b$ and $1/\sqrt{(b)}$ with **domain-reduced PWPA**, then compute a/b as $a \cdot PWPA(b)$

Piecewise Polynomial Approximation



LEGEND

— Ideal GELU

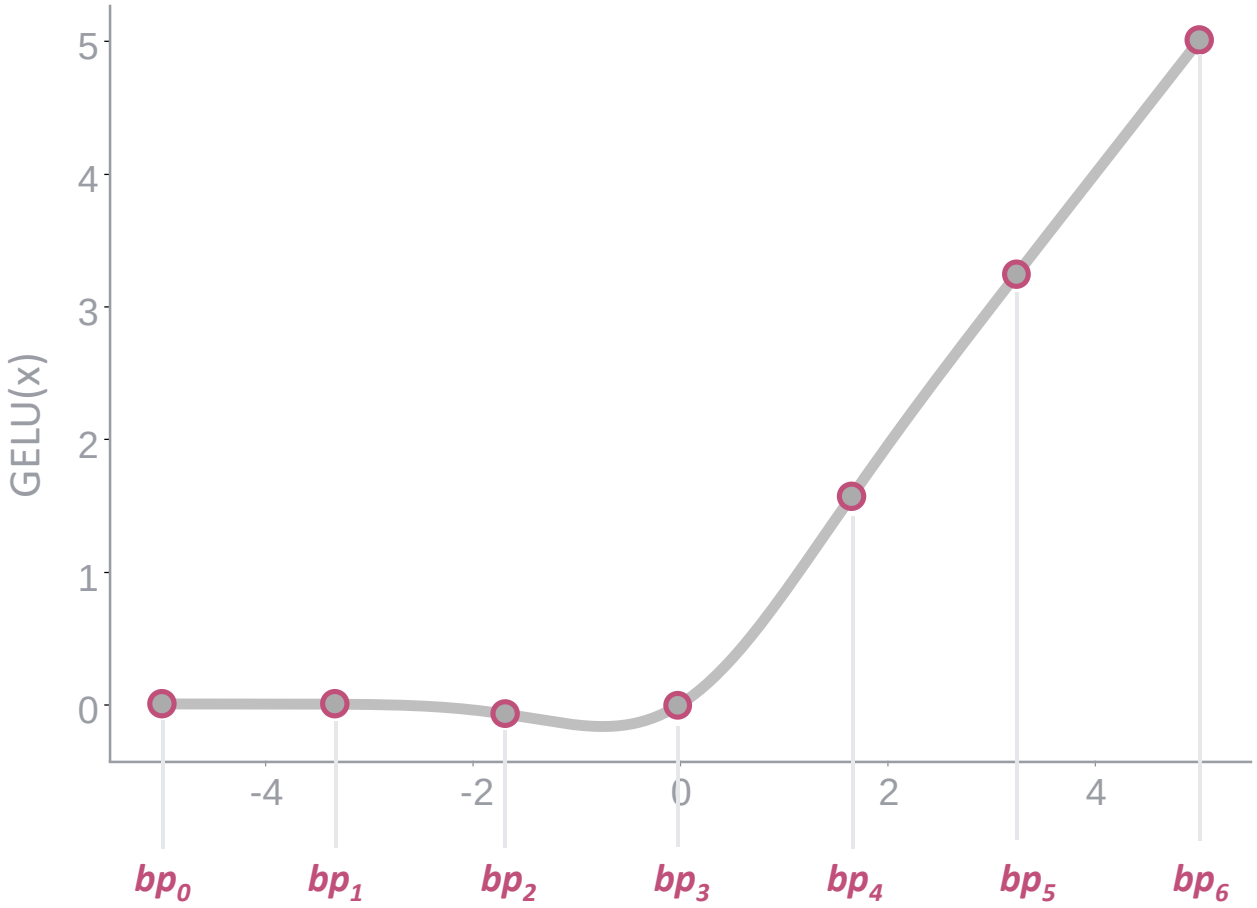
EXAMPLE (D=3, P=8)

Degree 3, Partitions 8

6 piecewise cubic segments

2 asymptotes (y = 0, y = x)

PWPA: Uniform Breakpoint Placement



LEGEND

- Ideal GELU
- Uniform breakpoints

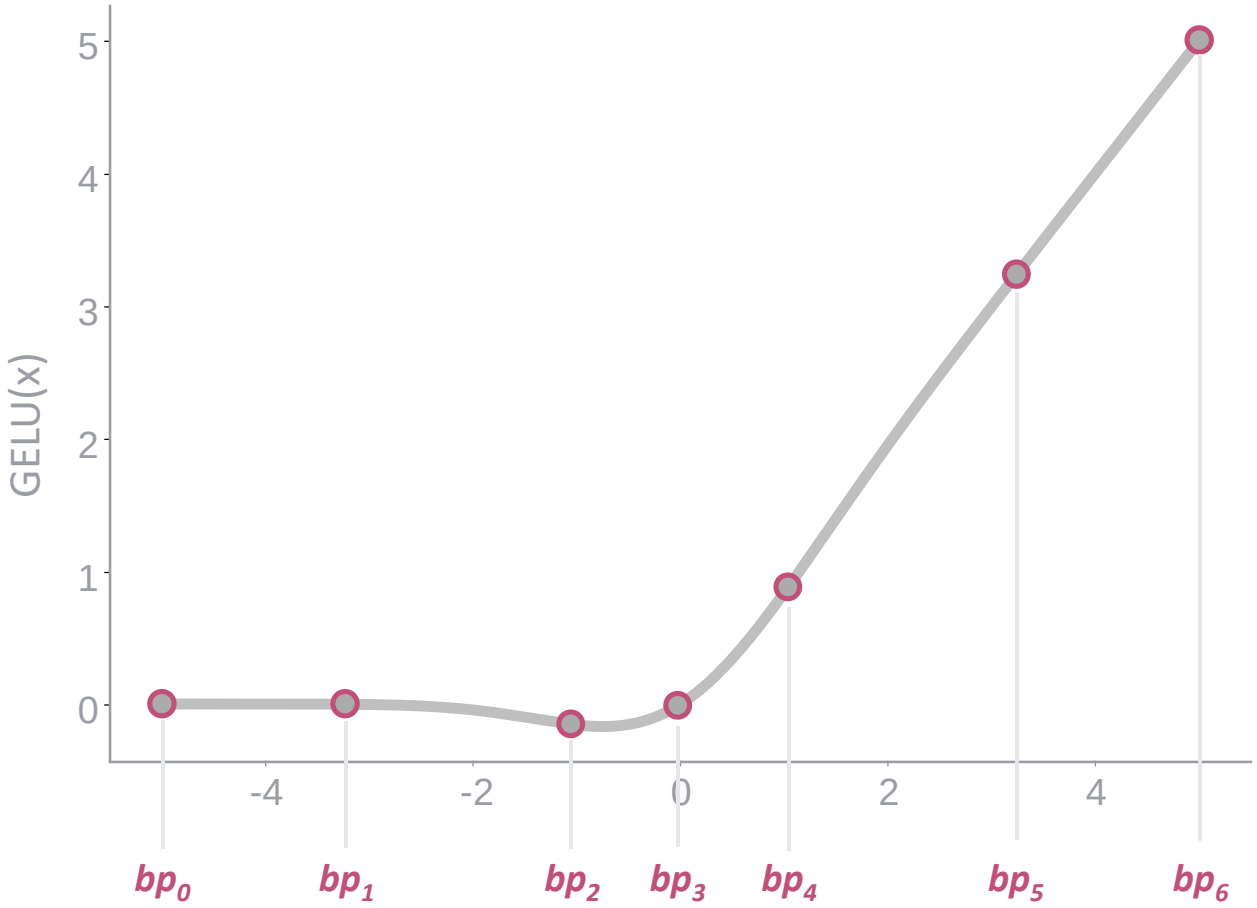
EXAMPLE (D=3, P=8)

Degree 3, Partitions 8

6 piecewise cubic segments

2 asymptotes (y = 0, y = x)

PWPA: Optimal Breakpoint Placement



LEGEND

- Ideal GELU
- Optimal breakpoints

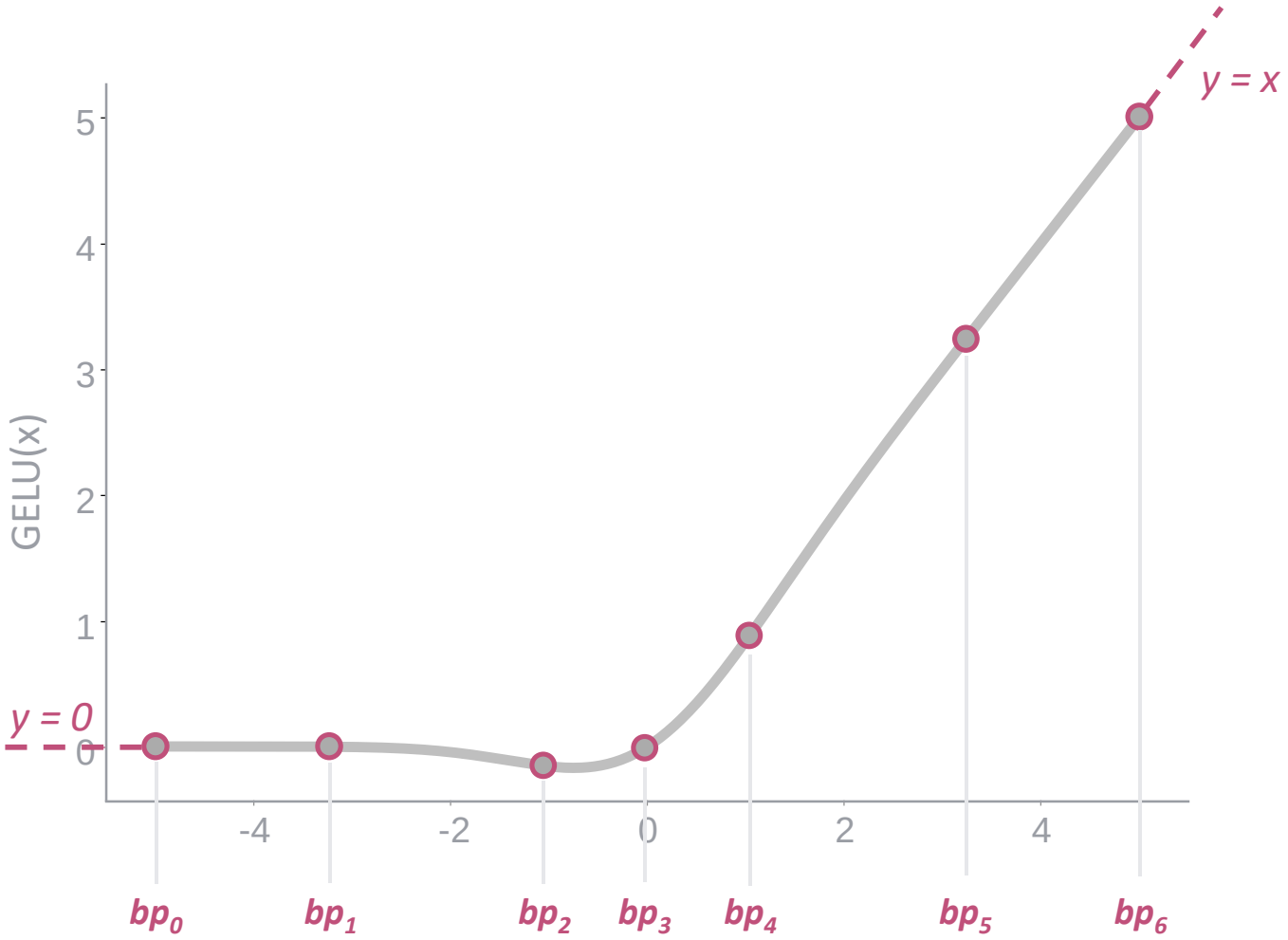
EXAMPLE (D=3, P=8)

Degree 3, Partitions 8

6 piecewise cubic segments

2 asymptotes (y = 0, y = x)

PWPA: Asymptote Handling



LEGEND

- Ideal GELU
- Optimal breakpoints
- Asymptote

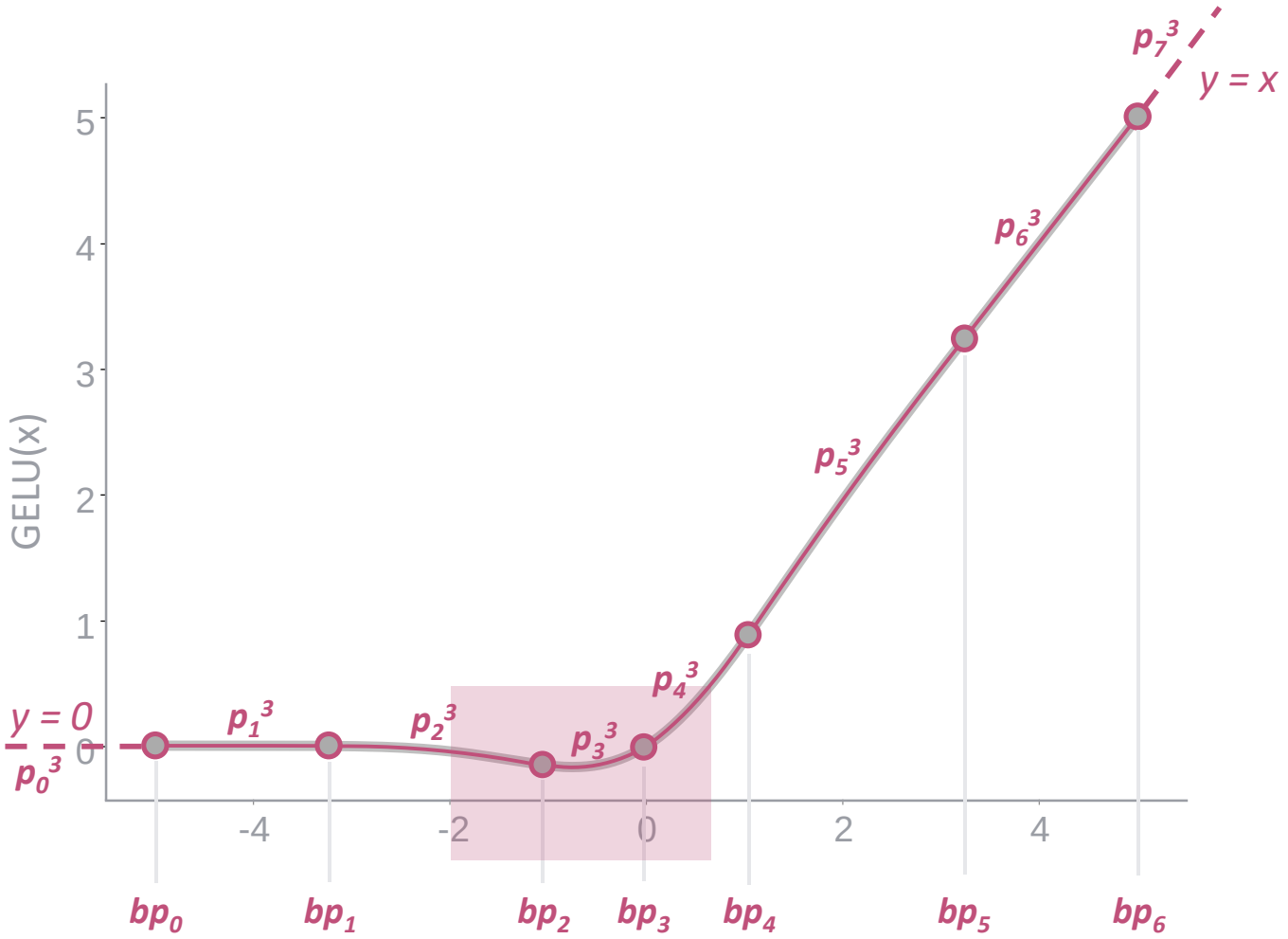
EXAMPLE (D=3, P=8)

Degree 3, Partitions 8

6 piecewise cubic segments

2 asymptotes (y = 0, y = x)

PWPA: Full Polynomial Coefficient Set



LEGEND

- Ideal GELU
- Optimal breakpoints
- Asymptote
- PACE approx.
- p_i^3 Polynomial segment

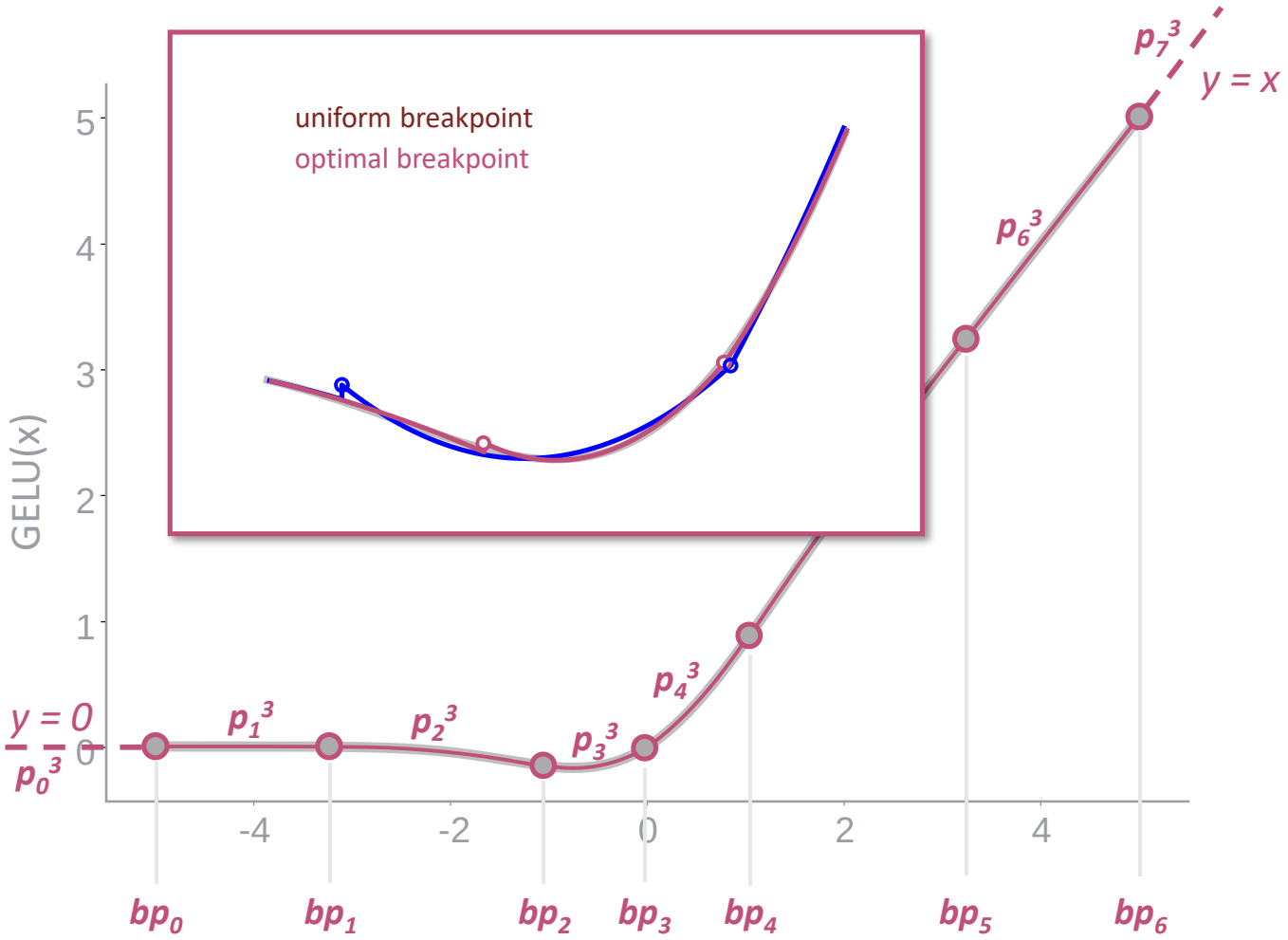
EXAMPLE (D=3, P=8)

Degree 3, Partitions 8

6 piecewise cubic segments

2 asymptotes (y = 0, y = x)

PWPA: Full Polynomial Coefficient Set



LEGEND

- Ideal GELU
- Optimal breakpoints
- Asymptote
- PACE approx.
- p_i^3 Polynomial segment

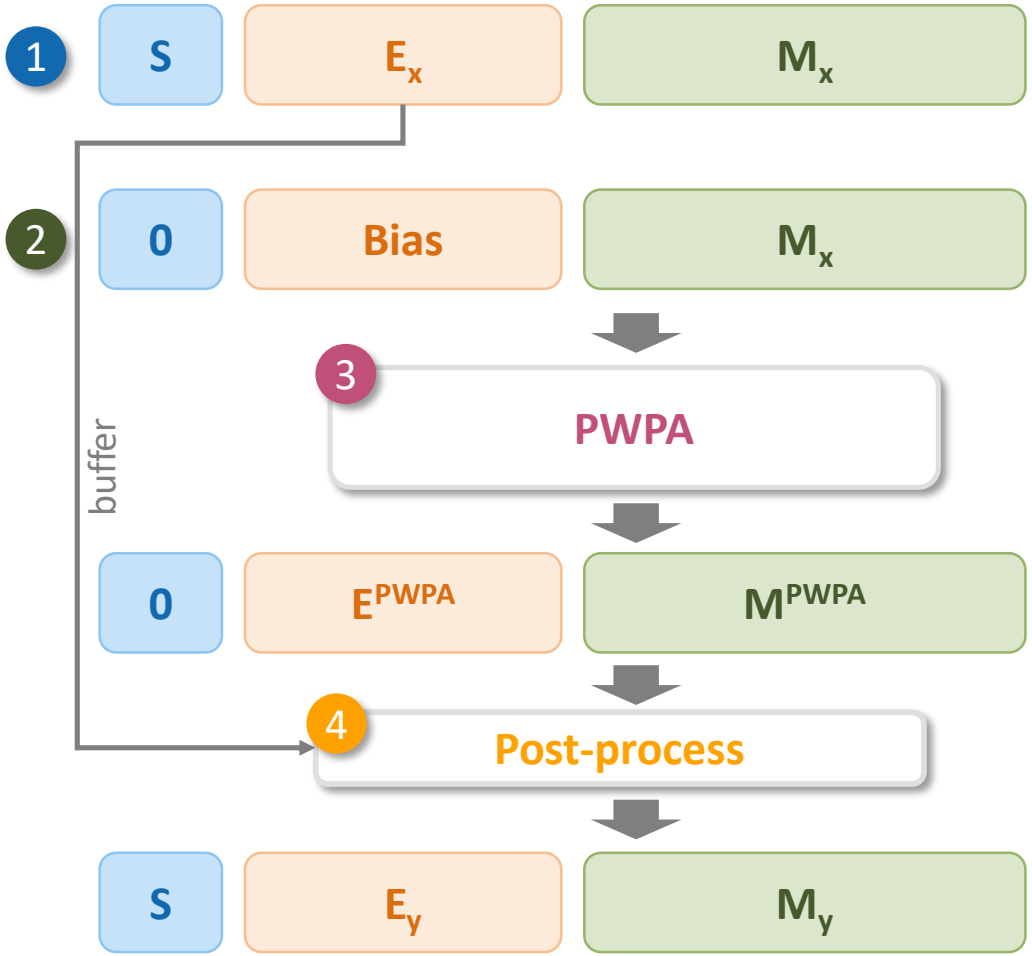
EXAMPLE (D=3, P=8)

Degree 3, Partitions 8

6 piecewise cubic segments

2 asymptotes (y = 0, y = x)

Extending PWPA for Reciprocal



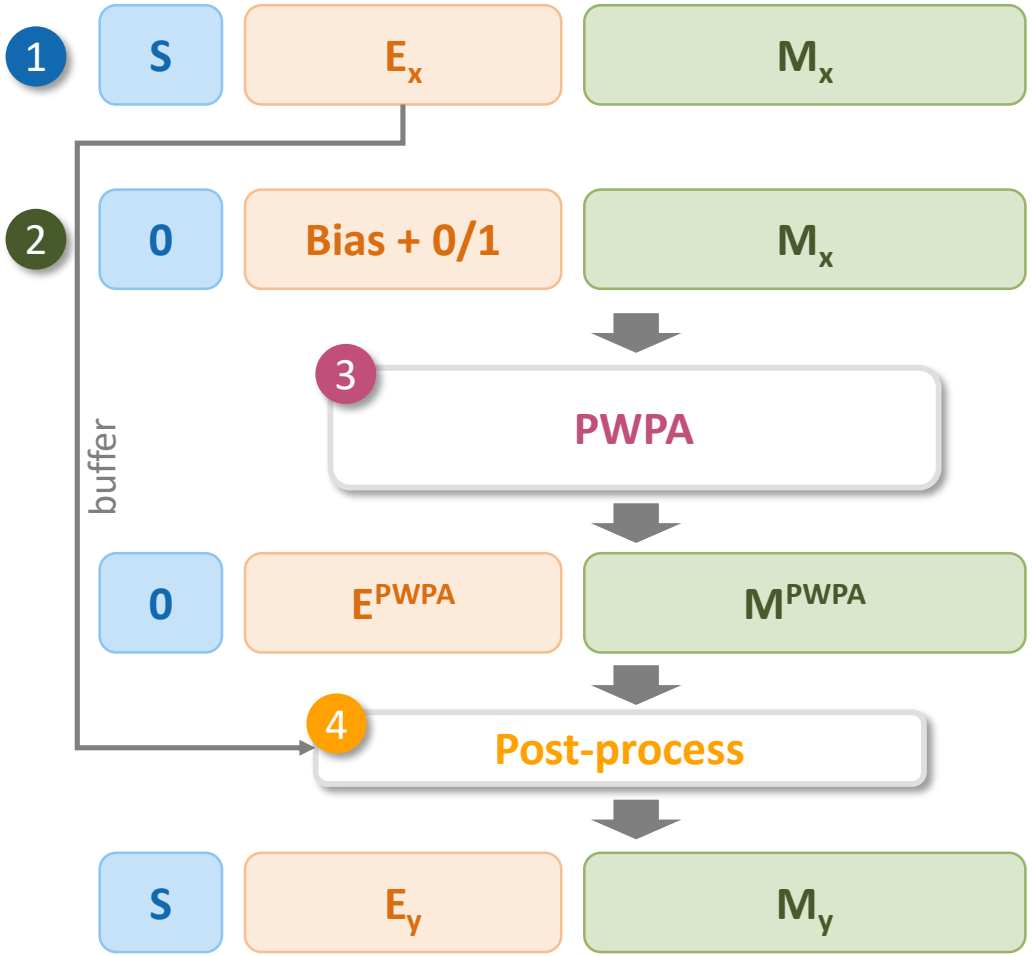
1. **Decompose Input:**
   Given a floating-point number **x**, decompose to $\mathbf{M_x}$ and $\mathbf{E_x}$

2. **Domain Reduction → [1, 2):**
   construct **1.$\mathbf{M_x}$** by assigning the exponent to bias and keeping the mantissa as it is (assumption normal input). Reduces the approximation interval, improving accuracy.

3. **Inverse Approximation (PWPA)**
   Approximate, on the reduced range that is $y = \frac{1}{x}, \; x \in [1, 2)$, through piecewise polynomial approximation

4. **Exponent Correction:**
   Adapt the exponent using the input exponent
   $E_y = E^{\mathrm{PWPA}} - E_x - bias$

Extending PWPA for Inverse Square Root



1 **Decompose Input:**
Given a floating-point number **x**, decompose to $\mathbf{M_x}$ and $\mathbf{E_x}$

2 **Domain Reduction → [1, 4):**
Copy the mantissa unchanged and adjust only the exponent to make it even, assuming a normalised input.

3 **Inverse Approximation (PWPA)**
Approximate, on the reduced range that is $y = \frac{1}{\sqrt{x}}, \; x \in [1, 4)$, through piecewise polynomial approximation

4 **Exponent Correction:**
Adapt the exponent using the input exponent
$E_y = E^{\mathrm{PWPA}} - ((E_x - bias) >> 1)$

Accuracy Validation on ViT-B & ViT-L

Evaluated on ViT-B and ViT-L (cast to FP16)

Seq len	layers	Embed dim
197	12/24	768/1024

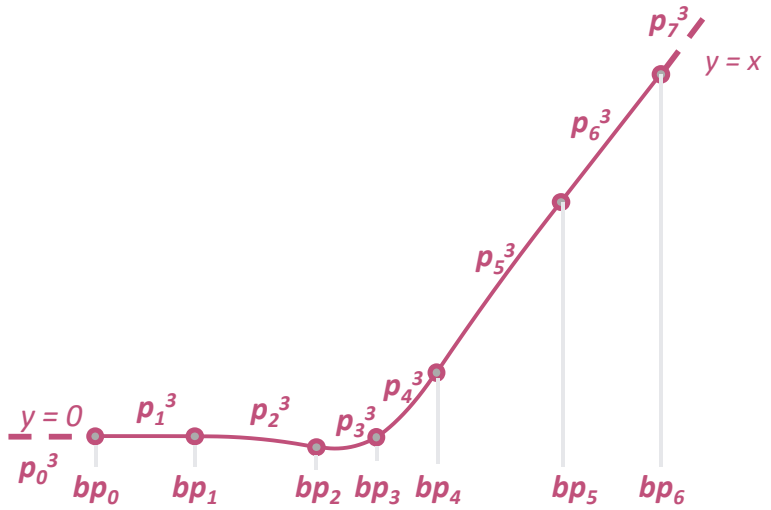
Classification on Imagenet validation set (Top-1)

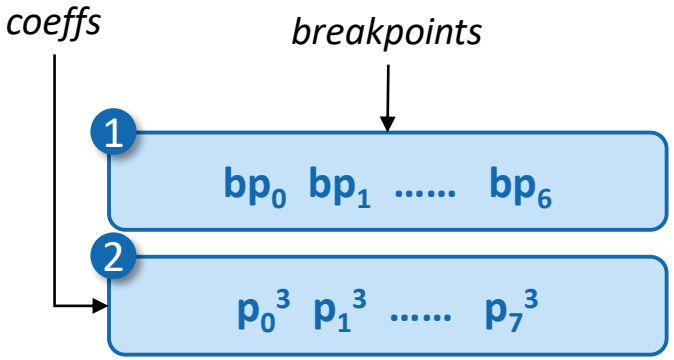
Nonlinearities replaced are GeLU, softmax, and Layer Normalisation at FP16 precision

Baseline	81.33 / 82.52	
Softmax (e^x)	81.33 / 82.53	99.89 / 99.85
Softmax (e^x & 1/x)	81.33 / 82.54	99.87 / 99.86
GELU	81.34 / 82.55	99.89 / 99.87
LayerNorm (1/√x)	81.31 / 82.52	99.94 / 99.94
All nonlinearities	**81.35 / 82.55**	99.86 / 99.82

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

Hardware Design: Programmable Parameters





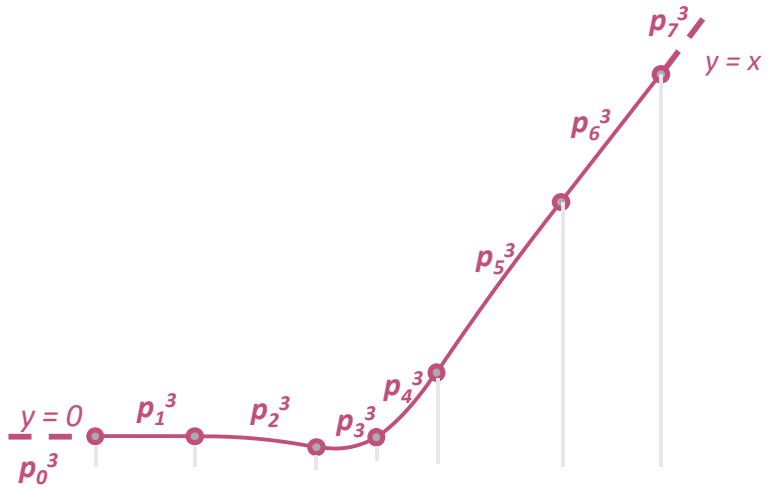
1 Programmable Bps & Coeffs

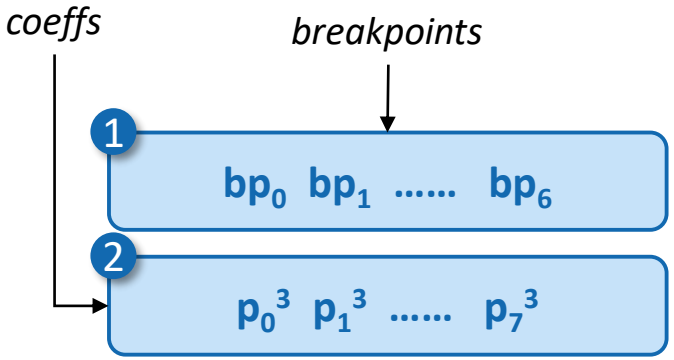
$bp_0 \ldots bp_7$ — segment breakpoints

$p_0^3 \ldots p_7^3$ — polynomial coefficients

Bps and coeffs loaded once per function

Hardware Design: Loading breakpoints





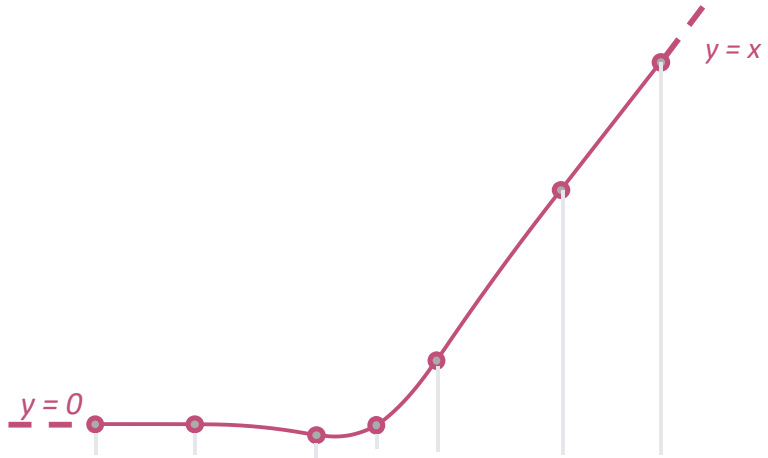
1 Programmable Bps & Coeffs

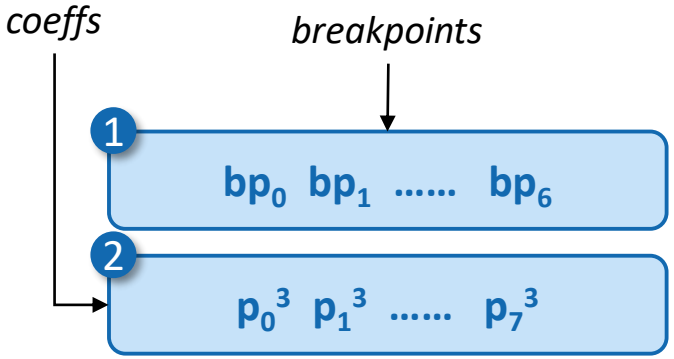
bp_0…bp_7 — segment breakpoints

p_0^3…p_7^3 — polynomial coefficients

Bps and coeffs loaded once per function

Hardware Design: Loading coefficients





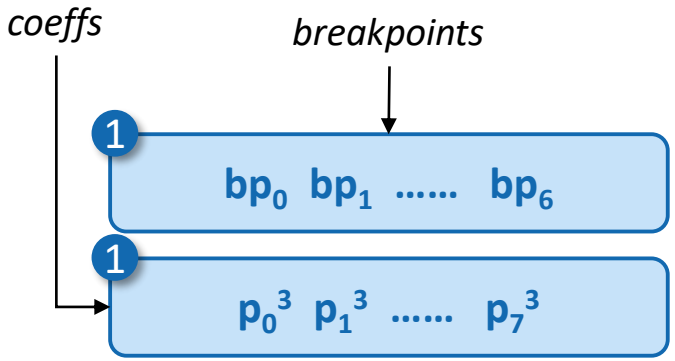
1 Programmable Bps & Coeffs

$bp_0 \ldots bp_7$ — segment breakpoints

$p_0^3 \ldots p_7^3$ — polynomial coefficients

Bps and coeffs loaded once per function

Hardware Design: Coefficient Storage



1

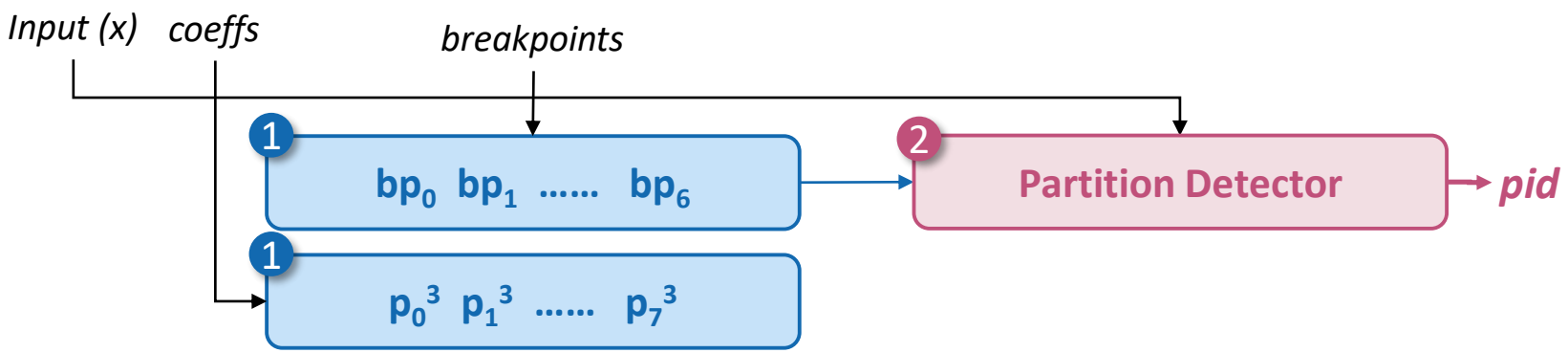
Programmable Bps & Coeffs

$bp_0 \ldots bp_7$ — segment breakpoints

$p_0^3 \ldots p_7^3$ — polynomial coefficients

Bps and coeffs loaded once per function

Hardware Design: Full 3-Stage Pipeline



1

Bps & coeffs loaded once per function

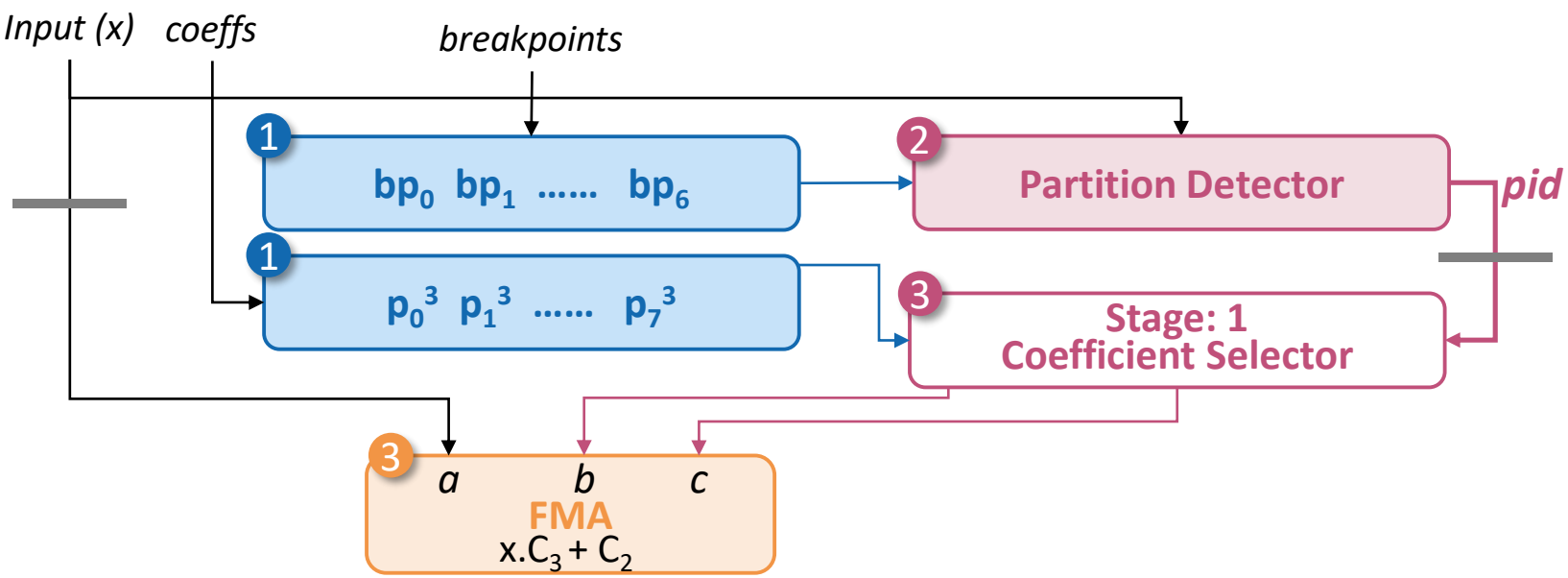
2

Partition Detector

Binary tree of **$\log_2(P)$ comparators**

identifies which segment x falls into.

Hardware Design: Full 3-Stage Pipeline



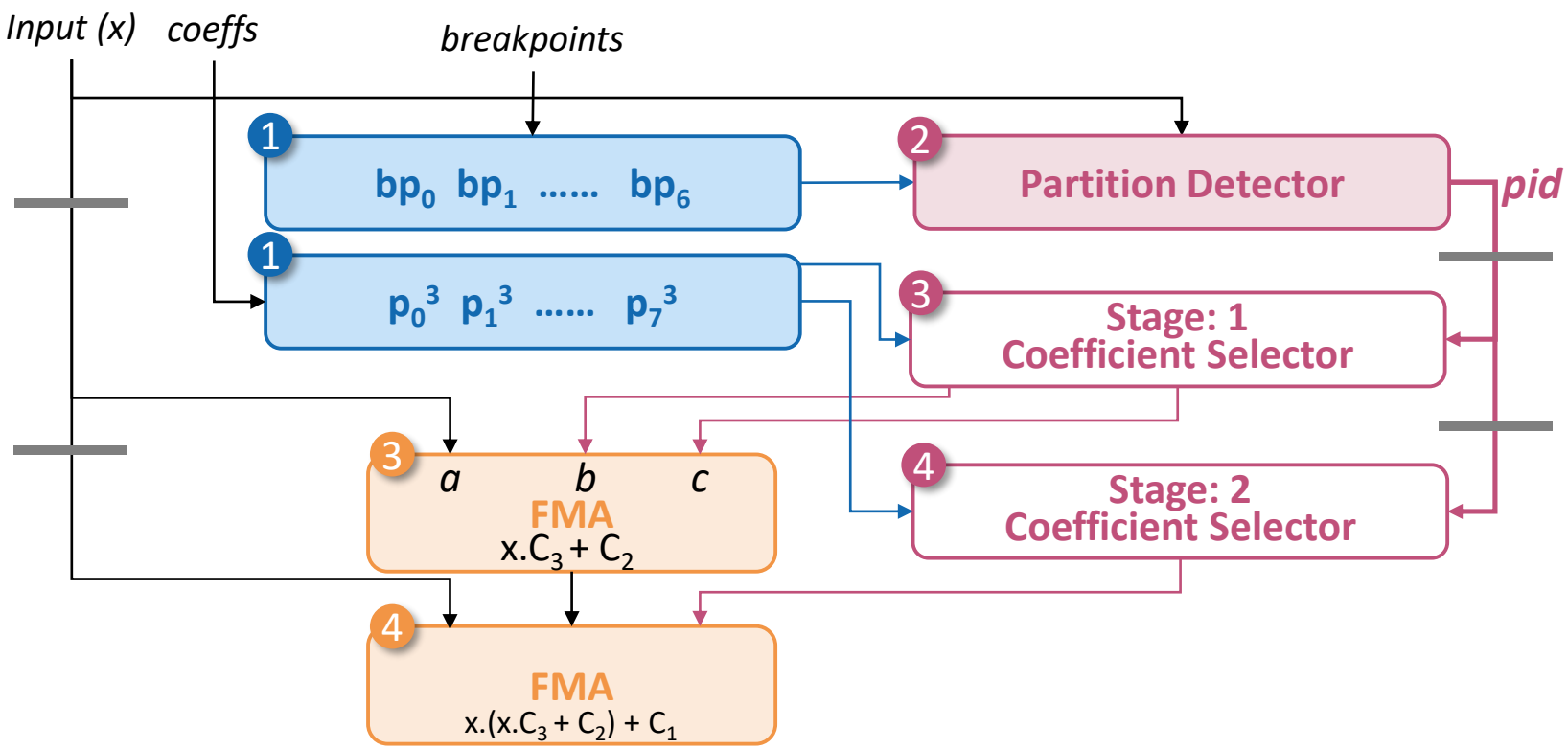
1. **Bps & coeffs loaded once per function**

2. **identifies which segment x falls into**

3. **Computation (Degree 1)**

MUX selects **2** × **P** coefficient entries for the identified segment. FMA computes degree 1 polynomial.

Hardware Design: Full 3-Stage Pipeline



1 **Bps & coeffs loaded once per function**

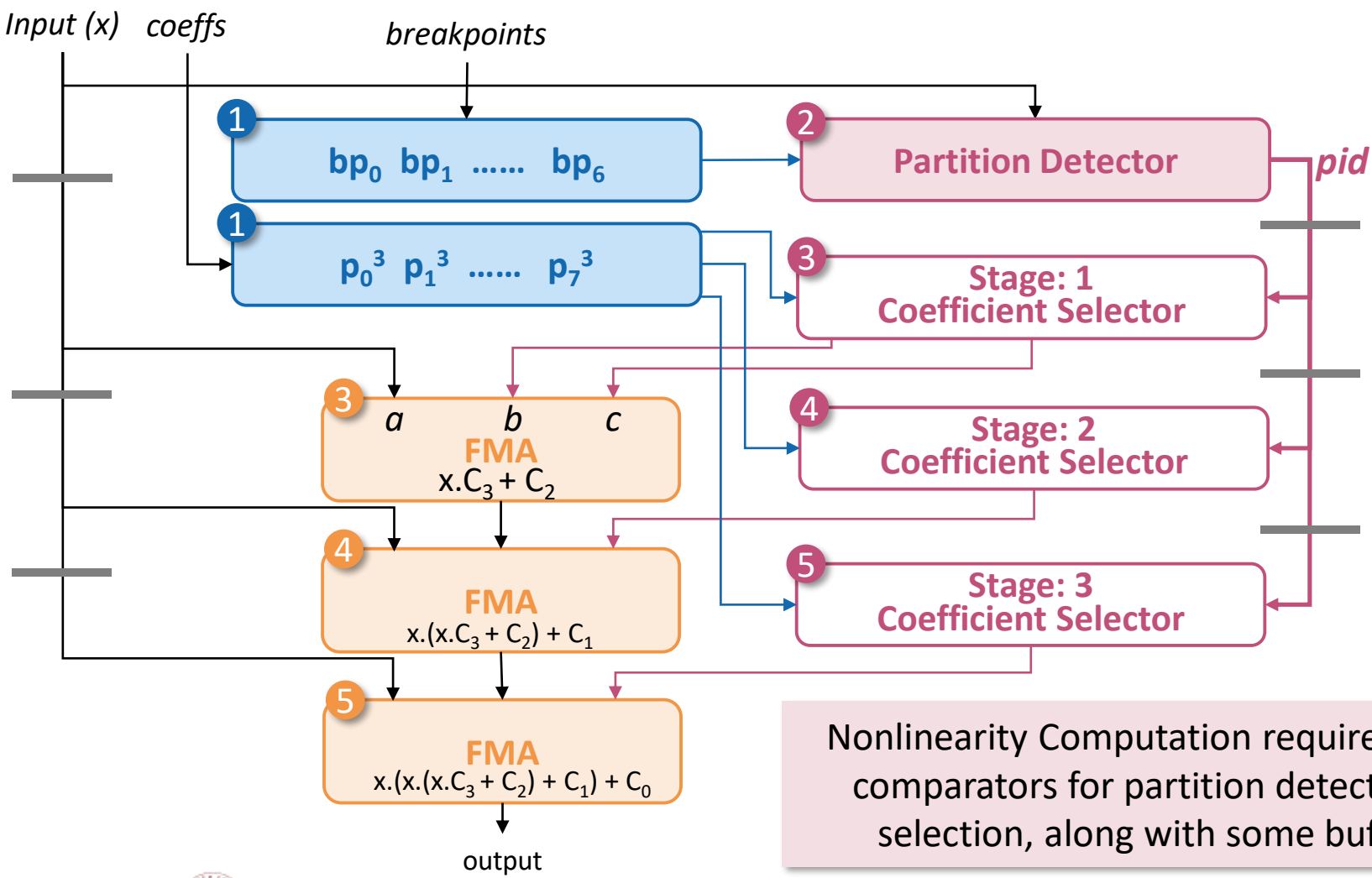
2 **identifies which segment x falls into**

3 **Computation (Degree 1)**

4 **Horner's Method (Degree 2)**

A **1×P MUX** selects coefficients using the stored **pid**; the Stage-1 FMA result and stored **x** feed the final FMA.

Hardware Design: Full 3-Stage Pipeline



1. Bps & coeffs loaded once per function
2. identifies which segment x falls into
3. Computation (Degree 1)
4. Horner's Method (Degree 2)
5. Final Output (Degree 3)

Nonlinearity Computation requires **FMAs and lightweight logic** comparators for partition detection and MUX for coefficient selection, along with some buffers for data orchestration

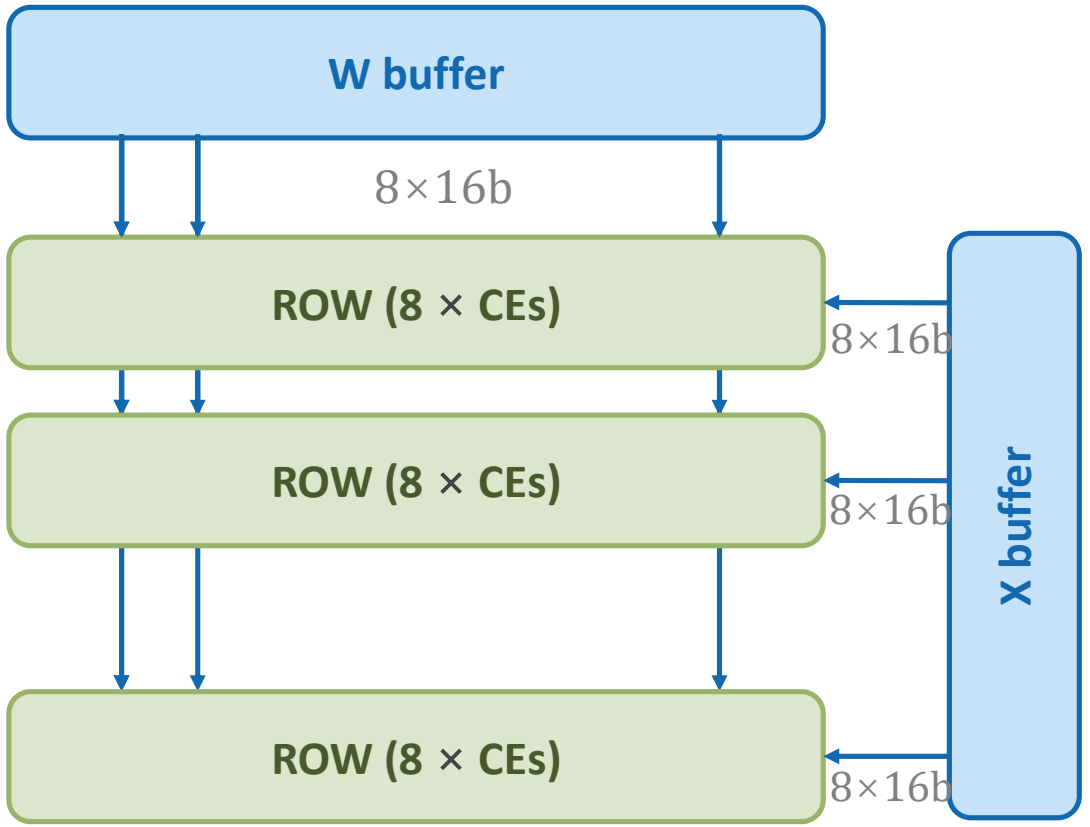
RedMule GEMM Accelerator Architecture

Open-source HWPE catering to GEMM acceleration

We use an 8 × 8 grid, with FP16 precision

Weight is streamed and broadcast across rows

Input X is unique to each CE and stays stationary



RedMule: Adding the Input and Output Buffer

Open-source HWPE catering to GEMM acceleration

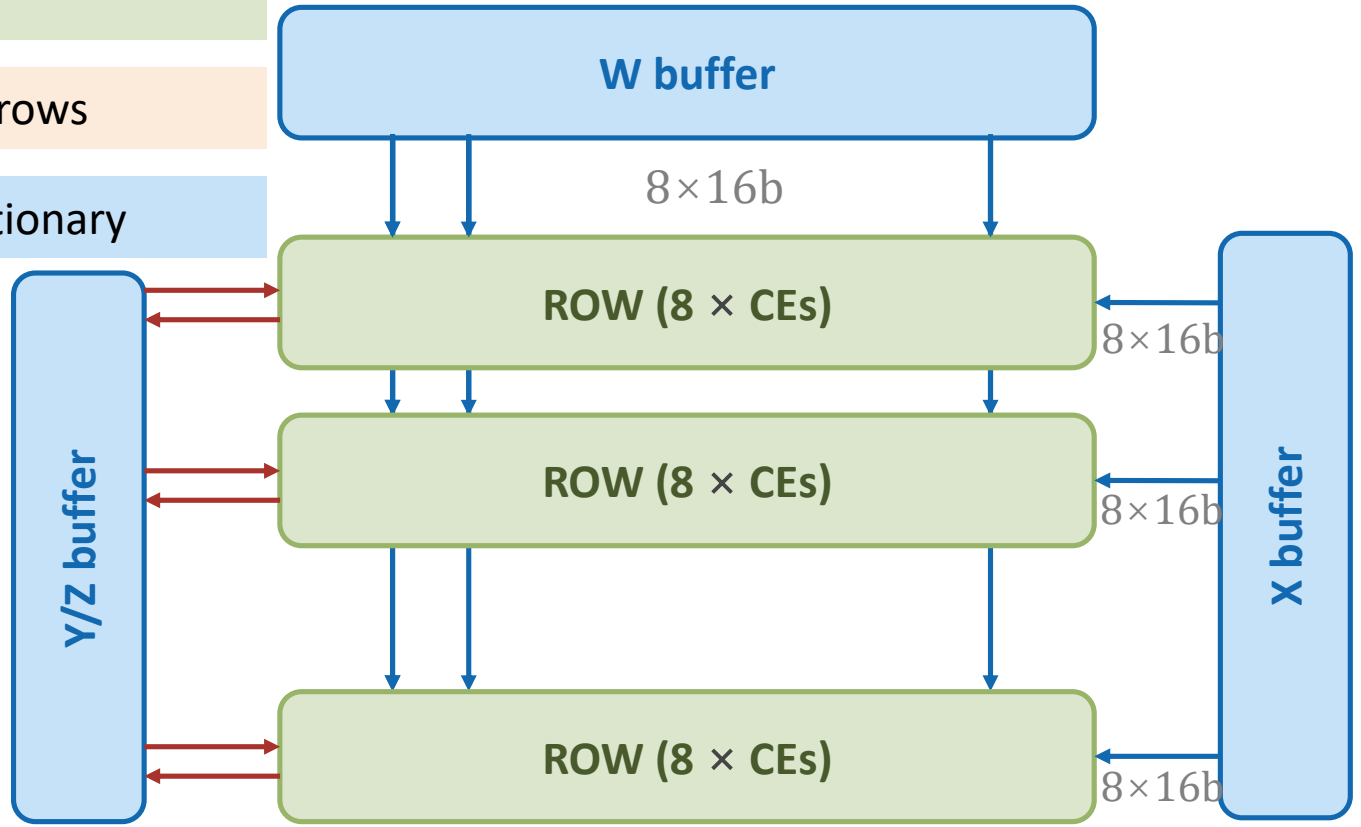
We use an 8 × 8 grid, with FP16 precision

Weight is streamed and broadcast across rows

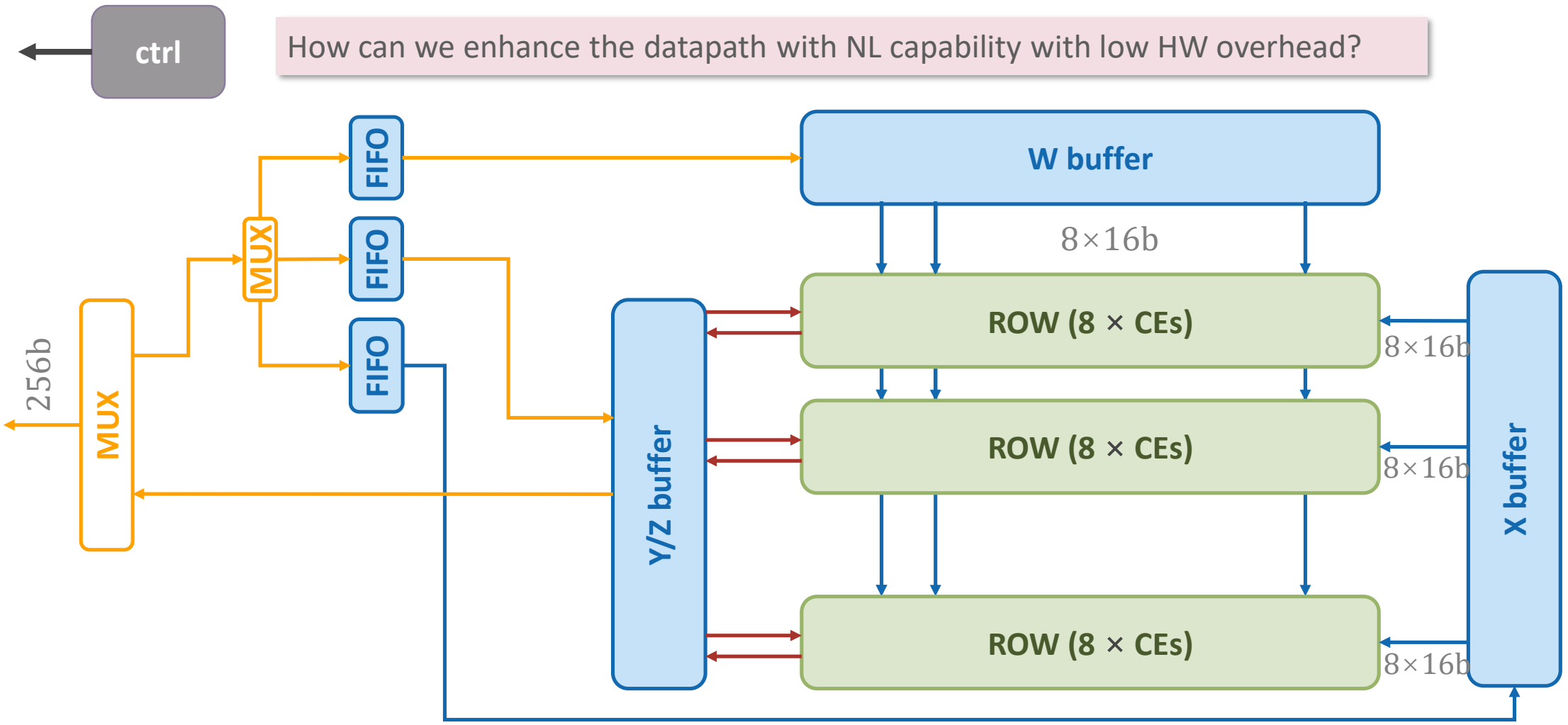
Input X is unique to each CE and stays stationary

16b partial sum is fed to each row

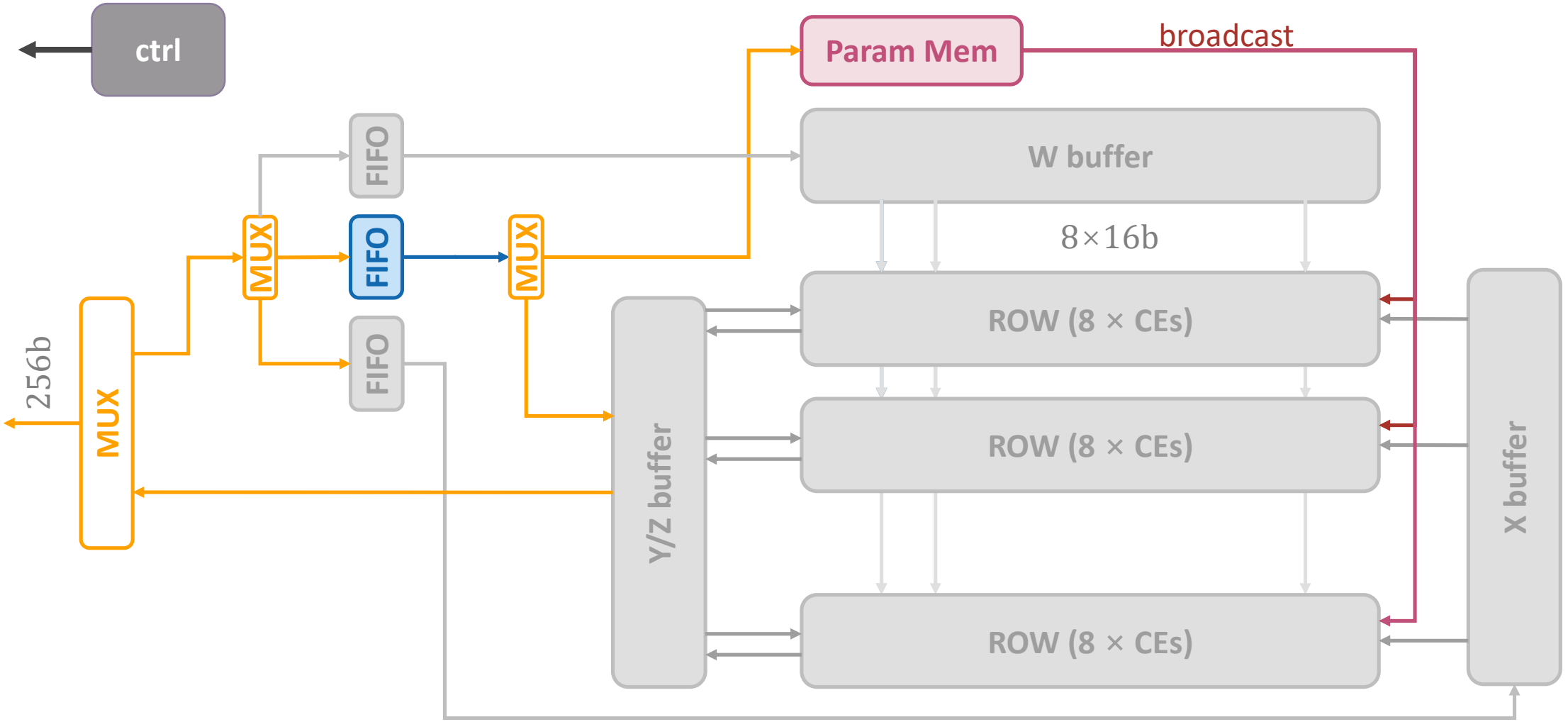
16b output is written to Y / Z buffer



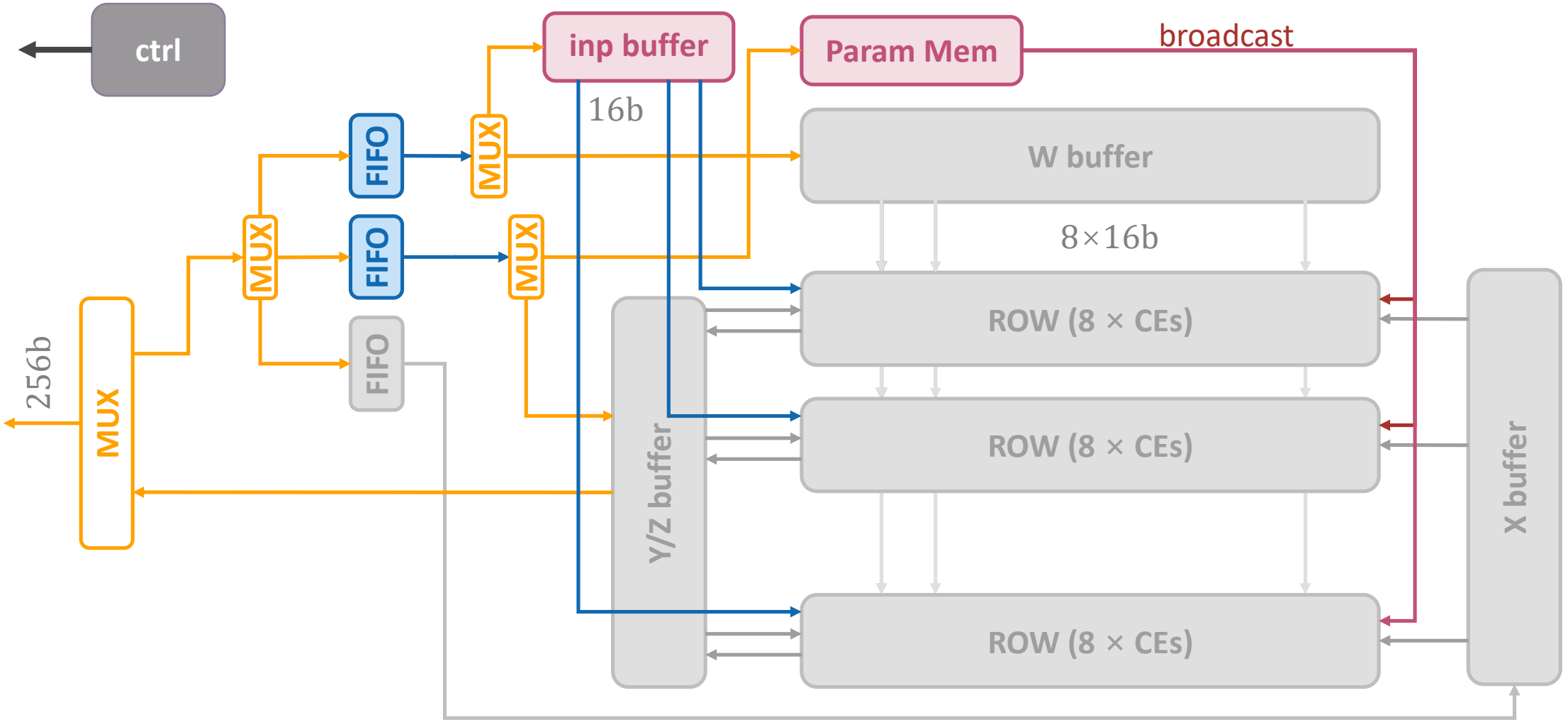
Enhancing RedMule for Nonlinearity



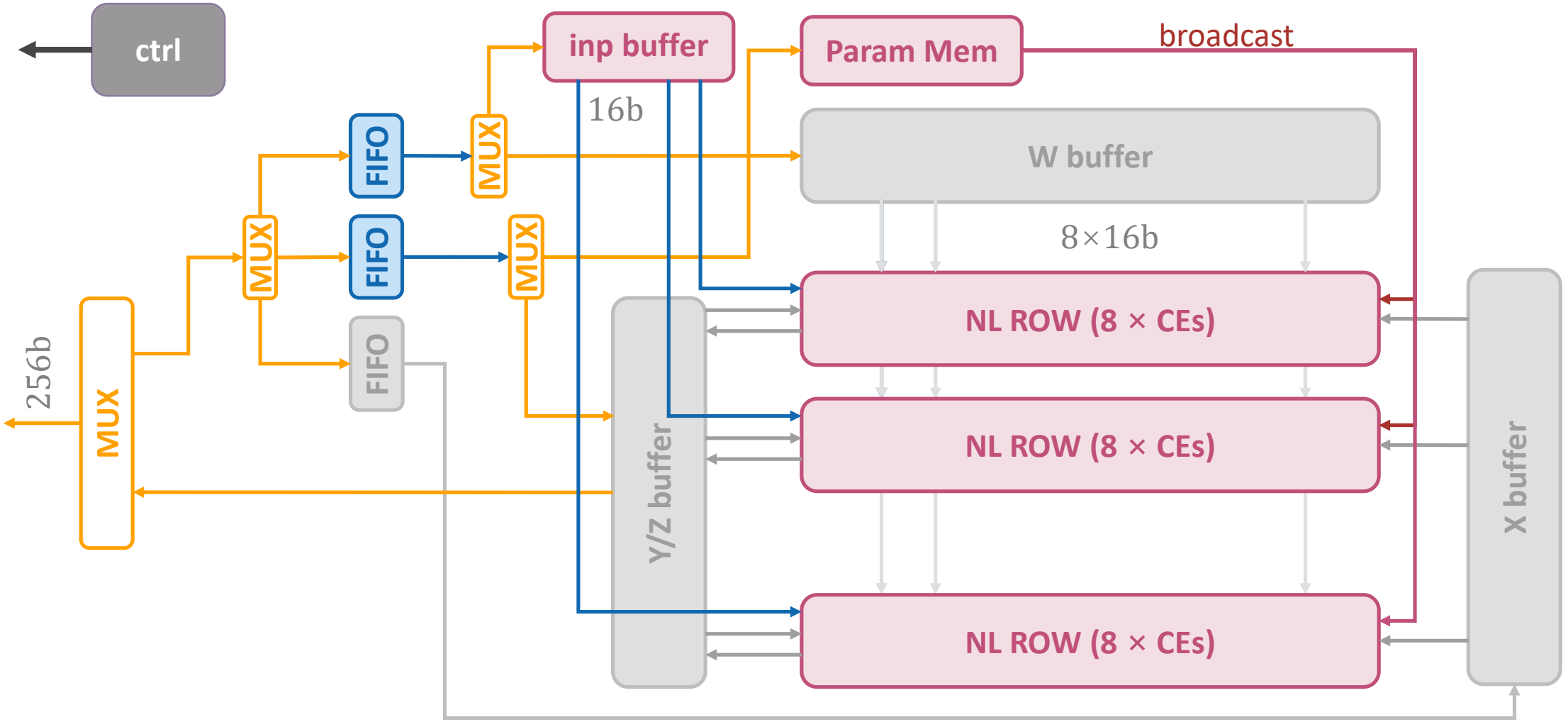
RedMule: Parameter Memory



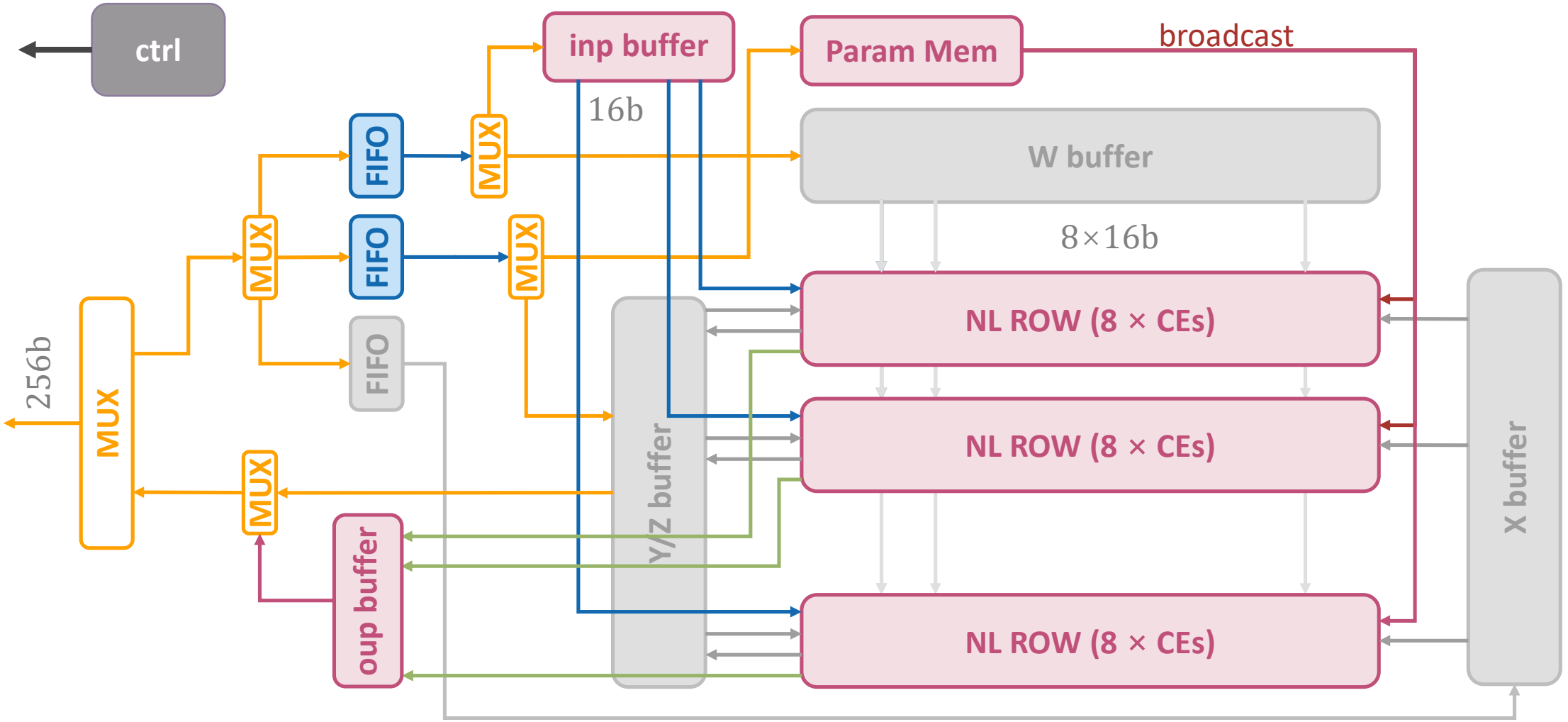
RedMule: Input Buffer



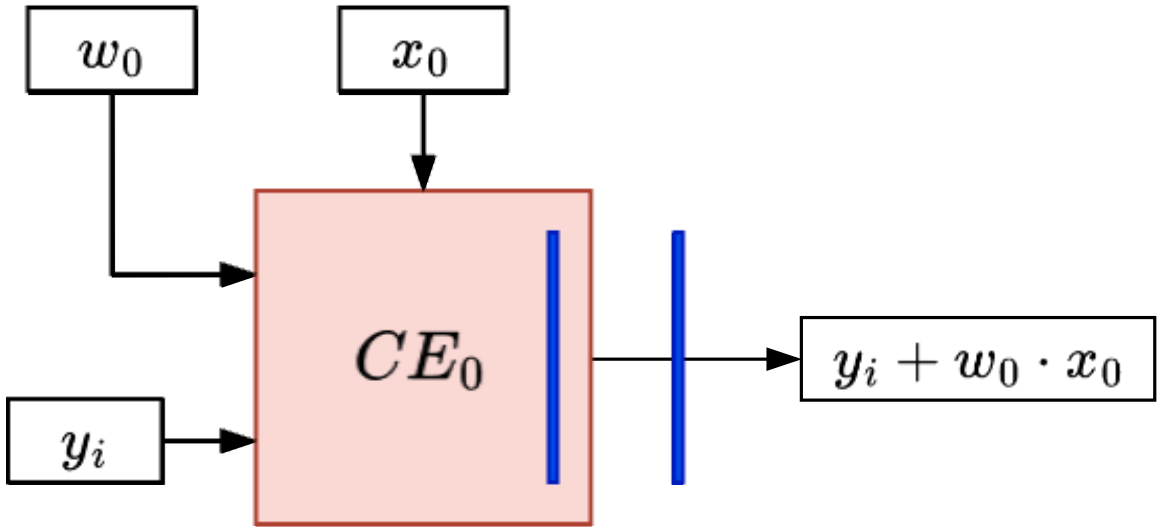
RedMule: Enhancing Rows to nonlinearity (NL) Rows



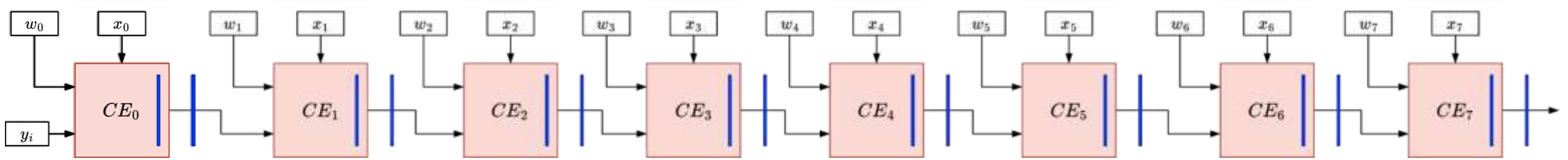
NL-Enhanced RedMule: Complete Design



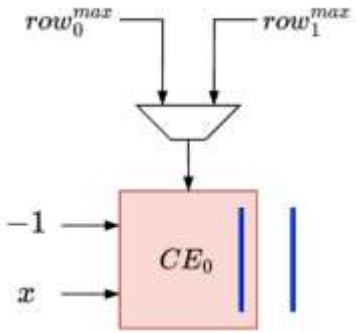
Basic Compute Element (CE): An FP16 FMA



Vanilla RedMule Row Architecture

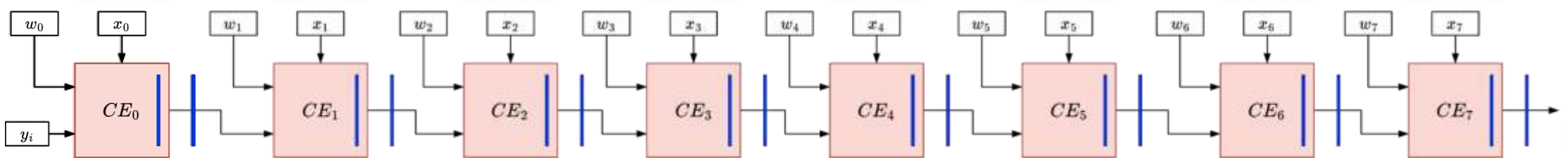


- $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$

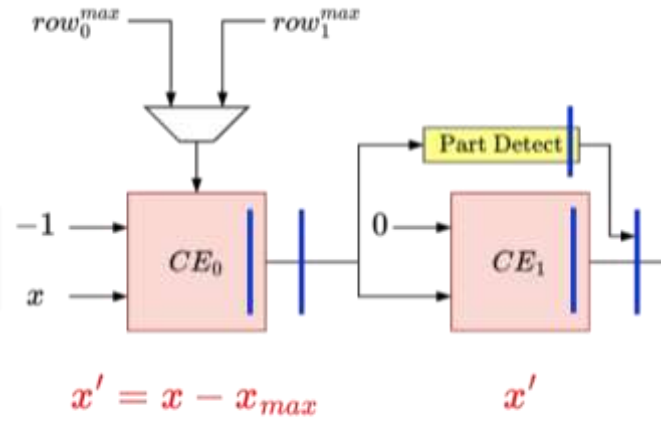


$x' = x - x_{max}$

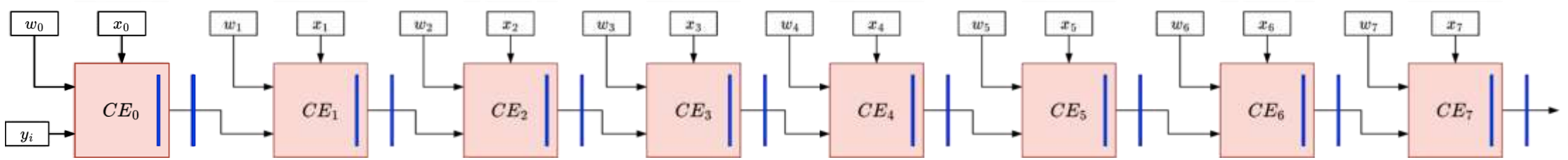
Softmax: exponentiation (part detection)



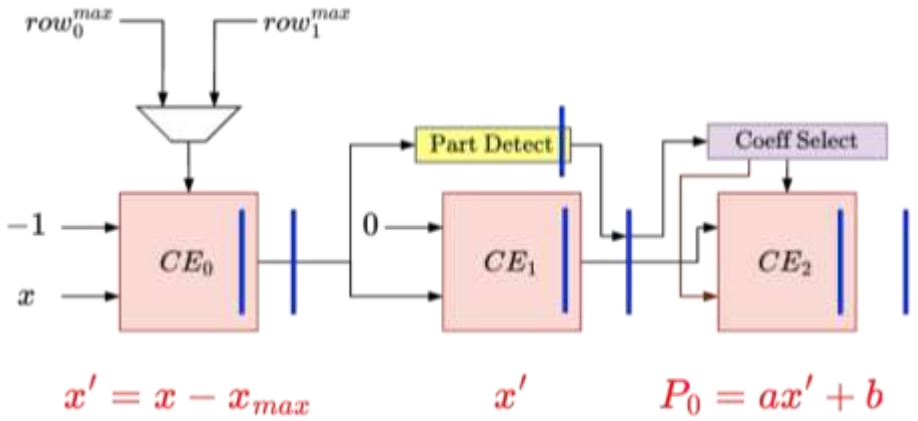
- $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$



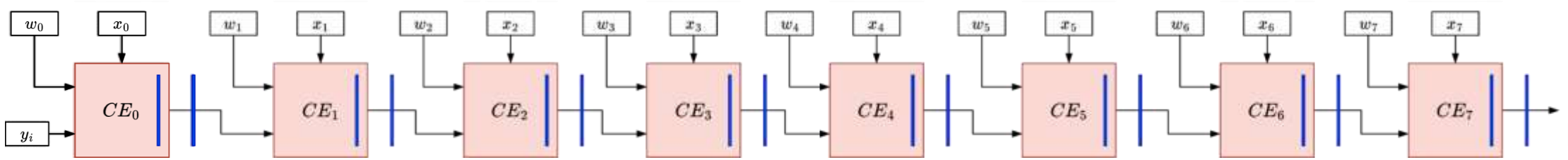
Softmax: exponentiation (coeff selection)



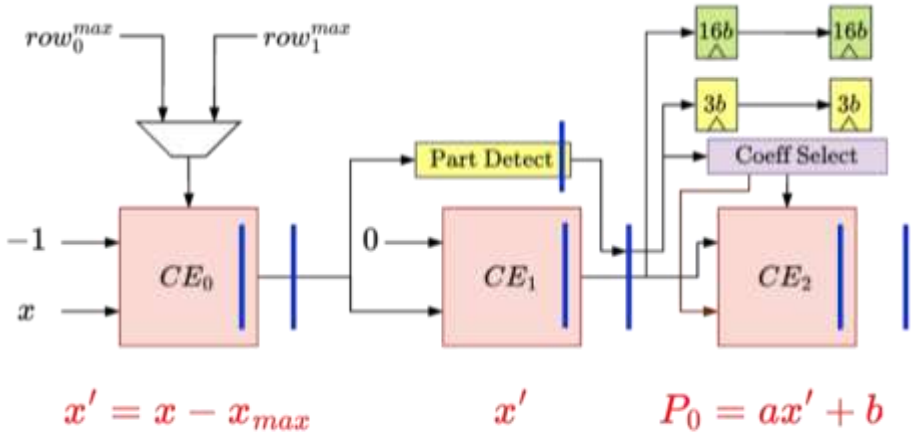
- $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$



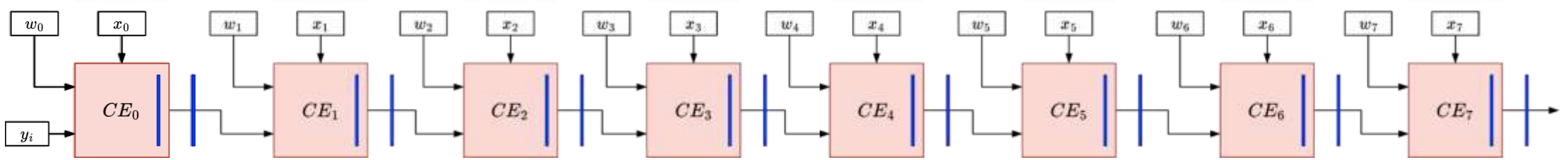
Softmax: exponentiation (buffer x and part id)



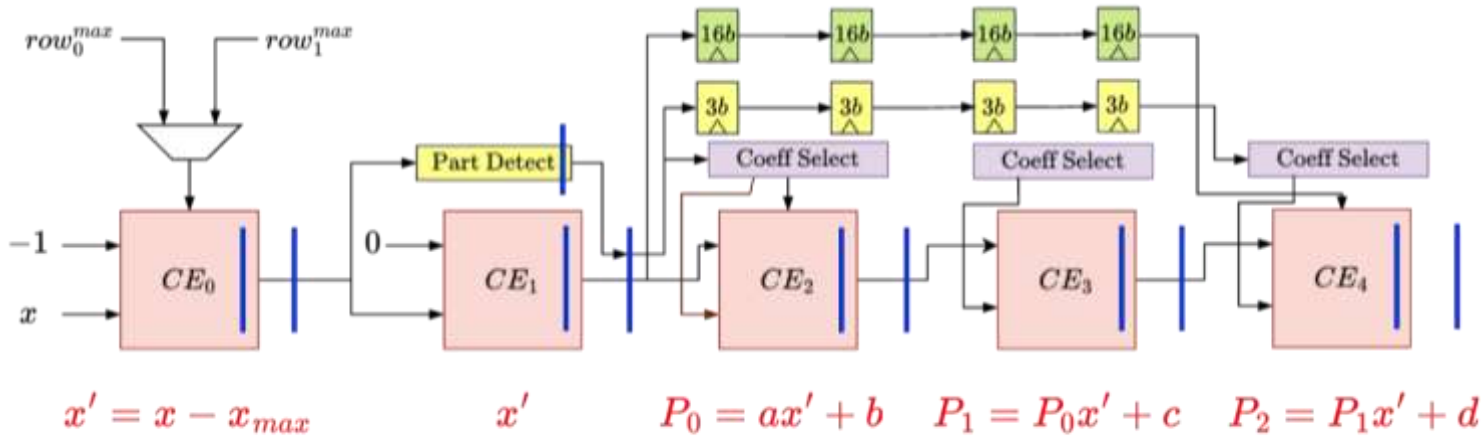
- $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$



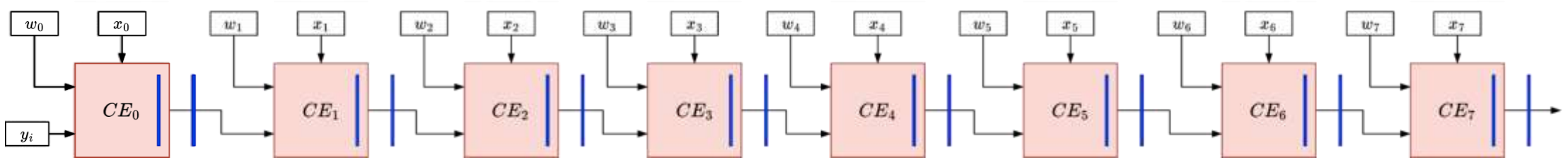
Softmax: exponentiation through PWPA



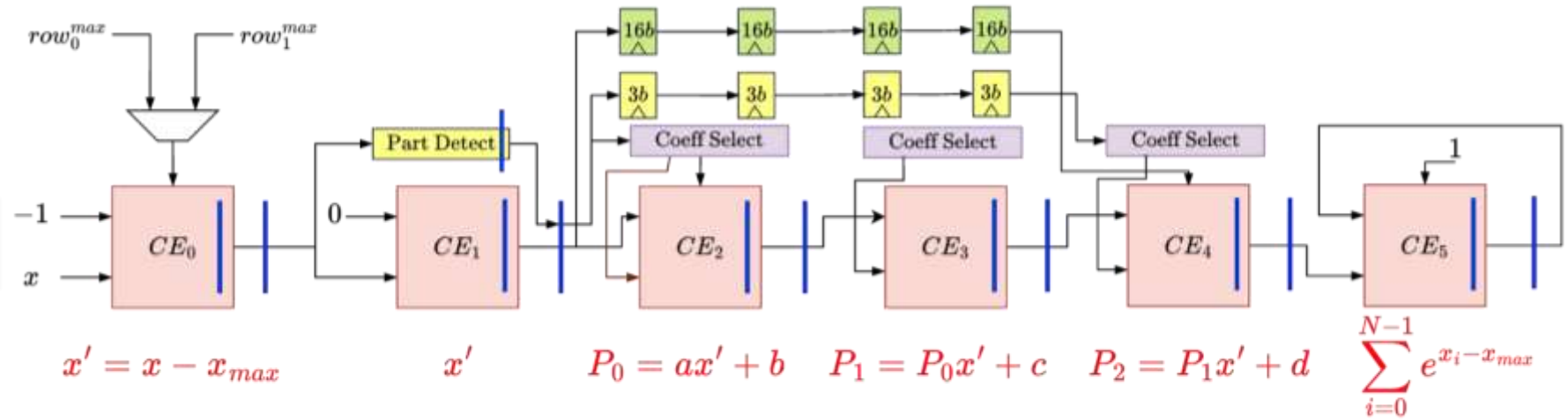
- $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$



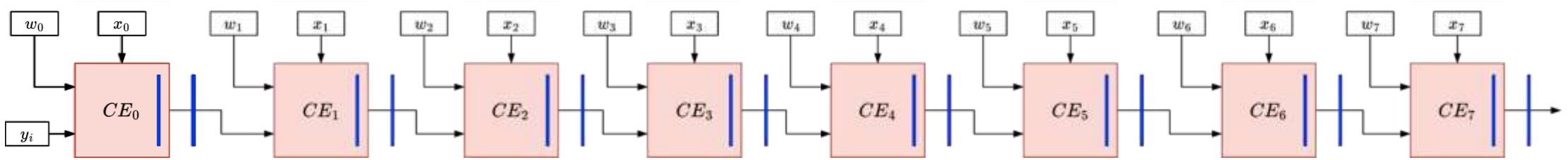
Softmax: sum to compute denominator



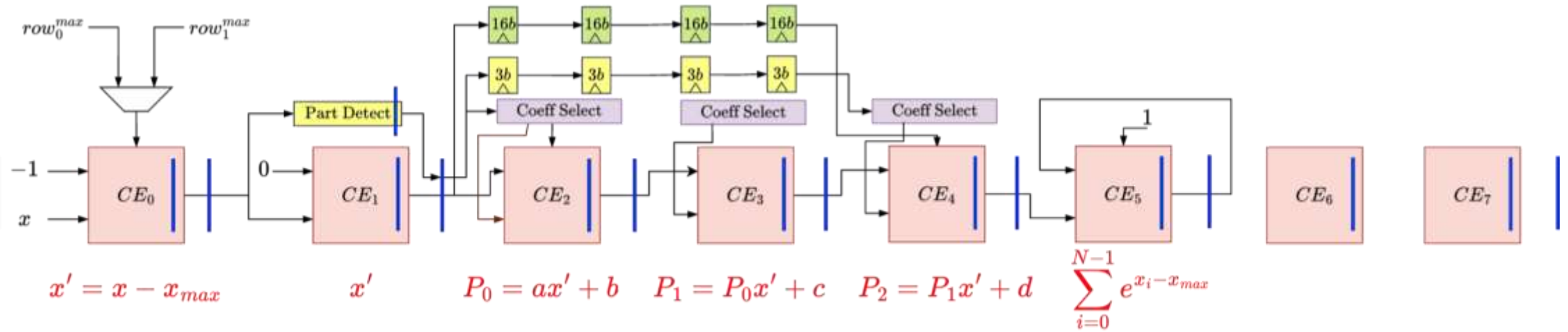
- $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$



Softmax: sum to compute denominator

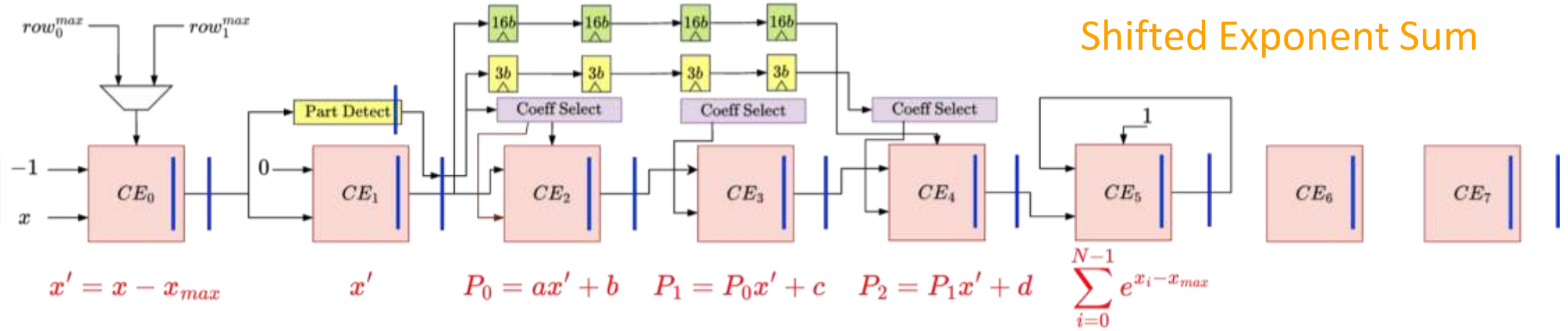


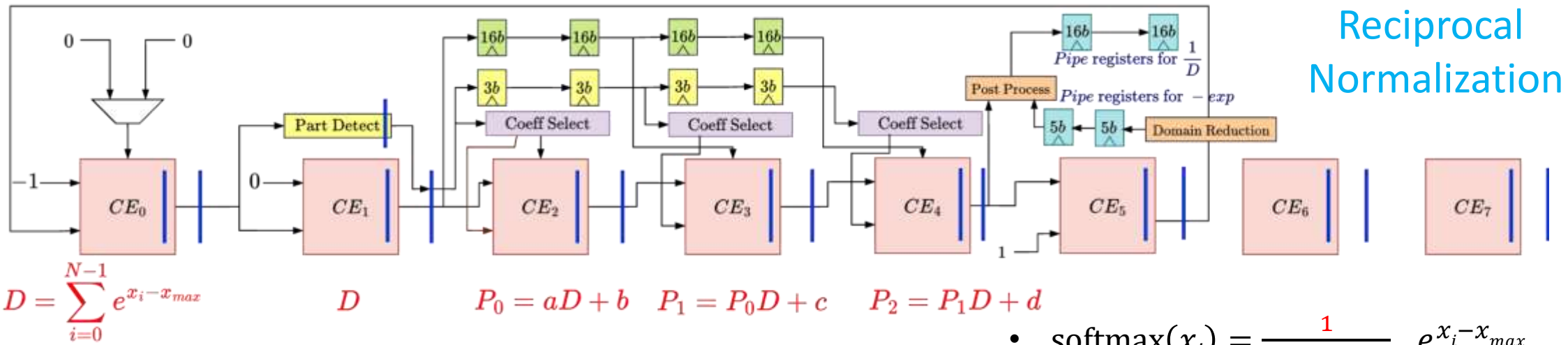
- $\text{softmax}(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_i - xmax}}$



No stream out to reduce traffic towards memory

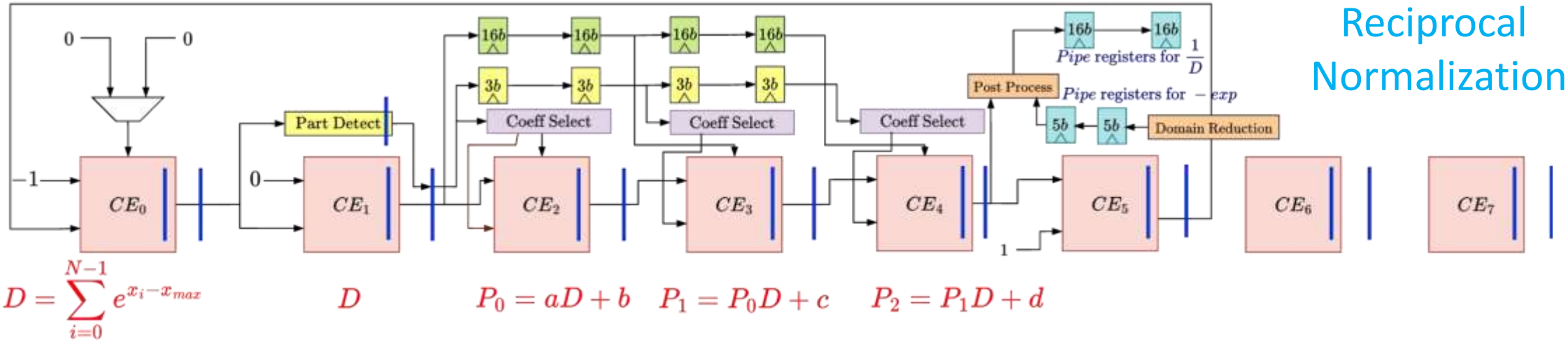
Softmax: Exponent Sum & Reciprocal



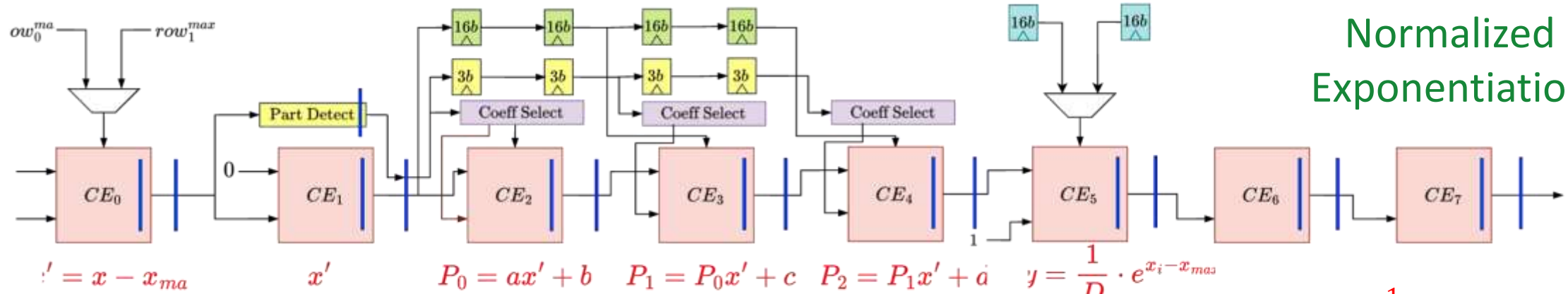


- $\text{softmax}(x_i) = \frac{1}{\sum e^{x_i - xmax}} \cdot e^{x_i - x_{max}}$

Softmax: Recomputing for 25% Traffic Reduction



Reciprocal Normalization

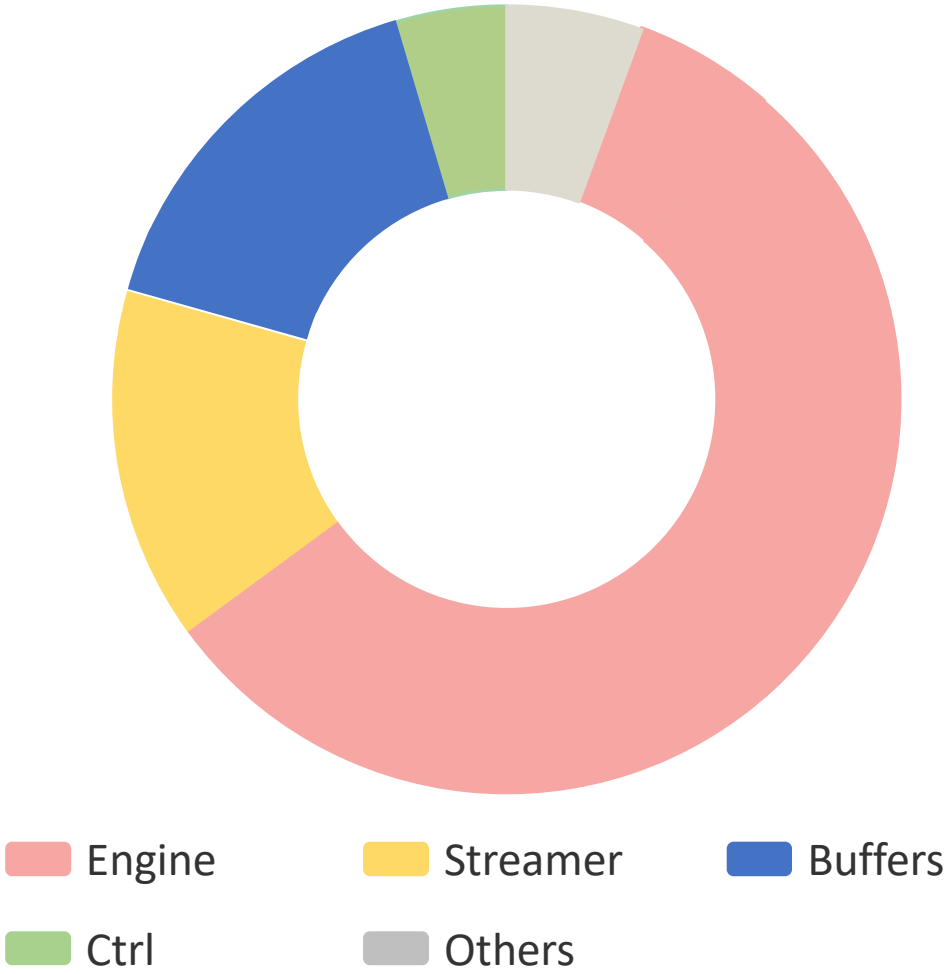


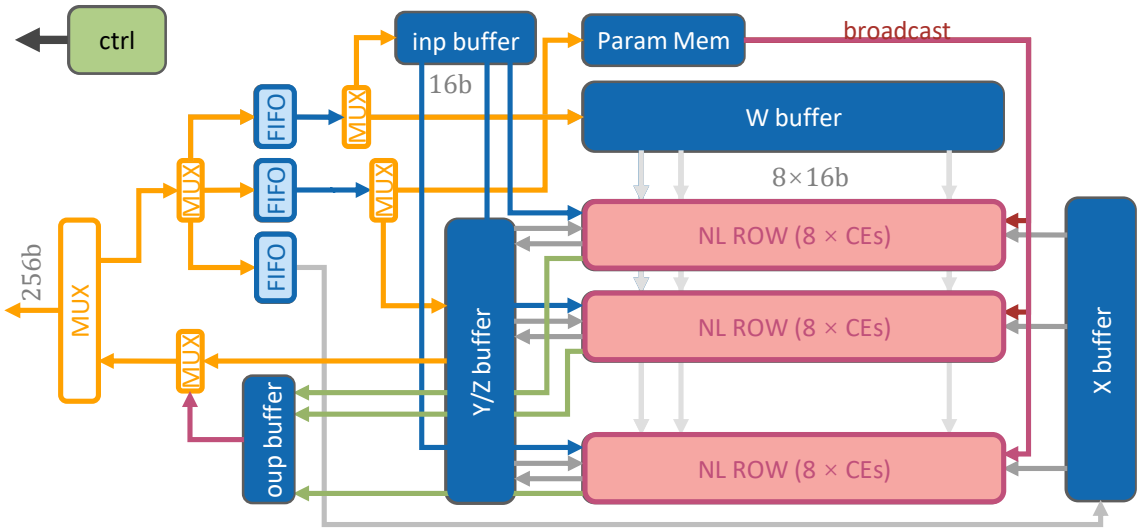
Normalized Exponentiation

- $\text{softmax}(x_i) = \frac{1}{\sum e^{x_i - xmax}} \cdot e^{x_i - x_{max}}$

Recomputing to **reduce** the memory traffic by **25%**

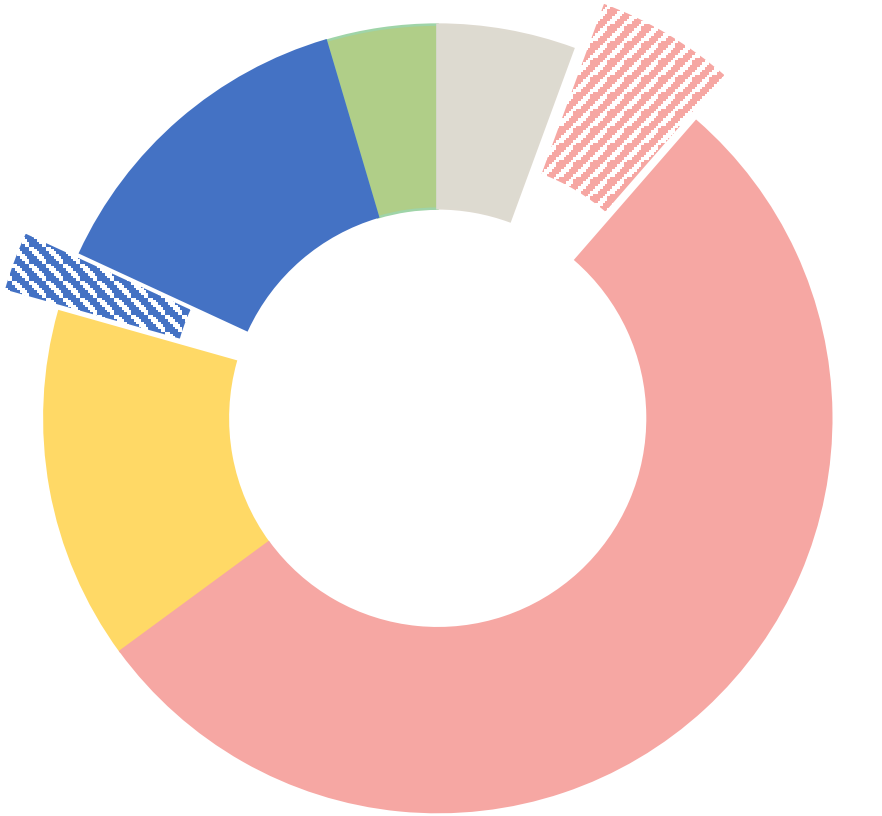
Area Overhead vs. Vanilla RedMule

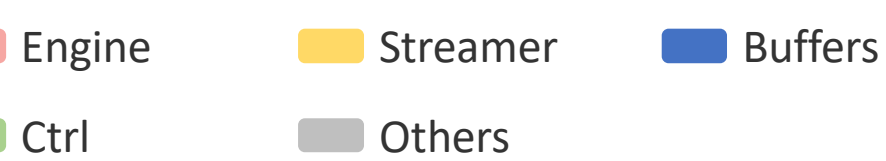


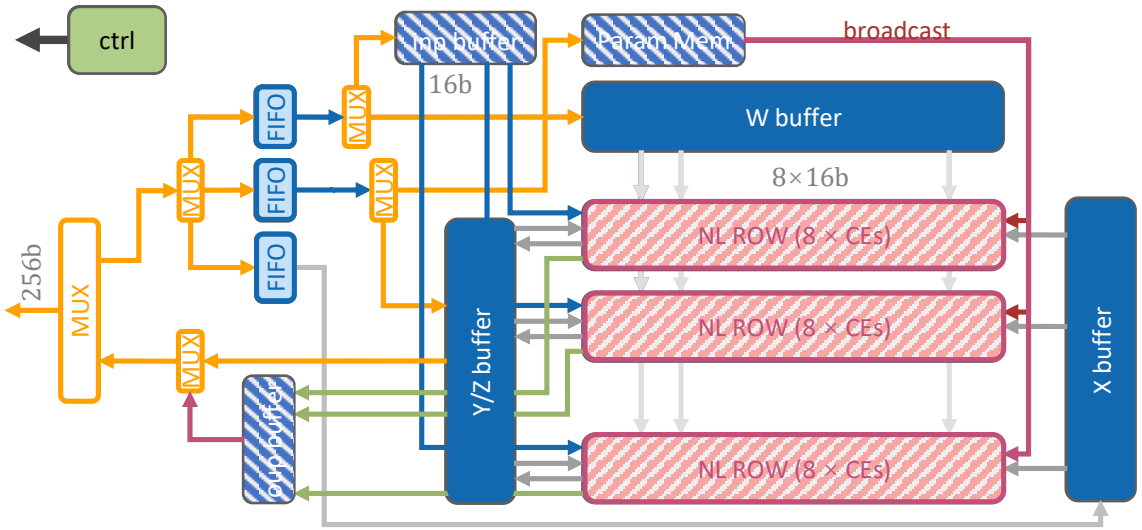


- Post route results → TSMC 7, 0.75V, TT, 1GHz

Area Overhead vs. Vanilla RedMule

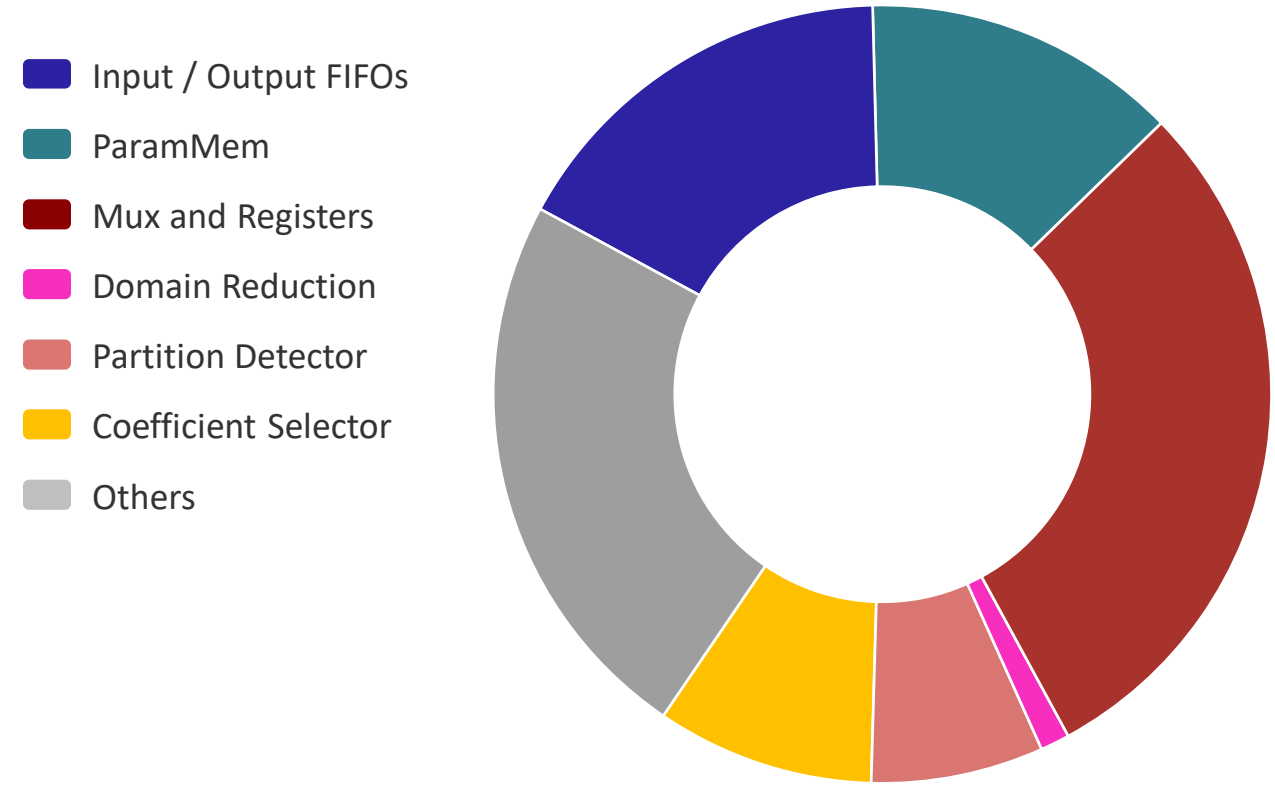






- Post route results → TSMC 7, 0.75V, TT, 1GHz
- Approximately 10% area overhead compared to vanilla RedMule

Nonlinearity Area Breakdown



Input-output FIFOs together contribute to 16% of the overhead, scales with the number of rows

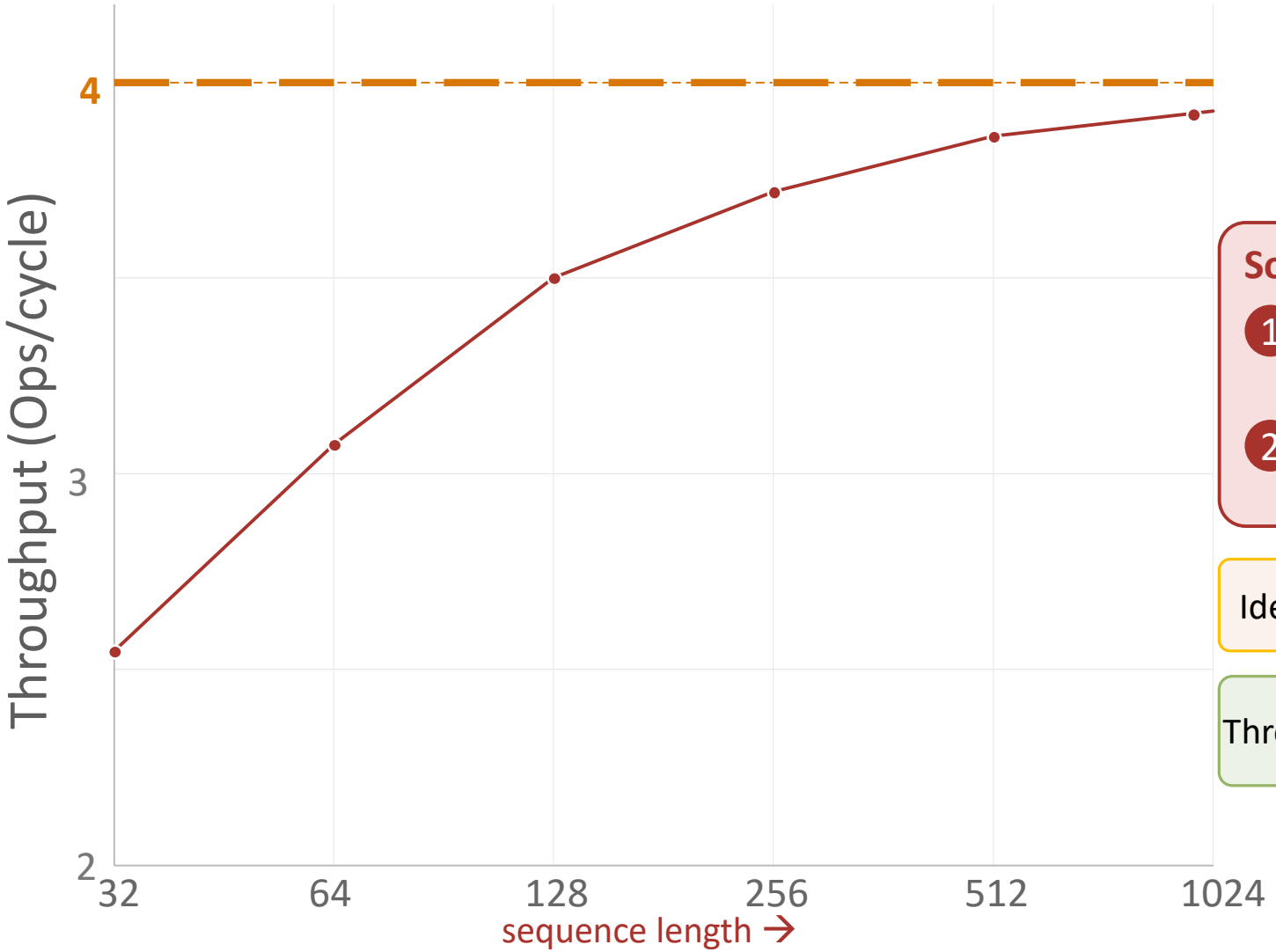
Shared Parameter Memory is 7.3 kGE and constitues to 13% of the area overhead

MUX and registers constitute nearly 30% of the overhead as they are added for each row

Domain Reduction for inverse and inverse square root operations adds negligible area overhead

The **Partition Detector** and **Coefficient Selector** contribute to 7.1 % and 9 % of the overhead

Softmax throughput of RedMule



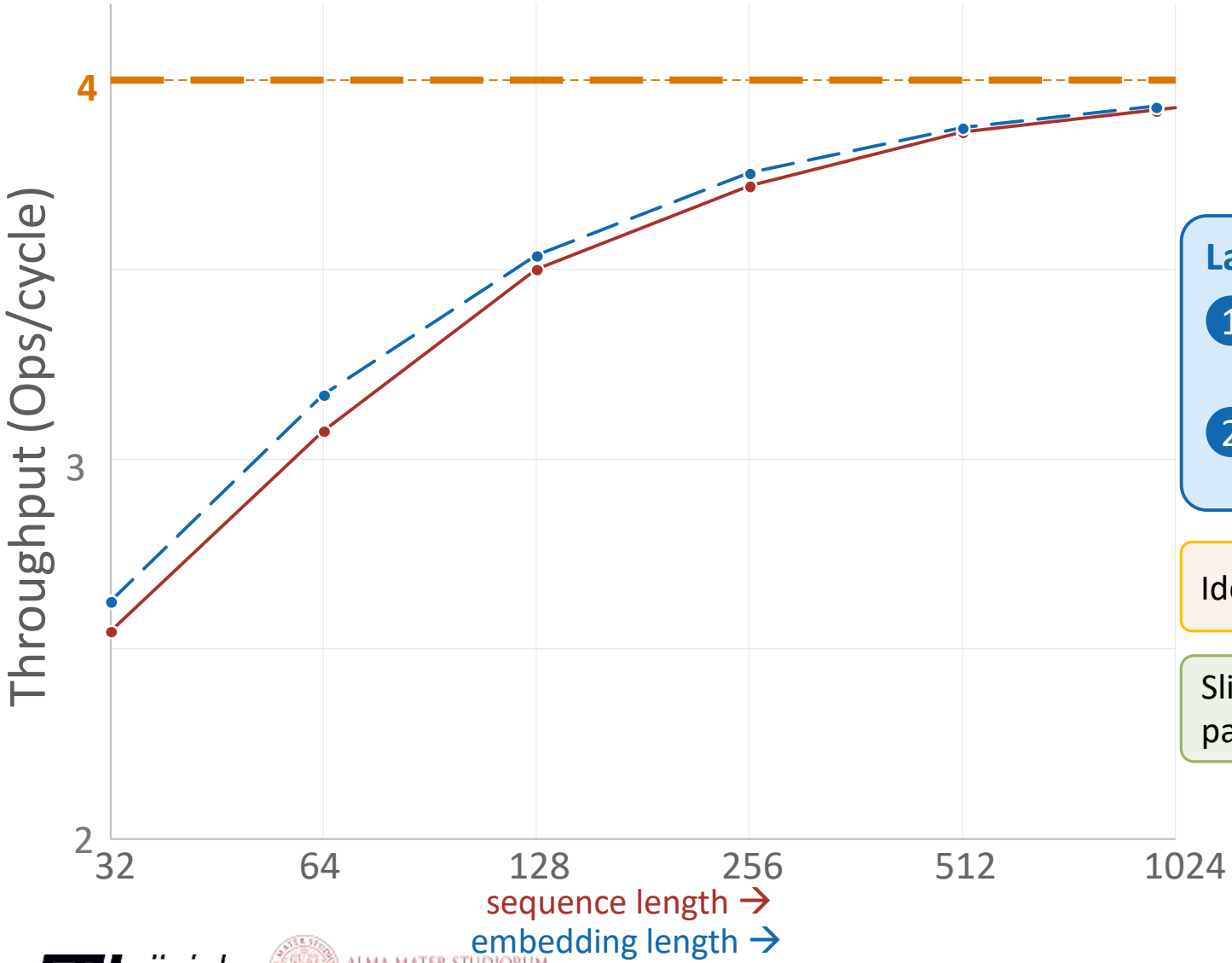
Softmax Kernel: (2 passes)

1. **compute softmax denominator:** evaluate $e^{(x-x_{max})}$ and reduce-sum across elements.
2. **Normalize:** scale exponentials by inverse denominator.

Ideal throughput is 4 Ops/cycle → 4 Gop/s @ 1GHz

Throughput approaches ideal for longer sequence length

Layer Normalization throughput of RedMule



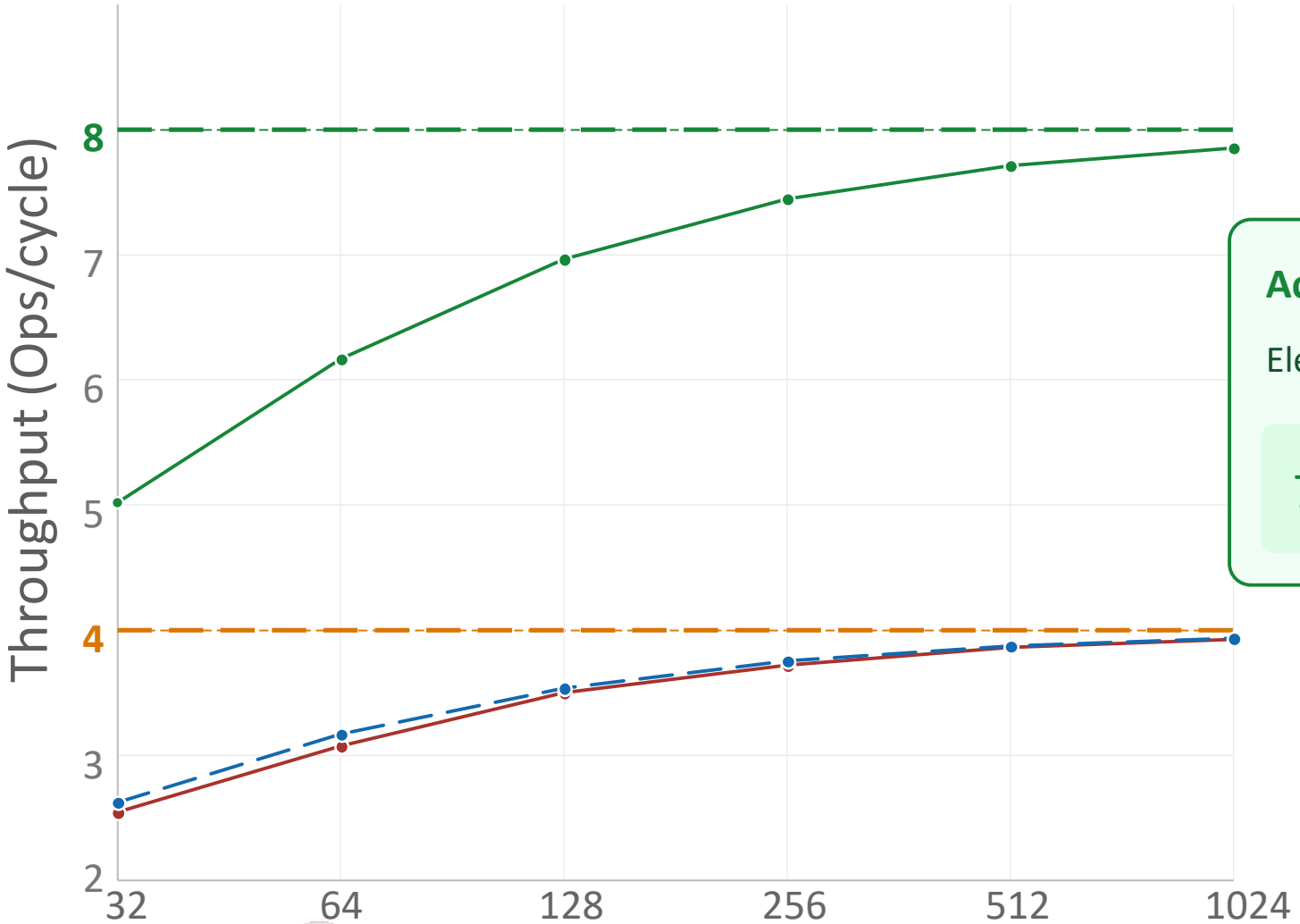
LayerNorm Kernel: (2 passes)

1. **compute statistics:** evaluate mean and variance
2. **Normalize:** offset by mean and scale by inverse of square root of variance

Ideal throughput is 4 Ops/cycle → 4 Gop/s @ 1GHz

Slightly higher throughput than softmax due to one parameter-memory initialization instead of two.

Activation throughput of RedMule



Activations: 2× higher throughput

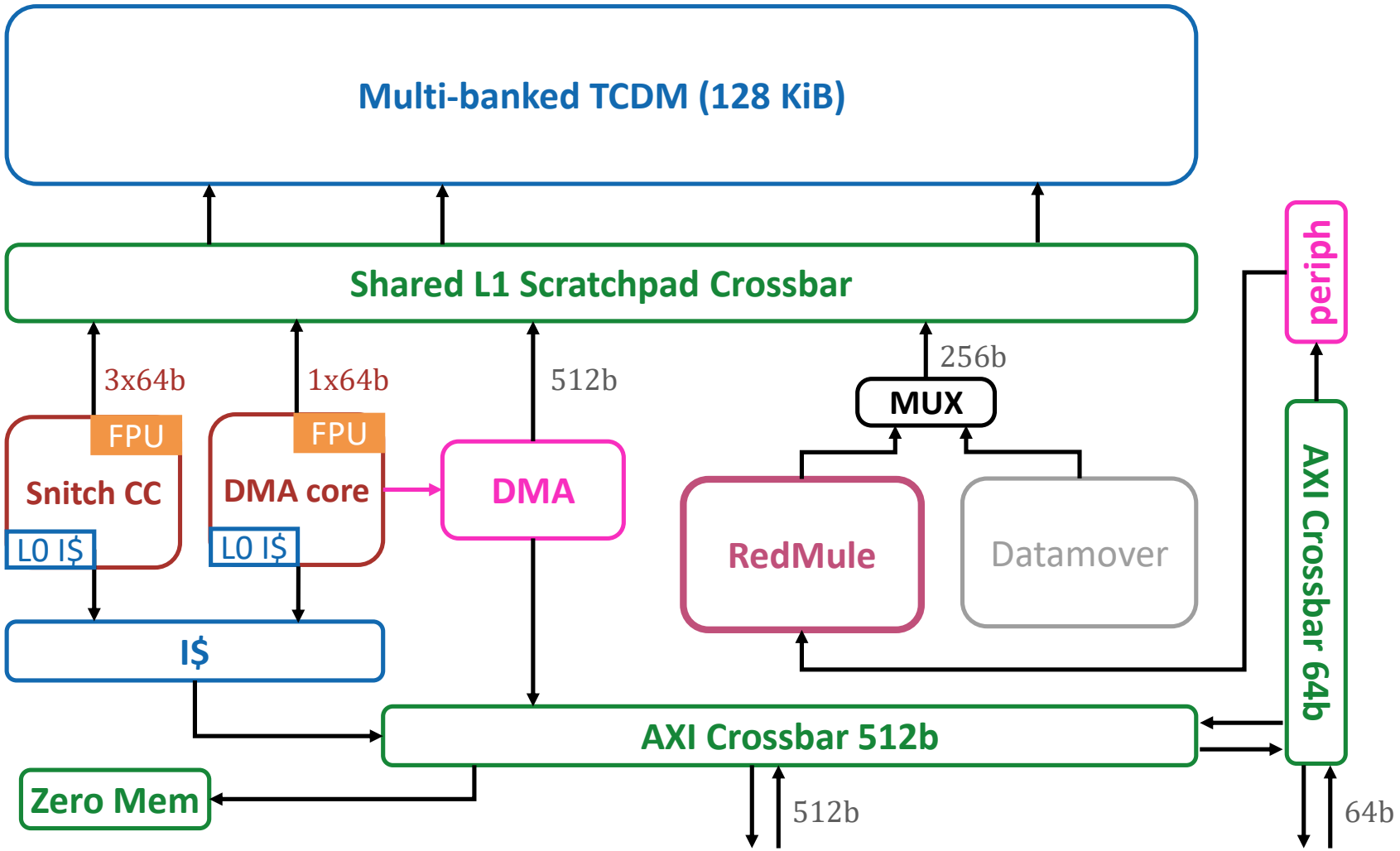
Elementwise ops (GeLU, SiLU) → single-pass streaming

7.85 Gop/s

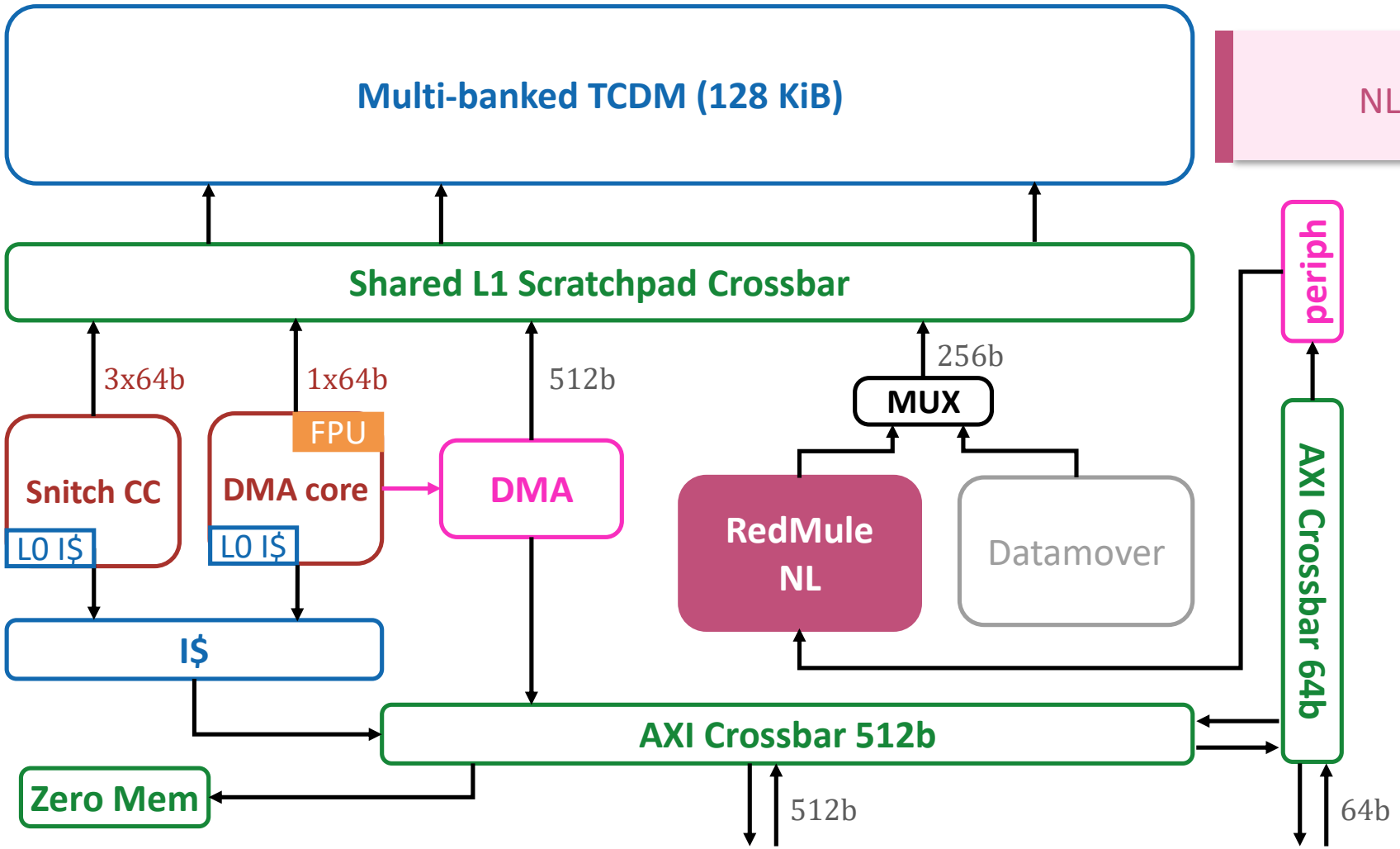
Ideal: 8 Ops/cycle

Efficiency: **~98%**

Snitch Cluster: System Level Integration

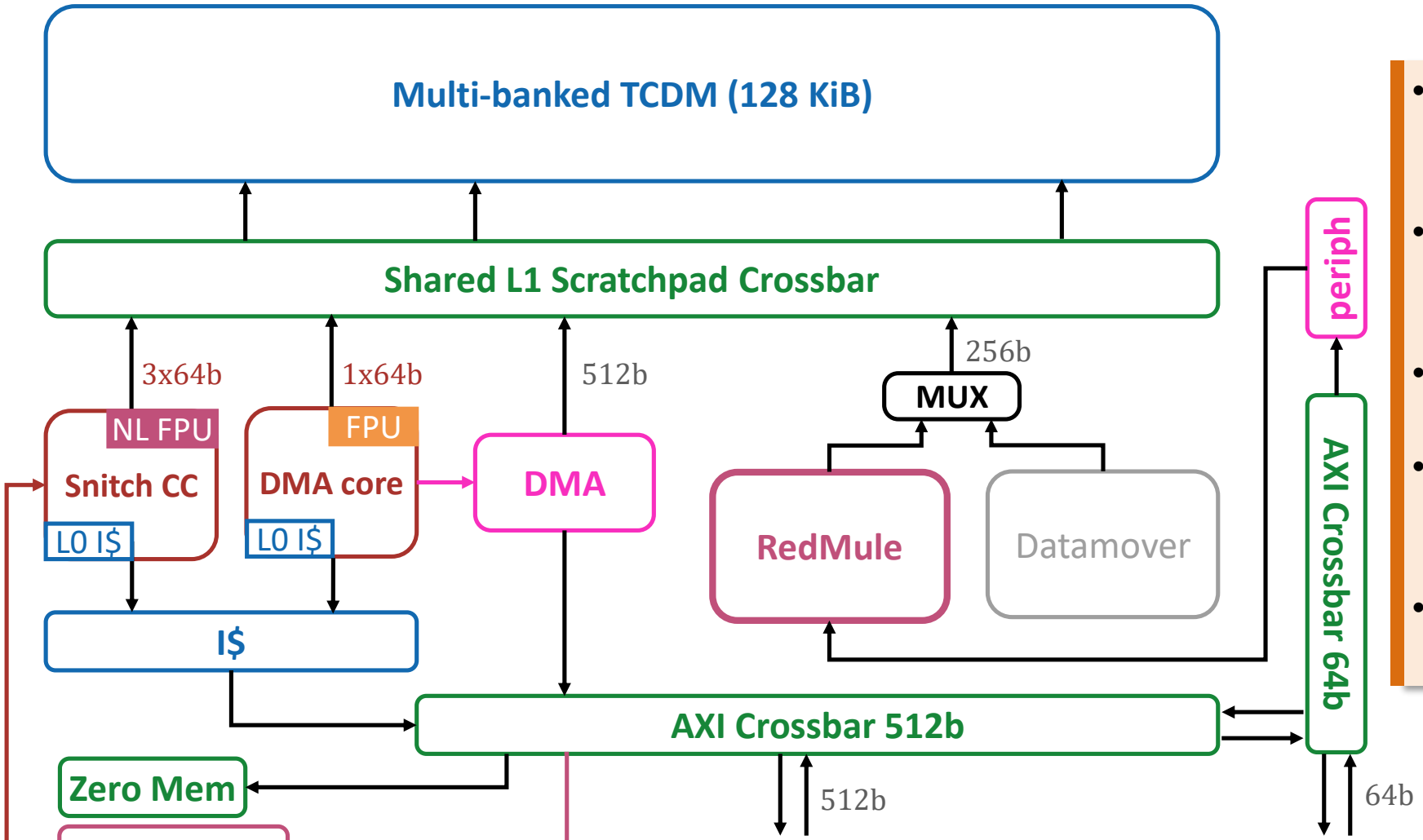


System-1: NL-Enhanced RedMule



NL enhanced in RedMule

System-2: NL enhanced FPU



- NL support in FPU + vanilla RedMule
- 64b FMA in 4-way 16b SIMD, enhanced with Nonlinearity
- Throughput 4 / 3 (NL / cycle)
- This solution requires FPU, which is ~200 kGE.
- 6 times slower than RedMule area overhead is nearly 4×

NL on RedMule vs. FPU: Energy Efficiency

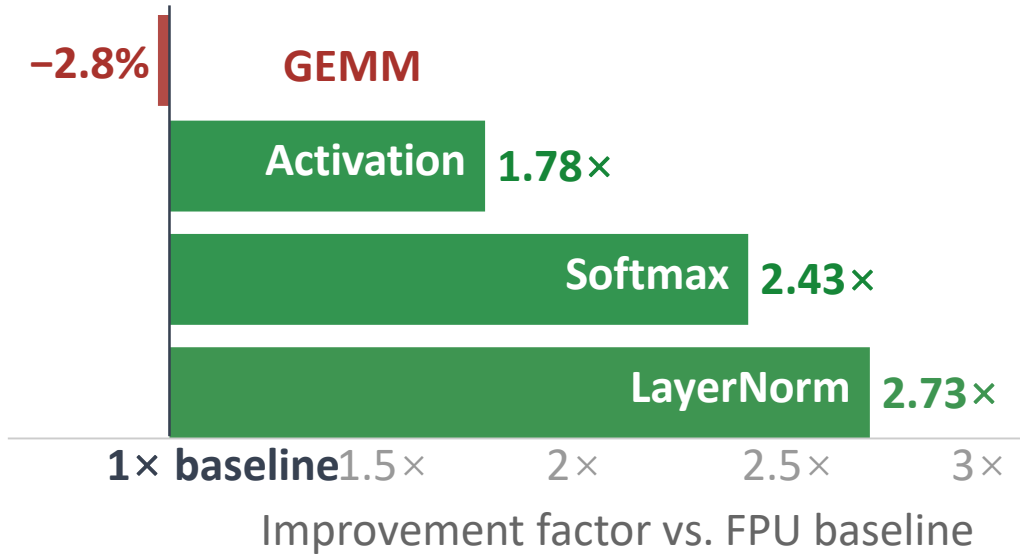
Baseline cluster

Vanilla RedMule
GEMM ops

FPU
NL ops

RedMule-NL cluster

RedMule-NL
GEMM + NL by FMA resue → no FPU



NL on RedMule vs. FPU: Area Efficiency

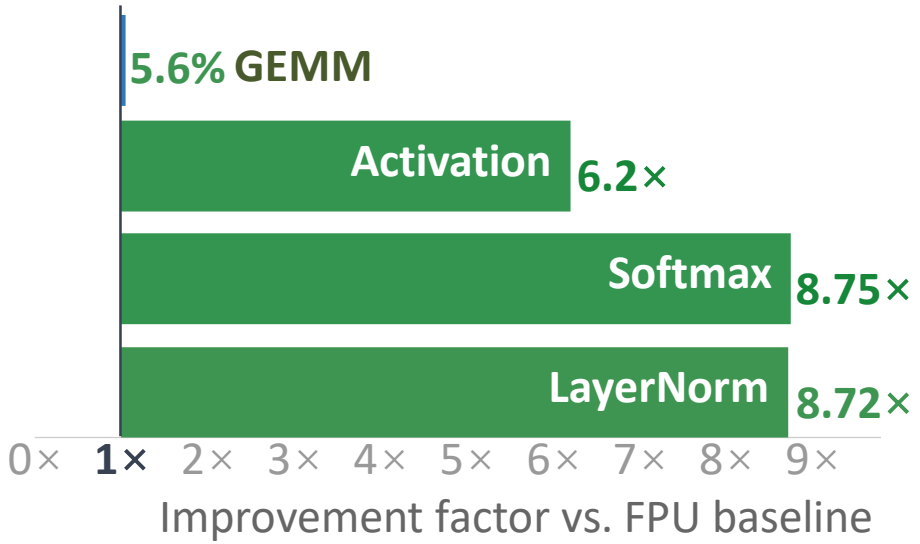
Baseline cluster

Vanilla RedMule
GEMM ops

FPU
NL ops

RedMule-NL cluster

RedMule-NL
GEMM + NL by FMA resue → no FPU



Softmax: State-of-the-Art Comparison

Reference	Precision	Soft./LN/Act	Tech (nm)	Freq (GHz)	Area (µm²)	Throughput (GOp/s)	Area Eff. (GOp/s/mm²)	Energy Eff. (GOp/J)
Zhu et al. [23]	**FX16**	✓ ✗ ✗	**28**	**1.64**	**18392**	**13.12**	**713**	—
Wang et al. [20]	BFP16	✓ ✗ ✗	12	1	968	0.45	465	63
Belano et al. [11]	BFP16	✓ ✗ ✓	12	1.12	39000	2.25	57.6	42
This Work	FP16	✓ ✓ ✓	7	1	3095	3.93	1270	92

[23]: our method supports diverse nonlinearities and is a drop-in replacement.

Softmax: State-of-the-Art Comparison

Reference	Precision	Soft./LN/Act	Tech (nm)	Freq (GHz)	Area (µm²)	Throughput (GOp/s)	Area Eff. (GOp/s/mm²)	Energy Eff. (GOp/J)
Zhu et al. [23]	FX16	✓ ✗ ✗	28	1.64	18392	13.12	713	—
Wang et al. [20]	**BFP16**	✓ ✗ ✗	**12**	**1**	**968**	**0.45**	**465**	**63**
Belano et al. [11]	BFP16	✓ ✗ ✓	12	1.12	39000	2.25	57.6	42
This Work	FP16	✓ ✓ ✓	7	1	3095	3.93	1270	92

[20]: Extends the FPU only for exponent approximation; our method is more general-purpose

Softmax: State-of-the-Art Comparison

Reference	Precision	Soft./LN/Act	Tech (nm)	Freq (GHz)	Area (µm²)	Throughput (GOp/s)	Area Eff. (GOp/s/mm²)	Energy Eff. (GOp/J)
Zhu et al. [23]	FX16	✓ ✗ ✗	28	1.64	18392	13.12	713	—
Wang et al. [20]	BFP16	✓ ✗ ✗	12	1	968	0.45	465	63
Belano et al. [11]	**BFP16**	✓ ✗ ✓	**12**	**1.12**	**39000**	**2.25**	**57.6**	**42**
This Work	FP16	✓ ✓ ✓	7	1	3095	3.93	1270	92

[11]: More general-purpose, but relies on the core FPU for SiLU/GELU, adding throughput and area overhead

Softmax: State-of-the-Art Comparison

Reference	Precision	Soft./LN/Act	Tech (nm)	Freq (GHz)	Area (µm²)	Throughput (GOp/s)	Area Eff. (GOp/s/mm²)	Energy Eff. (GOp/J)
Zhu et al. [23]	FX16	✓ ✗ ✗	28	1.64	18392	13.12	713	—
Wang et al. [20]	BFP16	✓ ✗ ✗	12	1	968	0.45	465	63
Belano et al. [11]	BFP16	✓ ✗ ✓	12	1.12	39000	2.25	57.6	42
This Work	FP16	✓ ✓ ✓	7	1	3095	3.93	1270	92

Technology differences make energy comparison unfair, but in absolute terms, our method is **1.46× more energy-efficient than [20]** and **1.73× more area-efficient than [23]**.

Activation: State-of-the-Art Comparison

Reference	Precision	Soft./LN/Act	Tech (nm)	Freq (GHz)	Area (µm²)	Throughput (GOp/s)	Area Eff. (GOp/s/mm²)	Energy Eff. (GOp/J)
Reggiani et al. [8]	**I/FP32/16/8**	✗ ✗ ✓	**28**	**0.6**	**14857**	**1.2**	**81**	**324**
Prasad et al. [19]	FP32/16	✗ ✗ ✓	12	1	28178	15.6	553	269
Yu et al. [18]	I32/FP16/32	✓ ✓ ✓	7	0.74	498	0.74	1486	—
This Work	FP16	✓ ✓ ✓	7	1	3095	7.85	2536	172

Excludes FMA power; area comparisons are node-dependent. Our design shares parameter memory and reduces partition/selection logic via higher-degree polynomials, thereby being architecturally more area-efficient.

Activation: State-of-the-Art Comparison

Reference	Precision	Soft./LN/Act	Tech (nm)	Freq (GHz)	Area (µm²)	Throughput (GOp/s)	Area Eff. (GOp/s/mm²)	Energy Eff. (GOp/J)
Reggiani et al. [8]	I/FP32/16/8	✗ ✗ ✓	28	0.6	14857	1.2	81	324
Prasad et al. [19]	**FP32/16**	✗ ✗ ✓	**12**	**1**	**28178**	**15.6**	**553**	**269**
Yu et al. [18]	I32/FP16/32	✓ ✓ ✓	7	0.74	498	0.74	1486	—
This Work	FP16	✓ ✓ ✓	7	1	3095	7.85	2536	172

[19]: Lightweight datapath, but **4.5× less area-efficient**. It's 1.55× standalone energy gain drops to ~1.1× after accounting for data-movement/synchronisation overhead.

Activation: State-of-the-Art Comparison

Reference	Precision	Soft./LN/Act	Tech (nm)	Freq (GHz)	Area (µm²)	Throughput (GOp/s)	Area Eff. (GOp/s/mm²)	Energy Eff. (GOp/J)
Reggiani et al. [8]	I/FP32/16/8	✗ ✗ ✓	28	0.6	14857	1.2	81	324
Prasad et al. [19]	FP32/16	✗ ✗ ✓	12	1	28178	15.6	553	269
Yu et al. [18]	**I32/FP16/32**	✓ ✓ ✓	**7**	**0.74**	**498**	**0.74**	**1486**	—
This Work	FP16	✓ ✓ ✓	7	1	3095	7.85	2536	172

[18]: Uses Piecewise Linear with only **16 partitions** and validates **2 workloads**; [8] suggests **32+ partitions** for 700+ workloads. Despite post-synthesis vs. our post-layout results, we achieve **1.7× higher area efficiency**.

Conclusion: FMA-centric approach to accelerate NL

1. **Unified Nonlinearity Hardware Extension:**
   Hardware extension to existing GEMM unit enabling diverse nonlinearities exp(x), SiLU, GELU etc. including 1/x, and 1/√x — all primitives needed for Softmax, LayerNorm, and Activations

2. **Low Area Overhead Solution**
   Demonstrated on an open-source GEMM accelerator with <10% standalone RedMule area overhead, saving 5% area versus FPU-based nonlinear support at system level.

3. **High Accuracy**
   Evaluated on ViT-B/L with no accuracy loss and >99.8% similarity, covering exp, SiLU, inverse for Softmax, and inverse square root for LayerNorm — all without fine-tuning.

4. **Efficiency & SoA**
   Achieves a peak energy efficiency of 92 / 98/ 172 Gop/J for softmax, layernorm and activation
   Achieves 1.7× higher area efficiency and 1.46 × higher energy efficiency across softmax solutions

References

[8] E. Reggiani et al., “Flex-SFU: Accelerating DNN activation functions by non-uniform piecewise approximation,” in Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC), 2023

[23] D. Zhu et al., “Efficient precision-adjustable architecture for softmax function in deep learning,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 12, pp. 3382–3386, 2020.

[20] R. Wang et al., “VEXP: A low-cost RISC-V ISA extension for acceler-ated softmax computation in transformers,” in Proceedings of the IEEE 32nd Symposium on Computer Arithmetic (ARITH), 2025.

[11] A. Belano et al., “A flexible template for edge generative AI with high- accuracy accelerated softmax and GELU,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 15, no. 2, pp. 200–216, 2025.

[19] A. S. Prasad et al., “PACE-Lite: Compact and efficient piecewise polynomial approximation for transformer nonlinearity acceleration,” in Proceedings of the IEEE International Conference on Computer Design (ICCD), 2025.

[18] J. Yu et al., “NN-LUT: Neural approximation of nonlinear operations for efficient transformer inference,” in Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC), 2022.