FlashAttention 原理

简介和相关前置

FlashAttention 的核心目标是通过融合(fusing)注意力计算中的多个操作(如 Q, K, V 矩阵乘法、softmax 归一化、与 V 的加权求和等)，直接在硬件(如 GPU)的高速缓存（SRAM）中完成计算，从而最小化对慢速全局内存(如 HBM)的读写次数。这被称为“IO 感知”(IO-aware)。 具体而言，FlashAttention使用平铺和重计算等经典技术，将输入块从HBM加载到SRAM(快速缓存)，在SRAM上执行注意力操作，并将结果更新回HBM。FlashAttention减少了内存读写量，从而实现了2-4倍的时钟时间加速。

- HBM(high bandwidth memory) 和 SRAM（static random-access memory）。速度上 SRAM>HBM>DRAM
- MAC(Memory Access Cost) 是指在计算机系统中，访问内存或存储器所需的时间和资源开销。它是衡量计算机程序或算法性能的重要指标之一。 MAC的值取决于多个因素，包括内存层次结构、缓存命中率、内存带宽、存储器延迟等。较低的MAC值表示访问内存的开销较小，而较高的MAC值表示访问内存的开销较大。

原理

传统Attention

对于输入序列 $Q, K, V \in R^{N \times d}$，$N$是序列长度，$d$是token尺寸，$N >> d$。

self-attention计算输出 $O \in R^{Nd}$, 计算公式为：

$$S = QK^T \in R^{N \times d},\ P = softmax(S) \in R^{N \times N}, O = PV \in R^{N \times d}$$

传统attention计算流程为:

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 $\mathbf{O}$.

FlashAttention

核心思想是传统减少HBM的访问，将QKV切分为小块后放入SRAM中
核心方法是tiling和recomputation

Tiling(平铺):分块计算

Softmax计算方法：

$$softmax(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{k} e^{x_i}}$$

softmax操作是row-wise的，即每行都算一次softmax，所以需要用到平铺算法来分块计算softmax。
原始softmax数值不稳定，为了数值稳定性，FlashAttention采用safe softmax，向量 $\in R$ 的safe softmax 计算如下

$$m(x) := \max_i x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \dots & e^{x_B - m(x)} \end{bmatrix}, \quad \ell(x) := \sum_i f(x)_i$$

$$\mathrm{softmax}(x) := \frac{f(x)}{\ell(x)}$$

同理，则$x = [x^{(1)} x^{(2)}] \in R^{2B}$ 的softmax也可以通过分解进行计算：

$$\begin{aligned} m(x) &= m\left(\left[x^{(1)} x^{(2)}\right]\right) = \max\left(m\left(x^{(1)}\right), m\left(x^{(2)}\right)\right), \\ f(x) &= \left[e^{m\left(x^{(1)}\right) - m(x)} f\left(x^{(1)}\right) \quad e^{m\left(x^{(2)}\right) - m(x)} f\left(x^{(2)}\right)\right], \\ \ell(x) &= \ell\left(\left[x^{(1)} x^{(2)}\right]\right) = e^{m\left(x^{(1)}\right) - m(x)} \ell\left(x^{(1)}\right) + e^{m\left(x^{(2)}\right) - m(x)} \ell\left(x^{(2)}\right), \\ \mathrm{softmax}(x) &= \frac{f(x)}{\ell(x)} \end{aligned}$$

$f(x)$和$l(x)$都可以通过分块计算得出，所以FlashAttention在计算时通过分块将Q，K，V分块后，按块加载到内存中。

前向计算步骤

伪代码：

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

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

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

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

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

  end for

end for

Return $\mathbf{O}$.

Algorithm 1 returns $O = softmax(QK^T)V$ with $O(N^2d)$ FLOPs and requires $O(b)$ additional memory beyond inputs and output.

▶ 前向计算Python代码