DEEP LEARNING JP
[DL Papers]

Scalable MatMul-free Language Modeling

Rui-Jie Zhu et al., University of California, Santa Cruz

Alfredo Solano, Matsuo Laboratory

Overview

- Introduction
- Method
- Experiments
- FPGA
- Conclusion

References:

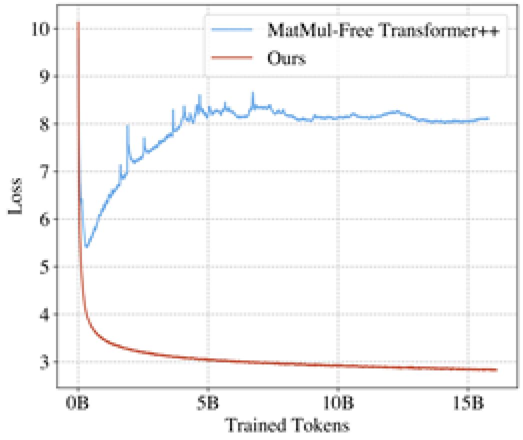
– Paper: https://arxiv.org/pdf/2406.02528
– Code: https://github.com/ridgerchu/matmulfreellm/tree/master

Introduction

- Motivation
  - General matrix-matrix multiplication (GEMM) is a dominant operation in neural networks
  - It has O(N^3) time complexity in the worst case
    - twice the input results in eight times the cost
  - GPUs are designed for this
    - hardware lottery
  - Used in both training and inference
  - If GEMM use could be reduced / simplified, training and inference time should be reduced

Method

- To simplify GEMM:
  - replace with simpler operations
    - AdderNet replaces multiplication with addition in CNNs
  - use binary / ternary quantization
    - binary: w = { 0, 1 }
    - ternary: w = { -1, 0, 1 }
    - Binary activations in Spiking Neural Networks (SNN)
    - Binary / Ternary weights in Binary / Ternary Neural Networks (BNN / TNN)
  - BitNet showed it is possible to scale binarized transformers to 3B
    - Replace Linear with BitLinear
    - Keep standard attention
      - Q, K calculated dynamically
      - require custom CUDA kernels for optimization
  - When testing a ternary quantization of Bitnet attention layers, it failed to converge



Method (2)

- Replace linear layers BitLinear layers with ternary values
  - multiplication is replaced by addition and negation
- Replace weight matrix W values with w = { -1, 0, 1 }
  - GEMM is then conceptually similar to:
    - `np.where(x==1) - np.where(x==-1)`
- Not efficient enough, use custom kernel
- Optimize memory access with fused root mean squared normalization (RMSNorm) and quantization of activations
  - reduce HMB I/O costs
  - memory size is already reduced by ternary values

$$\widetilde{\mathbf{Y}}_i = \sum_{j=1}^{u} x_j \widetilde{\mathbf{W}}_{ij} = \sum_{j:\widetilde{\mathbf{W}}_{ij}=1} x_j - \sum_{j:\widetilde{\mathbf{W}}_{ij}=-1} x_j, \quad \text{for } i = 1, 2, \ldots, m$$

Method (3)

Algorithm 1 Fused RMSNorm and BitLinear Algorithm with Quantization

Define FORDWARDPASS($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \epsilon$)
  $\mathbf{X} \in \mathbb{R}^{M\times N}, \mathbf{W} \in \mathbb{R}^{N\times K}, \boldsymbol{b} \in \mathbb{R}^{K}$

  function forward_pass($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \epsilon$)
    Load $\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \epsilon$ from HBM
    On Chip: $\widetilde{\mathbf{Y}}, \mu, \sigma^2, r \leftarrow \texttt{rms\_norm\_fwd}(\mathbf{X})$
    On Chip: $\widetilde{\mathbf{W}} \leftarrow \texttt{weight\_quant}(\mathbf{W})$
    On Chip: $\mathbf{O} \leftarrow \widetilde{\mathbf{Y}} \circledast \widetilde{\mathbf{W}} + \boldsymbol{b}$
    Store $\mathbf{O}, \mu, \sigma^2, r$ to HBM
    return $\mathbf{O}, \mu, \sigma^2, r$

  function rms_norm_fwd($\mathbf{X}$)
    $\mu, \sigma^2 \leftarrow \texttt{mean}(\mathbf{X}), \texttt{variance}(\mathbf{X})$
    $r \leftarrow \frac{1}{\sqrt{\sigma^2+\epsilon}}$
    $\widetilde{\mathbf{Y}} \leftarrow \texttt{activation\_quant}(r(\mathbf{X}-\mu))$
    return $\widetilde{\mathbf{Y}}, \mu, \sigma^2, r$

  function activation_quant($\mathbf{X}$)
    $s \leftarrow \frac{127}{\texttt{max}(|\mathbf{X}|)}$ $\triangleright$ $\lfloor\cdot\rceil\,|_{\cdot}$ means round then clamp
    $\widetilde{X} \leftarrow \lfloor s\mathbf{X} \rceil\,|_{[-128,127]} \cdot \frac{1}{s}$
    return $\widetilde{X}$

  function weight_quant($\mathbf{W}$)
    $s \leftarrow \frac{1}{\texttt{mean}(|\mathbf{W}|)}$
    $\widetilde{\mathbf{W}} \leftarrow \lfloor s\mathbf{X} \rceil\,|_{[-1,1]} \cdot \frac{1}{s}$
    return $\widetilde{\mathbf{W}}$
  return $\mathbf{O}$

Define BACKWARDPASS($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \mathbf{O}, \mathrm{d}\mathbf{O}, \mu, \sigma^2, r$)
  $\mathbf{X} \in \mathbb{R}^{M\times N}, \mathbf{W} \in \mathbb{R}^{N\times K}, \boldsymbol{b} \in \mathbb{R}^{K}$
  $\mathbf{O} \in \mathbb{R}^{M\times K}, \mathrm{d}\mathbf{O} \in \mathbb{R}^{M\times K}$

  function backward_pass($\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \mathbf{O}, \mu, \sigma^2, r, \mathrm{d}\mathbf{O}$)
    Load $\mathbf{X}, \mathbf{W}, \boldsymbol{b}, \mathbf{O}, \mu, \sigma^2, r, \mathrm{d}\mathbf{O}$ from HBM
    On Chip: $\mathrm{d}\mathbf{Y} \leftarrow \mathrm{d}\mathbf{O} \times \mathbf{W}^\top$
    On Chip: $\mathbf{dX}, \widetilde{\mathbf{Y}} \leftarrow \texttt{rms\_norm\_bwd}(\mathrm{d}\mathbf{Y}, \mathbf{X}, \mu, \sigma^2, r)$
    On Chip: $\mathrm{d}\mathbf{W} \leftarrow \mathrm{d}\mathbf{O}^\top \times \widetilde{\mathbf{Y}}$
    On Chip: $\mathrm{d}\boldsymbol{b} \leftarrow \texttt{sum}(\mathrm{d}\mathbf{O})$
    Store $\mathbf{dX}, \mathrm{d}\mathbf{W}, \mathrm{d}\boldsymbol{b}$ to HBM
    return $\mathbf{dX}, \mathrm{d}\mathbf{W}, \mathrm{d}\boldsymbol{b}$

  function rms_norm_bwd($\mathrm{d}\mathbf{Y}, \mathbf{X}, \mu, \sigma^2, r$)
    $\widetilde{\mathbf{Y}} \leftarrow \texttt{activation\_quant}(r(\mathbf{X}-\mu))$
    $\mathrm{d}\sigma^2 \leftarrow \texttt{sum}(\mathrm{d}\mathbf{Y} \times (\mathbf{X}-\mu) \times -0.5 \times r^3)$
    $\mathrm{d}\mu \leftarrow \texttt{sum}(-r\mathrm{d}\mathbf{Y}) + \mathrm{d}\sigma^2 \times \texttt{mean}(\mathbf{X}-\mu)$
    $\mathbf{dX} \leftarrow r\mathrm{d}\mathbf{Y} + 2\mathrm{d}\sigma^2(\mathbf{X}-\mu)/N + \mathrm{d}\mu/N$
    return $\mathbf{dX}, \widetilde{\mathbf{Y}}$

Method (4)

- Replace attention with other mechanism
- View the transformer as:
  - token mixer: sequence / temporal information: self-attention, mamba, etc.
  - channel mixer: embedding / spatial information: feed-forward, GLU, etc.
- For the token mixer:
  - ternarize Q, K matrices to get a ternary attention map
    - fails to converge
  - replace self-attention with a modified gated recurrent unit (GRU)
    - simpler RNN-based architecture
    - replaces GEMM with element-wise operations and accumulation

Method (5)

Standard GRU

$$\boldsymbol{r}_t = \sigma\left(\boldsymbol{x}_t \mathbf{W}_{xr} + \boldsymbol{h}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r\right) \in \mathbb{R}^{1\times d}, \tag{1}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{x}_t \mathbf{W}_{xf} + \boldsymbol{h}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f\right) \in \mathbb{R}^{1\times d}, \tag{2}$$

$$\boldsymbol{c}_t = \tanh\left(\boldsymbol{x}_t \mathbf{W}_{xc} + (\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) \mathbf{W}_{cc} + \mathbf{b}_c\right) \in \mathbb{R}^{1\times d}, \tag{3}$$

$$\boldsymbol{h}_t = \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{f}_t) \odot \boldsymbol{c}_t \in \mathbb{R}^{1\times d}, \tag{4}$$

$$\boldsymbol{o}_t = \boldsymbol{h}_t \tag{5}$$

Method (6)

- MatMul-free GRU
  - remove hidden-state related weights (Wcc, Whr, Whf)
  - remove activation between hidden states (tanh)
  - enables parallel computation
  - add a data-dependent gate between hidden state and output
  - decouple candidate state from hidden state

Method (7)

MatMul-free GRU

$$
\begin{aligned}
\boldsymbol{f}_t &= \sigma\left(\boldsymbol{x}_t \circledast \mathbf{W}_f + \mathbf{b}_f\right) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{c}_t &= \tau\left(\boldsymbol{x}_t \circledast \mathbf{W}_c + \mathbf{b}_c\right) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{h}_t &= \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{f}_t) \odot \boldsymbol{c}_t \in \mathbb{R}^{1\times d}, \\
\boldsymbol{g}_t &= \sigma(\boldsymbol{x}_t \circledast \mathbf{W}_g + \mathbf{b}_g) \in \mathbb{R}^{1\times d}, \\
\boldsymbol{o}'_t &= \boldsymbol{g}_t \odot \boldsymbol{h}_t \in \mathbb{R}^{1\times d}, \\
\boldsymbol{o}_t &= \boldsymbol{o}'_t \circledast \mathbf{W}_o + \mathbf{b}_o \in \mathbb{R}^{1\times d}.
\end{aligned}
$$

Method (8)

- For the channel mixer
  - use a gated linear unit (GLU), similar to latest LLMs like Llama, Mistral, etc.
    - uses only dense layers
  - make it use BitLinear layers

Experiments

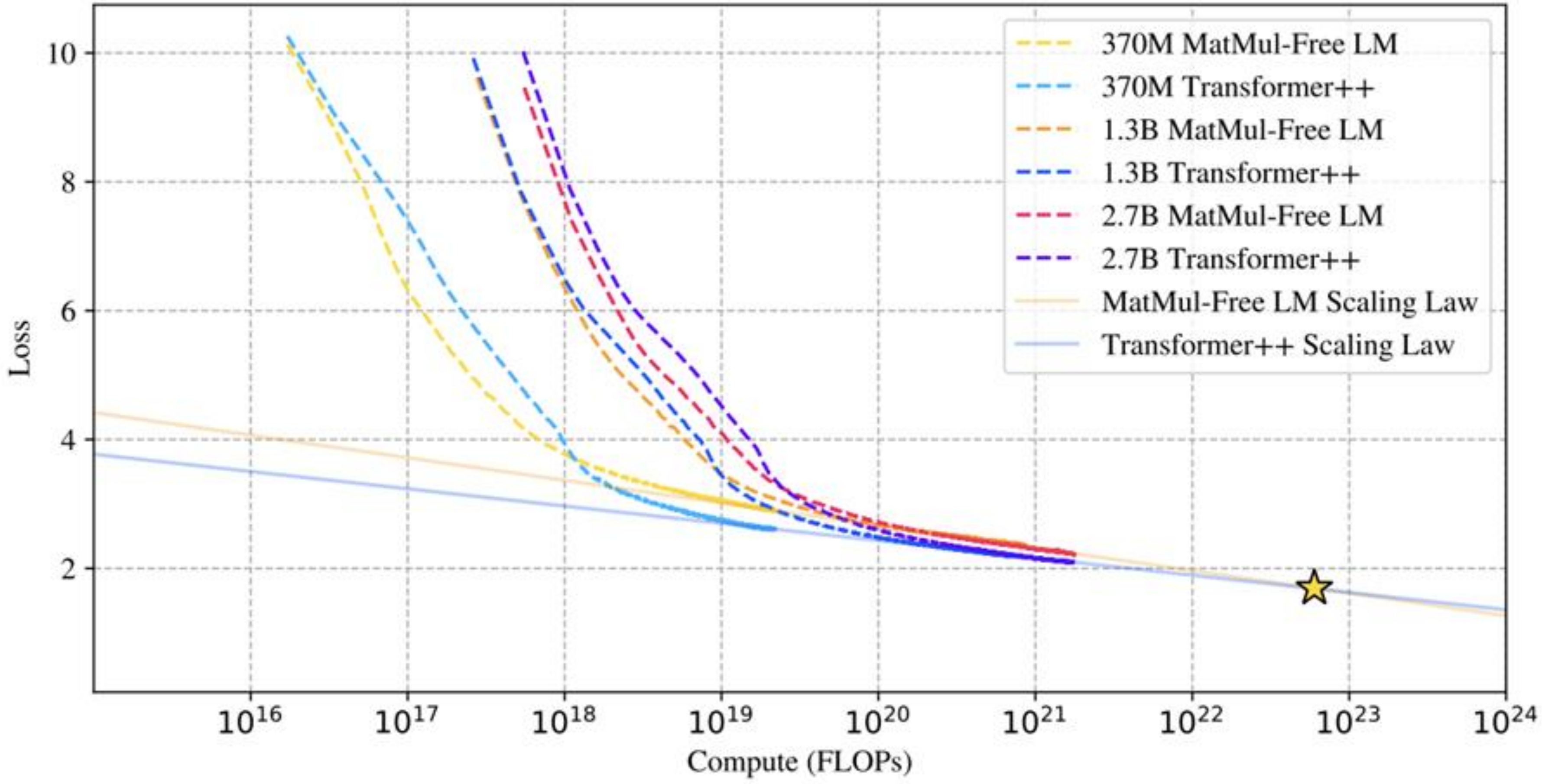
- Training details
  - use a surrogate gradient to handle non-differentiable functions like sign, clip, etc.
    - via Straight-Through Estimator
  - larger learning rate than traditional transformers
    - small LRs may lead to no weight updates after clipping
  - learning rate scheduler
    - shows different learning dynamics
    - cosine scheduler
    - halve midway through

Experiments (2)

- Compare against advanced transformer architecture from Llama 2
  - named Transformer++ in the charts
  - MatMul-free
- Three model sizes: 370M, 1.3B, and 2.7B
- All models pre-trained on the SlimPajama dataset
  - 370M model trained on 15 billion tokens, and the 1.3B and 2.7B models trained on 100 billion
- x8 NVIDIA H100 GPUs
  - ~5 hours for the 370M model
  - ~84 hours for the 1.3B model
  - ~173 hours for the 2.7B model

Experiments (3)

Loss graph



Experiments (4)

- Loss curve initially better for MatMul-free
- Then is taken over by Transformer++
- Scaling projections seem to indicate a steeper descent
  - more efficient resource usage
  - projected to intersect at 10^23 flops (similar to Llama 3 8B)
    - but only 3 data points
- Downstream tasks
  - multiple benchmarks: ARC-Challenge, Hellaswag, Winogrande, etc.
  - zero-shot
  - results show competitive performance

Experiments (5)

Downstream tasks

Table 1: Zero-shot accuracy of MatMul-free LM and Transformer++ on benchmark datasets.

Models	**Size**	**ARCe**	**ARCc**	**HS**	**OQ**	**PQ**	**WGe**	**Avg.**
370M parameters with 15B training tokens, Layer=24, d=1024								
Transformer++	370M	45.0	24.0	34.3	29.2	64.0	49.9	41.1
MatMul-free RWKV-4	370M	44.7	22.8	31.6	27.8	63.0	50.3	40.0
Ours	370M	42.6	23.8	32.8	28.4	63.0	49.2	40.3
1.3B parameters with 100B training tokens, Layer=24, d=2048								
Transformer++	1.3B	54.1	27.1	49.3	32.4	70.3	54.9	48.0
MatMul-free RWKV-4	1.3B	52.4	25.6	45.1	31.0	68.2	50.5	45.5
Ours	1.3B	54.0	25.9	44.9	31.4	68.4	52.4	46.2
2.7B parameters with 100B training tokens, Layer=32, d=2560								
Transformer++	2.7B	59.7	27.4	54.2	34.4	72.5	56.2	50.7
Ours	2.7B	58.5	29.7	52.3	35.4	71.1	52.1	49.9

Experiments (6)

- Training efficiency
  - Vanilla BitLinear compared to Fused BitLinear
  - Fused operator benefits from larger batch sizes
    - faster speed: 25.6% speedup for the 1.3B
    - reduced memory: 61% reduction for the 1.3B
    - more samples are being processed in a time step
- Inference efficiency
  - MatMul-free LM compared to Transformer++
  - Lower memory usage and latency
    - 4.9 GB vs 48.5 GB for the 1.3B
    - 695 ms vs 3184 ms for the 1.3B

FPGA

- Field-programmable gate array
  - configurable integrated circuit (IC)
  - lower level than GPU, higher than ASIC (application specific)
- To test efficiency on hardware that supports ternary operators
- Programmed in Verilog
- Deployed on Intel Cloud

FPGA (2)

Verilog RTl

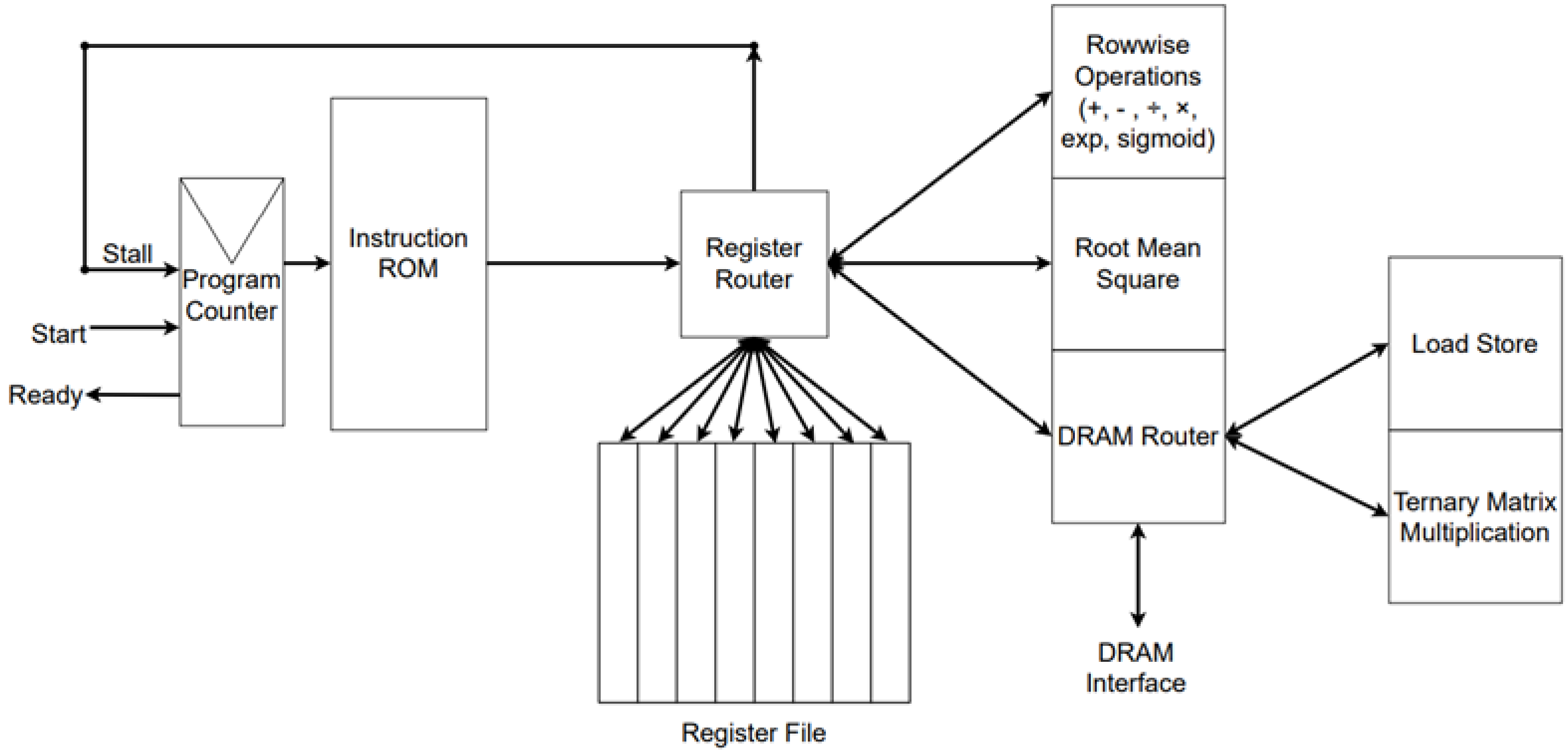


Figure 5: RTL implementation for running MatMul-free token generation

FPGA (3)

- Clock rate of 60Hz
- Around 13W
- Implemented single core, estimate the multi-core setting based on that
- 1.3B model projected at 42ms and 23.8 tokens/second
  - human reading speed
  - low power consumption

Conclusion

- MatMul-free models are feasible
- Performance can be comparable to standard transformers
  - reduces memory usage and latency
- GPUs are optimized for GEMM though, custom hardware may be needed
- Code is available on GitHub:
  - https://github.com/ridgerchu/matmulfreellm
  - compatible with HuggingFace libraries
  - CUDA kernels implemented with Triton language
- Needs to be tested on larger-scale models (100B+ parameters)