About GPU fuzed kernel

Paul Guillermit and Yann-Eric Choho

May 2024

1 Abstract

Recent developments in machine learning rely heavily on matrix operations computed on GPUs. These operations can be accelerated at a low level through tricks such as parallelization, quantization and cache usage optimization. In this project, we explore a Llama toy model and implement these technics with `Triton`. A GitHub is associated with this project: https://github.com/yann-Choho/projet_PPML

2 The Llama toy model

We build a small Llama toy model using the library `transformers` (see 1 for the parameters).

The computational scheme is:

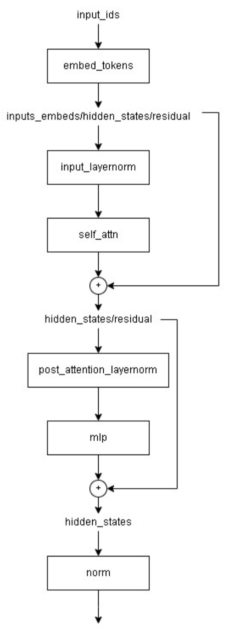

Figure 1: Computational scheme

We compare this scheme to the Netron trace analysis (see 5) to identify optimization opportunities.

3 Inside the MLP: triple elementwise multiplication

We investigate the MLP which corresponds to a gated multi-layer perception (see 6 for the structure visualized with Netron, and [1] for the literature). The MLP returns:

$$(\sigma(X \times G) \odot (X \times U)) \times D$$

with X the input and G, U, D the MLP parameters.
σ corresponds to the SiLU activation function: $\sigma(X) = X \odot sigmoid(X)$

If we note $Z = X \times G$, and $\tilde{Z} = \sigma(Z) \odot (X \times U)$, we propose the following computation scheme:

- Compute Z and $X \times U$ in parallel
- Compute $sigmoid(Z)$
- Compute the triple elementwise multiplication for $\tilde{Z}$
- Compute $\tilde{Z} \times D$

All steps appear to be natively parallelized, except for the triple elementwise multiplication.

Triple elementwise multiplication: The model parameters and input are already on the GPU Ram (using the corresponding `torch` options). Thanks to `triton`, we are able to propose the following improvements:

- Import corresponding blocks of the three matrices on the cache
- Perform the elementwise multiplication of the three blocks using parallel threads
- Save the results

We use `triton.autotune` to choose the block size. We benchmark our implementation against pytorch and get:

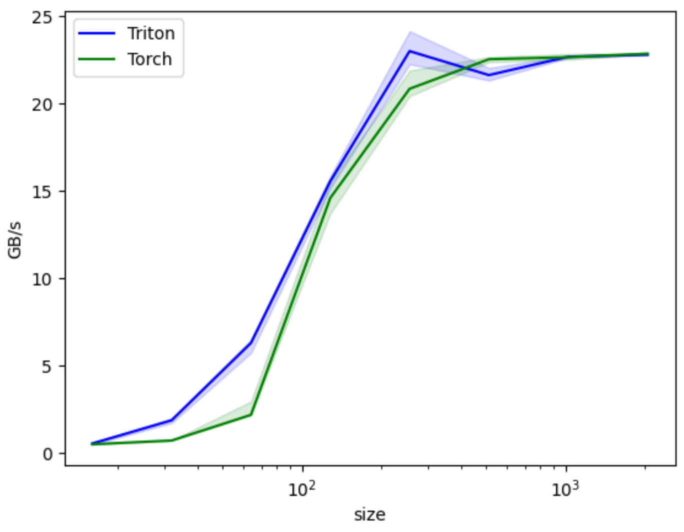

Figure 2: Triple elementwise multiplication fuzed kernel benchmark

4 Fusing the whole MLP with the RMS Normalisation

We further extend the kernel to include RMS normalization: The idea is to combine two calculation steps into a single optimized loop, thus reducing redundancies of loading the output after normalization to improve the overall efficiency of the model.

Here are the steps and the general idea of what we want to code:

- **RMS Normalization (RMSNorm)**: Apply RMS normalization to the input only once, and reuse this normalized output for subsequent operations. Currently, 'x_norm' is calculated once and then used in two separate matrix multiplications : **the novelty is that we calculate and apply the RMS statistics in the same loop as the matrix multiplications, thereby reducing the number of passes needed over the data**.
- **Chained Matrix Multiplications**: Chain the two matrix multiplications in a single loop to reduce memory accesses and improve efficiency. The first matrix multiplication is followed by a SiLU activation (which combines a sigmoid activation and a multiplication), then a second matrix multiplication is performed.
- **Operation Fusion**: Fuse the SiLU operation with the output of the first matrix multiplication to avoid redundant operations. This fusion involves combining the elementary operations so that they are performed in a single step.
- **Performance Improvement**: By streamlining these operations, the new kernel will be faster than the current PyTorch implementation. The optimizations aim to reduce execution time by minimizing redundant operations and improving resource utilization.

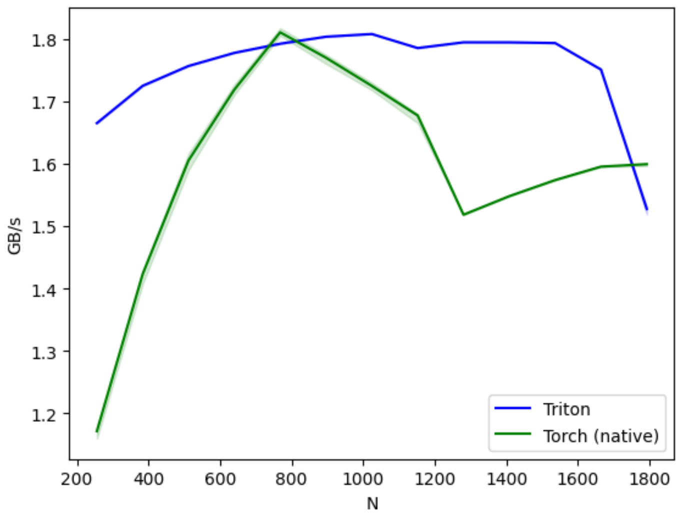

Figure 3: MLP (without RMSNorm, without quantization) kernel benchmark

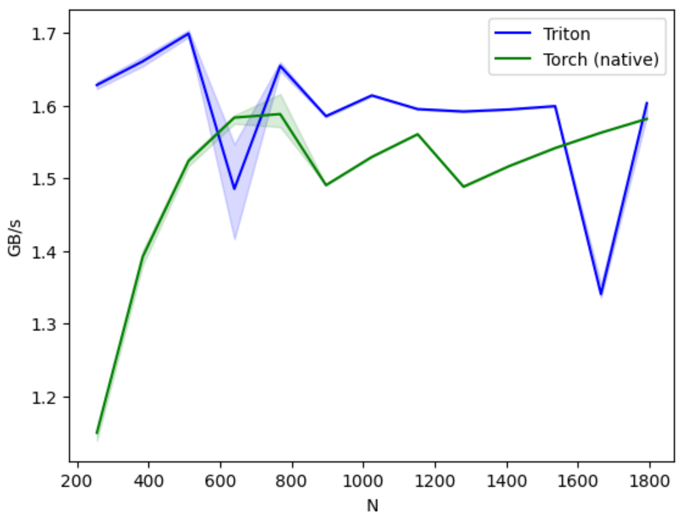

Figure 4: MLP + RMSNorm kernel benchmark (without quantization)

We do reach GPU execution time increase by 13% using Triton compared with PyTorch for a fixed input size (1,16, 4096) on the fused kernel for MLP, increase by 12% when fused with RMSNormalisation (both without using any quantization). We also observe variable performance when we modify the size, so the performance of the implemented kernel depends on the choice of parameters.

5 Extra : Quantization

To further optimize the model, we explore quantization techniques, focusing particularly on reducing precision without significant loss of accuracy. Reducing the precision from float 16 to float 8 increases the overall GPU execution time by 38% in Triton, while asserting that the error is within a certain range. The cons are that we observe the error due to the reduction in precision increases with the size of the input.

References

[1] Noam Shazeer. Glu variants improve transformer, 2020.

Appendix

Llama toy model parameters

Parameter	**Value**
hidden_size	1024
num_hidden_layers	1
vocab_size	32000
intermediate_size	11008
max_position_embeddings	2048
num_attention_heads	4

Table 1: Model Configuration

Llama Netron vizualization

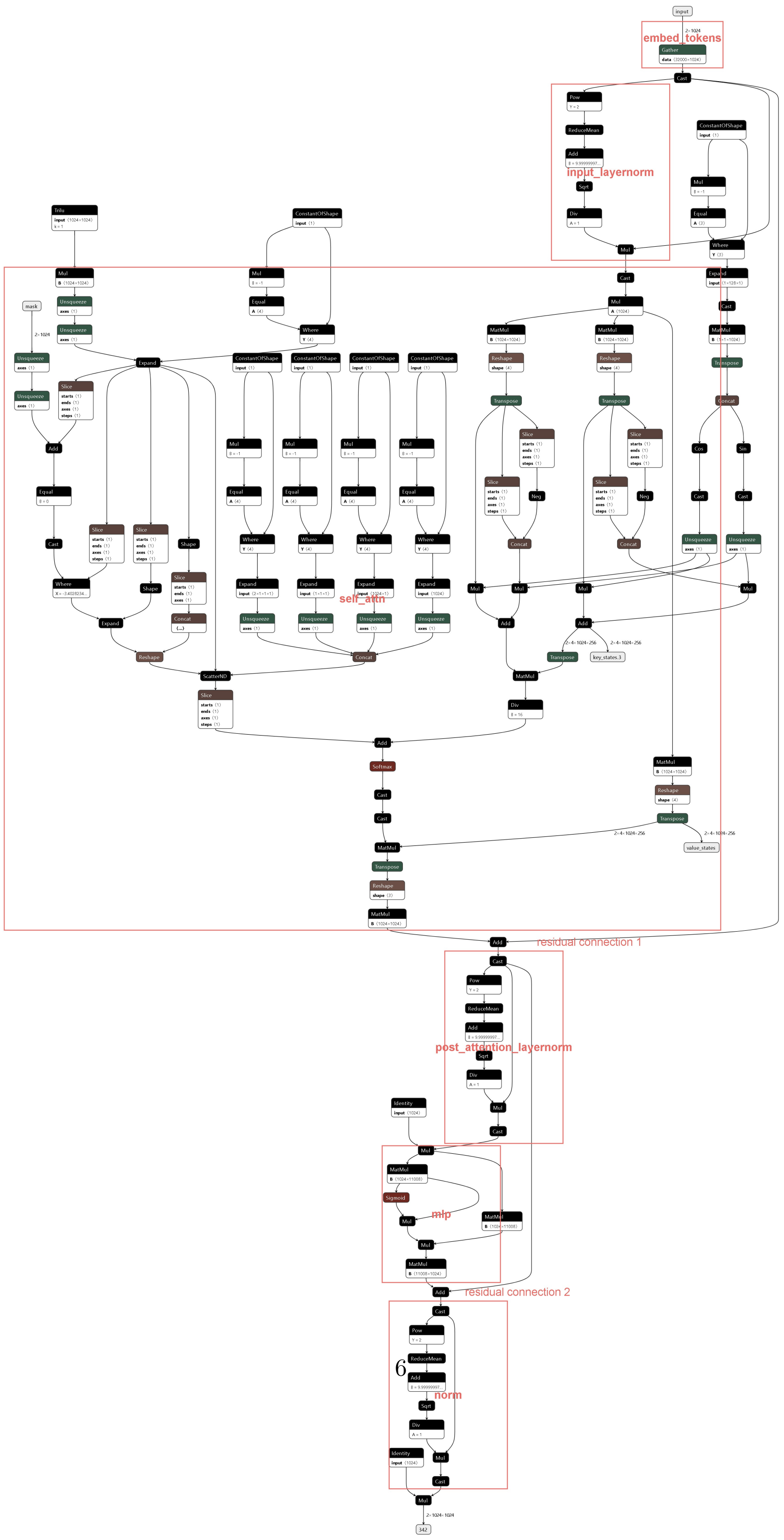

Figure 5: Netron vizualization labeled

MLP Netron vizualization

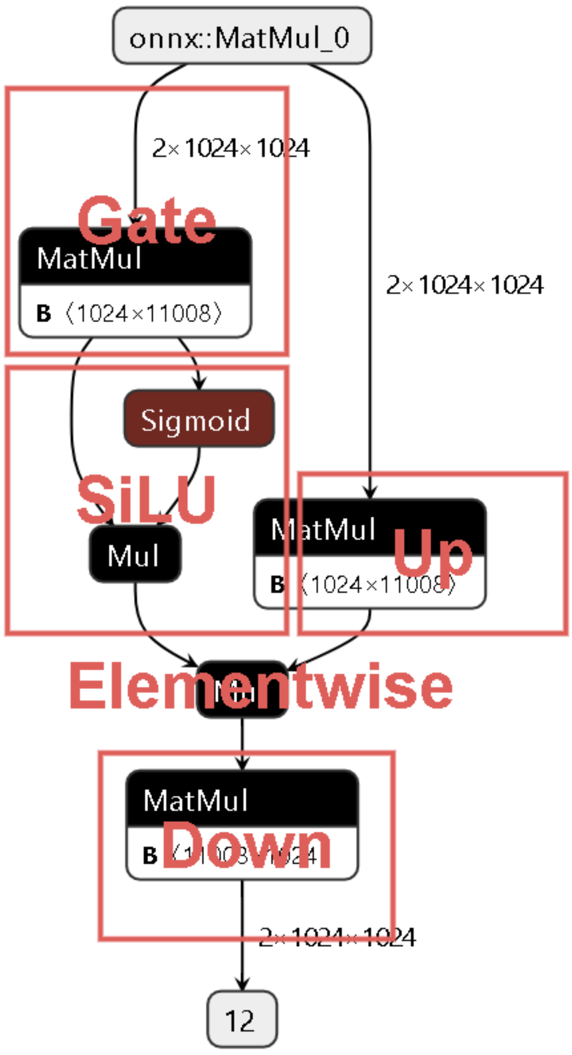

Figure 6: MLP Netron vizualization