Supervisor: Vladimir Krylov

Desember’ 14 2023

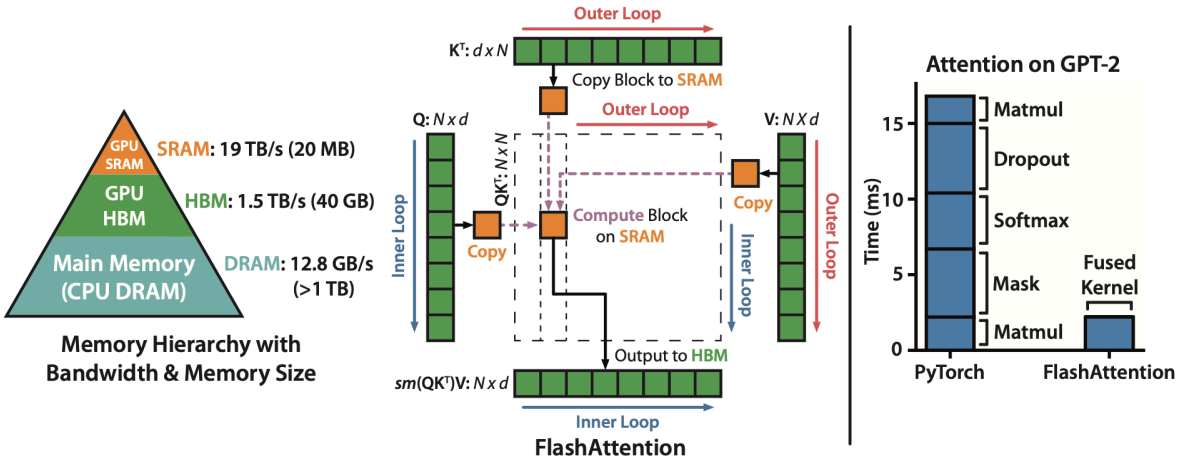
LLM compression

Abstract

In this work the several methods of LLMs inference speed up were discovered: Flash-attention, Flash-attention-2, kv-caching, Tensor Parallelism. Using of Flash-attention-2 decreased inference time on about 60% and decreased consuming of memory on 35%. KV-catching gave about 78% inference time speed up.

Methods

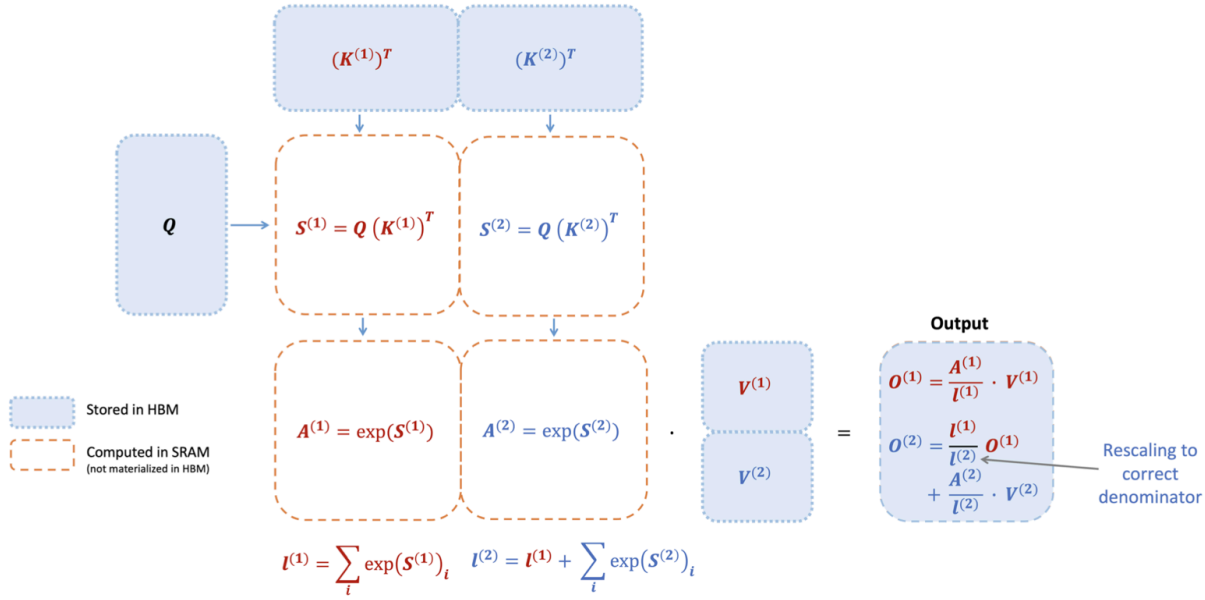
1. Flash-attention



FlashAttention is a algorithm that computes exact attention with far fewer memory accesses. The main goal is to avoid reading and writing the attention matrix to and from HBM. This requires computing the softmax reduction without access to the whole input not storing the large intermediate attention matrix for the backward pass.There were applied two well-established techniques to address these challenges. The attention computation was restructured to split the input into blocks and make several passes over input blocks, thus incrementally performing the softmax reduction.

The softmax normalization factor was stored from the forward pass to quickly recompute attention on-chip in the backward pass, which is faster than the standard approach of reading the intermediate attention matrix from HBM.

The basic idea is that we split the input data Q, K, V into blocks, load them from slow HBM to fast SRAM, then calculate the output data of attention relative to these blocks. By scaling the output of each block by the correct normalization factor before summing them, we get the correct result at the end.



Experiments

According to the results of our experiments, it turned out to accelerate the inference of the neural network by 14% due to the use of the algorithm, the results show that the size of the memory involved has slightly increased, but this can be explained by the fact that the GPT-2 model has an attention layer different from larger LLMs.

2. Flash-attention-2

FlashAttention-2 reduces the amount of non-matrix multiplication. The decrease in the number of nonmatrix operations is related to how they update and scale data in the process. Instead of constantly resizing this data at each step, they decided to do so only at the very end of the process.

Flash-attention

We do not have to rescale both terms of the output update by $\mathrm{diag}(\ell^{(2)})^{-1}$:

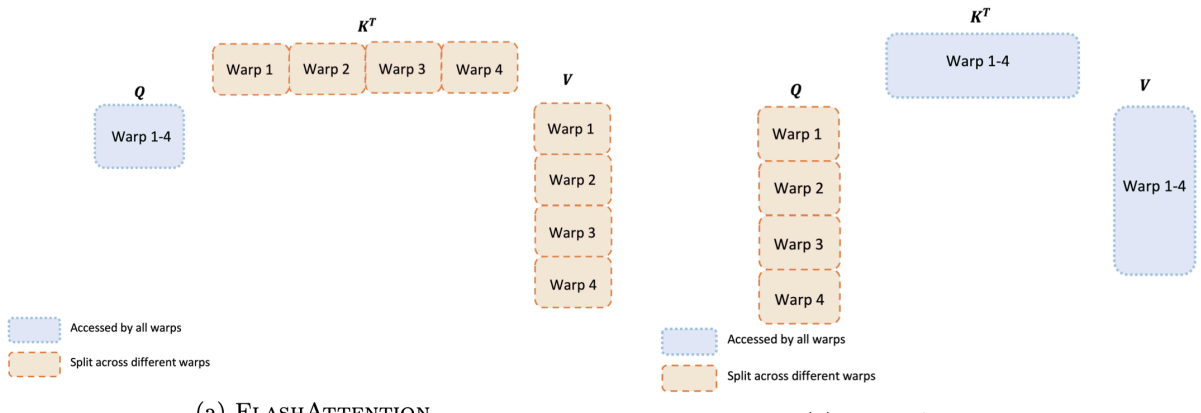
$$\mathbf{O}^{(2)} = \mathrm{diag}(\ell^{(1)}/\ell^{(2)})^{-1}\mathbf{O}^{(1)} + \mathrm{diag}(\ell^{(2)})^{-1}e^{\mathbf{S}^{(2)}-m^{(2)}}\mathbf{V}^{(2)}.$$

Flash-attention-2

$$\tilde{\mathbf{O}}^{(2)} = \mathrm{diag}(\ell^{(1)})^{-1}\mathbf{O}^{(1)} + e^{\mathbf{S}^{(2)}-m^{(2)}}\mathbf{V}^{(2)}.$$

In Flash Attention, keys (K) and values (V) were distributed across different warps, while queries (Q) were available to all warps. This required writing intermediate results to shared memory and then synchronizing them.

In FlashAttention-2, the approach has been changed: now Q is distributed between warps, and K and V are available to all warps. This allows each warp to perform matrix multiplication to obtain its own segment of results, without requiring additional communication between the warps.



(a) FlashAttention (b) FlashAttention-2

These yield around 2× speedup compared to FlashAttention, reaching 50-73% of the theoretical maximum FLOPs/s on A100 and getting close to the efficiency of GEMM operations.

Experiments

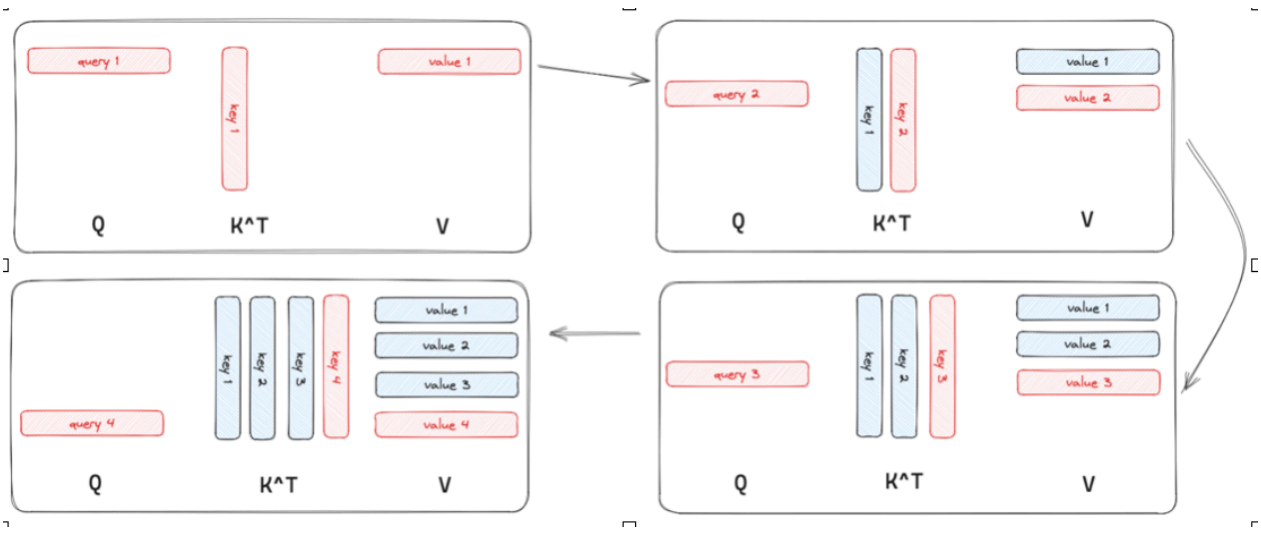
The using of the FlashAttention-2 gave about 60% acceleration of the inference time and reduced the memory used by 35%

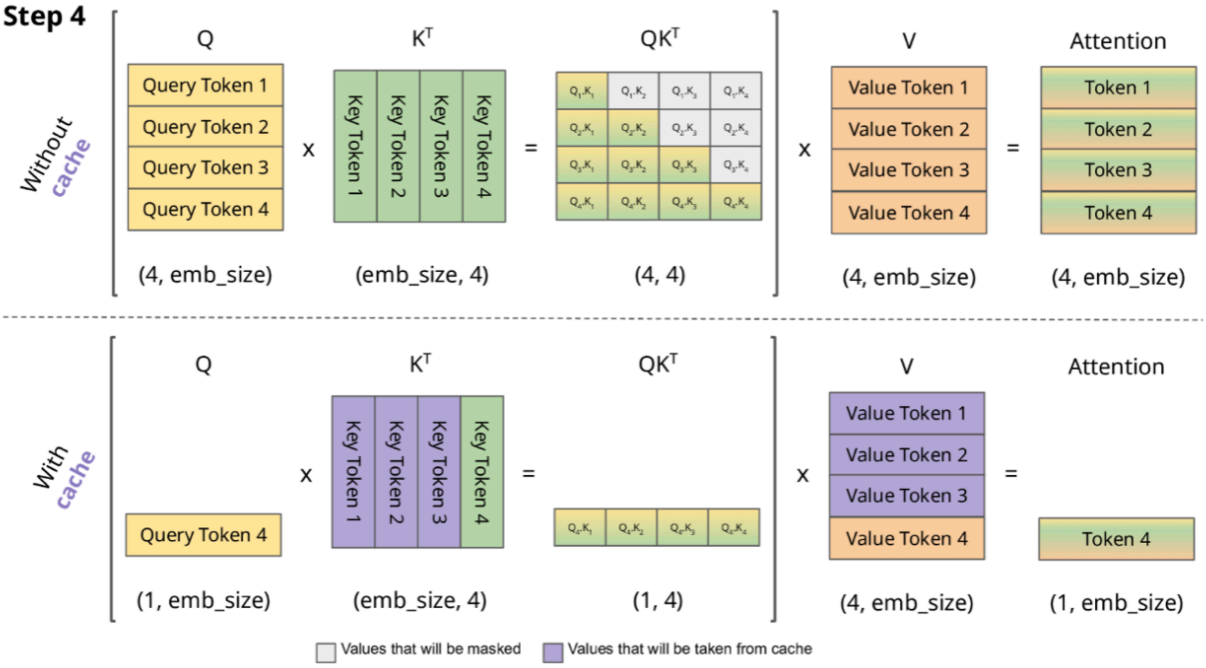
3. CV-caching

When calculating self-attention, we calculate the scalar product of the vector q by the vectors k_i of each other token in front of it in the input sequence. When adding a new element to the sequence, we have to recalculate the weighted sums for all previous elements. This is inefficient, especially when the length of the sequence increases.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

KV-caching is the preservation of already calculated key–value pairs for the previous elements of the sequence. When a new element is added, instead of going through the entire sequence again, we use the stored values. This greatly speeds up the process.





Experiments

We launched the GPT2 inference using KV-caching and without it and measured the running time:

with KV caching: 12.26 +- 1.367 seconds

without KV caching: 54.873 +- 0.393 seconds

The inference time was reduced by 78%

4. Tensor Parallelism

Tensor Parallelism using is MLP layers in transformer blocks.

```
TransformerBlock(
  (input_layernorm): LayerNorm((4096,), eps=1e-05, elementwise_affine=True)
  (self_attention): Attention(
    (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (k_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (v_proj): Linear(in_features=4096, out_features=4096, bias=False)
    (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
  )
  (post_attention_layernorm): LayerNorm((4096,), eps=1e-05, elementwise_affine=True)
  (mlp): MLP(
    (dense_h_to_4h): Linear(in_features=4096, out_features=16384, bias=False)
    (dense_4h_to_h): Linear(in_features=16384, out_features=4096, bias=False)
  )
)
```

To calculate linear layers in parallel on several devices, you need to divide the weights into as many parts as there are devices, and multiply the submatrices separately on each of the devices. At the end, you need to combine all the outputs together.

- h to 4h column parallel linear->(1, 10, 4096) . (4096, 8192) = (1, 10, 8192)
- 4h to h row parallel linear->(1, 10, 8192) . (8192, 4096) = (1, 10, 4096)
- All reduce operation

Experiments

With using Tensor Parallelism we got about a 16% decrease in the inference time.

The practical part

The following methods of accelerating the inference of neural networks have been studied and reproduced in practice:

- Flash-attention
- Flash-attention-2
- kv-caching
- Tensor-parallelism

Two of the most effective methods were chosen - flash-attention-2 and kv-caching, they were used in the final application.

To demonstrate how the methods work, the ChatGLM2-6B neural network was selected and an application was written in Flask.

Resources:

- Tri Dao, Daniel Y. Fu. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

- Tri Dao, Daniel Y. Fu. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning

- Zhihao Jia, Todd Warszawski . TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph

- https://huggingface.co/blog/optimize-llm, https://medium.com/@joaolages/kv-caching-explained-276520203249

- Efficient Memory Management for Large Language Model Serving with Paged-Attention, https://arxiv.org/pdf/2309.06180.pdf

- https://www.deepspeed.ai/tutorials/automatic-tensor-parallelism/#example-script

- https://blog.gopenai.com/understanding-tensor-parallelism-to-fit-larger-models-on-multiple-devices-d4da1821d41b

- https://huggingface.co/blog/optimize-llm