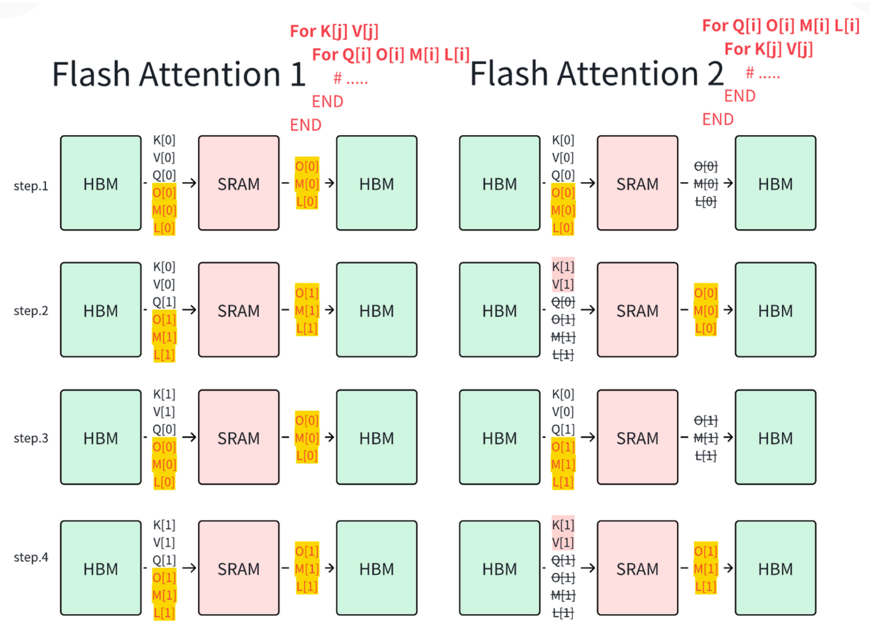


关于作者
小冬瓜AIGC
原创课程➡️ 公众号：手撕L...
回答 8 文章 37 关注者 4,858
关注他 发私信

【手撕LLM-FlashAttention2】只因For循环优化的太美

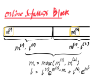
小冬瓜AIGC
原创课程➡️ 公众号：手撕LLM
关注他

来自专栏 · 手撕LLM >

> 我是小冬瓜AIGC，原创超长文知识分享，原创课程已帮助多名同学速成上岸LLM赛道：手撕LLM+RLHF
> 研究方向：LLM、RLHF、Safety⁺、Alignment⁺

目录

阅读本文前需要先掌握 `Online Softmax和Flash Attention 1`



小冬瓜AIGC：【手撕online softmax】Flash Attention前传，一撕一个不吱声
101 赞同 · 13 评论 文章

小冬瓜AIGC：【手撕LLM-FlashAttention】从softmax说起，保姆...
859 赞同 · 44 评论 文章

导读

1. 本文不需要有任何 `CUDA` 背景知识，尽可能少公式和符号，看懂图解和代码就够了
2. 本文需要熟悉 `Flash Attention1` 至少能理解 `Online-Softmax` 和 `Flash Attention Forward` 代码实现逻辑

1 论文解析

> FlashAttentio
> Stanford

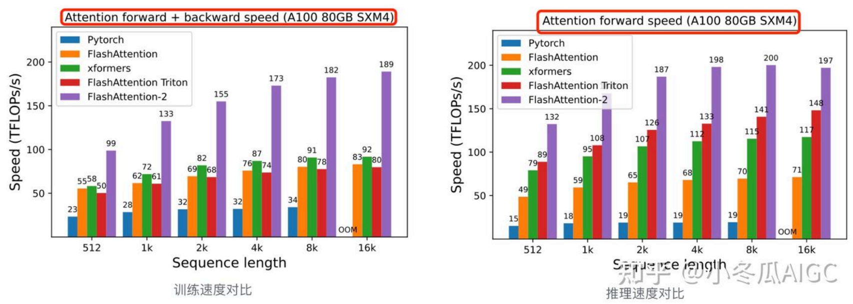
1.1 Paper Abstract & Key Point：

1. `Flash Attention 2` 比 `Flash Attention 1` 加速 `2x`，计算效率达到GEMM性能的 `50~73%`
2. 改进1：减少非乘法计算【讲这个】
3. 改进2: 优化 `QKV for` 循环顺序【讲这个】
4. 改进3: 采用 `shared memory` 减少通信

baselines), with no approximation. However, FlashAttention is still not nearly as fast as optimized matrix-multiply (GEMM) operations, reaching only 25-40% of the theoretical maximum FLOPs/s. We observe that the inefficiency is due to suboptimal work partitioning between different thread blocks and warps on the GPU, causing either low-occupancy or unnecessary shared memory reads/writes. We propose FlashAttention-2, with better work partitioning to address these issues. In particular, we (1) tweak the algorithm to reduce the number of non-matmul FLOPs (2) parallelize the attention computation, even for a single head, across different thread blocks to increase occupancy, and (3) within each thread block, distribute the work between warps to reduce communication through shared memory. 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. We empirically validate that when used end-to-end to train GPT-style models, FlashAttention-2 reaches training speed of up to 225 TFLOPs/s per A100 GPU (72% model FLOPs utilization).[1]

1.2 Flash Attention-2 性能

- 训练时： `forward+backward`， `4k` 文本， `FA2` 较 `FA1` 增速 `2.27x`， `FA2` 较 `Pytorch` 增速 `5.4x`
- 推理时： `forward`， `4k` 的 `FA2` 较 `FA1` 增速 `2.91x`， `FA2` 较 `Pytorch` 版本增速 `10x`



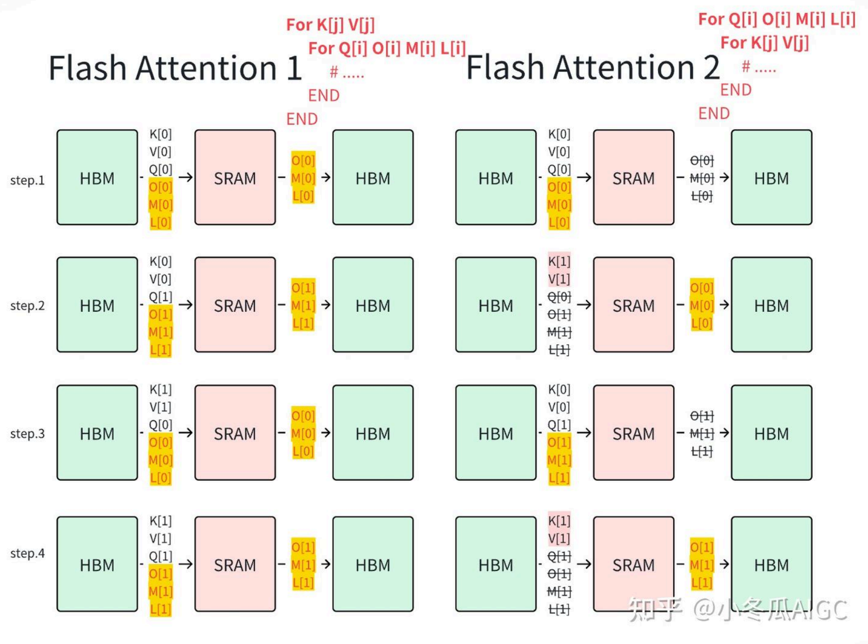
训练速度对比 推理速度对比

Pytorch哭晕在厕所

2 Flash Attention-2 算法原理解析

2.1 优化内外循环读写流程

- `Flash Attention 1` 最大的问题在于 `O[i]` 需要频繁的读写 SRAM 缓存，如左图，那么考虑 `O[i]` 的在一个 `Q[i]` 周期算完，就可以减少 `O[i]` 的频繁读写缓存
- `Flash Attention 2` 将 `Q` 当作外循环， `KV` 当作内循环， 将 `O[i]` 的在一个 `Q[i]` 周期算完， `O[i]{t}<—O[i]{t-1}` 如右图
- 从 `O` 的缓存 `write/read` 次数从 `2 x B_q x B_kv -> 2 x B_q` 次
- ***此图对应文末代码***



Left:Flash Attention 1 中的O、M、L读写8次，Right：Flash Attention 2 中的O、M、L读写4次

Stanford Tri Dao官方版动图可见 `Flash Attention2` 的动图与上述流程一样(章节 Multi-head attention⁺ for decoding)

https://crfm.stanford.edu/2023/10/12/flashdecoding.html
crfm.stanford.edu/2023/10/12/flashdecoding.html

2.2 减少非乘法(non-matmul)计算量

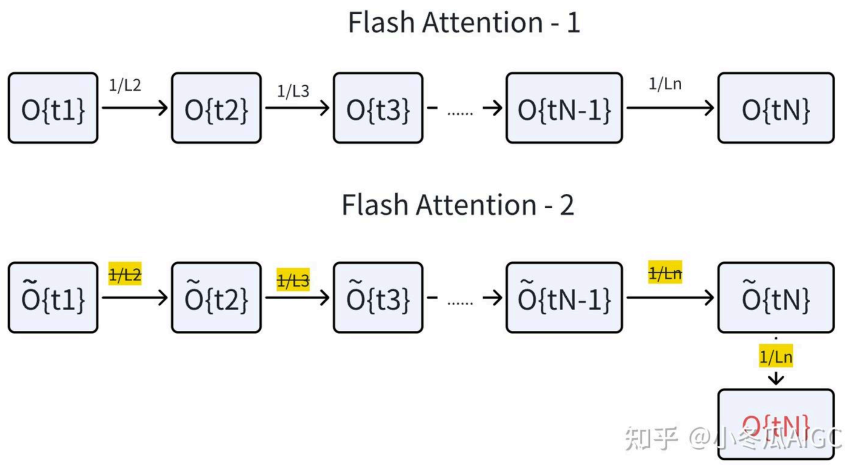
下图可以直接对比出来， `Flash Attnetion 1` 每次都要做 `Scaled` (对O * L_i^{-1})，都是额外的非乘法计算。

在Flash Attention中，计算O时存在非乘法(non-matmul)的计算，非常耗时，如 $drag(l^{(2)})^{-1}$

$$O^{(2)} = diag(l^{(1)}/l^{(2)})^{-1}O^{(1)} + diag(l^{(2)})^{-1}e^{S^{(2)}-m^{(2)}}V^{(2)}$$

提取出 $drag(l^{(2)})^{-1}$ 系数，计算 `O` 的过程 `un-scaled` ，对 `O` 最后时刻做 `scaled`

$$\begin{aligned}\tilde{O}^{(2)} &= diag(l^{(1)})^{-1}O^{(1)} + e^{S^{(2)}-m^{(2)}}V^{(2)} \\ O^{(2)} &= diag(l^{(2)})^{-1}\tilde{O}^{(2)} \\ O^{(N)} &= diag(l^{(N)})^{-1}\tilde{O}^{(N)}\end{aligned}$$

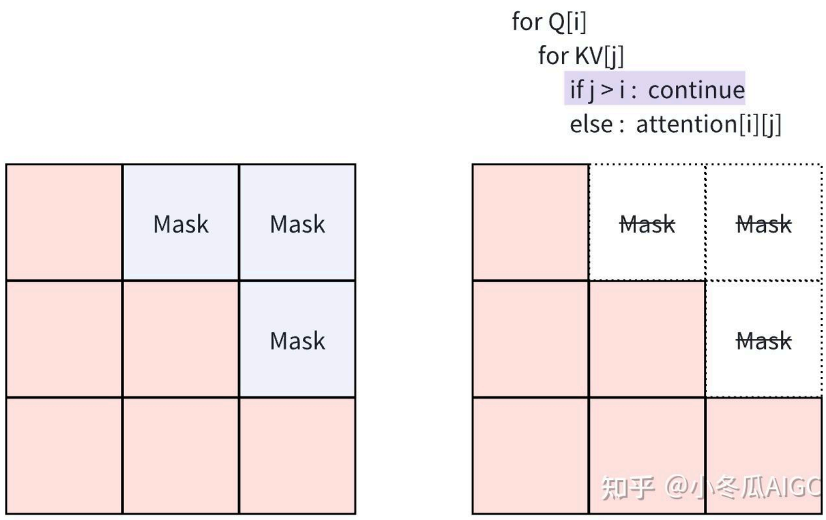


2.3 忽略mask block的Attention计算

- 跳过绝对需要完全 mask 的 block 的计算
- 加速 1.7-1.8x ， Decode-Only 确实可以忽略“**上三角**”块的计算

Causal masking. One common use case of attention is in auto-regressive language modeling, where we need to apply a causal mask to the attention matrix $\mathbf{S}$ (i.e., any entry $\mathbf{S}_{ij}$ with $j > i$ is set to $-\infty$).

1. As FlashAttention and FlashAttention-2 already operate by blocks, for any blocks where all the column indices are more than the row indices (approximately half of the blocks for large sequence length), we can skip the computation of that block. This leads to around 1.7-1.8× speedup compared to attention without the causal mask.



Left:原始Masked Self Attention Right：Flash2 Ignore Masked Self Attention

3. 手撕 Flash Attention 2

手撕 Flash Attention 2 ，看结果的打印顺序，豁然开朗

```python
# author: 小冬瓜AIGC

import torch
from einops import rearrange

Q_BLOCK_SIZE = 3
KV_BLOCK_SIZE = 3
NEG_INF = -1e10  # -infinity
EPSILON = 1e-10
Q_LEN = 6
K_LEN = 6
Tr = Q_LEN // Q_BLOCK_SIZE
Tc = K_LEN // KV_BLOCK_SIZE

Q = torch.randn(1, 1, 6, 4, requires_grad=True).to(device='cpu')
K = torch.randn(1, 1, 6, 4, requires_grad=True).to(device='cpu')
V = torch.randn(1, 1, 6, 4, requires_grad=True).to(device='cpu')
O = torch.zeros_like(Q, requires_grad=True)
l = torch.zeros(Q.shape[:-1])[..., None]
m = torch.ones(Q.shape[:-1])[..., None] * NEG_INF

Q_BLOCKS = torch.split(Q, Q_BLOCK_SIZE, dim=2)
K_BLOCKS = torch.split(K, KV_BLOCK_SIZE, dim=2)
V_BLOCKS = torch.split(V, KV_BLOCK_SIZE, dim=2)
O_BLOCKS = list(torch.split(O, Q_BLOCK_SIZE, dim=2))
l_BLOCKS = list(torch.split(l, Q_BLOCK_SIZE, dim=2))
m_BLOCKS = list(torch.split(m, Q_BLOCK_SIZE, dim=2))

# start with ^
for i in ran
```

```python
        Qi = Q_BLOCKS[i]
        Oi = O_BLOCKS[i]
        li = l_BLOCKS[i]
        mi = m_BLOCKS[i]
        # li_cache = l_cache_BLOCKS[i]

        for j in range(Tc):
            #if j>i:
            #    continue    # ignore masked
            Kj = K_BLOCKS[j]
            Vj = V_BLOCKS[j]

            S_ij = Qi @ Kj.transpose(2,3)
            m_block_ij, _ = torch.max(S_ij, dim=-1, keepdims=True)
            mi_new = torch.maximum(m_block_ij, mi)
            P_ij_hat = torch.exp(S_ij - mi_new)
            l_block_ij = torch.sum(P_ij_hat, dim=-1, keepdims=True) + EPSILON

            li_new = torch.exp(mi - mi_new) * li  + l_block_ij
            Oi = torch.exp(mi - mi_new) * Oi + P_ij_hat @ Vj

            li = li_new
            mi = mi_new
            print(f'-----------O{i} = attn( Q{i}, KV[{j}])---------')
            print(Oi)

        O_BLOCKS[i] = Oi / li_new # 最后做Scaled
        l_BLOCKS[i] = li_new
        m_BLOCKS[i] = mi_new

    O = torch.cat(O_BLOCKS, dim=2)
    l = torch.cat(l_BLOCKS, dim=2)
    m = torch.cat(m_BLOCKS, dim=2)

    print(O)
```

结果, 可以看见对于 `O[i]` 在 `Q[i]` 周期内，直至算完才结束， 与FlashAttention-V1和标准Attention最终结果一致。

```
-----------O0 = attn( Q0, KV[0])---------
tensor([[[[-0.5712,  0.5756,  0.0109, -0.5217],
          [ 0.3267,  0.1504, -0.4310, -0.5974],
          [ 0.3851,  0.5882,  0.9223, -0.1432]]]], grad_fn=<AddBackward0>)
-----------O0 = attn( Q0, KV[1])---------
tensor([[[[-0.5769,  0.5776,  0.0344, -0.5117],
          [-0.1262,  0.2778,  3.9363,  1.1742],
          [-0.3885,  0.8162,  3.8338,  1.5820]]]], grad_fn=<AddBackward0>)
-----------O1 = attn( Q1, KV[0])---------
tensor([[[[ 0.5361,  0.5217,  1.1288,  0.0461],
          [-0.2164, -0.0970, -1.2126, -0.7168],
          [ 0.4848,  0.5008,  0.8824, -0.0961]]]], grad_fn=<AddBackward0>)
-----------O1 = attn( Q1, KV[1])---------
tensor([[[[-0.1802,  0.2877,  4.0247,  1.3121],
          [-0.3674, -0.0465, -0.9725, -0.5489],
          [-0.6276,  0.9017,  3.6402,  1.7995]]]], grad_fn=<AddBackward0>)
------------------------------------------
tensor([[[[-0.3839,  0.3844,  0.0229, -0.3405],
          [-0.0877,  0.1931,  2.7363,  0.8162],
          [-0.1728,  0.3630,  1.7052,  0.7037],
          [-0.1631,  0.2604,  3.6420,  1.1873],
          [-0.2228, -0.0282, -0.5896, -0.3328],
          [-0.2500,  0.3592,  1.4500,  0.7168]]]], grad_fn=<CatBackward0>)
```

《手撕RLHF》 解析如何系统的来做LLM对齐工程

小冬瓜AIGC：【手撕RLHF-Safe RLHF】带着脚镣跳舞的PPO

小冬瓜AIGC：【手撕RLHF-Rejection Sampling】如何优雅的从SFT过渡到PPO

《手撕LLM》 系列文章+原创课程：LLM原理涵盖Pretrained/PEFT/RLHF/高性能计算

小冬瓜AIGC：【手撕LLM-QLoRA】NF4与双量化-源码解析

小冬瓜AIGC：【手撕LLM-RWKV】重塑RNN 效率完爆Transformer

小冬瓜AIGC：【手撕LLM-FlashAttention】从softmax说起，保姆级超长文！！

小冬瓜AIGC:【手撕LLM-Generation】Top-K+重复性惩罚

小冬瓜AIGC：【手撕LLM-KVCache】显存刺客的前世今生--文末含代码

小冬瓜AIGC：【手撕LLM-FlashAttention2】只因For循环优化的太美

《手撕Agent》 从代码和工程角度，探索能够通向AGI的Agent方法

小冬瓜AIGC：【手撕Agent-ReAct】想清楚再行动、减轻LLM幻觉

> 我是小冬瓜AIGC，原创超长文知识分享，原创课程已帮助多名同学速成上岸LLM赛道：手撕LLM+RLHF
> 研究方向：LLM、RL、RLHF、AIGC和Agent

编辑于 2025-04-22 12:23 · 美国

LLM 大模型 RLHF

DefTruth

赞

2023-12-03 · 广东 回复 1

山

大佬

2024-03-28 · 上海 回复 喜欢

DefTruth ▸ 山与水你和我

flashattention系列的已经填坑了哈

2024-03-28 · 广东 回复 喜欢

smile2game

手撕的代码实际跑了一下是错的，可以用下面这个代码测下结果。

```
import torch.nn.functional as F
Sd = F.softmax(Q @ K.transpose(2,3), dim=-1)
Od = Sd @ V
print(f"Od.shape is {Od.shape}")
print(f"O is {O}\nOd is {Od}" )
assert torch.allclose(Od,O,atol=1e-5)
```

修正了一下，在内层j的列循环种，需要及时更新li和mi，附上我纠正好的代码

```
import torch
torch.manual_seed(456)
NEG_INF = -1e10
EPSILON = 1e-10

#原始的 QKV尺寸 6x4
N = 6
d = 2

#切分求解 block_size ,Bc = M/4d,Br = min(M/4d,d)
Br = 2 #6//2,
Bc = 2
#根据block_size得到，Q,K,V 切出来，Tr行和Tc列
Tr = N//Br
Tc = N//Bc

#创建 QKV，Olm等矩阵
Q = torch.randn(N,d,requires_grad=True)
K = torch.randn(N,d,requires_grad=True)
V = torch.randn(N,d,requires_grad=True)

O = torch.zeros_like(Q, requires_grad=True)
l = torch.zeros(Q.shape[:-1])[..., None] #删减后在增减一个为1的维度
m = torch.ones(Q.shape[:-1])[..., None] * NEG_INF

#切分成tiling
Q_blocks = torch.split(Q,Br,dim = 0) #沿着序列维度去切分他,变成两个元组了，这里
split第二个参数是 尺寸大小
# print(f"Q_blocks is {Q_blocks}")
K_blocks = torch.split(K,Bc,dim = 0)
V_blocks = torch.split(V,Bc,dim = 0)

O_blocks = list(torch.split(O,Br,dim = 0))
l_blocks = list(torch.split(l,Br,dim = 0))
m_blocks = list(torch.split(m,Br,dim = 0))

for i in range(Tr): #行循环
Qi = Q_blocks[i]
Oi = O_blocks[i]
li = l_blocks[i]
mi = m_blocks[i]

for j in range(Tc): #列循环
Kj = K_blocks[j] #加载进来 K和V
Vj = V_blocks[j]

S_ij = Qi@Kj.T
m_block_ij,_ = torch.max(S_ij,dim = -1,keepdims = True) #行最大值
#下面这里开始和v1不一样
mi_new = torch.maximum(m_block_ij,mi) #提前算出来 mi_new用来更新 P_ij_hat和
l_block_ij ,减少了两次换底乘法
P_ij_hat = torch.exp(S_ij - mi_new)
l_block_ij = torch.sum(P_ij_hat,dim = -1,keepdim = True) + EPSILON #行求和，防止
是0所以加个极小值，要做除数
#9-step
li_new = li * torch.exp(mi-mi_new) + l_block_ij #行前和修正 + 当前块修正
#10-step
Oi = Oi *
```

```
li = li_new
mi = mi_new

O_blocks[i] = Oi/li_new
l_blocks[i] = li_new
m_blocks[i] = mi_new


O = torch.cat(O_blocks,dim = 0)


print(f"O.shape is {O.shape}")


import torch.nn.functional as F
Sd = F.softmax(Q @ K.T, dim=-1)
Od = Sd @ V
print(f"Od.shape is {Od.shape}")
print(f"O is {O}\nOd is {Od}" )
assert torch.allclose(Od,O,atol=1e-5)
```

04-04 · 陕西 回复 1

小冬瓜AIGC 作者
感谢share，检验结果一致
04-22 · 美国 回复 喜欢

珠穆拉玛峰
non-matmul 这个应该是减少非矩阵相乘
2024-03-26 · 北京 回复 1

Sakura
减少非乘法(non-matmul)计算量 那里， 公式是不是错了。
diag(l1 / l2) 那里 多了个逆(-1)
2024-07-25 · 北京 回复 喜欢

Bowen-zzb610
for j in range(Tc):
请教下这个内层循环的结束是不是少了对 mi 和 li 的更新呀, 试了下和 v1 的结果对不上
2024-05-22 · 上海 回复 喜欢

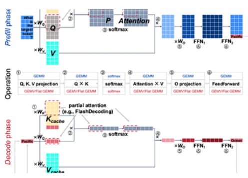
smile2game
可以看下我的回复，解决了这个问题
04-04 · 陕西 回复 1

qqqhhhbbb
是的，讲道理按照算法应该进行更新
2024-07-27 · 北京 回复 喜欢

展开其他 1 条回复 〉

理性发言，友善互动

推荐阅读



[LLM推理优化]🔥FlashDecoding++: 比...
DefTr... 发表于高性能计算...

RAG-Fusion 提高 LLM 生成文本的质量和深度
1. 介绍 检索增强生成（RAG）显着先进了人工智能。它结合了预训练的密集检索和序列到序列模型的功能来生成响应。在此基础上，出现了一种称为 RAG-Fusion 的新方法，旨在弥合传统搜索范式与...
大数据杂货铺

【FlashAttention-V4，非官方】FlashDecoding++
建议先阅读专栏：LLM加速 https://zhuanlan.zhihu.com/p/66... Introdcution为了提高softmax并行性，...
Austi... 发表于LLM加速

LLM在Text2SQL任(Agent版)
王大锤