Flase attention-代码

参考https://www.zhihu.com/question/611236756/answer/3132304304

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.

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

2: 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.

3: 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.

4: 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.

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

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

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

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

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

10:         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), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

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

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

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

14:     **end for**

15: **end for**

16: Return $\mathbf{O}$.

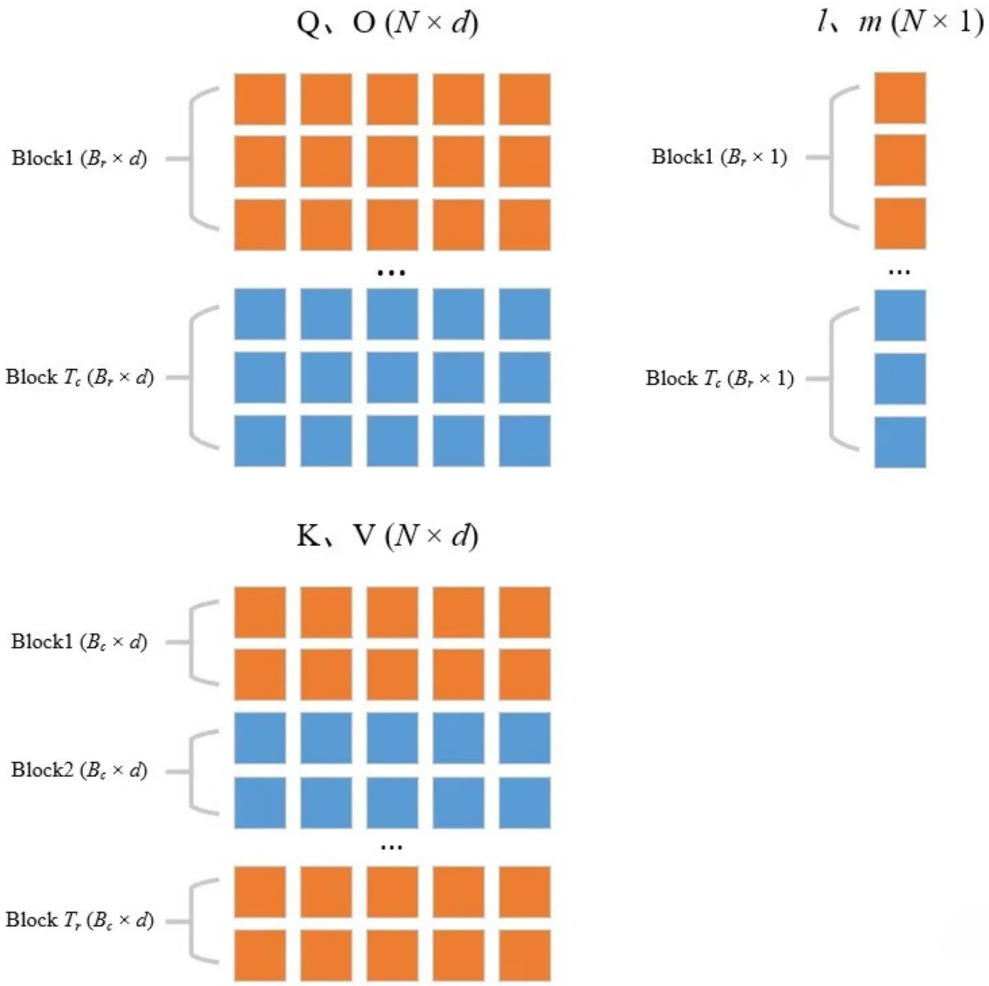
输入：在HBM中的$Q, K, V \in R^{N \times d}$，SRAM大小M

1-4行：计算Q O和K V的分块大小B_c和B_r，之所以是$ceil(M/4d)$，因为每次计算要存储的q k v o四个向量都是d维的，至少需要4d的空间。

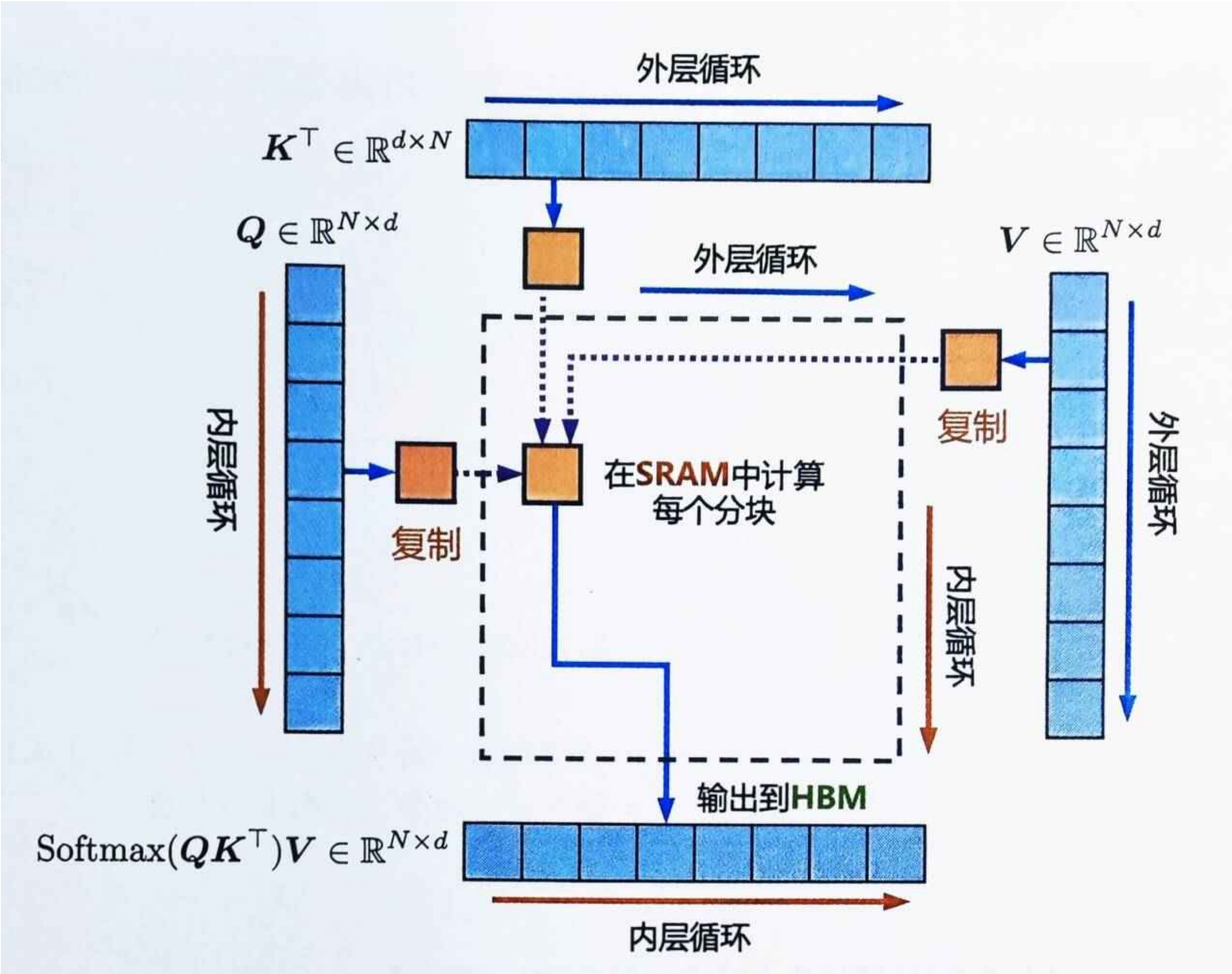
初始化最终输出$O \in R^{N \times d}$，N维向量l和m，分别记录每行行的exp求和、每行的最大值。

将矩阵 Q、O 沿着行方向分为 T_r 块，每一分块的大小为 $B_r \times d$； 将向量 l 和向量 m 分为 T_r 块，每一个子向量大小为 B_r。

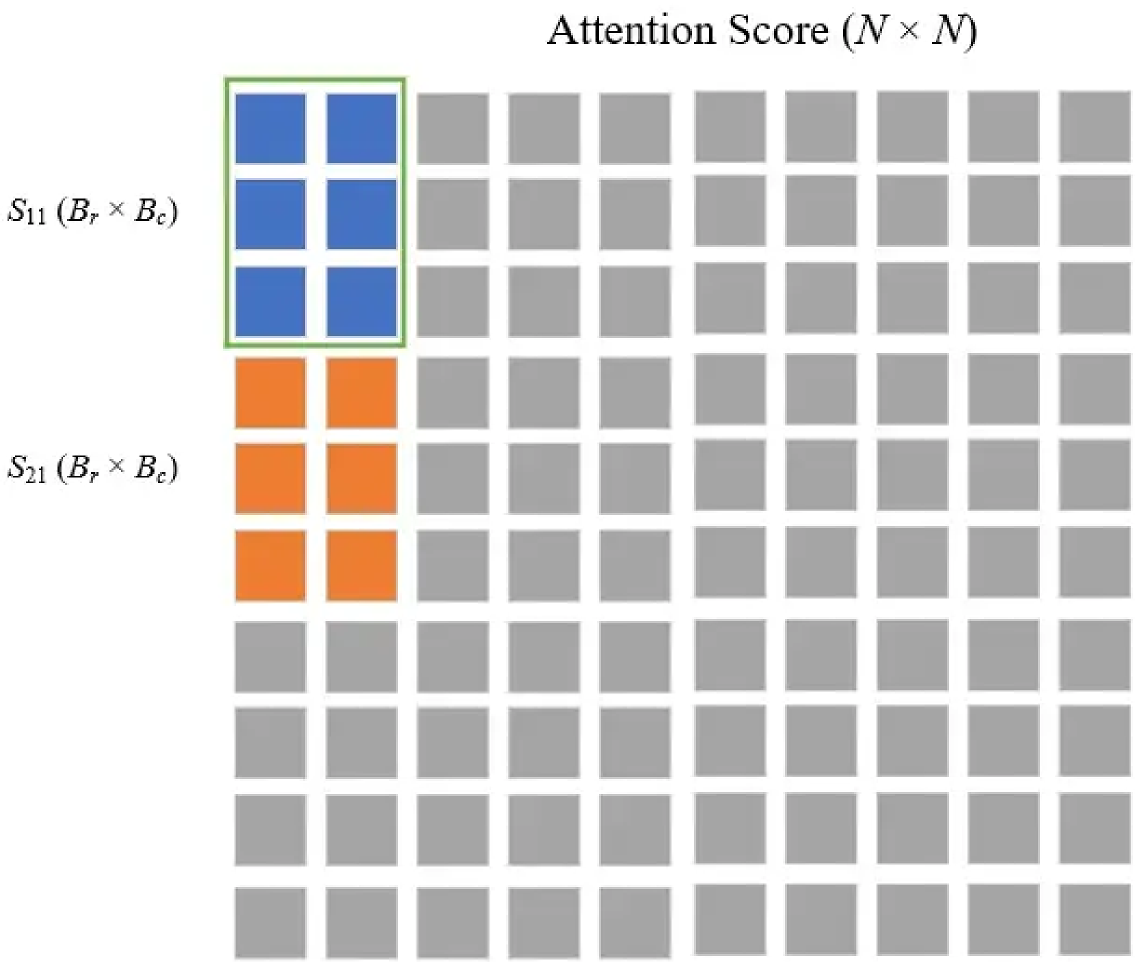
将矩阵 K, V 沿着行方向分为 T_c 块，每一块的大小为 $B_c \times d$。



5-8行：将K V作为外层循环，Q O作为内层循环



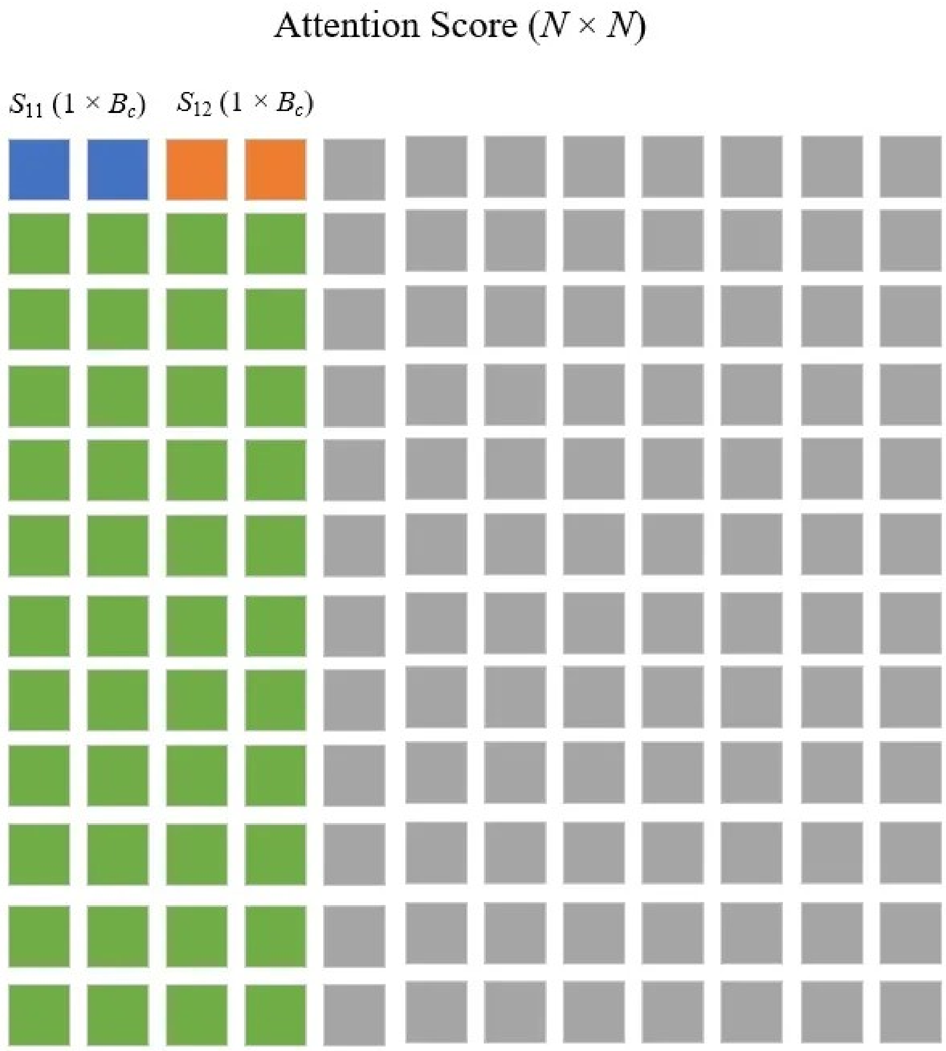
9行： 分块计算Attention score $s_{ij}=Q_iK_j^T \in R^{B_r\times B_c}$.标准的Transformer需要计算的Attention Score包括整个矩阵（灰色）。各分块计算出的Attention Score如图中蓝色和橙色区域所示.



10-11行： 计算局部 $\hat{m}_{ij}$ 和 $\hat{l}_{ij}$, 并利用其更新全局 m_i^{new} 和 l_i^{new}

12行：

为了更好地理解这一行的公式，首先得明白多行一起计算的目的是Batch计算。上图中的 S_{ij} 中虽然有3行，但是行之间的数据没有交互，真正的分块意义是在列上，因为softmax是沿着列方向进行的。为简便起见，先设置 $B_r = 1$ ，用 SM_i 表示每一行的softmax，初始化为全是0的 $1 \times d$ 的向量。



首先基于（9）-（12）计算出 S_{11} ，此时 SM_1 只有蓝色位置有值，相同方法处理下方的每一行。

$$(9)\ m(x^{(1)}) = \max([x_1^{(1)}, x_2^{(1)}, \ldots, x_B^{(1)}])$$

$$(10)\ f(x^{(1)}) = [e^{x_1^{(1)} - m(x^{(1)})}, \ldots, e^{x_B^{(1)} - m(x^{(1)})}]$$

$$(11)\ l(x^{(1)}) = \sum_i f(x^{(1)})_i$$

$$(12)\ \mathrm{softmax}(x^{(1)}) = \frac{f(x^{(1)})}{l(x^{(1)})}$$

接着计算 S_{12} ,此时橙色部分的计算就如（22），其中 $\hat{P}_{12}$ 就是公式（14）中的 $f(x^{(2)})$ 。而为了消除蓝色位置的局部性，采用(23)只需要标量 SM_1 和 l_1 而不用再把 $x^{(1)}$ 加载到SRAM中。将两项求和得到新的 SM_1^{new} :

$$(14)\ f(x^{(2)}) = [e^{x_1^{(2)} - m(x^{(2)})}, \ldots, e^{x_B^{(2)} - m(x^{(2)})}]$$

$$(22)\ softmax^{new}(x^{(2)}) = \frac{f^{new}(x^{(2)})}{l_{all}^{new}} = \frac{f(x^{(2)}) \cdot e^{m(x^{(2)})-m_{max}^{new})}}{l_{all}^{new}}$$

$$(23) = \frac{softmax(x^{(2)}) \cdot l(x^{(2)}) \cdot e^{m(x^{(2)})-m_{max}^{new})}}{l_{all}^{new}}$$

$$\mathrm{SM}_1^{(new)} = \frac{\mathrm{SM}_1 \cdot l_1 \cdot e^{m_1 - m_1^{new}}}{l_1^{new}} + \frac{\hat{P}_{12} \cdot e^{m_{12} - m_1^{new}}}{l_1^{new}}$$

> 这里的第一项长得像[x,y,0,0,0,0], 第二项长得像[0,0,p,q,0,0]，相加得到了[x,y,p,q,0,0]

首先计算出 O_1 ，接下来计算到橙色部分时更新 O_1^{new} 的方法类似，只要在每次动态更新完softmax，乘上其对应的V的值即可

$$O_1^{(new)} = \frac{\mathrm{O}_1 \cdot l_1 \cdot e^{m_1 - m_1^{new}}}{l_1^{new}} + \frac{\hat{P}_{12} \cdot e^{m_{12} - m_1^{new}}}{l_1^{new}} \cdot V_2$$

和上面的伪代码进行对比，可知伪代码中的公式仅仅是此公式的矩阵版本。

13行： 更新 l_i 和 m_i 。

计算flash attention的内存访问复杂度：

- 内循环访问Q，开销为Nd
- 外循环执行 T_c 次， $T_c = \left\lceil \frac{N}{B_c} \right\rceil = \left\lceil \frac{4dN}{M} \right\rceil$ ，总开销为 $O(N^2 d^2 M^{-1})$.
- 由于分配给一次运算的M=100KB远大于d（一般为64或128），因此内存访问复杂度也低于传统的attention

总结

- **为什么加快了计算？Fast**
  - 降低了耗时的HBM（显存）访问次数。采用Tiling技术分块从HBM加载数据到SRAM缓存进行融合计算。
- **为什么节省了内存？Memory-Efficient**
  - 不再对中间矩阵S，P进行存储。在反向的时候通过Recomputation重新计算来计算梯度。
- **为什么是精准注意力？Exact Attention**
  - 算法流程只是分块计算，无近似操作。