FlashAttention

1 FlashAttention1

1.1 目的

FlashAttention 旨在通过减少内存访问和计算的开销来加速 Transformer 模型中的注意力机制。它通过将注意力计算与内存访问紧密结合，减少了中间结果的存储需求，从而提高了计算效率。

1.2 算法解析

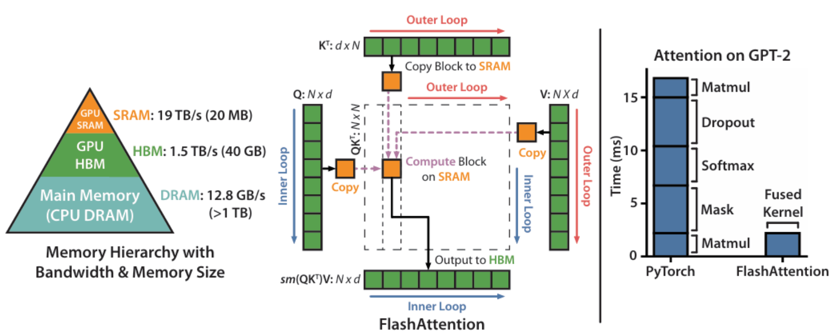


图 1: FlashAttention 的计算流程示意图

注：（1）第一幅图中的 SRAM 代指 GPU 的高速缓存（如 L1、L2 等），而 HBM 代指 GPU 的高带宽内存（DRAM、显存）。（2）由第三幅图可知，Attention 机制的瓶颈主要在于 Dropout, Softmax 和 Mask 操作，这些操作需要大量的内存访问和计算。

1.2.1 标准 attention 机制

给定输入序列 $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$，其中 N 为序列长度，d 为特征维度，标准的注意力机制计算如下：

Algorithm 0 Standard Attention Implementation

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

Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.

Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.

Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.

Return $\mathbf{O}$.

图 2: 标准注意力机制的计算流程示意图

标准 attention 机制反向传播的计算流程如下：

Algorithm 3 Standard Attention Backward Pass

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

Load $\mathbf{P}, \mathbf{dO}$ by blocks from HBM, compute $\mathbf{dV} = \mathbf{P}^\top \mathbf{dO} \in \mathbb{R}^{N \times d}$, write $\mathbf{dV}$ to HBM.

Load $\mathbf{dO}, \mathbf{V}$ by blocks from HBM, compute $\mathbf{dP} = \mathbf{dO}\mathbf{V}^\top \in \mathbb{R}^{N \times N}$, write $\mathbf{dP}$ to HBM.

Read $\mathbf{P}, \mathbf{dP}$ from HBM, compute $\mathbf{dS} \in \mathbb{R}^{N \times N}$ where $dS_{ij} = P_{ij}(dP_{ij} - \sum_l P_{il} dP_{il})$, write $\mathbf{dS}$ to HBM.

Load $\mathbf{dS}$ and $\mathbf{K}$ by blocks from HBM, compute $\mathbf{dQ} = \mathbf{dS}\mathbf{K}$, write $\mathbf{dQ}$ to HBM.

Load $\mathbf{dS}$ and $\mathbf{Q}$ by blocks from HBM, compute $\mathbf{dK} = \mathbf{dS}^\top \mathbf{Q}$, write $\mathbf{dK}$ to HBM.

Return $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$.

图 3: 标准注意力机制反向计算流程示意图

1.2.2 FlashAttention 的计算流程

Tiling

利用分块操作进行 softmax 计算，向量 $x \in \mathbb{R}^B$ 的 softmax 计算如下：

$$
\begin{aligned}
m(x) &= \max_i x_i \\
f(x) &= [e^{x_1 - m(x)}, ..., e^{x_B - m(x)}] \\
l(x) &= \sum_i f(x)_i \\
softmax(x) &= \frac{f(x)}{l(x)}
\end{aligned}
$$

注：（1）此处减去 $m(x)$ 是为了计算数值的稳定性，避免指数函数计算时出现溢出。

对于向量 $x^{(1)}, x^{(2)} \in \mathbb{R}^B$，将其拼接成 $x = [x^{(1)} x^{(2)}] \in \mathbb{R}^{2B}$ 进行 softmax 计算：

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

分块计算的好处在于可以减少内存访问和计算开销。具体来说，分块计算可以将向量分成多个小块，每个小块单独计算，然后将结果合并。这种方式可以减少对大向量（HBM）的内存访问，在 SRAM 中完成计算，提高计算效率。

Recomputation

Attention 机制为了在反向传播时节省内存，采用了重计算（Recomputation）策略。具体来说，在前向传播中不保存中间结果，如矩阵 **S P**，避免将矩阵写回 HBM 带来的开销；而是在前向传播的过程中保存中间变量 $m(x)$ $l(x)$，在反向传播时在 SRAM 上重新计算 **S** 和 **P**。

FlashAttention 的前向计算流程如图4所示。FlashAttention 的反向计算流程如图5所示。

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 $\textsc{mask}$, dropout probability p_{drop}.

Initialize the pseudo-random number generator state $\mathcal{R}$ and save to HBM.

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 $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ 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.

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.

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

    On chip, compute $\mathbf{S}_{ij}^{\text{masked}} = \textsc{mask}(\mathbf{S}_{ij})$.

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

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

    On chip, compute $\tilde{\mathbf{P}}_{ij}^{\text{dropped}} = \text{dropout}(\tilde{\mathbf{P}}_{ij}, p_{\text{drop}})$.

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

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

  end for

end for

Return $\mathbf{O}, \ell, m, \mathcal{R}$.

图 4: FlashAttention 的前向计算流程示意图

Algorithm 4 FlashAttention Backward Pass

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO} \in \mathbb{R}^{N \times d}$ in HBM, vectors $\ell, m \in \mathbb{R}^N$ in HBM, on-chip SRAM of size M, softmax scaling constant $\tau \in \mathbb{R}$, masking function MASK, dropout probability p_{drop}, pseudo-random number generator state $\mathcal{R}$ from the forward pass.

Set the pseudo-random number generator state to $\mathcal{R}$.

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

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.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide $\mathbf{dO}$ into T_r blocks $\mathbf{dO}_i, \dots, \mathbf{dO}_{T_r}$ 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.

Initialize $\mathbf{dQ} = (0)_{N \times d}$ in HBM and divide it into T_r blocks $\mathbf{dQ}_1, \dots, \mathbf{dQ}_{T_r}$ of size $B_r \times d$ each. Initialize $\mathbf{dK} = (0)_{N \times d}, \mathbf{dV} = (0)_{N \times d}$ in HBM and divide $\mathbf{dK}, \mathbf{dV}$ in to T_c blocks $\mathbf{dK}_1, \dots, \mathbf{dK}_{T_c}$ and $\mathbf{dV}_1, \dots, \mathbf{dV}_{T_c}$, of size $B_c \times d$ each.

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

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

  Initialize $\tilde{\mathbf{dK}}_j = (0)_{B_c \times d}, \tilde{\mathbf{dV}}_j = (0)_{B_c \times d}$ on SRAM.

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

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

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

    On chip, compute $\mathbf{S}_{ij}^{\text{masked}} = \text{MASK}(\mathbf{S}_{ij})$.

    On chip, compute $\mathbf{P}_{ij} = \text{diag}(l_i)^{-1} \exp(\mathbf{S}_{ij}^{\text{masked}} - m_i) \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute dropout mask $\mathbf{Z}_{ij} \in \mathbb{R}^{B_r \times B_c}$ where each entry has value $\frac{1}{1 - p_{\text{drop}}}$ with probability $1 - p_{\text{drop}}$ and value 0 with probability p_{drop}.

    On chip, compute $\mathbf{P}_{ij}^{\text{dropped}} = \mathbf{P}_{ij} \circ \mathbf{Z}_{ij}$ (pointwise multiply).

    On chip, compute $\tilde{\mathbf{dV}}_j \leftarrow \tilde{\mathbf{dV}}_j + (\mathbf{P}_{ij}^{\text{dropped}})^\top \mathbf{dO}_i \in \mathbb{R}^{B_c \times d}$.

    On chip, compute $\mathbf{dP}_{ij}^{\text{dropped}} = \mathbf{dO}_i \mathbf{V}_j^\top \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\mathbf{dP}_{ij} = \mathbf{dP}_{ij}^{\text{dropped}} \circ \mathbf{Z}_{ij}$ (pointwise multiply).

    On chip, compute $D_i = \text{rowsum}(\mathbf{dO}_i \circ \mathbf{O}_i) \in \mathbb{R}^{B_r}$.

    On chip, compute $\mathbf{dS}_{ij} = \mathbf{P}_{ij} \circ (\mathbf{dP}_{ij} - D_i) \in \mathbb{R}^{B_r \times B_c}$.

    Write $\mathbf{dQ}_i \leftarrow \mathbf{dQ}_i + \tau \mathbf{dS}_{ij} \mathbf{K}_j \in \mathbb{R}^{B_r \times d}$ to HBM.

    On chip, compute $\tilde{\mathbf{dK}}_j \leftarrow \tilde{\mathbf{dK}}_j + \tau \mathbf{dS}_{ij}^\top \mathbf{Q}_i \in \mathbb{R}^{B_c \times d}$.

  end for

  Write $\mathbf{dK}_j \leftarrow \tilde{\mathbf{dK}}_j, \mathbf{dV}_j \leftarrow \tilde{\mathbf{dV}}_j$ to HBM.

end for

Return $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$.

图 5: FlashAttention 的反向计算流程示意图

2 FlashAttention-2

2.1 FlashAttention 的思考

2.1.1 问题

FlashAttention 在 A100 GPU 上，前向计算只达到理论最大计算吞吐量的 $30-50\%$，而反向计算只达到理论最大计算吞吐量的 $25-35\%$。相比之下，GEMM 可以达到理论最大计算吞吐量的 $80-90\%$。

2.1.2 原因

在 GPU 的不同线程块和线程束之间的工作优化还未达到最佳，导致低占用率或不必要的内存读写。

2.2 FlashAttention-2 的改进

2.2.1 减少非矩阵计算

现代 GPU 对矩阵计算有专门的计算单元进行加速优化，而非矩阵计算（如 softmax、dropout 等）则没有专门的加速单元。因此，FlashAttention-2 通过将非矩阵计算转换为矩阵计算来提高效率。

FlashAttention 将矩阵进行分块计算，以注意力矩阵 $\mathbf{S} = [\mathbf{S}^{(1)}\ \mathbf{S}^{(2)}]$ 为例，进行分块计算，其中 $\mathbf{S}^{(1)}, \mathbf{S}^{(2)} \in \mathbb{R}^{B_r \times B_c}$，value 矩阵 $\mathbf{V}^{(1)}, \mathbf{V}^{(2)} \in \mathbb{R}^{B_c \times d}$。

$$
\begin{aligned}
m &= \max(\mathrm{rowmax}(\mathbf{S}^{(1)}), \mathrm{rowmax}(\mathbf{S}^{(2)})) \in \mathbb{R}^{B_r} \\
\ell &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) \in \mathbb{R}^{B_r} \\
\mathbf{P} &= \begin{bmatrix} \mathbf{P}^{(1)} & \mathbf{P}^{(2)} \end{bmatrix} = \mathrm{diag}(\ell)^{-1} \begin{bmatrix} e^{\mathbf{S}^{(1)}-m} & e^{\mathbf{S}^{(2)}-m} \end{bmatrix} \in \mathbb{R}^{B_r \times 2B_c} \\
\mathbf{O} &= \begin{bmatrix} \mathbf{P}^{(1)} & \mathbf{P}^{(2)} \end{bmatrix} \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix} = \mathrm{diag}(\ell)^{-1} e^{\mathbf{S}^{(1)}-m} \mathbf{V}^{(1)} + e^{\mathbf{S}^{(2)}-m} \mathbf{V}^{(2)} \in \mathbb{R}^{B_r \times d}.
\end{aligned}
$$

图 6: 标准 softmax 计算

标准 softmax 计算如图6所示，而 FlashAttention 的 softmax 计算如图7所示。

$$
\begin{aligned}
m^{(1)} &= \mathrm{rowmax}(\mathbf{S}^{(1)}) \in \mathbb{R}^{B_r} \\
\ell^{(1)} &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m^{(1)}}) \in \mathbb{R}^{B_r} \\
\tilde{\mathbf{P}}^{(1)} &= \mathrm{diag}(\ell^{(1)})^{-1} e^{\mathbf{S}^{(1)}-m^{(1)}} \in \mathbb{R}^{B_r \times B_c} \\
\mathbf{O}^{(1)} &= \tilde{\mathbf{P}}^{(1)}\mathbf{V}^{(1)} = \mathrm{diag}(\ell^{(1)})^{-1} e^{\mathbf{S}^{(1)}-m^{(1)}}\mathbf{V}^{(1)} \in \mathbb{R}^{B_r \times d} \\
m^{(2)} &= \max(m^{(1)}, \mathrm{rowmax}(\mathbf{S}^{(2)})) = m \\
\ell^{(2)} &= e^{m^{(1)}-m^{(2)}}\ell^{(1)} + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m^{(2)}}) = \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) = \ell \\
\tilde{\mathbf{P}}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1} e^{\mathbf{S}^{(2)}-m^{(2)}} \\
\mathbf{O}^{(2)} &= \mathrm{diag}(\ell^{(1)}/\ell^{(2)})^{-1}\mathbf{O}^{(1)} + \tilde{\mathbf{P}}^{(2)}\mathbf{V}^{(2)} = \mathrm{diag}(\ell^{(2)})^{-1} e^{s^{(1)}-m}\mathbf{V}^{(1)} + \mathrm{diag}(\ell^{(2)})^{-1} e^{s^{(2)}-m}\mathbf{V}^{(2)} = \mathbf{O}.
\end{aligned}
$$

图 7: FlashAttention 的 softmax 计算

前向计算的改动

（a）将如下操作：

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

替换为：

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

$$
\begin{aligned}
m^{(1)} &= \mathrm{rowmax}(\mathbf{S}^{(1)}) \in \mathbb{R}^{B_r} \\
\ell^{(1)} &= \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m^{(1)}}) \in \mathbb{R}^{B_r} \\
\mathbf{O}^{\tilde{(1)}} &= e^{\mathbf{S}^{(1)}-m^{(1)}}\mathbf{V}^{(1)} \in \mathbb{R}^{B_r \times d} \\
m^{(2)} &= \max(m^{(1)}, \mathrm{rowmax}(\mathbf{S}^{(2)})) = m \\
\ell^{(2)} &= e^{m^{(1)}-m^{(2)}}\ell^{(1)} + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m^{(2)}}) = \mathrm{rowsum}(e^{\mathbf{S}^{(1)}-m}) + \mathrm{rowsum}(e^{\mathbf{S}^{(2)}-m}) = \ell \\
\tilde{\mathbf{P}}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1} e^{\mathbf{S}^{(2)}-m^{(2)}} \\
\tilde{\mathbf{O}}^{(2)} &= \mathrm{diag}(e^{m^{(1)}-m^{(2)}})\tilde{\mathbf{O}}^{(1)} + e^{\mathbf{S}^{(2)}-m^{(2)}}\mathbf{V}^{(2)} = e^{s^{(1)}-m}\mathbf{V}^{(1)} + e^{s^{(2)}-m}\mathbf{V}^{(2)} \\
\mathbf{O}^{(2)} &= \mathrm{diag}(\ell^{(2)})^{-1}\tilde{\mathbf{O}}^{(2)} = \mathbf{O}.
\end{aligned}
$$

图 8: FlashAttention-2 的 softmax 计算

应用后，FlashAttention-2 的 softmax 计算如图8所示。不在每个 block

的每次迭代中执行 rescale 操作，而是在最后进行 rescale。

（b）在前向计算的过程中，不再将 $m^{(j)}$ 的最大值和 $l^{(j)}$ 的指数和都存储下来，用于反向计算，而是只存储 longsumexp $L^{(j)} = m^{(j)} + log(l^{(j)})$。

Algorithm 1 FlashAttention-2 forward pass

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

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.

Divide the output $\mathbf{O} \in \mathbb{R}^{N \times d}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, and divide the logsumexp L into T_r blocks $L_i, \dots, L_{T_r}$ of size B_r each.

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

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

  On chip, initialize $\mathbf{O}_i^{(0)} = (0)_{B_r \times d} \in \mathbb{R}^{B_r \times d}, \ell_i^{(0)} = (0)_{B_r} \in \mathbb{R}^{B_r}, m_i^{(0)} = (-\infty)_{B_r} \in \mathbb{R}^{B_r}$.

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

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

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

    On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \mathrm{rowmax}(\mathbf{S}_i^{(j)})) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \mathrm{rowsum}(\tilde{\mathbf{P}}_i^{(j)}) \in \mathbb{R}^{B_r}$.

    On chip, compute $\mathbf{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)} - m_i^{(j)}}) \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_i^{(j)} \mathbf{V}_j$.

  end for

  On chip, compute $\mathbf{O}_i = \mathrm{diag}(\ell_i^{(T_c)})^{-1} \mathbf{O}_i^{(T_c)}$.

  On chip, compute $L_i = m_i^{(T_c)} + \log(\ell_i^{(T_c)})$.

  Write $\mathbf{O}_i$ to HBM as the i-th block of $\mathbf{O}$.

  Write L_i to HBM as the i-th block of L.

end for

Return the output $\mathbf{O}$ and the logsumexp L.

图 9: FlashAttention-2 的前向计算流程示意图

2.2.2 并行化计算

在 FlashAttention 中，以 batch size 和 head 数量为单位进行并行化计算，每个线程块处理一个 attention head，因此总共有 batch size × number of heads 个线程块，每个线程块在一个 SM 上运行。以 A100 GPU 为例，共有 108 个 SM，当线程块数量足够大时（比如说大于等于 80），计算效率才会足够高。

但是当序列长度较大，对应 batch size 或 number of heads 较小时，计算效率会较低，因此将并行化粒度拓展到序列长度的维度。

Algorithm 2 FlashAttention-2 Backward Pass

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO} \in \mathbb{R}^{N \times d}$ in HBM, vector $L \in \mathbb{R}^N$ in HBM, block sizes B_c, B_r.

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 $\mathbf{dO}$ into T_r blocks $\mathbf{dO}_i, \ldots, \mathbf{dO}_{T_r}$ of size $B_r \times d$ each, and divide L into T_r blocks $L_i, \ldots, L_{T_r}$ of size B_r each.

Initialize $\mathbf{dQ} = (0)_{N \times d}$ in HBM and divide it into T_r blocks $\mathbf{dQ}_1, \ldots, \mathbf{dQ}_{T_r}$ of size $B_r \times d$ each. Divide $\mathbf{dK}, \mathbf{dV} \in \mathbb{R}^{N \times d}$ in to T_c blocks $\mathbf{dK}_1, \ldots, \mathbf{dK}_{T_c}$ and $\mathbf{dV}_1, \ldots, \mathbf{dV}_{T_c}$, of size $B_c \times d$ each.

Compute $D = \mathrm{rowsum}(\mathbf{dO} \circ \mathbf{O}) \in \mathbb{R}^d$ (pointwise multiply), write D to HBM and divide it into T_r blocks $D_1, \ldots, D_{T_r}$ 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.

  Initialize $\mathbf{dK}_j = (0)_{B_c \times d}, \mathbf{dV}_j = (0)_{B_c \times d}$ on SRAM.

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

    Load $\mathbf{Q}_i, \mathbf{O}_i, \mathbf{dO}_i, \mathbf{dQ}_i, L_i, D_i$ from HBM to on-chip SRAM.

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

    On chip, compute $\mathbf{P}_i^{(j)} = \exp(\mathbf{S}_{ij} - L_i) \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\mathbf{dV}_j \leftarrow \mathbf{dV}_j + (\mathbf{P}_i^{(j)})^\top \mathbf{dO}_i \in \mathbb{R}^{B_c \times d}$.

    On chip, compute $\mathbf{dP}_i^{(j)} = \mathbf{dO}_i \mathbf{V}_j^\top \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\mathbf{dS}_i^{(j)} = \mathbf{P}_i^{(j)} \circ (\mathbf{dP}_i^{(j)} - D_i) \in \mathbb{R}^{B_r \times B_c}$.

    Load $\mathbf{dQ}_i$ from HBM to SRAM, then on chip, update $\mathbf{dQ}_i \leftarrow \mathbf{dQ}_i + \mathbf{dS}_i^{(j)} \mathbf{K}_j \in \mathbb{R}^{B_r \times d}$, and write back to HBM.

    On chip, compute $\mathbf{dK}_j \leftarrow \mathbf{dK}_j + \mathbf{dS}_i^{(j)\top} \mathbf{Q}_i \in \mathbb{R}^{B_c \times d}$.

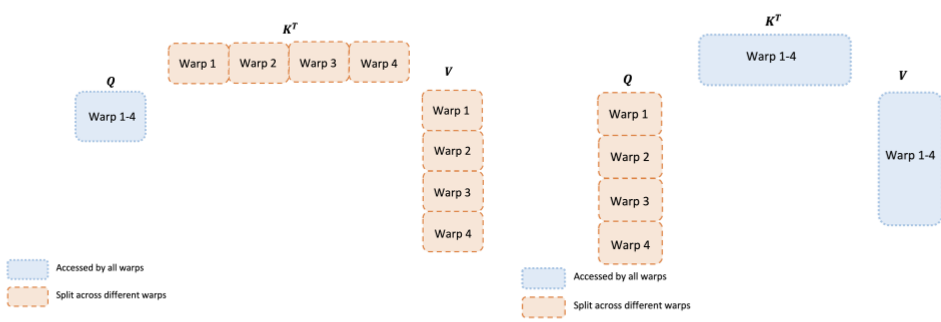
  end for

  Write $\mathbf{dK}_j, \mathbf{dV}_j$ to HBM.

end for

Return $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$.

图 10: FlashAttention-2 的反向计算流程示意图



(a) FlashAttention (b) FlashAttention-2

图 11: 前向计算中 warp 之间的工作划分示意图

2.2.3 warp 之间的工作划分

FlashAttention-1 是 KV 做外循环，Q 做内循环，而 FlashAttention-2 则是 Q 做外循环，KV 做内循环。

如11所示:

（a）FlashAttention-1 采取 split-K 方案：

1. 将 K 和 V 分配给 4 个 warp，Q 对所有 warp 可见 2. 每个 warp 计算一个 QK^T 的分块（这里每个 warp 只计算了列方向上的结果，行方向上的结果需要通过通信获得）3. 所有 warp 需要通信，将中间结果写回 shared memory，然后相加得到最终结果 4. 这个方案的缺点是需要大量的 shared memory 读写操作，以及 warp 之间的通信

（b）FlashAttention-2 采取 split-Q 方案：1. 将 Q 分配给 4 个 warp，K 和 V 对所有 warp 可见 2. 每个 warp 计算一个 QK^T 的分块（行方向上计算完全独立）3. 所有 warp 直接和 V 相乘得到对应的结果，无需通信 4. 这个方案的优点是减少了 shared memory 读写操作和 warp 之间的通信，提高了计算效率

3 FlashAttention-3

4 参考

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

FlashAttention-2: Faster and Memory-Efficient Exact Attention with IO-Awareness

FlashAttention-3: Faster and Memory-Efficient Exact Attention with IO-Awareness

FlashAttention 详解

[Attention 优化][2w 字] 原理篇: 从 Online-Softmax 到 FlashAttention V1/V2/V3