ReTransformer

ReRAM-based Processing-in-Memory Architecture for Transformer Acceleration

Reza Adinepour

Amirkabir University of Technology (Tehran Polytechnic)

adinepour@aut.ac.ir

Computer Engineering Department

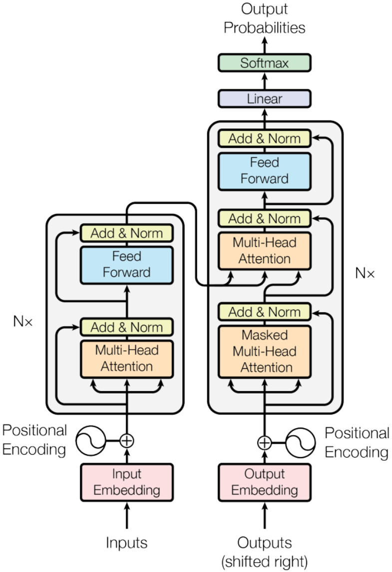
June 28, 2024

Presentation Overview

1. Introduction to Transformer Network
   - Why Use Transformers?
   - Strengths of Transformer
   - Weaknesses of Transformers
2. Motivation
3. ReRAM Concept
   - Vector-Matrix Multiplication (VMM)
   - Matrix-Matrix Multiplication (MatMul)
   - In-Memory Logic Operations
4. Overall Architecture
   - Key Features
   - Workflow
5. Experimental Setup
6. Results and Analysis

What is a Transformer Network?

1. Introduced in **"Attention is All You Need"** (2017)
2. Unlike traditional RNNs and LSTMs, Transformers **do not** process data sequentially but use a mechanism called **"self-attention"** to draw global dependencies between input and output.



Why Use Transformers?

1. **Parallelization:**
   Unlike RNNs, Transformers can process input data in parallel, leading to faster training times.
2. **Self-Attention Mechanism:**
   This allows the model to weigh the importance of different words in a sentence, capturing long-range dependencies more effectively.
3. **Scalability:**
   Transformers can be scaled up effectively, leading to improved performance with more data and larger models.
4. **Versatility:**
   They are used in various applications, from machine translation and text generation to image processing and more.

Strengths of Transformer

1. **Efficiency:**
   Due to parallel processing, they train faster on large datasets.
2. **Accuracy:**
   State-of-the-art performance in many tasks, particularly in NLP.
3. **Flexibility:**
   Applicable to a wide range of tasks beyond language, such as image and speech processing.
4. **Transfer Learning:**
   Pre-trained models like BERT and GPT can be fine-tuned for specific tasks with relatively small amounts of data.

Weaknesses of Transformers

1. **Resource-Intensive:**
   Require significant computational power and memory, especially for large models.
2. **Complexity:**
   More challenging to understand and implement compared to simpler models.
3. **Data Requirements:**
   Performance often hinges on the availability of large-scale datasets for pre-training.
4. **Inference Speed:**
   Performance bottlenecks during inference due to the scaled dot-product attention mechanism.

Motivation

In this paper:

1. Developed **ReTransformer, a ReRAM-based Processing-in-Memory (PIM) architecture** specifically designed to accelerate Transformer models.
2. Implemented **optimized MatMul operations** to reduce data dependency and intermediate result handling
3. Designed a hybrid softmax mechanism combining in-memory logic and look-up tables for efficient softmax calculations.
4. Introduced a **sub-matrix pipeline design** for better utilization of ReRAM crossbars and improved throughput.

Motivation (Cont.)

And Improvements Made is:

1. **Computing Efficiency:**
   23.21x improvement over GPU, 3.25x over PipeLayer.
2. **Power Consumption:**
   1086x reduction compared to GPU, 2.82x compared to PipeLayer.
3. **Latency Reduction:**
   1.32x for smaller models, 1.16x for larger models.
4. **Softmax Efficiency:**
   32% lower power consumption compared to traditional CMOS-based designs.
5. **Throughput Enhancement:**
   1.18x increase in computational throughput.

Motivation (Cont.)

This concept use in CNNs and RNNs. but we can't and cant be directly applied to Transformer due to the following reasons:

1. **Matrix-Matrix Multiplication:**
   Transformers require frequent matrix-matrix multiplications, causing potential slowdowns and reduced efficiency due to intermediate result storage.
2. **Different Computations:**
   Unlike CNNs, Transformers use scaled dot-product attention, necessitating different computational approaches.
3. **Finer Pipeline Granularity:**
   Transformer accelerators need a more detailed pipeline design compared to the layer-level granularity used in previous designs.

ReRAM Concept

ReRAM Basics:

1. Is a Non-volatile memory with:
   - High density
   - Low access energy
   - And support for multi-level cell and 3D integration
2. ReRAM-based Vector-Matrix & Matrix-Matrix Multiplication:
   - Vector-Matrix Multiplication (VMM)
   - Matrix-Matrix Multiplication (MatMul)
3. In-Memory Logic Operations:
   - NOR Logic
   - XOR Logic
   - And other Logic Operations

Vector-Matrix Multiplication (VMM)

1. Conductance of ReRAM cells represents elements in a matrix.
2. Input voltage vector ($V_I = [v_0, v_1, v_2, v_3]$) fed to word lines (WLs) generates output current through bit lines (BLs).
3. One read cycle completes the VMM operation.

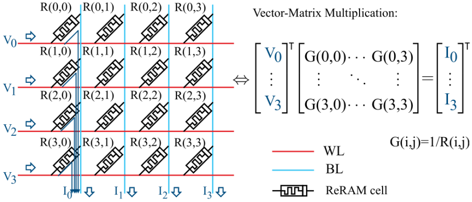


Figure: ReRAM-based vector-matrix multiplication

According to Kirchhoff's law the output current calculate as bellow:

$$i_j = \sum_{i=0}^{3} \frac{v_i}{R(i,j)} = \sum_{i=0}^{3} v_i G(i,j)$$

Matrix-Matrix Multiplication (MatMul)

1. Input matrix separated into vectors for sequential VMM operations
2. Results of VMM operations combined to obtain MatMul results

In-Memory Logic Operations

❶ **NOR Logic:**
Uses high and low conductance values to represent logic states. Operations performed using specific voltage settings.

❷ **XOR Logic:**
Implemented using a combination of OR and NAND operations, leveraging ReRAM's programmable conductance.

❸ **Other Logic Operations:**
INV and OR implemented in one or two cycles respectively.

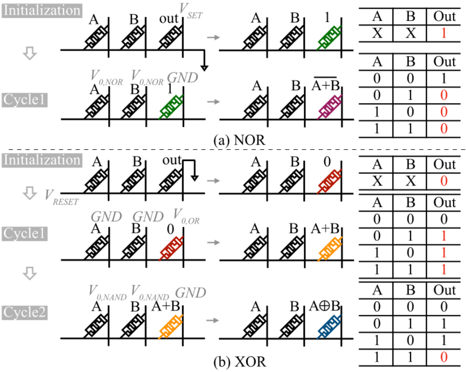


Figure: ReRAM-based in-memory logic: (a) NOR, (b) XOR.

Overall Architecture

A ReRAM-based PIM module is divided into three types of functional components:

❶ **Processing Subarrays:** Execute computations such as MatMul and feed-forward operations.

❷ **Buffer Subarrays:** Serve as caches to store intermediate data and results.

❸ **Memory Subarrays:** Store original input data and final output results.

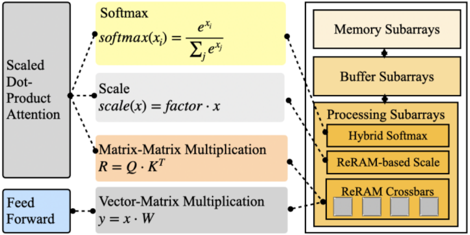


Figure: Overview of the proposed ReRAM-based PIM design for Transformer

Key Features

1. **ReRAM Crossbars:** Enable efficient in-memory computations.
2. **Optimized MatMul:** Reduces data dependency and intermediate writes.
3. **Hybrid Softmax Mechanism:** Combines in-memory logic with look-up tables for efficiency.
4. **Sub-Matrix Pipeline:** Slices input matrices for parallel processing.

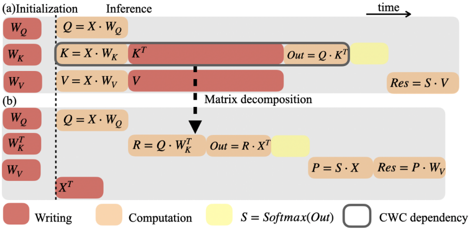


Figure: Remove data dependency in scaled dot-product attention layer: (a) a CWC dependency caused by the in- termediate result K. (b) The optimized MatMul eliminates the CWC dependency by decomposing the computation into two cascaded multiplications.

Workflow

1. **Data Loading:**
   Input data is stored in memory subarrays.
2. **Computation:**
   Processing subarrays perform operations using data from buffer subarrays.
3. **Intermediate Handling:**
   Buffer subarrays temporarily store intermediate results, reducing the need for frequent memory writes.
4. **Output Storage:**
   Final results are stored back in memory subarrays.

Experimental Setup

1. **Configurations:**
   - GPU: NVIDIA TITAN RTX, 24GB memory, 672 GB/s bandwidth.
   - ReTransformer: ReRAM crossbar arrays, 2-bit cell precision, 128x128 subarray size.
2. **Evaluation Metrics:**
   - Computing Efficiency: Operations per second per watt.
   - Power Consumption: Total power used during computations.

MatMul Optimization:
Reduces computation latency by 1.32x for Model A and 1.16x for Model B.

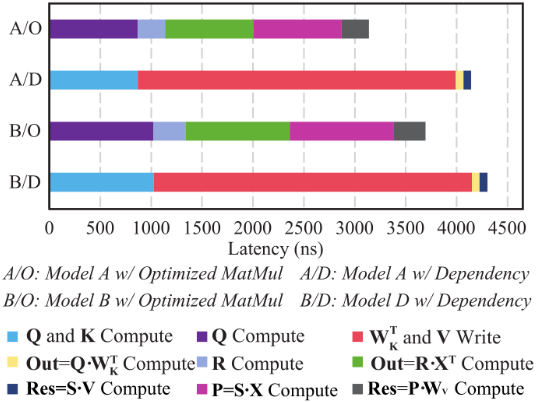


Figure: MatMul computation latency comparisons

Hybrid Softmax Efficiency:
Lowers power consumption by 32% compared to CMOS-based design.

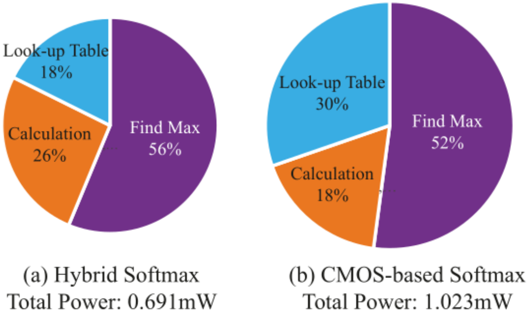


Figure: Softmax design comparison

Results and Analysis (Cont.)

Pipeline Performance:
Finer granularity pipeline improves throughput by 1.18x for both models.

Model	Layer	Finer	Improvement
Model A	69.24 GOPs/s	81.85 GOPs/s	1.18x
Model B	67.89 GOPs/s	80.07 GOPs/s	1.18x

Table: Performance comparison of two pipeline designs

Results and Analysis (Cont.)

Overall Comparison:

1. ReTransformer achieves 23.21x improvement in computing efficiency and 1086x reduction in power consumption compared to GPU.
2. Compared to PipeLayer, ReTransformer improves computing efficiency by 3.25x and reduces power by 2.82x.

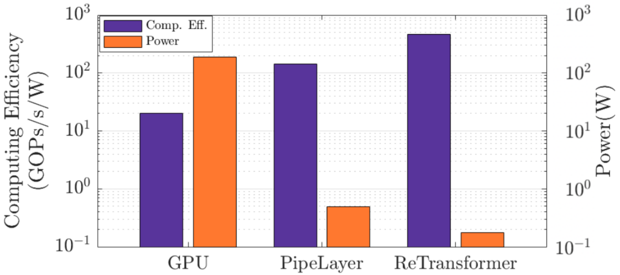


Figure: Performance comparison with GPU and PipeLayer

The End

Questions? Comments?

You can find this slides here:

github.com/M-Sc-AUT/M.Sc-Computer-Architecture/Memory Technologies