Designing an Energy-Efficient Hardware Accelerator for Deep Learning

Kai Breese kbreese@ucsd.edu
Justin Chou jtchou@ucsd.edu
Katelyn Abille kabille@ucsd.edu
Lukas Fullner lfullner@ucsd.edu
Mentor: Rajesh Gupta rgupta@ucsd.edu

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Motivation

As modern AI systems grow in scale and complexity, their computational processes require increasingly large and unsustainable amounts of energy.

- ChatGPT required **564 MWh per day** (Feb 2023), with two days nearly amounting to the total 1,287 MWh used throughout the training phase. [3]
- Google reports **60% of ML energy use** goes to inference. [5]

Proposal

Building on the linear-complexity multiplication ($\mathcal{L}$-Mul) algorithm [4], **we designed a hardware accelerator to optimize floating-point operations to improve energy efficiency and computational speed in neural network inference, while maintaining strong accuracy.**

Background: Framework for Floating-Point Arithmetic

BF16 is widely adopted in ML for its ability to retain the same dynamic range as FP32 while reducing memory usage and computational requirements.

Bit Type	S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 1. BF16 sign, exponent, and mantissa breakdown

Floating Point Multiplication

To multiply two floats together, the standard is to use the following equation:

$$Mul(x,y) = (1 + x_m + y_m + x_m \cdot y_m) \cdot 2^{x_e + y_e}$$

Incorporating our sign bit, we have:

$$p = (s_1 \oplus s_2) \times 2^{(e_1 + e_2 - 2b)} \times (1 + m_1 + m_2 + m_1 \times m_2)$$

$$\begin{aligned} e_{\text{unbiased}} &= e_1 + e_2 - 2b \\ e_{\text{bits}} &= e_{\text{unbiased}} + b \\ e_{\text{bits}} &= e_1 + e_2 - b \end{aligned}$$

The Linear-Complexity Multiplication Algorithm ($\mathcal{L}$-Mul)

In comparison, $\mathcal{L}$-Mul eliminates the multiplication of $m_1 \times m_2$ by approximating it with a term $L(M)$ where M is the number of mantissa bits:

$$L(M) = \begin{cases} M, & M \leq 3 \\ 3, & M = 4 \\ 4, & M > 4 \end{cases}$$

$$\mathcal{L}\text{-Mul} = (s_1 \oplus s_2) \times 2^{(e_1 + e_2 - b)} \times (1 + m_1 + m_2 + L(M))$$

In practice, we can perform this algorithm by taking the XOR, performing a bitwise addition on the combined exponent and mantissa parts, adding the $L(M)$ term, and subtracting the bias bit shifted to the left to align with the exponent part. This enables us to skip the normalization step since the mantissa carry is automatically added to the exponent.

Methods: Using PyRTL, Yosys, and Openroad

- PyRTL streamlined development with **Pythonic Syntax** allows seamless integration with Python-based utility libraries, testing suites, and visualizations. [2]
- **Efficient simulation and optimization** in the framework's built-in features smoothened simulation and refinement.

- **Performance and power evaluation** were conducted with Yosys and Openroad to extract key performance metrics, including speed and power consumption, to assess our implementations.

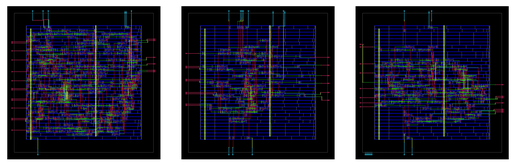


Figure 2. Left to right: adder, $\mathcal{L}$-Mul, multiplier pipelines

Methods: System Architecture

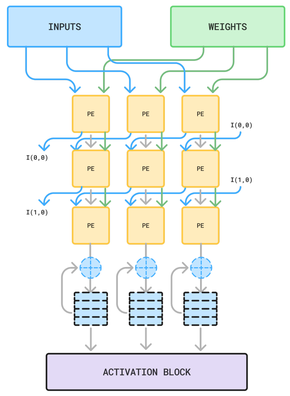


Figure 3. Systolic Array DiP Architecture [1]

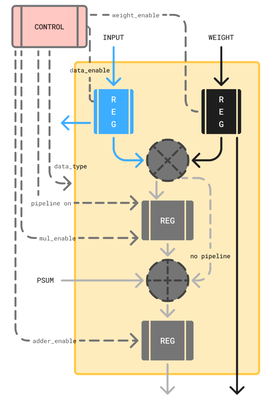


Figure 4. Processing Element [1]

Hardware Metrics

- The key metrics for our project are power, area, and delay. By lowering power requirements we lower costs and emissions, lowering delay increases model speed, and lowering area creates a smaller and cheaper physical chip.
- In optimizing our design, we tested multiple implementations of components such as the multipliers and L-mul modules to find the designs with the best performance in these key metrics.

Component-Level Accuracy

- L-mul increases speed by approximating; we're building out our hardware to test the speed/efficiency vs. accuracy trade-off. The paper claims that the error created by the algorithm is too small to have significant effects on outputs.
- In theory, lowering the precision of the model from FP32 to FP16 or FP8 would lead to errors, but in practice a standard model will not have a lower accuracy[6]

Comparative Analysis of Configurations

- Depending on the Neural Network, changing parameters such as the systolic array size, floating point format, and degree of pipelining will lead to better performance.
- A larger model will see speed increases with the pipelined modules, whereas a smaller model will be faster on the fast module.
- The model may require higher precision, usually due to outliers in the data, which would require a higher floating point format.
- One tradeoff to consider is the pipelined models are have greater delay, but allow for more data to be processed in parallel.

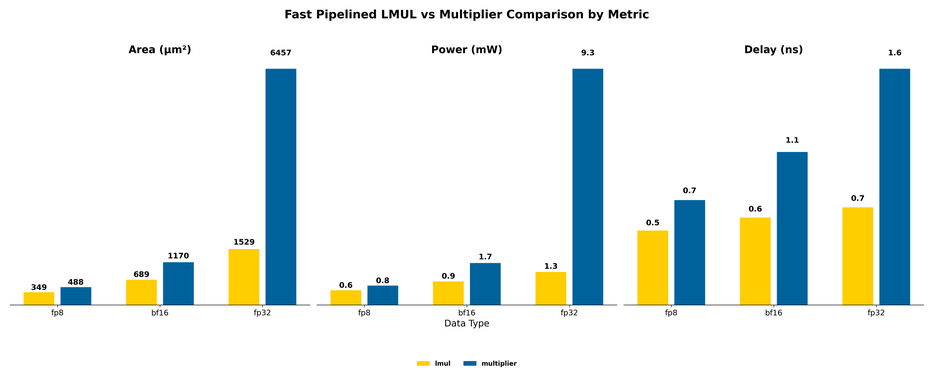
Multiplier	Weight Type	Activation Type	Accuracy (%)
Baseline	Float32	Float32	97.81
Baseline	Float8	BF16	97.43
Baseline	BF16	BF16	97.46
L-mul	Float8	BF16	96.89
L-mul	BF16	BF16	97.37

Table 1. Accuracy Results

Results

Design	fp8			bf16			fp32		
	Area (μm^2)	Power (mW)	Delay (ns)	Area (μm^2)	Power (mW)	Delay (ns)	Area (μm^2)	Power (mW)	Delay (ns)
L-mul Comb.	112.784	0.111	0.360	255.626	0.253	0.480	702.506	0.532	0.550
L-mul Pipelined	348.726	0.583	0.510	688.674	0.928	0.600	1529.230	1.300	0.670
Multiplier Comb.	347.396	1.055	1.290	1067.720	7.460	1.940	6311.910	133.398	2.850
Multiplier Pipelined	487.578	0.762	0.720	1169.600	1.654	1.050	6457.420	9.311	1.620
Multiplier Stage 2	162.260	0.161	0.550	552.482	1.184	0.910	4149.600	29.274	1.460
Multiplier Stage 3	71.820	0.027	0.230	134.064	0.048	0.290	319.466	0.080	0.420
Multiplier Stage 4	160.132	0.118	0.650	352.982	0.216	0.690	1253.660	0.553	1.070

Table 2. Area, Power, and Delay for Different Data Types (fp32, bf16, fp8)



Conclusion

When comparing the $\mathcal{L}$-Mul algorithm to the standard IEEE floating point multiplication algorithm, we saw significant jumps in all of our key metrics: area, power, and delay. Specifically, we saw increasing percentage improvements as we scaled up in data types (table 2). This makes sense when considering the time complexity of each algorithm – $\mathcal{L}$-Mul is an O(n) algorithm while IEEE is O(n^2), meaning the difference in operations should scale up as the number of bits increases.

We additionally found that the loss in accuracy (table 1) from IEEE to $\mathcal{L}$-Mul (0.092% for BF16) was extremely small compared to other key metrics (52% in area, 56% in power, 55% in delay for BF16). This is a strong indication that the hypothesis we were testing – that the speed/efficiency for accuracy tradeoff we were making would be worth it – indeed rang true.

Future Projections & Feasibility

- We see practical performance of L-mul in a small scale here, but we would like to evaluate its performance on larger and more complex models such as LMMs.
- As the models expand, we would be able to evaluate trends in memory usage and compute requirements.
- We would also like to scale up our power savings estimates to the size of a data center to see practical cost and power savings.
- Beyond the theoretical, we could tapeout our designs from a theoretical simulation to a physical chip.

References and Website

[1] Ahmed J Abdelmaksoud, Shady Agwa, and Themis Prodromakis. Dip: A scalable, energy-efficient systolic array for matrix multiplication acceleration. *arXiv preprint arXiv:2412.09709*, 2024.

[2] John Clow, Georgios Tzimpragos, Deeksha Dangwal, Sammy Guo, Joseph McMahan, and Timothy Sherwood. A pythonic approach for rapid hardware prototyping and instrumentation. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[3] Alex de Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023.

[4] Hongyin Luo and Wei Sun. Addition is all you need for energy-efficient language models. *arXiv preprint arXiv:2410.00907*, 2024.

[5] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7):18–28, 2022. doi: 10.1109/MC.2022.3148714.

[6] Mart van Baalen, Andrey Kuzmin, Suparna S Nair, Yuwei Ren, Eric Mahurin, Chirag Patel, Sundar Subramanian, Sanghyuk Lee, Markus Nagel, Joseph Soriaga, and Tijmen Blankevoort. Fp8 versus int8 for efficient deep learning inference, 2023. URL `https://arxiv.org/abs/2303.17951`.

github.com/ninjakaib/hardware-accelerators nakschou.github.io/hardware-accelerators-site/