Is GEMM Enough for Transformers? A Template for Edge GenAI with Accelerated Softmax & GELU

Andrea Belano

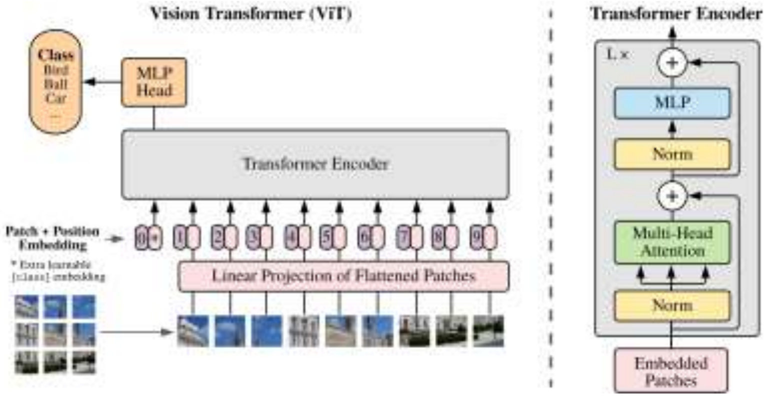
PULP Platform
Open Source Hardware, the way it should be!

About Me

- Received the Bachelor's and Master's Degree in Computer Engineering at the University of Bologna in 2022 and 2024 respectively
- Started a PhD in Microelectronics in October 2024 under the group of Prof. Luca Benini
- My research focuses on the hardware acceleration of Artificial Intelligence applications on energy efficient platforms

The Transformer

- **Transformers are the main models driving the evolution of modern Artificial Intelligence**
  - Both in **perceptive** task and in **generative** applications
- **However, this performance uplift comes at a cost**
  - Transformers generally use more parameters than previous-gen neural networks
  - Each layer of a Transformer is more complex, featuring multi-head self-attention (MHSA) and additional projections



Why Transformers at the Edge?

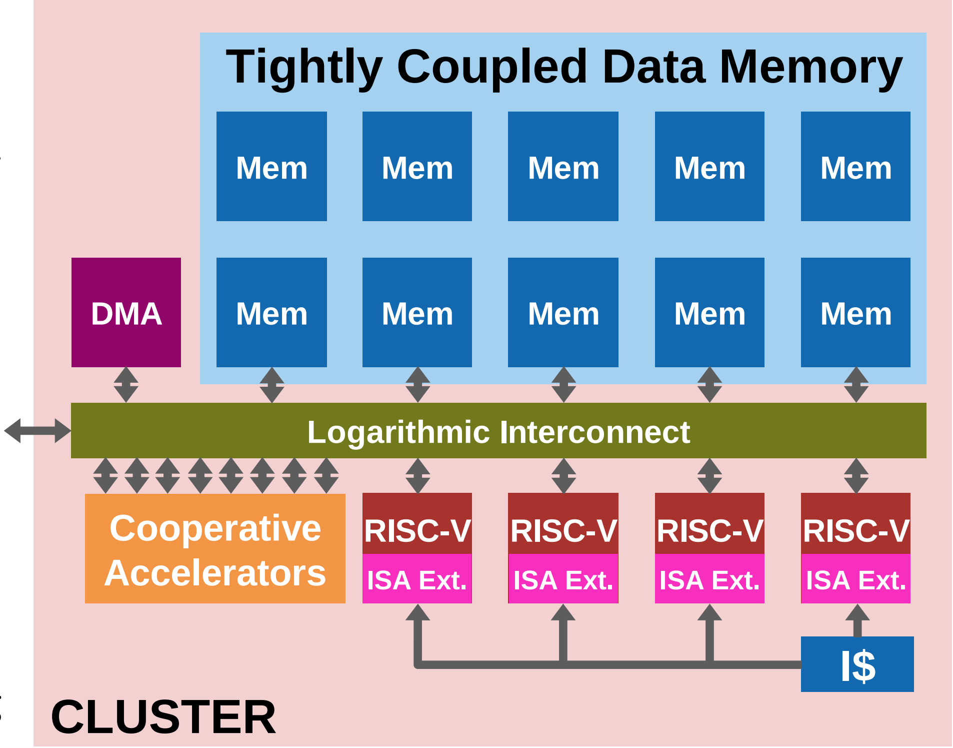
- **State-of-the-art models are in the order of $10^{11}, 10^{12}$ parameters**
  - Edge inference is unthinkable, not even remotely near the required performance and memory capacity on embedded devices
  - Cloud is the natural choice for these models
- **Significant interest in running smaller models ($10^{8}, 10^{9}$ parameters) at the Edge**
- **Why running such models at the edge?**
  - Low latency applications
  - Reduce wireless traffic congestion
  - Improve privacy and security in GenAI applications

A Fully Integer Transformer?

- **Fully-integer CNNs are the standard**
  - Quantization greatly cuts the model size
  - Boosts inference speed and efficiency
  - Comparable performance to non-quantized models
- **What about Transformers?**
  - Orders of magnitude more expensive to train compared to CNNs
  - Often trained on huge, non-public datasets and/or with human feedback
  - Activation quantization still not mature
- **Quantization is often unfeasible!**
- **For this reason, we will focus on the acceleration of transformers in their native format (BF16)**

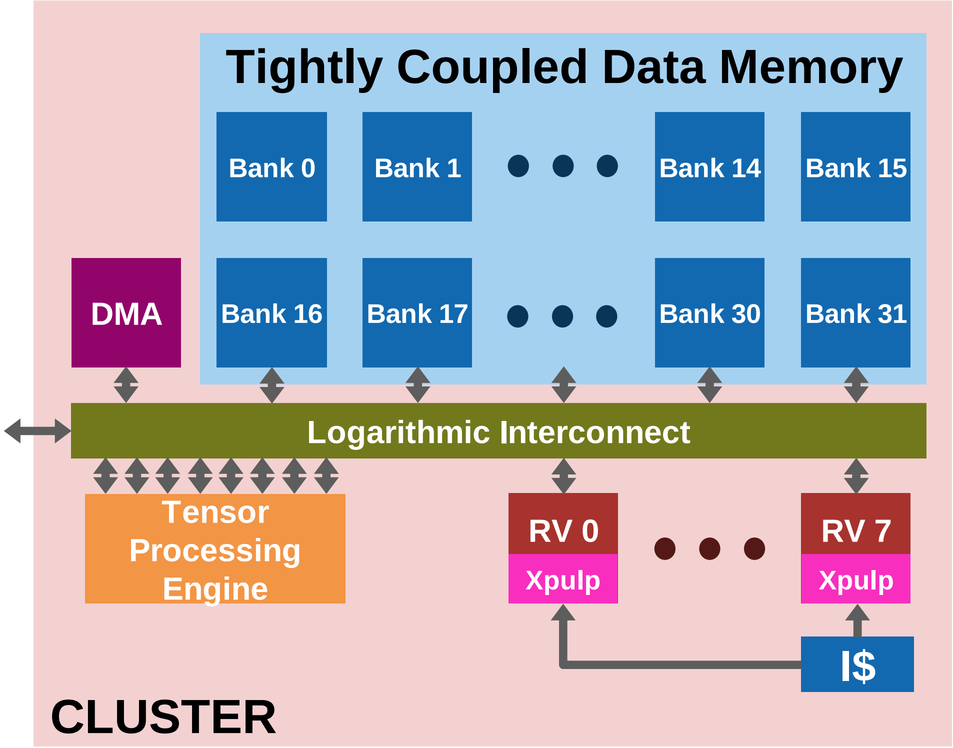
The PULP Cluster Template

- **2-8 RISC-V digital signal processing cores**
- **Shared L1 Scratchpad Memory (Tightly Coupled Data Memory)**
    - **bank interleaving** to maximize available bandwidth in typical parallel computing scenarios
- **32KiB shared instruction cache**
- **Accelerate specific tasks through:**
    - ISA extensions
    - Cooperative HWPE (Hardware Processing Engines)



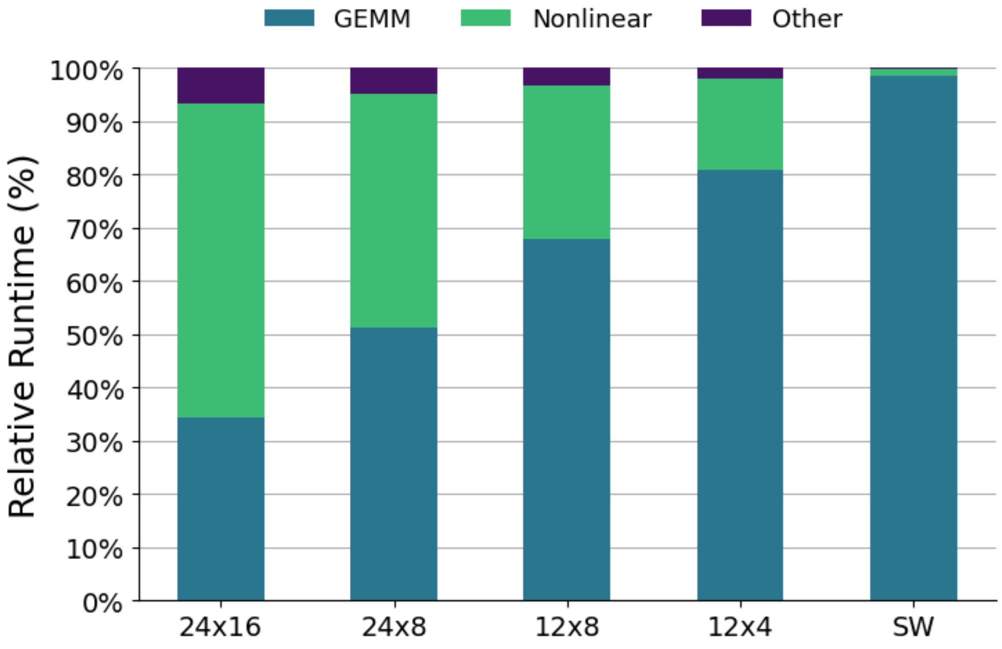
A Transformer-Ready PULP Cluster?

- **8 RI5CY 32-bit cores**
  - 4-stages, in-order pipeline
  - Xpulp extensions (HW loops, bit manipulations, SIMD)
  - Private FPU supporting FP32 and BF16 formats, with 2-way SIMD support for BF16
- **256KiB of TCDM split among 32 banks**
- **A Tensor Processing Engine based on the RedMulE architecture**



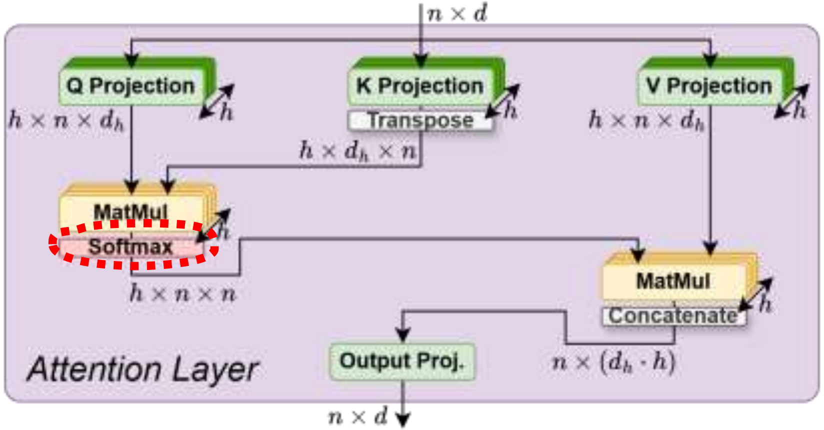
Is GEMM Really Enough?

- **Let's run ViT-Base on the proposed cluster and sweep RedMulE's number of CEs**
  - Nonlinear operations are approximated using the fastest possible method
- **Huge initial gains but diminishing returns as we enlarge the Tensor Processing Engine**
  - Utilization as low a 31%
- **Big relative contribution from non-GEMM operations when matmuls are accelerated**



The Attention Mechanism & Softmax

- **Attention consists of multiple GEMMs + softmaxs**
  - Unlike CNNs, softmax is applied multiple times every layer
- **It is fundamental to also accelerate softmax if we target Transformer-based models**
- **What's the deal with this function?**
  - It is based on the **exponential** function
  - It is **NOT** a **point-to-point** function



$$Softmax(x_i) = \frac{e^{x_i - x_{max}}}{\sum e^{x_j - x_{max}}}$$

Glibc’s Exponential Function

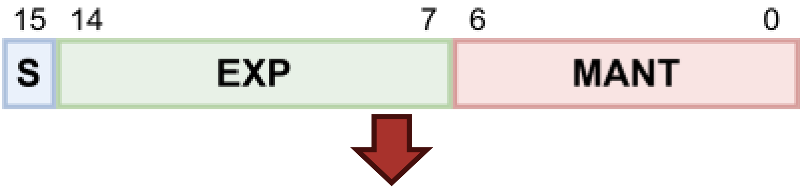
- How is exp normally implemented? Let’s have a look to glibc’s implementation…
- Look-up tables, polynomial approximations, double precision…
- Clearly not suitable for low-power applications, let alone hardware accelerators
- We need an alternative

Other exp Approximations

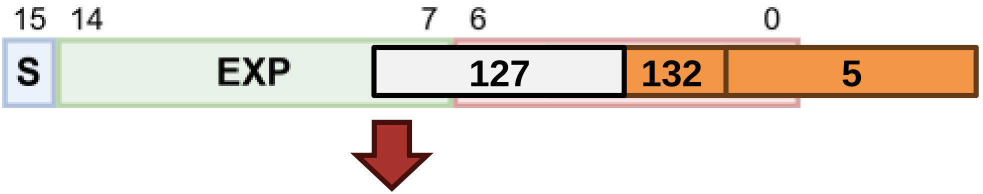
- **What about less computationally-intensive approximations?**
- **CORDIC**
  - Good accuracy and efficient
  - Slow convergence
- **LUT-based methods**
  - Perform no computation at all besides interpolation
  - Costly in terms of area
  - Work best with limited ranges
- **Polynomial approximations**
  - We can build optimal approximations using Chebyshev's polynomials
  - For a good result we must limit the input range

An Efficient Approximation – Schraudolph's Method

- **Think outside the box:**
  - How are floats stored?
  - How do we find their value?
  - Then what if add an integer the bias and replace the exponent with it?
- **We get a perfect base-2 exponentiation**
- **However:**
  - What about base-e exp?
    - → Just multiply the input by $\log_2 e$
  - **What if the input is not an integer?**



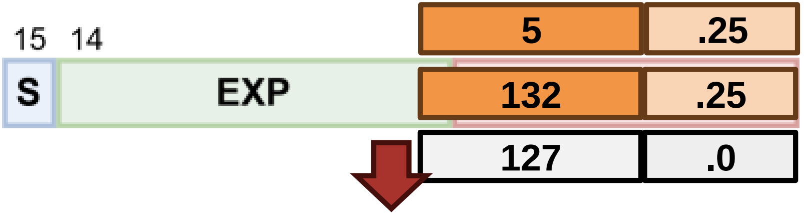
$$\text{value} = (-1)^{S} \cdot 2^{\text{EXP}-127} \cdot (1 + \text{MANT})$$



$$\text{value} = (-1)^{0} \cdot 2^{132-127} \cdot (1 + 0) = 2^5$$

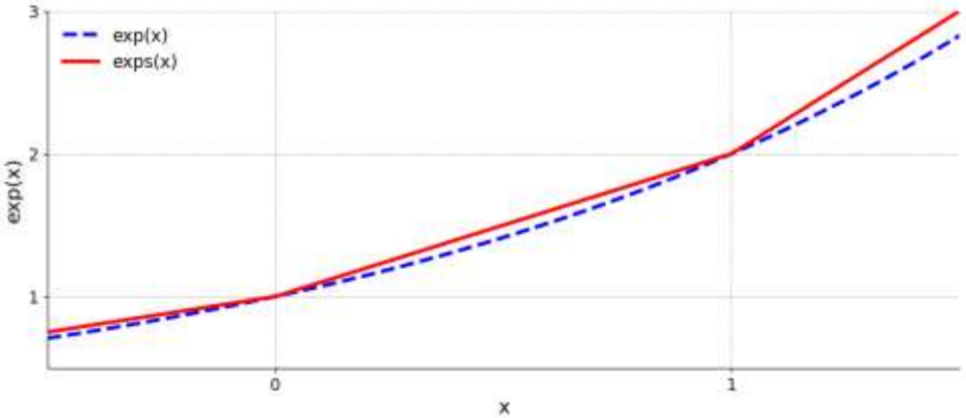
An Efficient Approximation – Schraudolph's Method

- **Let's apply the same function to a fixed-point number**
  - Same process as before but we shift the number until the integer part overlaps with the exponent bits
- **What happens to the result?**
  - The integer part is **perfectly exponentiated**
  - The fractional part becomes the mantissa
- **We get a linear interpolation of the 2 nearest integer powers of 2**



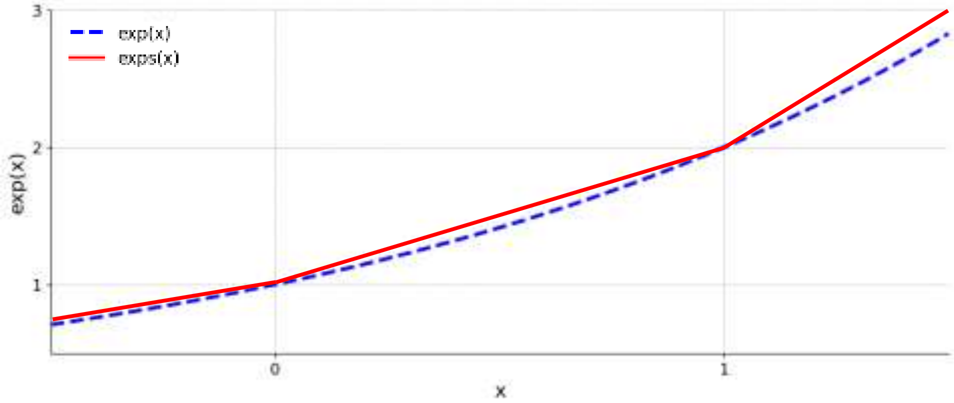
$$\text{value} = (-1)^{0} \cdot 2^{132-127} \cdot (1 + 0.25) = 2^{5} \cdot 1.25$$

$$2^{x} \approx 2^{\text{int}(x)} \cdot \big(1 + \text{frac}(x)\big) \coloneqq \text{exps}(x)$$



Improving Schraudolph's Accuracy

- **In the original paper, a constant is added to the result, shifting the function to minimize the average error**
  - Comes for free in software implementations
  - From now on we will use this function in software approximations



$$\mathrm{exps}(x) \coloneqq ax + (b - c)$$

- **We propose to enhance the accuracy of the approximation by processing the mantissa only**
  - We replace $\mathrm{frac}(x)$ with a polynomial $P(\mathrm{frac}(x))$ that approximates $2^{\mathrm{frac}(x)}$

$$2^{\mathrm{int}(x)} \cdot (1 + \mathrm{frac}(x)) \Rightarrow 2^{\mathrm{int}(x)} \cdot (1 + P(\mathrm{frac}(x)))$$

Our Proposed Enhancement (1)

- **We define $P(x)$ as a piecewise second-order polynomial**
  - For $x \in [0,0.5)$ we sum a straight-line tangent to $2^x - 1$ in 0 with a parabola centered in 0
  - For $x \in [0.5,1)$ we do as before, but the functions are centered in 1
- **Can we simplify the second polynomial to make it look more like the first?**
  - YES, if we approximate $1 - x$ with it's one's complement

$$P(x) = \log 2 \cdot x + \alpha x^2$$

$$\Downarrow$$

$$P(x) = \alpha x \left(x + \frac{\log 2}{\alpha} \right)$$

$$P(x) = 2\log 2 \cdot x + 1 - 2\log 2 + \beta(1-x)^2$$

$$\Downarrow$$

$$P(x) = 1 - \beta(1-x)\left(x + \frac{2\log 2}{\beta} - 1 \right)$$

$$P(x) = 1 - \beta(1-x)\left(x + \frac{2\log 2}{\beta} - 1 \right)$$

$$\Downarrow$$

$$P(x) = \text{not}\left(\beta\,\text{not}(x) \cdot \left(x + \frac{2\log 2}{\beta} - 1 \right) \right)$$

Our Proposed Enhancement (2)

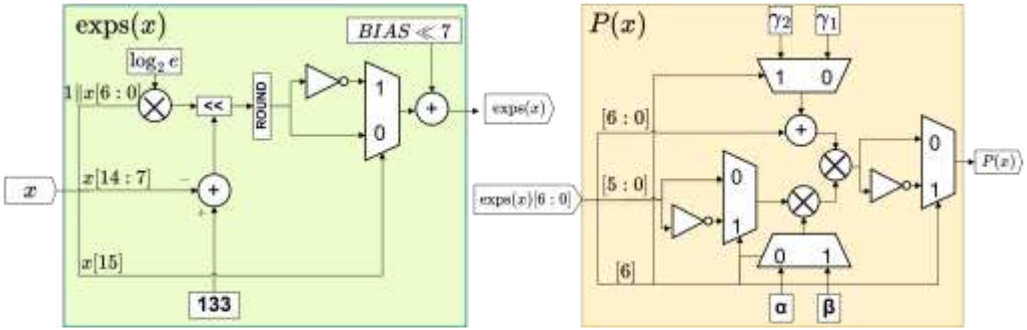
- **In practice, we replace the additive factors with 2 free parameters (γ_1,γ_2) and optimize them independently**

$$P(x) \doteq \alpha x\,(x + \gamma_1)\,, \qquad x \in [0, 0.5)$$
$$P(x) \doteq \mathrm{not}\big(\beta\,\mathrm{not}(x) \cdot (x + \gamma_2)\,\big), \qquad x \in [0.5, 1)$$

- **We minimize the error introduced by the approximation using a Montecarlo procedure**

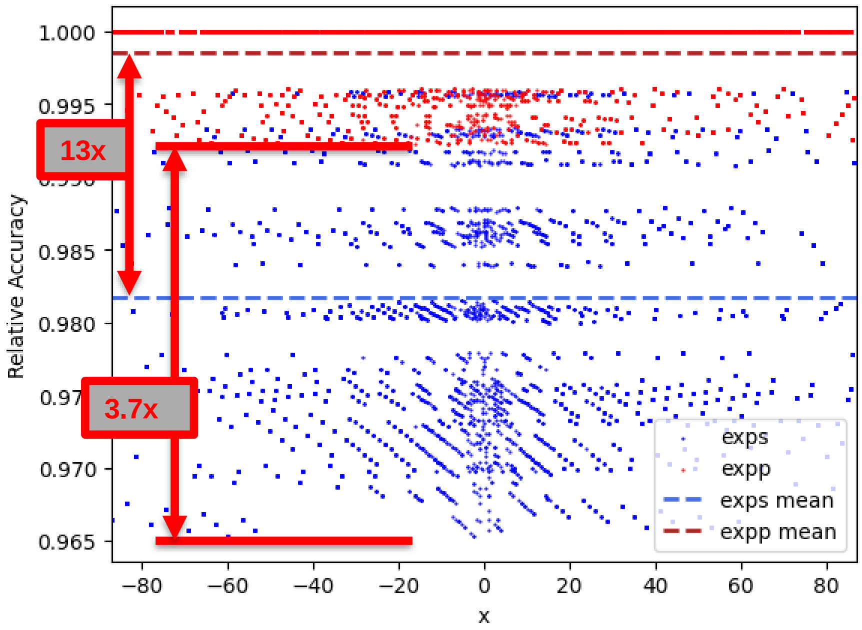
$\alpha = 0.21875 \quad \beta = 0.4375 \quad \gamma_1 = 3.296875 \quad \gamma_2 = 2.171875$

- **Very low final parameter bit width**
  - 4 bits for α and β
  - 8 bits for γ_1 and γ_2



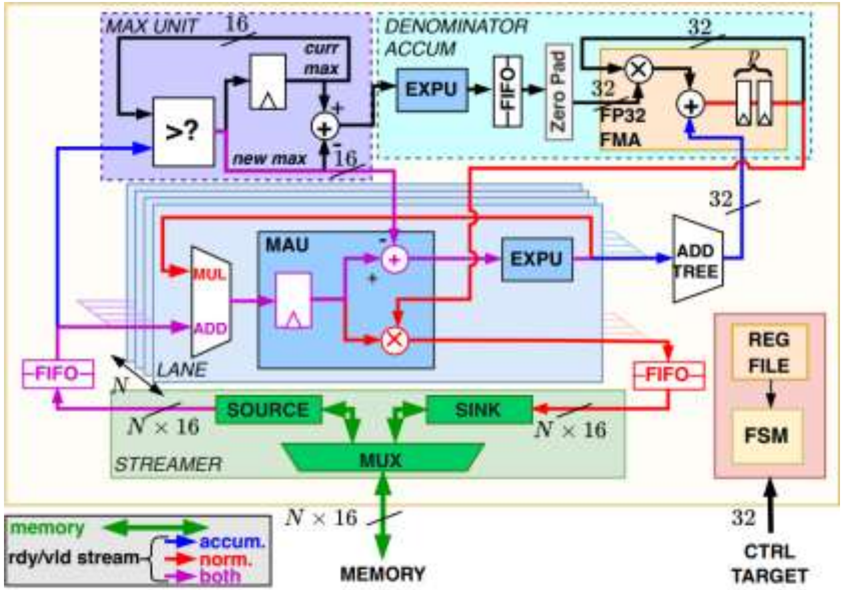
Accuracy Evalutation

- **Average relative error of 0.14%**
  - 13× decrease compared to Schraudolph's method
- **Relative accuracy of 99.86%**
- **The relative error is no greater than 0.78%**
  - 3.7× decrease compared to Schraudolph



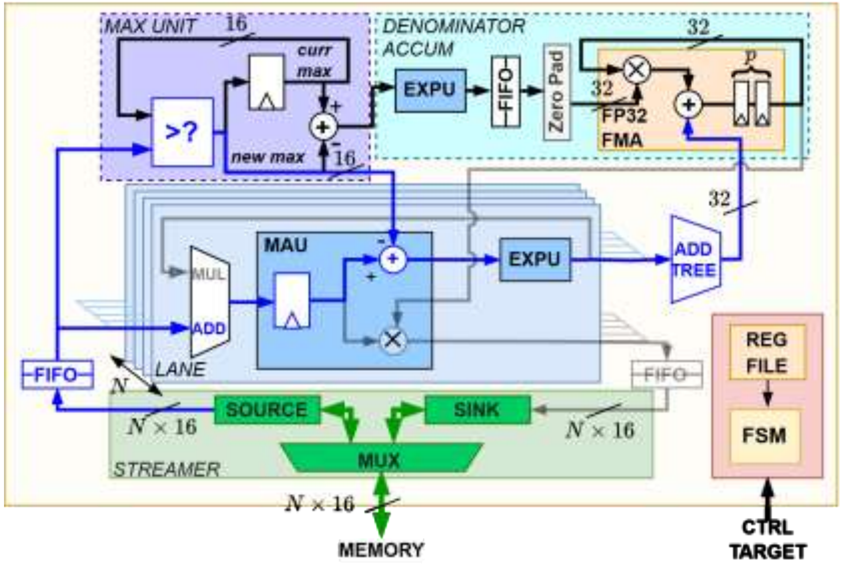
SoftEx

- **Parametric accelerator of softmax on BFloat16 vectors**
- **Organized as an HWPE**
- **The datapath features:**
  - N lanes containing a Multiplication and Addition Unit (MAU) and an Exponential Unit (EXPU)
  - an Accumulator module containing a single pipelined FP32 Fused Multiply-Add (FMA) unit
- **Softmax is split into: Accumulation, Inversion, and Normalization**



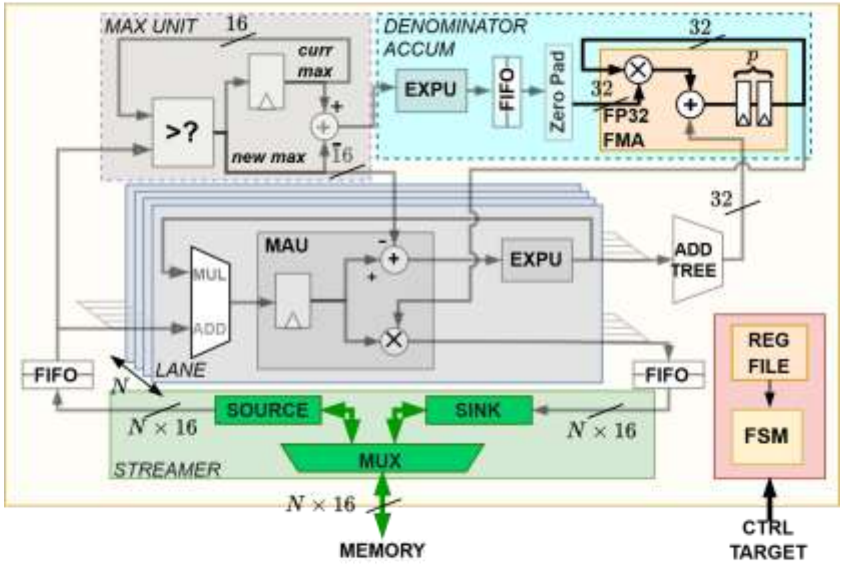
SoftEx – Accumulation

- **During this step we compute the denominator of softmax**
- **Each cycle N inputs are read, subtracted the maximum score, exponentiated and pushed into the accumulator**
- **To avoid a maximum search, we use an online normalization scheme**
  - Each score is subtracted the current maximum score
  - When the maximum is updated, all partial sums in the accumulator are rescaled by $e^{\text{oldmax}-\text{newmax}}$



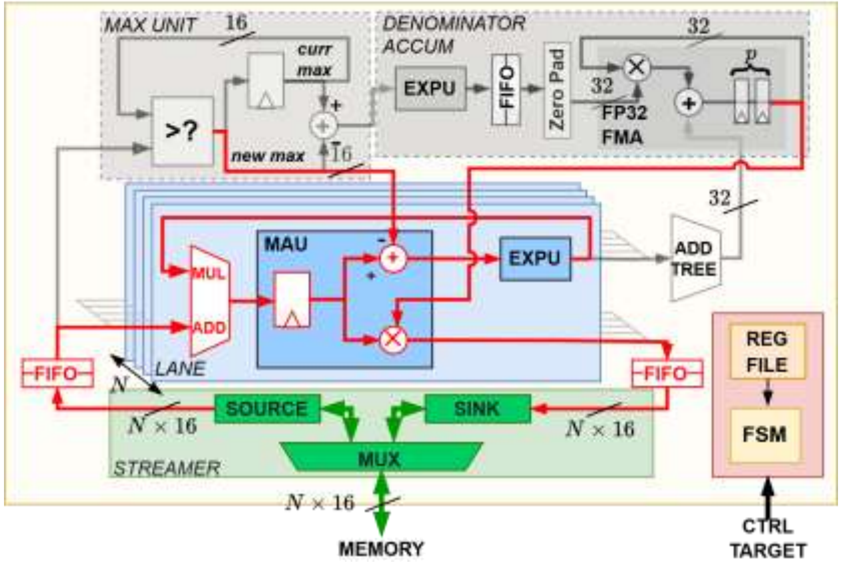
SoftEx – Inversion

- **Once all the scores have been read, we move to the Inversion step**
- **First, all partial sums in the accumulator are summed together**
- **Then, the reciprocal is computed using 2 Newton-Raphson iterations**
- **How do we choose the initial estimate?**
  - The exponent of the reciprocal can be computed exactly as $2\ \mathrm{BIAS} - 1 - \mathrm{EXP}$
  - The mantissa is estimated with the parabola $\frac{1}{2}(1 - \mathrm{MANT})^2$



SoftEx – Normalization

- **In the final step, the vector is read again and the exponentiated values are multiplied by the reciprocal of the denominator**
- **The MAUs are used to both subtract the maximum input and normalize the outputs**
  - To fully utilize the available memory bandwidth during accumulation and normalization, the load of a new vector of scores and the store of a vector of probabilities are alternated



Is Softmax Enough?

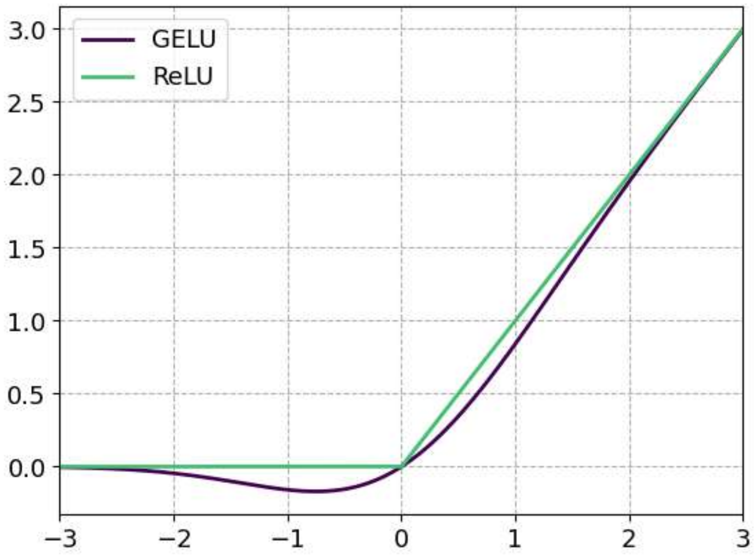
- If we look at the attention layer alone there are no major nonlinearities left
- However, there is still the feed-forward network…
- While the original Transformer employed ReLU, modern models forego this function in favor of more complex activation functions
- A commonly used function in high-performance models is GELU

Gaussian Error Linear Unit

- **GELU is an activation function consistently outperforming ReLU**
- **Instead of gating the input like ReLU, GELU weights the input by the value of the Guassian Cumulative Distribution Function**

$$\mathrm{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{1}{2}t^2\right) dt$$

- **Again, we need an approximation of this function!**



Calculating GELU in Practice

- **The original paper proposes 2 approximations**

$$\mathrm{GELU}(x) \approx x \cdot \frac{1}{2}\left(1 + \tanh\left(\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right)\right)$$

$$\mathrm{GELU}(x) \approx x \cdot \sigma(1.702x)$$

- **Both tanh and sig are based on exponentials! However…**

$$\tanh(x) \doteq \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$\sigma(x) \doteq \frac{1}{1 + e^{x}}$$

- **They both require a division of the terms, out of the question**
- **Are there other approximations solely based on exponentials and basic arithmetic?**

Φ as a Sum of Exponentials (1)

- **Let's focus on the complementary Gaussian CDF, the Q-function**

$$Q(x) \doteq 1 - \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_x^{+\infty} \exp\left(-\frac{1}{2}t^2\right) dt$$

- **An alternative formulation of the Q-function for positive arguments is:**

$$Q(x) = \frac{1}{\pi} \int_0^{\frac{\pi}{2}} \exp\left(-\frac{x^2}{2\sin^2(\theta)}\right) d\theta\,, \qquad x \geq 0$$

- **If we apply the rectangular integration formula as proposed by Chiani:**

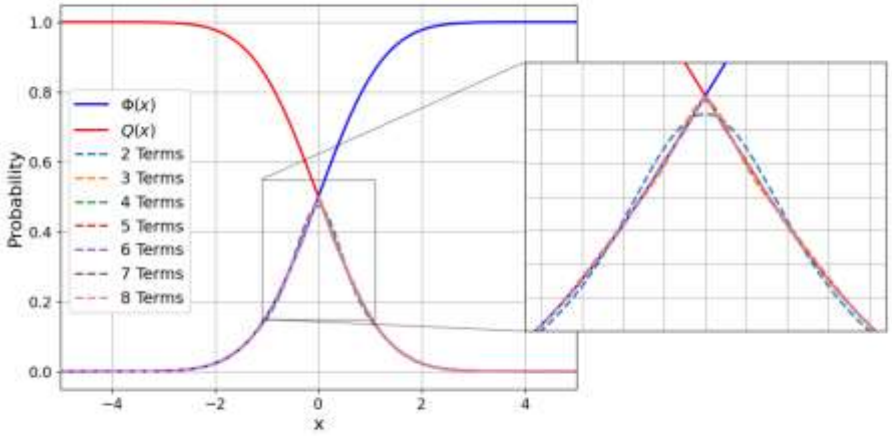
$$Q(x) \leq \frac{1}{\pi} \sum_{i=1}^{N} \int_{\theta_{i-1}}^{\theta_i} \exp\left(-\frac{x^2}{2\sin^2(\theta_i)}\right) d\theta = \sum_{i=1}^{N} a_i e^{-b_i x^2}\,, \qquad x \geq 0$$

- **We have an upper bound for Q (and Φ) expressed as a sum of exponentials!**
  - This formulation is symmetrical: for $x < 0$ it evaluates Φ
  - However, it is still an upper bound…

Φ as a Sum of Exponentials (2)

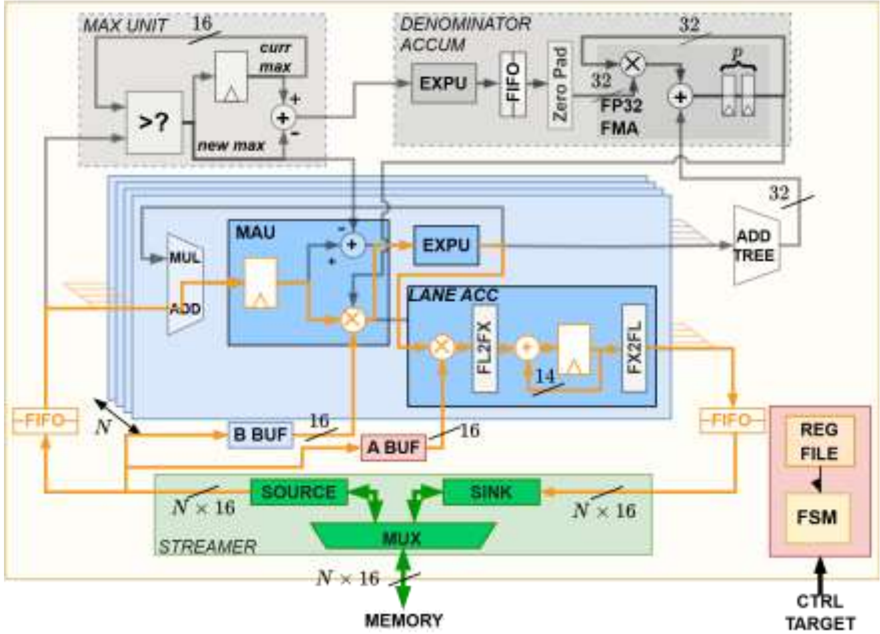
- **Can we turn Chiani's result it into an approximation?**
- **Yes, Tanash and Riihone propose a method to optimize the a and b parameters by solving an optimization problem**
  - Optimizes the relative error of the approximation for $x \leq x_{2N+1}$ given the rightmost extreme of the interval (x_{2N+1}) and the number of terms (N)
- **The resulting approximation is optimal in a minmax sense**
  - Also converges quickly!

$$
\begin{cases}
r'(x_k) = 0, & \text{for } k = 1, 2, ..., 2N, \\
r(x_k) = (-1)^{k+1} r_{\max}, & \text{for } k = 1, 2, ..., 2N, \\
\sum_{n=1}^{N} a_n = \frac{1}{2}, & \text{when } r(0) = 0, \\
\sum_{n=1}^{N} a_n = \frac{1}{2} - \frac{r_{\max}}{2}, & \text{when } r(0) = -r_{\max} \\
r(x_{2N+1}) = -r_{\max}.
\end{cases}
$$



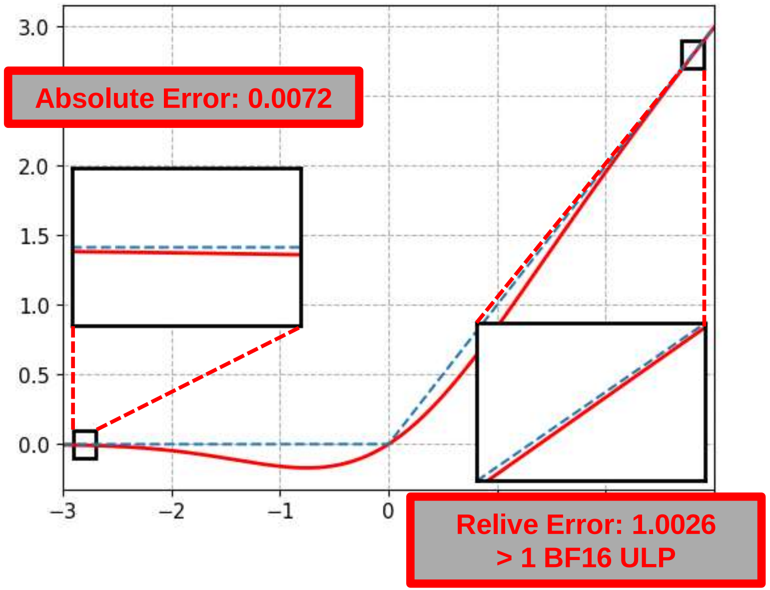
The Extended SoftEx

- **SoftEx accelerates only the sum of exponentials**
  - The remaining, simple steps are delegated to the cores
- **Little modifications compared to the softmax-only version**
  - 2 buffers for the a and b weights and accumulator per lane
- **The accumulators are NOT FMAs**
  - The accumulated value is bounded within the $(0,0.5]$ range, we can use fixed points
  - We just have to decide the number of bits to use for representing this value



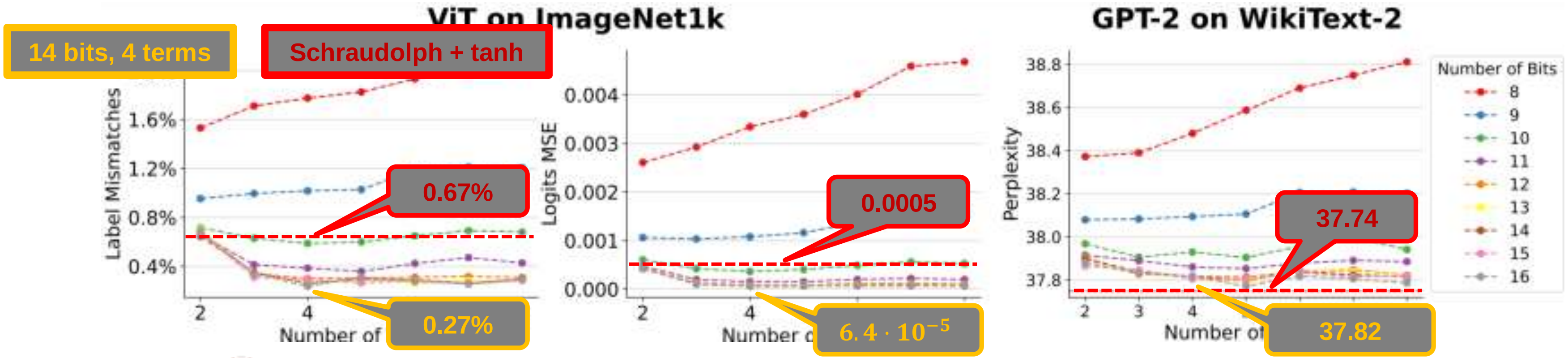
Where to Optimize the Function?

- **We need an approximation that is accurate near 0 and just good enough far from it**
  - After all, GELU behaves similarly to ReLU for $|x| \gg 0$
- **We solve the optimization problem for $|\boldsymbol{x}| \leq \mathbf{2.8}$**
  - For $x > 2.8$ the value of GELU in BF16 is exactly the value of the input
  - For $x < -2.8$ the value of GELU can be safely approximated with 0
- **Now we have to determine the optimal number of bits to use in the accumulator**



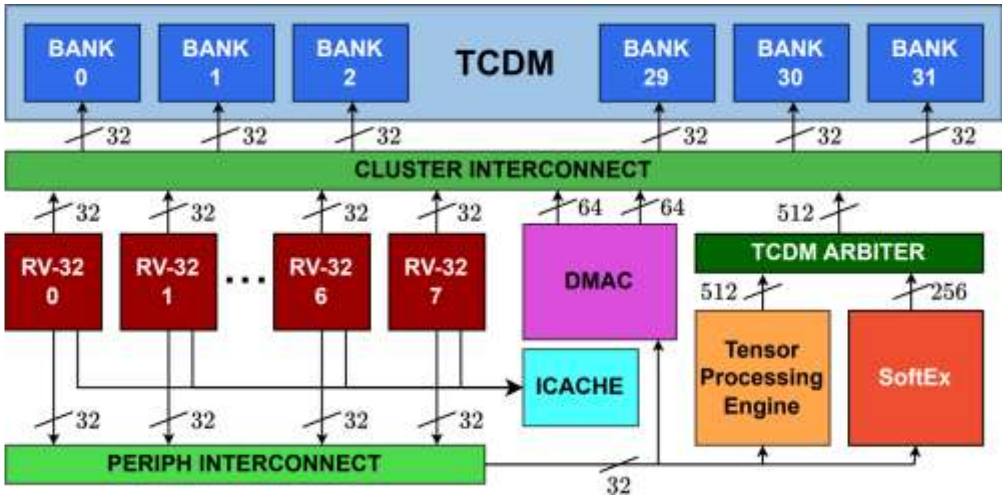
How Many Bits?

- **Swept both the number of bits and the number of terms and evaluated:**
  - Percentage of label mismatches and logits mean squared error on ViT on ImageNet1k
  - Perplexity on GPT2 on the WikiText benchmark
- **Using 10 or less bits results in significant deviations from the base models**
- **With 11 or more bits the deviations stabilize at around 4 terms**



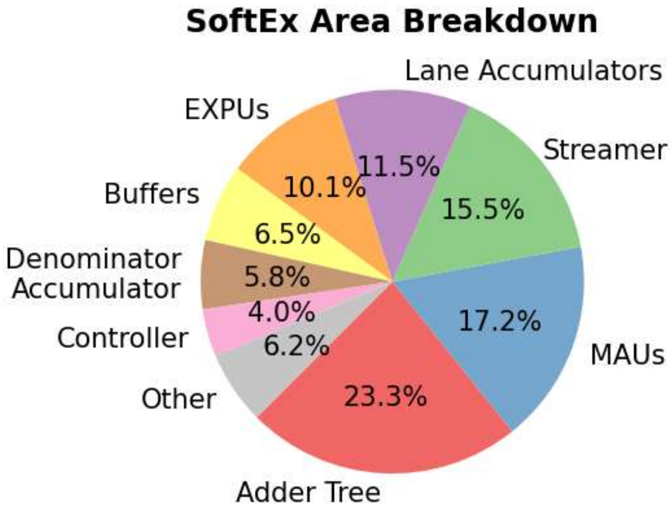
The Final Test System

- 8 RI5CY RISC-V cores with the Xpulp extension and private FPU
- 256KiB TCDM split among 32 banks
- 32 KiB of shared instruction cache
- RedMulE Tensor Processing Engine in 24x8 computing element configuration
- SoftEx softmax&GELU accelerator in 16 lanes configuration



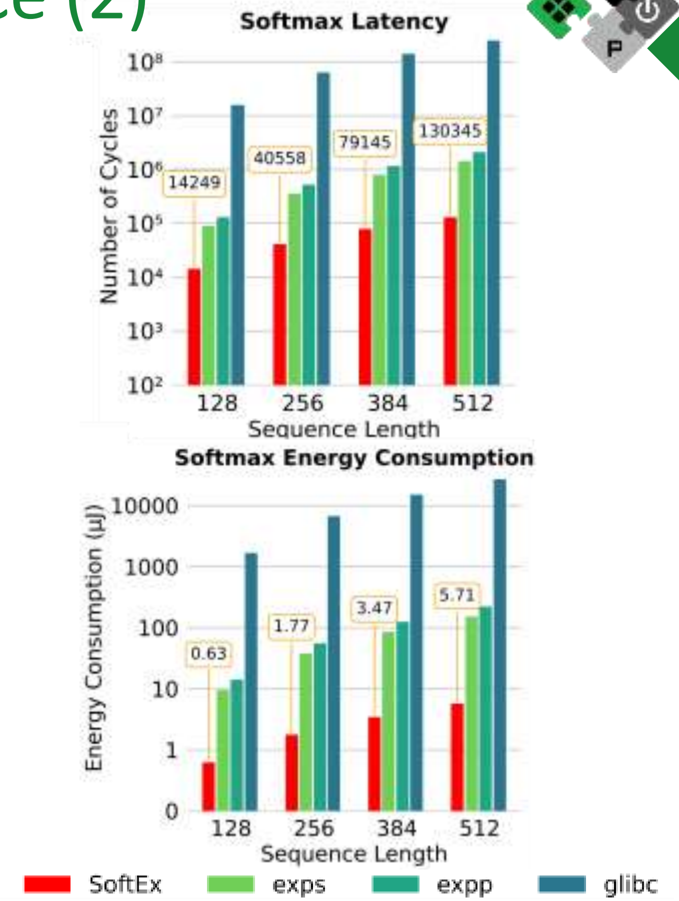
SoftEx – Area, Power and Performance (1)

- **Cluster implemented in GlobalFoundries 12LP+ technology**
- **Benchmarked typical conditions in 2 operating points:**
  - 0.8V and 1.12GHz for maximum performance
  - 0.55V and 460MHz for maximum efficiency
- **SoftEx area occupation: 0.039 mm^2**
  - 3.22% of the cluster area (1.21 mm^2)
  - 1/6 of RedMulE's area (0.24 mm^2)
- **SoftEx area dominated by the adder tree and MAUs**
  - Exponential units and accumulators account for only 10.1% and 11.5% of the total, respectively



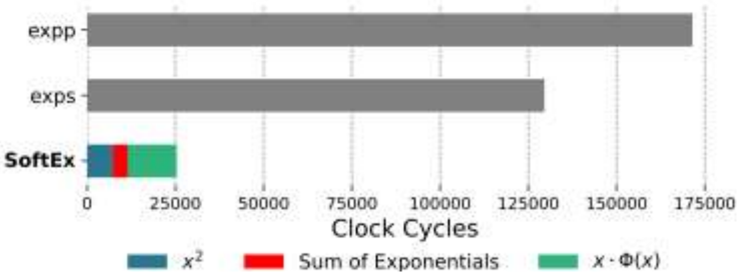
SoftEx – Area, Power and Performance (2)

- **Cluster power consumption during softmax:**
  - 278 mW @ 0.8V, 53.2 mW for SoftEx
  - 56.1 mW @ 0.55V, 9.87 mW for SoftEx
- **MAUs dominate the power consumption (24.2%)**
  - EXPUs only contribute by 13.7%
- **Benchmarked on activations from MobileBERT**
  - 6.2-10.8× faster compared to the 8 RISC-V cores using Schraudolph (*exps*)
  - 15.3-26.8× less power-hungry than the best software implementation
  - Software implementation of the algorithm (*expp*) on average 31% slower than *exps*



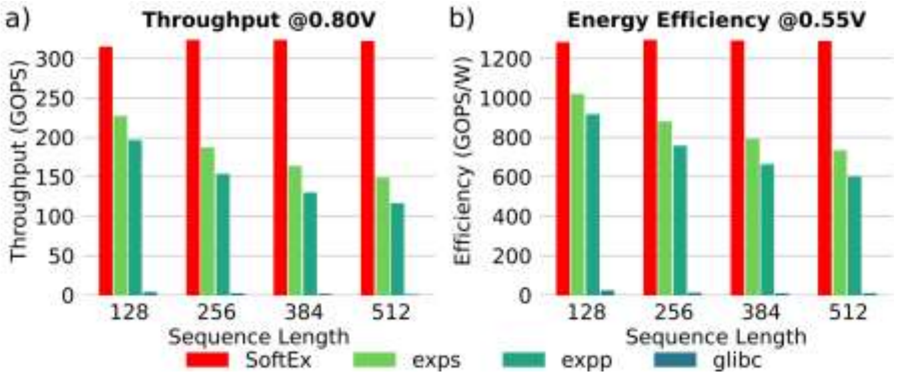
SoftEx – Area, Power and Performance (3)

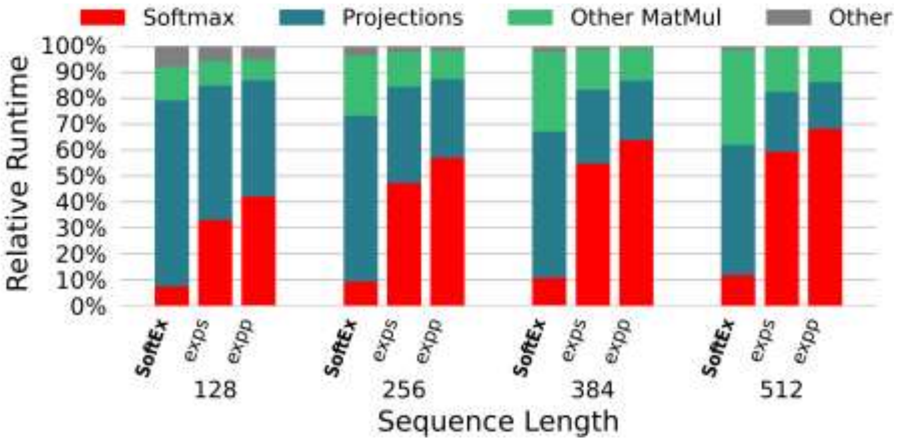
- **Cluster power consumption during sum of exp**
  - 276 mW @ 0.8V, 50.8 mW for SoftEx
  - 55.7 mW @ 0.55V, 9.46 mW for SoftEx
- **Accumulators dominate the power (22%) with the MAUs close behind (20%), higher EXPU contribution compared to softmax (16%)**
- **GELU benchmarked on ViT's FFN**
  - Software implementations use the sigmoid approximation
  - Φ approximated with a 4-term sum of exponentials
  - Even if partially performed in software, 5.11× speedup and a 5.29× higher energy efficiency compared to SW



Cluster Performance on MobileBERT's Attention

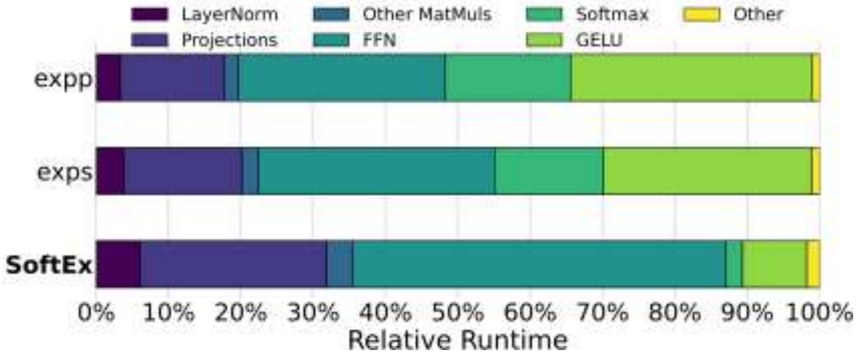
- **Bottleneck solely due to softmax**
- **Peak throughput of 324 GOPS**
  - 1.3-2.17× faster than the fastest software implementation
- **Peak energy efficiency of 1.30 TOPS/W**
  - 20.5-75.4% increment compared to the most efficient software implementation
- **Relative softmax runtime reduced by up to 4 times**

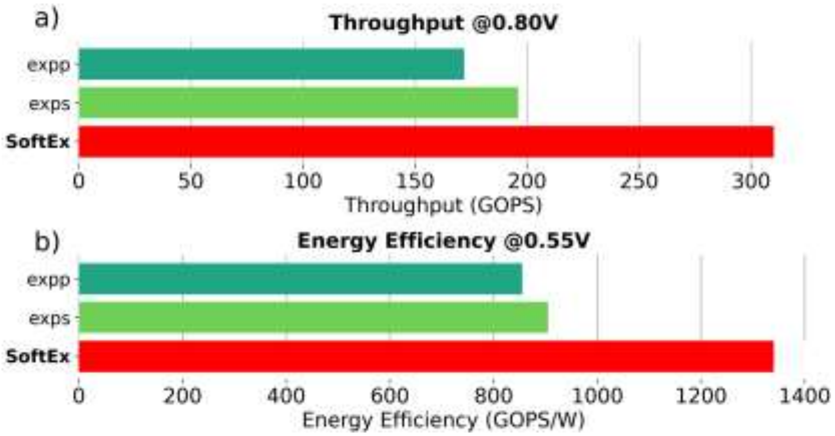




Cluster Performance on ViT

- **Bottleneck shared between softmax and GELU**
- **310 GOPS@0.8V**
  - End-to-end latency of 113 ms
  - 1.58× throughput increase on ViT base wrt software-only softmax & GELU
  - Using SoftEx-assisted GELU increases throughput by 1.30× wrt SW GELU
- **1.34 TOPS/W@0.55V**
  - 1.42× better efficiency compared to the approximate SW implementation





Conclusion

- **We presented a flexible acceleration template for Transformers at the edge, based on an 8-core RISC-V cluster augmented with:**
  - A 24×8 Computing Elements tensor processing engine
  - SoftEx, a novel accelerator for BFloat16 softmax and GELU non-linearities
- **Using SoftEx boost the system throughput by 1.58× and its energy efficiency by 1.42× on ViT**
  - 310 GOPS at 0.8V
  - 1.34 TOPS/W at 0.55V
- **SoftEx successfully achieves its design goal of alleviating the softmax and GELU bottleneck**

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

Andrea Belano andrea.belano2@unibo.it

Q&A

Institut für Integrierte Systeme – ETH Zürich
Gloriastrasse 35
Zürich, Switzerland
DEI – Università di Bologna
Viale del Risorgimento 2
Bologna, Italy

ETH zürich
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

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