编译论坛

FLASHATTENTION：一种具有 IO 感知，且兼具快速、内存高效的新型注意力算法

FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao†, Daniel Y. Fu†, Stefano Ermon†, Atri Rudra‡, and Christopher Ré†

†Department of Computer Science, Stanford University
‡Department of Computer Science and Engineering, University at Buffalo, SUNY

{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

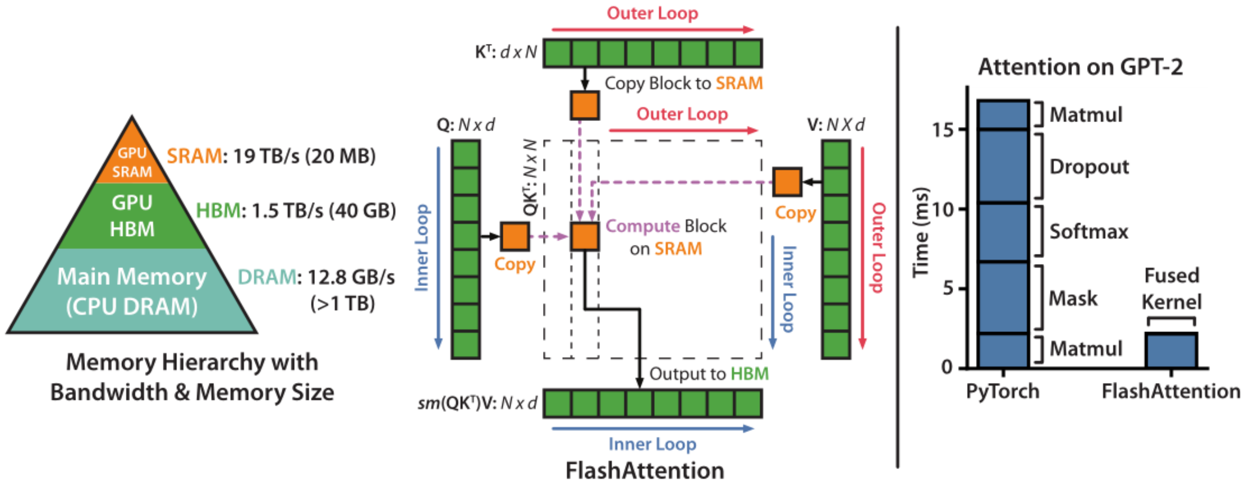
嘉宾：牛贺奔

目录

背景

Transformer 模型已是图像分类、自然语言处理等分支领域中最为常见的架构。这种模型核心的自注意力机制（self-attention）的时间和存储复杂度在序列长度上属于二次型。

于是有人提出近似注意力的方法，来减少注意力计算和内存需求。但它们过于关注降低每秒所执行的浮点运算次数（FLops），并且倾向于忽略来自内存访问(IO)的开销。



先进编译实验室
Advanced Compiler

传统Attention

在传统的Attention中，Q,K,V作为输入，大小为N×d，如下图所示，在计算中需要存储中间值S和P到HBM中，这会极大占用HBM（高带宽显存）。

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{N\times N}, \quad \mathbf{P} = \mathrm{softmax}(\mathbf{S}) \in \mathbb{R}^{N\times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N\times d},$$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

FLASH ATTENTION

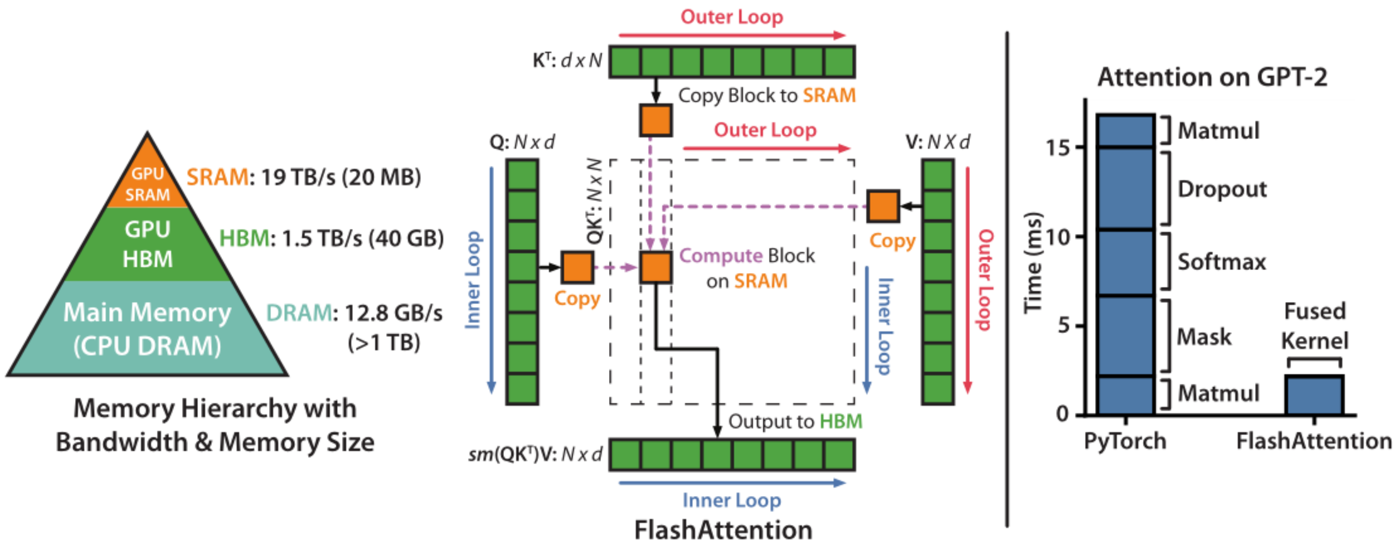
FlashAttention旨在避免从 HBM（High Bandwidth Memory）中读取和写入注意力矩阵。这需要做到：

(1)在不访问整个输入的情况下计算softmax函数的缩减；

(2)在后向传播中不能存储中间注意力矩阵。

实现思路

标准Attention算法由于要计算softmax，而softmax都是按行来计算的，按这个逻辑的话，在和V做矩阵乘之前，需要让 Q,K 的各个分块完成整一行分块的计算。得到Softmax的结果后，再和矩阵V分块做矩阵乘。而在Flash Attention中，将**输入分割成块**，并在输入块上进行多次传递，从而以增量方式执行softmax缩减。



实现思路

相比于标准Attention算法，Flash Attention并不需要存储中间注意力矩阵，存储前向传递的softmax归一化因子，以便在后向传递中快速重新计算芯片上的注意，这比从HBM读取中间注意矩阵的标准方法更快。

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Algorithm 2 FlashAttention Forward Pass

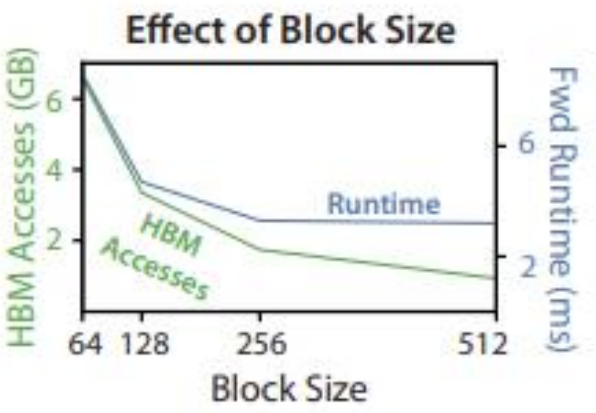
Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M, softmax scaling constant $\tau \in \mathbb{R}$, masking function MASK, dropout probability p_{drop}.

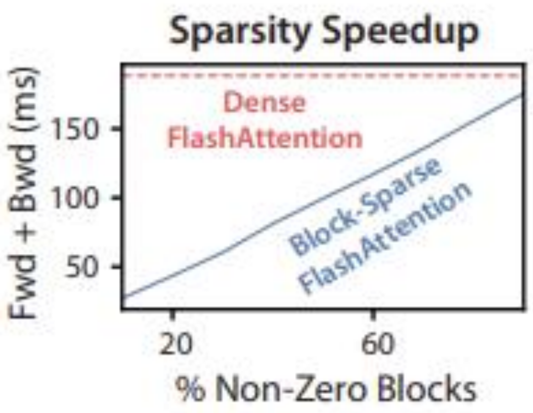
1: Initialize the pseudo-random number generator state $\mathcal{R}$ and save to HBM.
2: Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.
3: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
4: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
5: Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \dots, m_{T_r}$ of size B_r each.
6: **for** $1 \le j \le T_c$ **do**
7: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
8: **for** $1 \le i \le T_r$ **do**
9: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
10: On chip, compute $\mathbf{S}_{ij} = \tau \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
11: On chip, compute $\mathbf{S}_{ij}^{\mathrm{masked}} = \mathrm{MASK}(\mathbf{S}_{ij})$.
12: On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}^{\mathrm{masked}}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij}^{\mathrm{masked}} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
13: On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
14: On chip, compute $\tilde{\mathbf{P}}_{ij}^{\mathrm{dropped}} = \mathrm{dropout}(\tilde{\mathbf{P}}_{ij}, p_{\mathrm{drop}})$.
15: Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij}^{\mathrm{dropped}} \mathbf{V}_j)$ to HBM.
16: Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.
17: **end for**
18: **end for**
19: Return $\mathbf{O}, \ell, m, \mathcal{R}$.

结果

对比标准的Attention机制，Flash Attention虽然由于向后传播需要重新计算导致GFLOPs增加，但是Flash Attention对HBM的I/O和运行时间都有了显著的提高，如下图所示，我们可以看出Flash Attention在I/O减少和加速都有不错的效果。（A100GPU、GPT-2模型）。

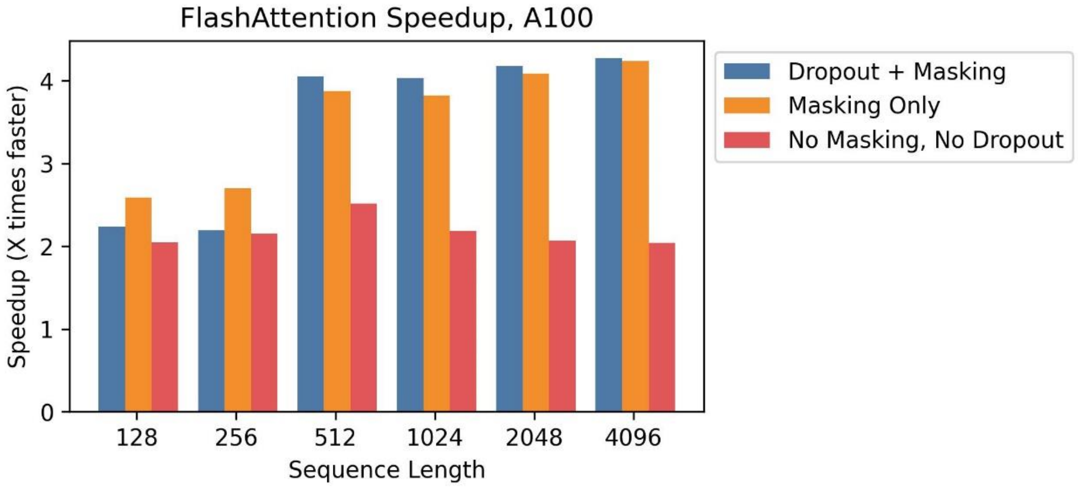
Attention	Standard	FlashAttention
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

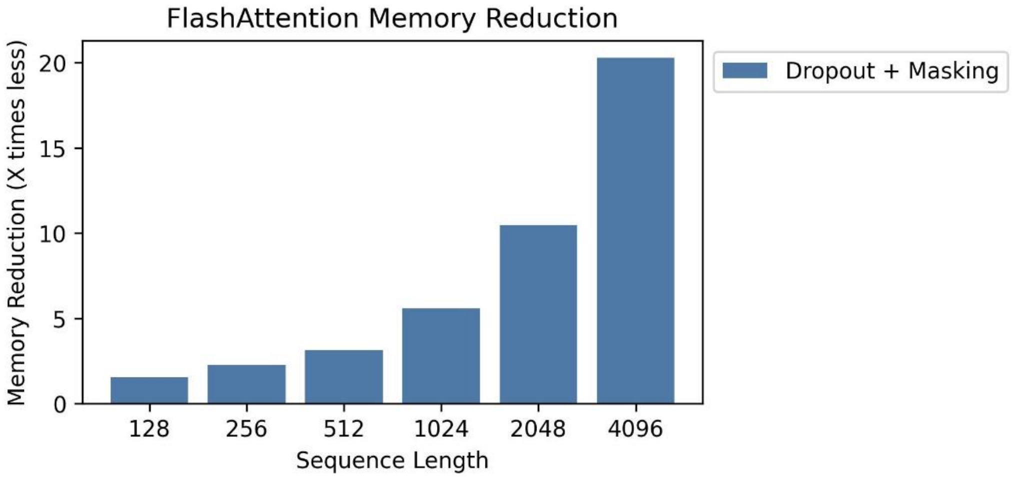




结果

如图所示，Flash-Attention算法在A100显卡上的加速效果，在不同的序列长度下都有不同程度的加速效果。而在右图中展示了随着序列长度的增加，Flash-Attention对于内存消耗有着不断提升的效果。





总结

Flash Attention的主要目的是**加速和节省内存。**主要贡献点：

1.计算softmax时候不需要全量input数据，可以分段计算。

2.反向传播的时候，不存储attention matrix (N^2的矩阵)，而是只存储softmax归一化的系数。

Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.

Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.

for $1 \le j \le T_c$ **do**

  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

  for $1 \le i \le T_r$ **do**

    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

    On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.

    Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.

    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.

  end for

end for

Return $\mathbf{O}$.

参考资料

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

https://gitcode.net/mirrors/HazyResearch/flash-attention?utm_source=csdn_github_accelerator

https://zhuanlan.zhihu.com/p/567167376