Flash attention formula

Original Attention

```
def scaled_dot_product_attention(query, key, value, attn_mask=No
        is_causal=False, scale=None, enable_gqa=False) -> torch
    L, S = query.size(-2), key.size(-2)
    scale_factor = 1 / math.sqrt(query.size(-1)) if scale is Nor
    attn_bias = torch.zeros(L, S, dtype=query.dtype)
    if is_causal:
        assert attn_mask is None
        temp_mask = torch.ones(L, S, dtype=torch.bool).tril(diag
        attn_bias.masked_fill_(temp_mask.logical_not(), float("-
        attn_bias.to(query.dtype)

    if attn_mask is not None:
        if attn_mask.dtype == torch.bool:
            attn_bias.masked_fill_(attn_mask.logical_not(), floa
        else:
            attn_bias += attn_mask

    if enable_gqa:
        key = key.repeat_interleave(query.size(-3)//key.size(-3)
        value = value.repeat_interleave(query.size(-3)//value.si

    attn_weight = query @ key.transpose(-2, -1) * scale_factor
    attn_weight += attn_bias
    attn_weight = torch.softmax(attn_weight, dim=-1)
    attn_weight = torch.dropout(attn_weight, dropout_p, train=Tr
    return attn_weight @ value
```

$S = QK^T$, $P = softmax(X)$, $O = PV$

Standard attention implementations materialize the matrices S and P to HBM, which takes $O(N_2)$ memory.

As some or most of the operations are memory-bound (e.g., softmax), the large number of memory accesses translates to slow wall-clock time.

Online softmax

safe softmax

$\frac{e^{x_i-m}}{\sum_{j=1}^{N} e^{x_j-m}}$

1. for $i \leftarrow 1, N$
   - $m_i = max(m_{i-1}, x_i)$
2. for $i \leftarrow 1, N$
   - $d_i = d_{i-1} + e^{x_i - m_N}$
3. for $i \leftarrow 1, N$
   - $a_i = \frac{e^{x_i-m_N}}{d_N}$

online softmax

$d'_i = \sum_{j=1}^{j} e^{x_j - m_i} = (\sum_{j=1}^{j-1} e^{x_j-m_i}) + e^{x_i-m_i}$
$= (\sum_{j=1}^{j-1} e^{x_j-m_{i-1}}) e^{m_{i-1}-m_i} + e^{x_i-m_i}$
$= d'_{i-1} e^{m_{i-1}-m_i} + e^{x_i-m_i}$

1. for $i \leftarrow 1, N$
   - $m_i = max(m_{i-1}, x_i)$
   - $d'_i = d'_{i-1} e^{m_{i-1}-m_i} + e^{x_i-m_i}$
2. for $i \leftarrow 1, N$
   - $a_i = \frac{e^{x_i-m_N}}{d'_N}$

Flash Attention

Multi-pass self-attention

inputs: $Q[k, :], K^T[:, i], V[i, :]$

output: $O[k, :]$

1. for $i \leftarrow 1, N$

$x_i = Q[k,:]K^T[:,i]$

$m_i = max(m_{i-1}, x_i)$

$d'_i = d'_{i-1}e^{m_{i-1}-m_i} + e^{x_i-m_i}$

2. for $i \leftarrow 1, N$

   $a_i = \frac{e^{x_i-m_N}}{d'_N}$

   $o_i = o_{i-1} + a_i V[i,:]$

3. $O[k,:] = o_N$

$o_i = \sum_{j=1}^{i}(\frac{e^{x_j-m_N}}{d'_N}V[j,:])$

let $o'_i = \sum_{j=1}^{i}\frac{e^{x_j-m_i}}{d'_i}V[j,:]$

$= (\sum_{j=1}^{i-1}\frac{e^{x_j-m_i}}{d'_i}V[j,:]) + \frac{e^{x_i-m_i}}{d'_i}V[j,:]$

$= (\sum_{j=1}^{i-1}\frac{e^{x_j-m_{i-1}}}{d'_{i-1}}\frac{e^{x_j-m_i}}{e^{x_j-m_{i-1}}}\frac{d'_{i-1}}{d'_i}V[j,:]) + \frac{e^{x_i-m_i}}{d'_i}V[j,:]$

$= (\sum_{j=1}^{i-1}\frac{e^{x_j-m_{i-1}}}{d'_{i-1}}V[j,:])\frac{d'_{i-1}}{d'_i}e^{m_{i-1}-m_i} + \frac{e^{x_i-m_i}}{d'_i}V[j,:]$

$= o'_{i-1}\frac{d'_{i-1}e^{m_{i-1}-m_i}}{d'_i} + \frac{e^{x_i-m_i}}{d'_i}V[j,:]$

Flash attention

1. for $i \leftarrow 1, N$

   $x_i = Q[k,:]K^T[:,i]$

   $m_i = max(m_{i-1}, x_i)$

   $d'_i = d'_{i-1}e^{m_{i-1}-m_i} + e^{x_i-m_i}$

   $o'_i = o'_{i-1}\frac{d' i-1 e^{mi-1-m_i}}{d'_i} + \frac{e^{x_i-m_i}}{d'_i}V[j,:]$

2. $O[k,:] = o_N$

Flash attention w/ Tiling

1. for $i \leftarrow 1, \#tiles$

   $x_i = Q[k,:]K^T[:,(i-1)b:ib]$

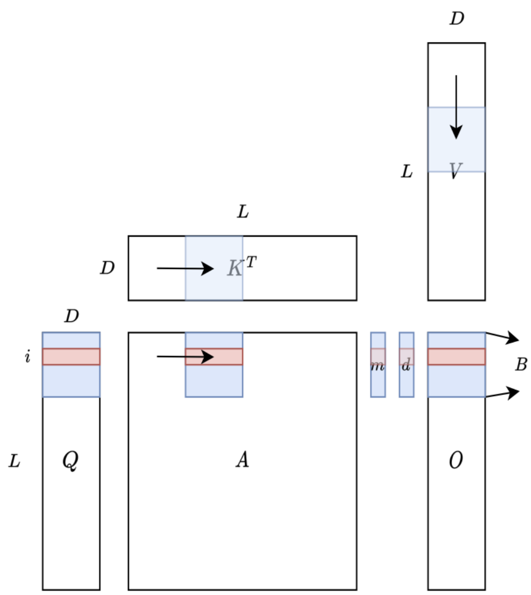
   $m_i^{(local)} = max_{j=1}^{b}(x_i[j])$

   $m_i = max(m_{i-1}, m_i^{(local)})$

$$d_i' = d_{i-1}' e^{m_{i-1}-m_i} + \sum_{j=1}^{b} e^{x_i[j]-m_i}$$

$$o_i' = o_{i-1}' \frac{d'i-1e^{m_{i-1}-m_i}}{d_i'} + \sum_{j=1}^{b} \frac{e^{x_i[j]-m_i}}{d_i'} V[j+(i-1)b,:]$$

2. $O[k,:] = o_{N/b}'$



The overall SRAM memory footprint depends only on B and D and is not related to L.

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 $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ 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 $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
    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 $\mathbf{O}$.