



Exploring Memory-Efficient Execution Strategies for Long-Sequence Protein Structure Prediction

侯少騫 · Advisor: Prof. Aman Sinha · Institute of Integrated Circuit Design, National Sun Yat-sen University

Abstract

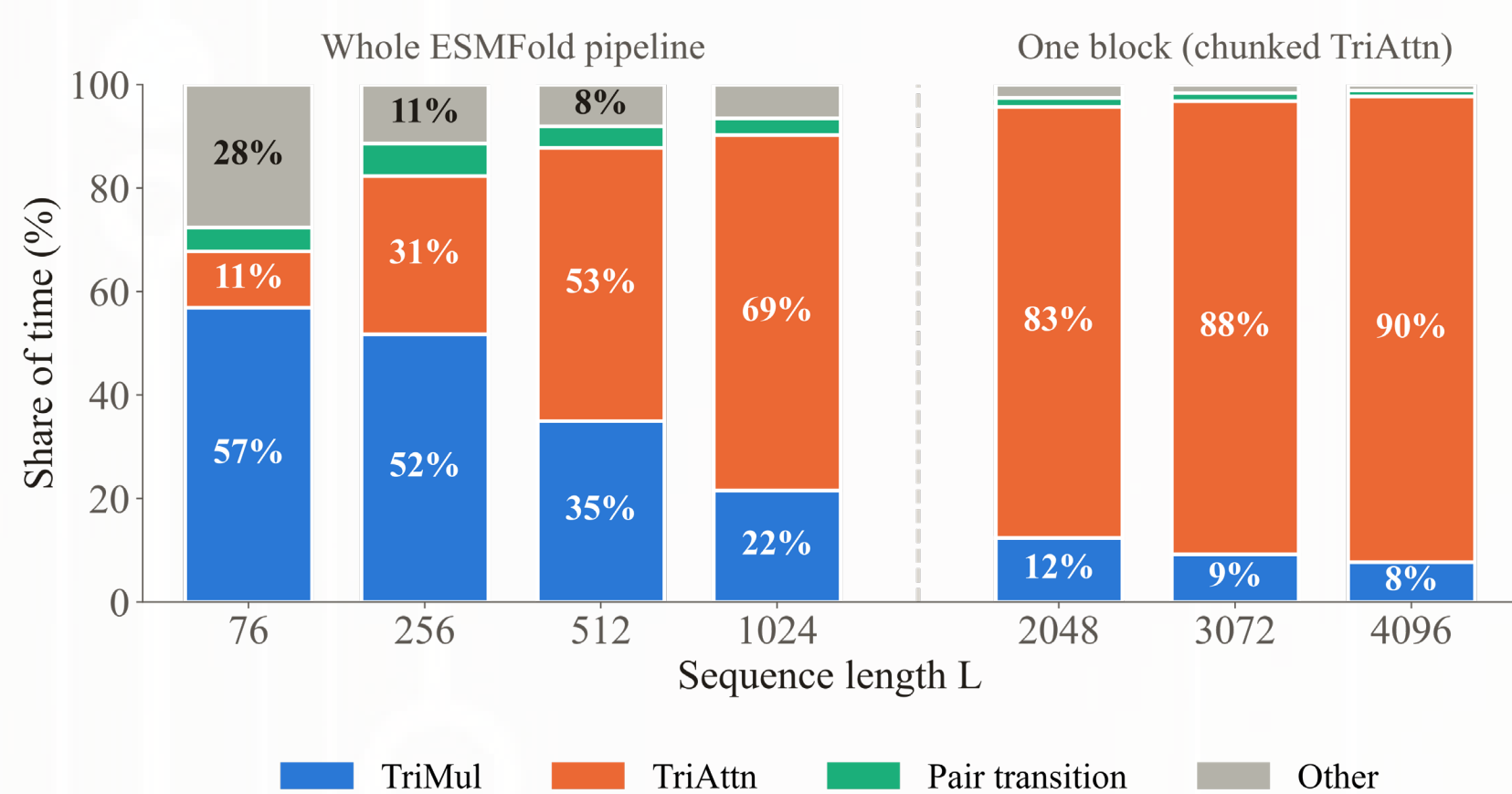
As the protein amino-acid sequences length grows, the computational and memory requirements for predicting 3D structures also increase. The key difference between PPMs and standard Transformers is that PPMs maintain a “pair representation” which describes the relationships between every pair of amino acids, this tensor shape is about (Ns,Ns,H_z). Therefore the increasing of memory consumption generally follows the sequence length N grows quadratically. Some attention score related computations can even increase cubically. The key bottleneck in PPMs is not primarily the model weights, but the intermediate activation. For a protein sequence of length 2034, activation memory is already 24.15 times larger than weight memory, require about 144GB, beyond the capacity of a single high-end GPU. Traditional chunking can reduce peak memory usage, but it just separate a big problem into small pieces, increase the data movement and

kernel calling, result in low speed of inference.

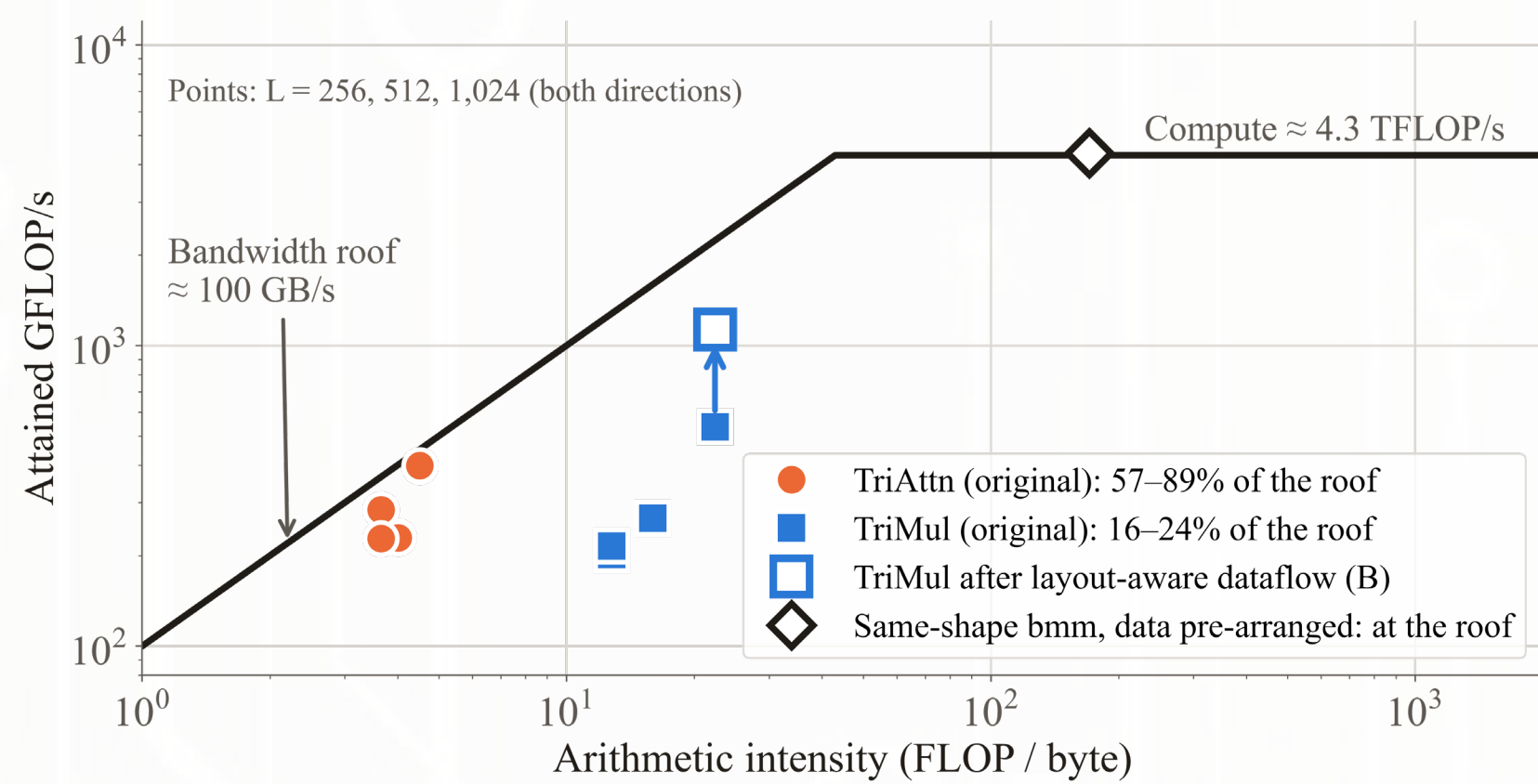
This project will investigate memory-efficient execution strategies for long-sequence PPMs. The initial step will profile Pair Representation operations and characterize their memory behavior under different sequence lengths. Based on this analysis, the project will evaluate chunking, reduced-precision activation storage, and tiled dataflows to reduce peak memory usage and off-chip memory traffic. Promising designs may later be prototyped on an available FPGA or high-bandwidth-memory platform. The goal is to identify a scalable hardware-software co-design direction for long-sequence protein structure prediction.

1 Introduction: Characterization & Bottleneck

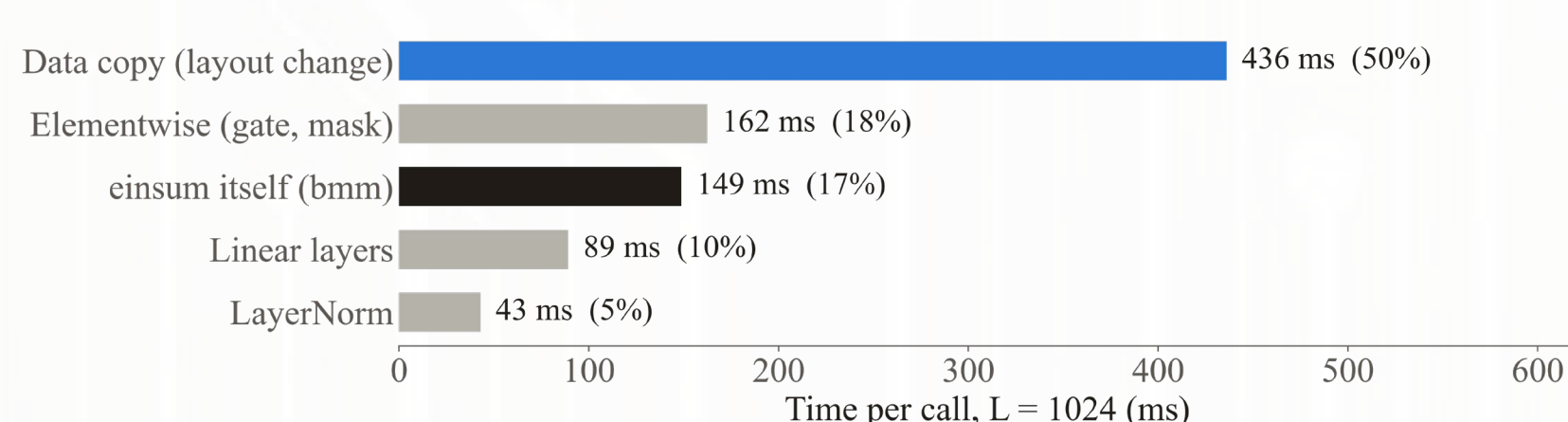
ESMFold’s [2] (an AlphaFold2-style [3] model) 48 folding blocks repeatedly update an $L \times L \times 128$ pair representation. We timed the full pipeline on a CPU server (FP32, $L = 76-1,024$) and one block up to $L = 4,096$.



- 1 Triangle operations take 82–90% of the whole pipeline ($L = 256-1,024$); **TriAttn**’s share grows with L.



- 2 **TriAttn** reaches 57–89% of the bandwidth roof; **TriMul** only 16–24% of its roof ($L = 256-1,024$).

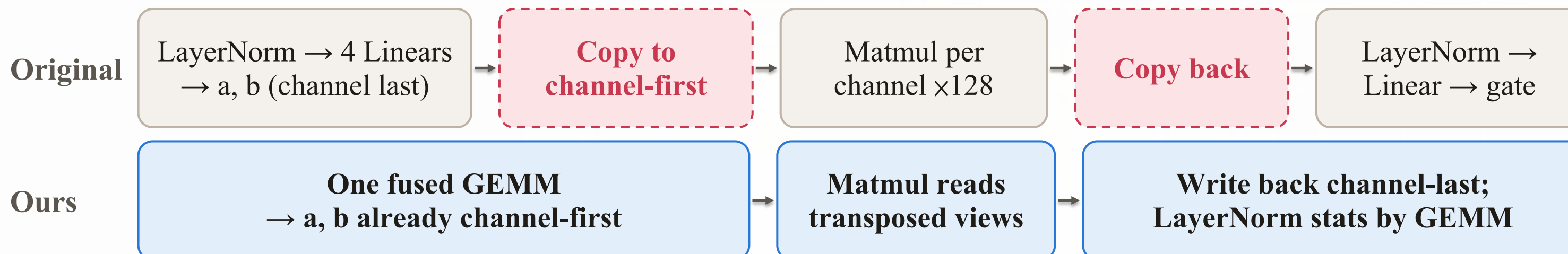


- 3 Half of **TriMul**’s time is spent copying data into the layout the matrix library expects.

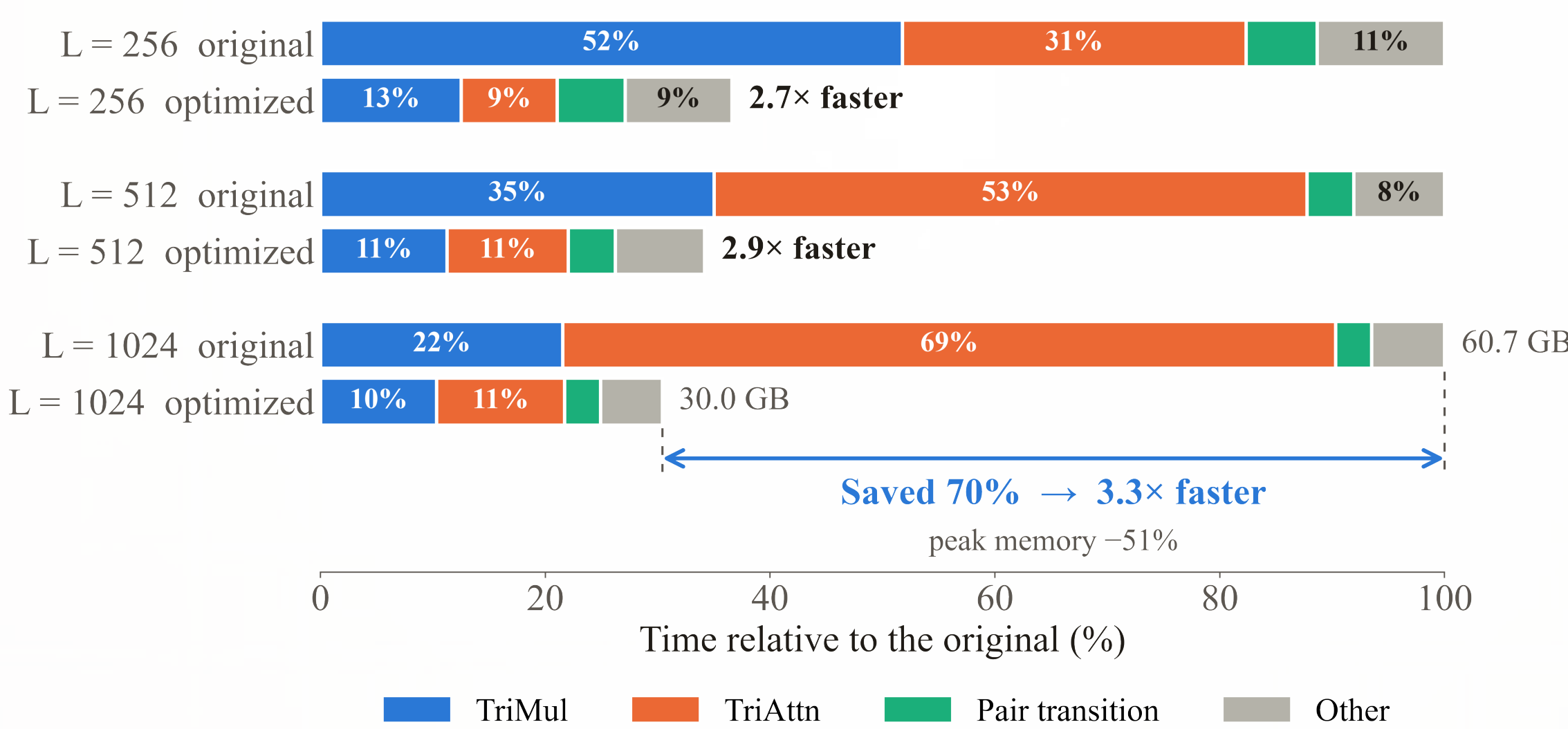
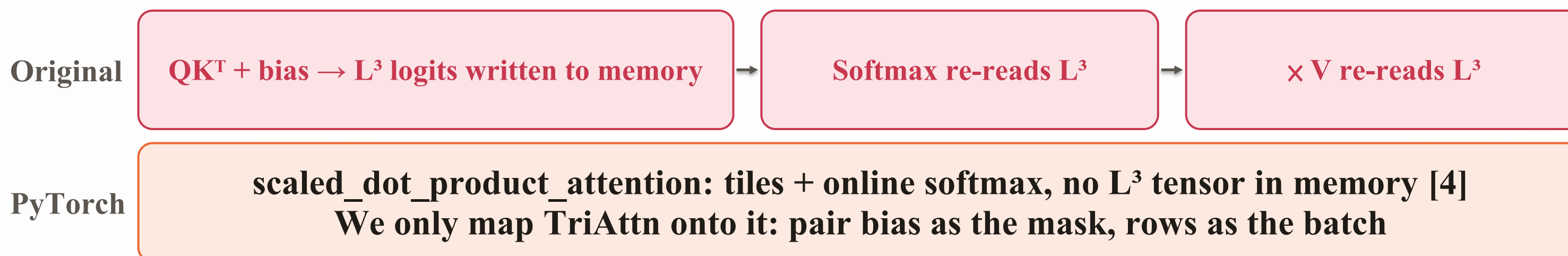
2 Design Changes I: Dataflow

TriMul’s new dataflow (A, B) is our design; TriAttn uses PyTorch’s existing kernel (E) as a control. The math is unchanged.

TriMul (our design, A + B: 2.1–4.3× faster per call, $L = 256-1,024$)



TriAttn (E, existing PyTorch kernel: 3.5–6.6× faster per call, $L = 256-1,024$)



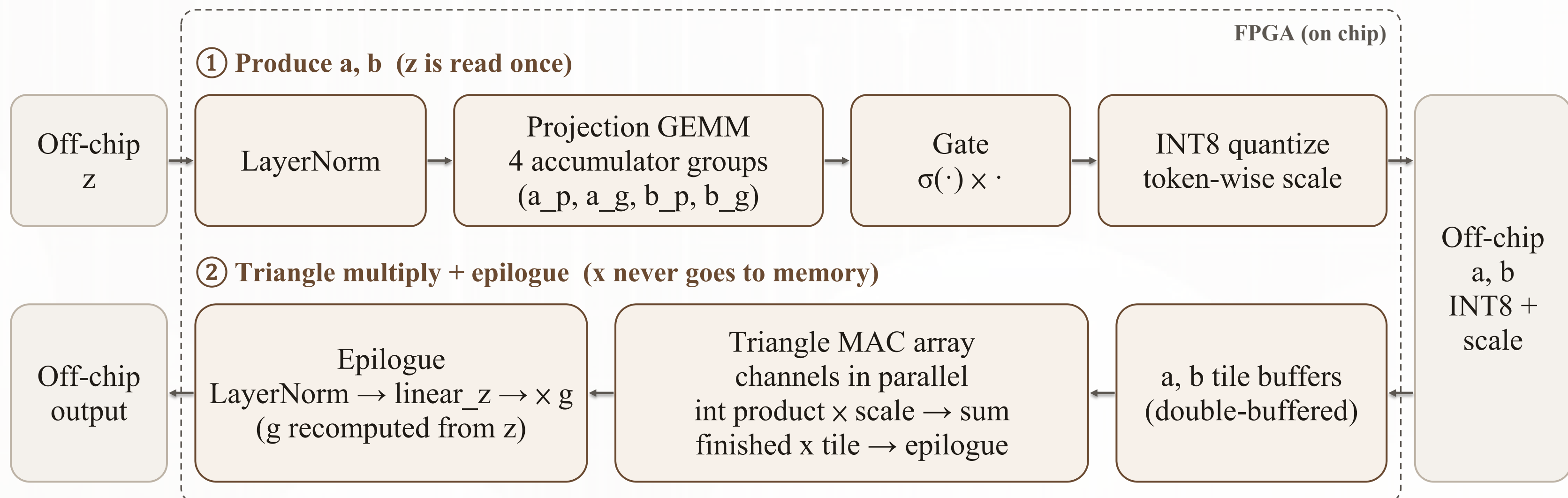
Time saved: 62% comes from TriMul (ours) at $L = 256$; 82% from TriAttn (PyTorch) at $L = 1,024$.

References

- [1] S. Han, S. Choi, J.-Y. Kim, “LightNobel: Improving Sequence Length Limitation in Protein Structure Prediction Model via Adaptive Activation Quantization,” ISCA 2025.
- [2] Z. Lin et al., “Evolutionary-scale prediction of atomic-level protein structure with a language model,” Science, 2023.
- [3] J. Jumper et al., “Highly accurate protein structure prediction with AlphaFold,” Nature, 2021.
- [4] T. Dao et al., “FlashAttention: Fast and memory-efficient exact attention with IO-awareness,” NeurIPS 2022.

3 Design Changes II: Why Hardware

Our software experiments show that TriMul’s cost is data movement and that changing the dataflow pays off. Yet on a general-purpose CPU, even after A + B + E, **TriMul** still takes about a third of the run time (measured: 33–34% at $L = 256-1,024$). We therefore propose two hardware directions (arrived at independently, and consistent with LightNobel’s [1] design):



- 1 **Layout set by the circuit.** The original spends about 50–58% of TriMul copying, and removing the copies alone (A) is 1.3–1.9× faster per call ($L = 512-2,048$, measured). A datapath built around the native [i][k][c] layout needs no reordering: one wide read feeds many channels.

- 2 **Intermediates kept on chip.** After B, 83% of TriMul’s time at $L = 1,024$ is outside the einsum, in the linear, gate and LayerNorm steps (measured). Their arithmetic needs only ~ 50 ms at the CPU’s peak (estimated); the rest is mostly reading and writing intermediates. Hardware chains them on chip, so only a and b go to off-chip memory.

a and b can also adopt LightNobel’s [1] token-wise quantization and be stored as INT8, cutting their traffic to $\frac{1}{4}$ of FP32.