Attentions and How to Run Them Fast

By Kaizhao

Agenda

- Basic math stuffs
- Exact Attentions
    - Flash Attention 1-3
    - Ring Attention
    - Tree Attention
- Attention as RNN
- Approximate versions
    - StreamingLLM
    - Linear Attention (fast weight perspective)
- Speculative Decoding
- Trend and future

Basic Math stuffs

$$\overrightarrow{S} = [s_0, s_1, s_2, ..., s_T]$$

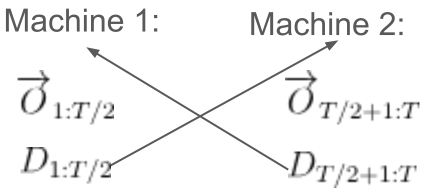
$$Softmax(\overrightarrow{S})_t = \frac{e^{\overrightarrow{S}_t}}{\sum_t^T e^{\overrightarrow{S}_t}}$$

Stare at it for a few seconds, tell me what you see?


wait, it's just RNN with growing memory?
always has been
imgflip.com

Basic MLE interview question 2:

Now imagine you have two machines, and a very long $\vec{S}$, how do you parallelize your softmax?



Corrected version:

$$\vec{O}'_{1:T/2} = \vec{O}_{1:T/2} \cdot \frac{D_{1:T/2}}{D_{1:T/2} + D_{T/2+1:T}}$$

$$\vec{O}'_{T/2+1:T} = \vec{O}_{T/2+1:T} \cdot \frac{D_{T/2+1:T}}{D_{1:T/2} + D_{T/2+1:T}}$$

Basic MLE interview question 2.5:

What if you have N machines, how do you build an attention mechanism that can run at $O(T/Nlog(T/N))$

What is the communication cost?

What is the memory complexity?

Exercise left for audience offline

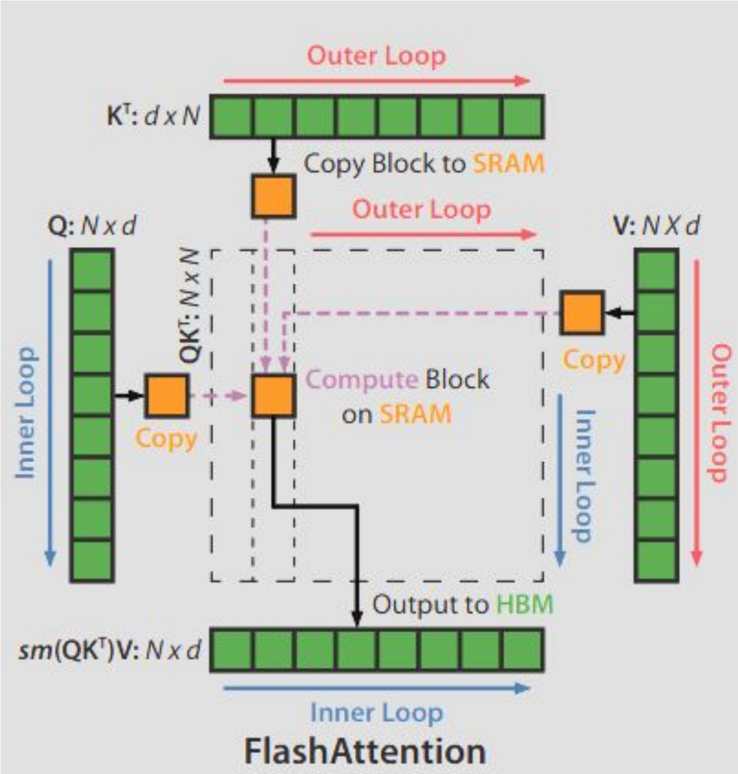
Basic MLE interview question 3:

What function's derivative is Softmax?

$$\frac{\partial\, logsumexp(S + x)}{\partial x}\Big|_{x=0} = Softmax(S)$$

Remember what we've learnt so far and let's get started

Flash Attention



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.

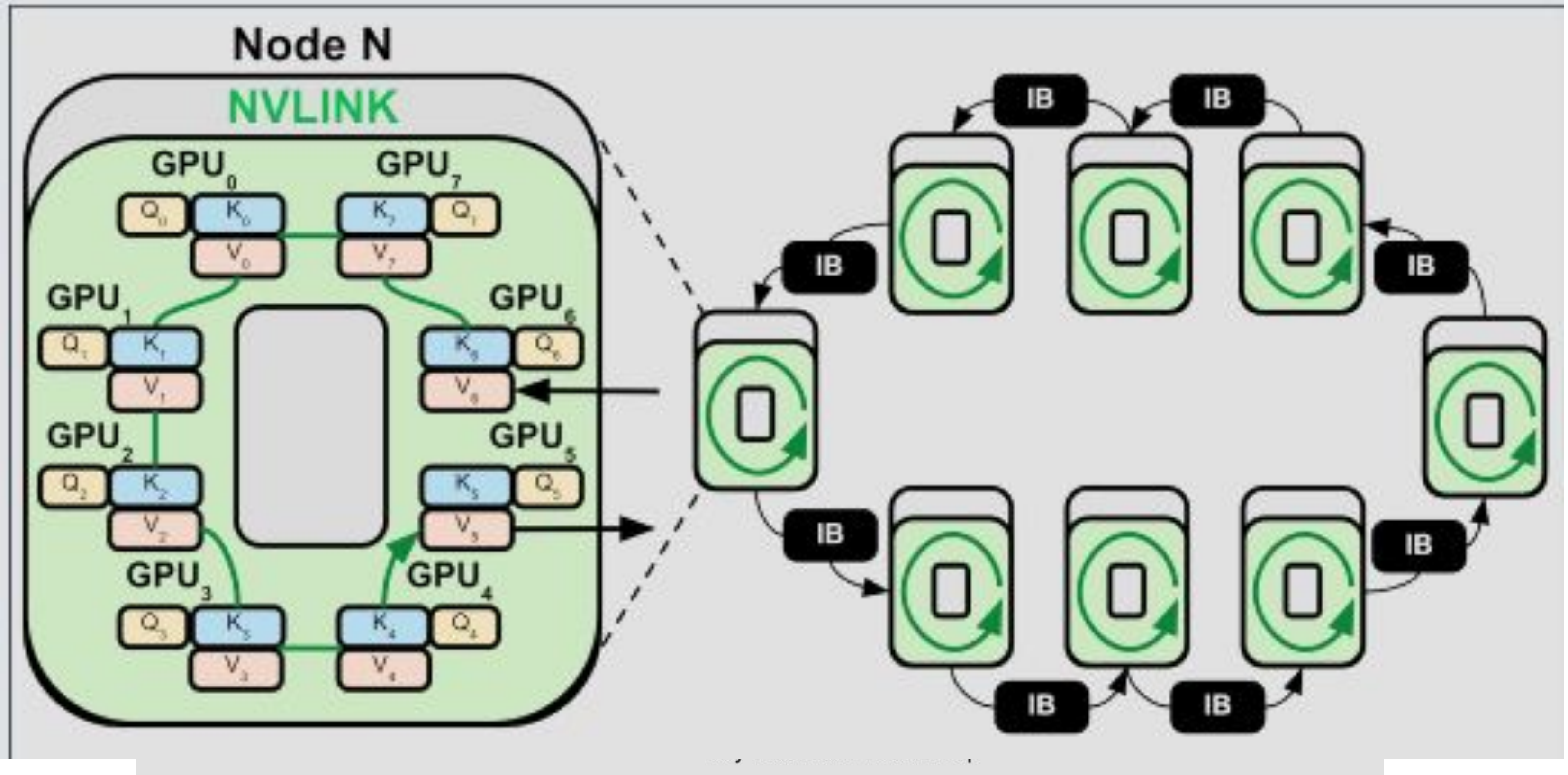
    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.

  end for

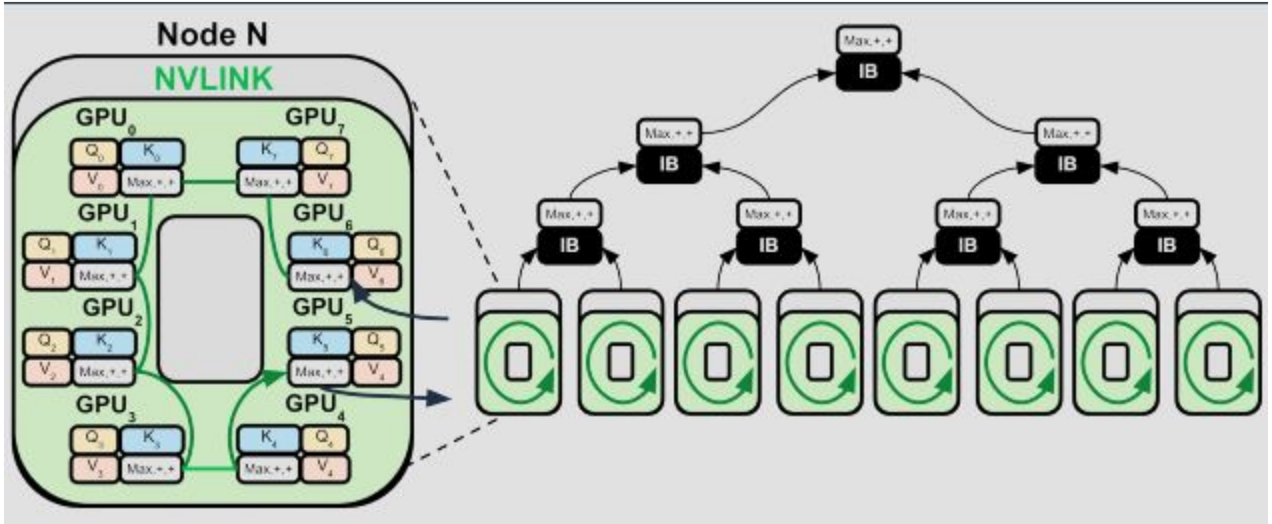
end for

Return $\mathbf{O}$.

Ring Attention



Tree Attention



Summary

All the modern memory optimization/parallelization leverage the mathematical property of softmax, which you learnt today, with some twists to the hardware/software settings.

I also encourage you to work out the numerically stable version of it yourself. You just need keep one more variable the **max** and correct for it.

In case we have more time: speculative decoding

Sample from a smaller model $p(x)$ to guess what a larger target model $q(x)$ wants to say

- If $p(x') > q(x')$, accept the token
- else reject it, and sample again from p - q renomralized

Proof of correctness

We will now show that for any distributions $p(x)$ and $q(x)$, the tokens sampled via *speculative sampling* from $p(x)$ and $q(x)$ are distributed identically to those sampled from $p(x)$ alone. Let β be the acceptance probability (Definition 3.1).

Note that as $p'(x) = norm(max(0, p(x) - q(x))) = \frac{p(x)-min(q(x),p(x))}{\sum_{x'}(p(x')-min(q(x'),p(x')))} = \frac{p(x)-min(q(x),p(x))}{1-\beta}$, the normalizing constant for the adjusted distribution $p'(x)$ is $1 - \beta$, where the last equation follows immediately from Lemma 3.3 and Theorem 3.5.

Now:

$$P(x = x') = P(guess\ accepted, x = x') + P(guess\ rejected, x = x')$$

Where:

$$P(guess\ accepted, x = x') = q(x') \min(1, \frac{p(x')}{q(x')}) = \min(q(x'), p(x'))$$

And:

$$P(guess\ rejected, x = x') = (1 - \beta)p'(x') = p(x') - \min(q(x'), p(x'))$$

Overall:

$$P(x = x') = \min(p(x'), q(x')) + p(x') - \min(p(x'), q(x')) = p(x').$$

As desired. $\square$

Next episode preview? If you still let me present