Agenda:

0) Some more on how flash attn uses GPU
1) Flash Decoding
2) Speculative Decoding
3) Paged Attention
4) IF time: KV cache compression

HOW FLASH ATTENTION USES THE GPU:

- key idea recap: compute attention in BLOCKS to reduce global memory access

- Main techniques:
  (1) Tiling - restructure algorithm to load q, k, v by block, from global HBM to streaming multiprocessor shared mem
  (2) recomputation - Don't store dotproduct (QK^T) or Attn matrix (softmax (QK^T)) from FW pass; recompute in bw pass
  (3) And fusion and efficient use of tensor cores

* tiling relies on these rescaling factors:

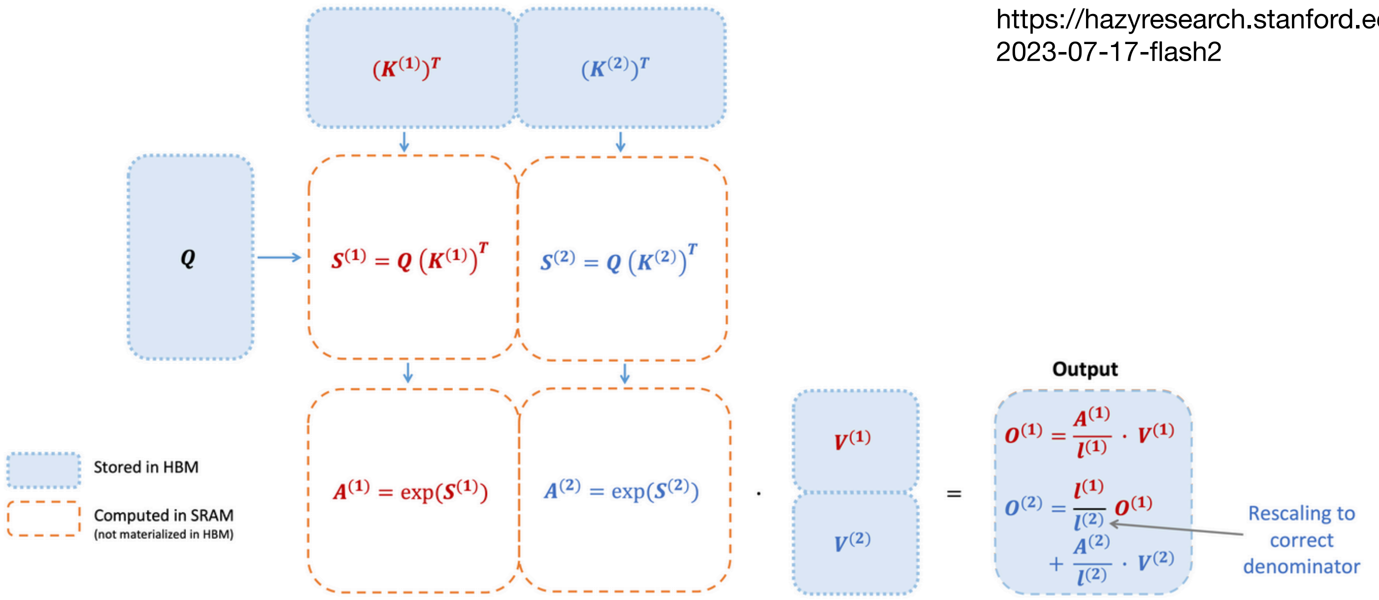
$$\text{softmax}(S_1, S_2) = [\alpha_1 \cdot \text{softmax}(S_1), \alpha_2 \cdot \text{softmax}(S_2)]$$

tiles of sgemm dot product output

$$\text{softmax}(S_1, S_2) \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \alpha_1 \text{softmax}(S_1) \cdot V_1 + \alpha_2 \cdot \text{softmax}(S_2) \cdot V_2$$

- and instead of storing $S_1 | S_2$ or softmax (S_1, S_2), JUST store rescaling factors $(a_1, a_2 \ldots) \rightarrow$ sized

https://hazyresearch.stanford.edu/blog/2023-07-17-flash2



($l^{(1)} = \sum_i \exp(S^{(1)})_i$, $l^{(2)} = l^{(1)} + \sum_i \exp(S^{(2)})_i$) $\rightarrow$ to calculate these, need to wait for all local comps to finish

*in FA, within a SINGLE BLOCK (so not all in parallel), we compute: (later we'll revisit whether we should do in //)

1) $\rightarrow$ numerators: $A^{(1)} = e^{S^{(1)}}$

$\rightarrow$ local denom: $L^{(1)} = \sum_{i=1}^{N} e^{S^{(1)}_i}$ $\rightarrow$ 1st rescaling factor

2) $\rightarrow$ local numerator: $A^{(2)} = e^{S^{(2)}}$

$\rightarrow$ 2nd rescaling factor: $L^{(2)} = \sum_{i=1}^{N} e^{S^{(2)}_i} + L^{(1)}$

...

and so on (for all tiles)

* but: FA1 still has some inefficiencies w.r.t suboptimal work partitioning btwn dif. thread blocks and warps

↳ causes low occupancy or unecessary HBM reads writes

- Better Parallelism in FA2:

→ A100 has 108 dif. streaming multiprocessors:

how do we use these?

- idea 1: split dif. attention heads across dif. SMs

  ↳ remember: each head operates on a split dimension → $\frac{d_{TOTAL}}{\# \text{ heads}}$

  → so Ks / Vs / Qs are independent

- what if we don't have 108 heads (usually 16-64)?

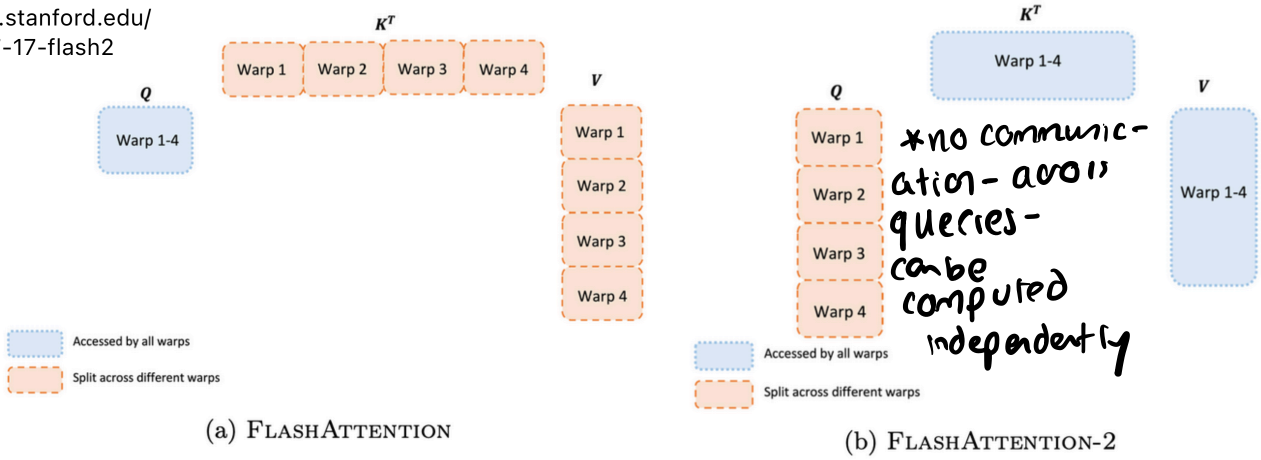
  ↳ idea 2: parallelize over sequence length (N)

  - To do this, we can split Q and assign dif. queries to dif. threadblock
  - Dot-product, causal mask, softmax is all row wise ⇒ (per query!)

- idea 3: better warp scheduling!

* within thread block, FA1 splits K/V across 4 warps, keeps Q accessible by all warps

- this means the dif. warps need to communicate https://hazyresearch.stanford.edu/blog/2023-07-17-flash2



(a) FlashAttention (b) FlashAttention-2

now, lets move onto inference...

· FLASH DECODING

* in training - FA(1 and 2) parallelizes BOTH across batch size and sequence / query length (you have access to entire sequence)

* what about inference, specifically, autoregressive decoding / next token prediction?

- can be thought of largely in 2 phases:

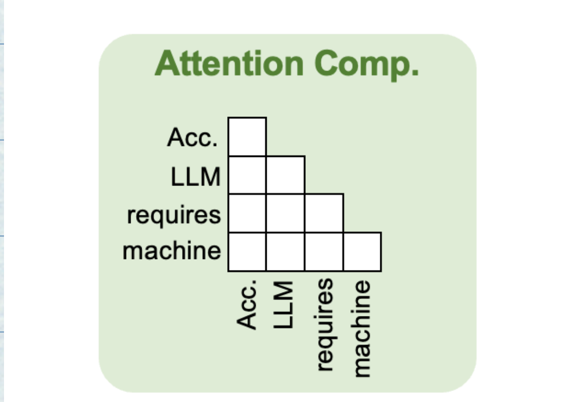
(1) Prefill - process all input tokens at once (e.g., prompt)

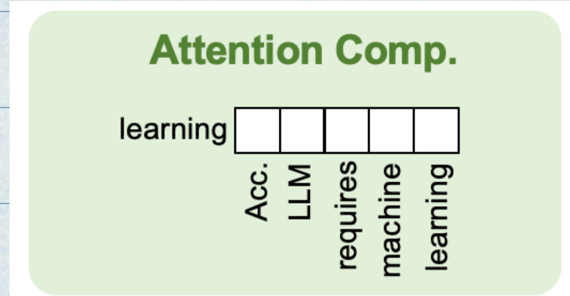
(2) Decode - process single token generated from previous iteration

- uses attention keys / values from prev iterations (which we can cache!)

- can we apply flash attention to inference?

(CS-15442 slides at CMU)



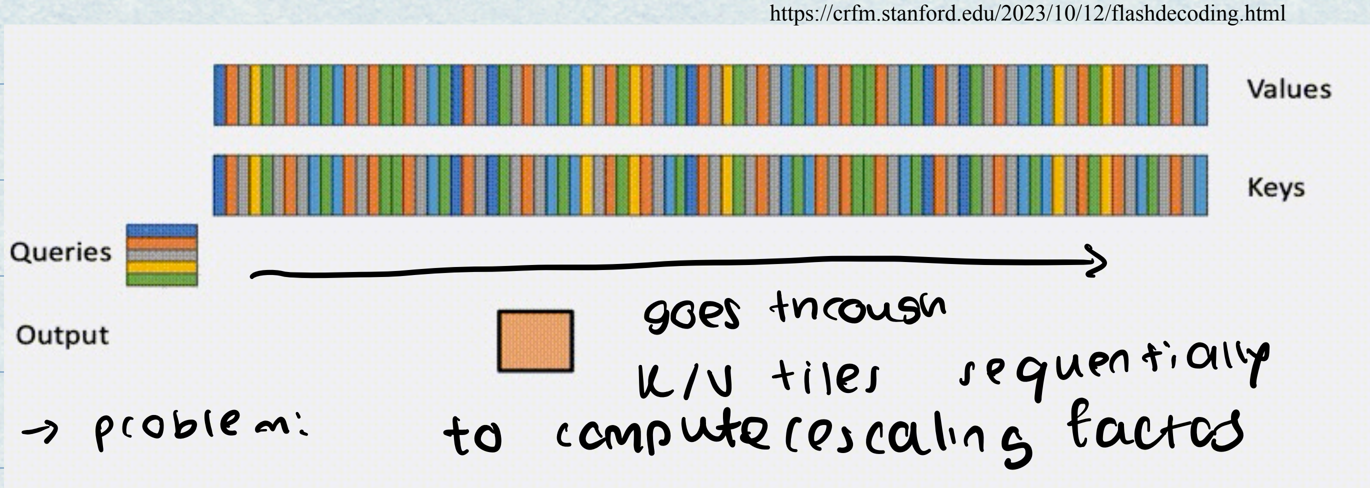


* yes! for prefill

* not really for decode- we have 1 query!

- recall: FA tries to split Q over dif. thread blocks, and within a thread block, to dif. warps
  - ↳ if we have 1 query, can't do this!
- can potentially split via batches- but batch size could be smaller, especially when prompts are long.

https://crfm.stanford.edu/2023/10/12/flashdecoding.html



goes through K/V tiles sequentially

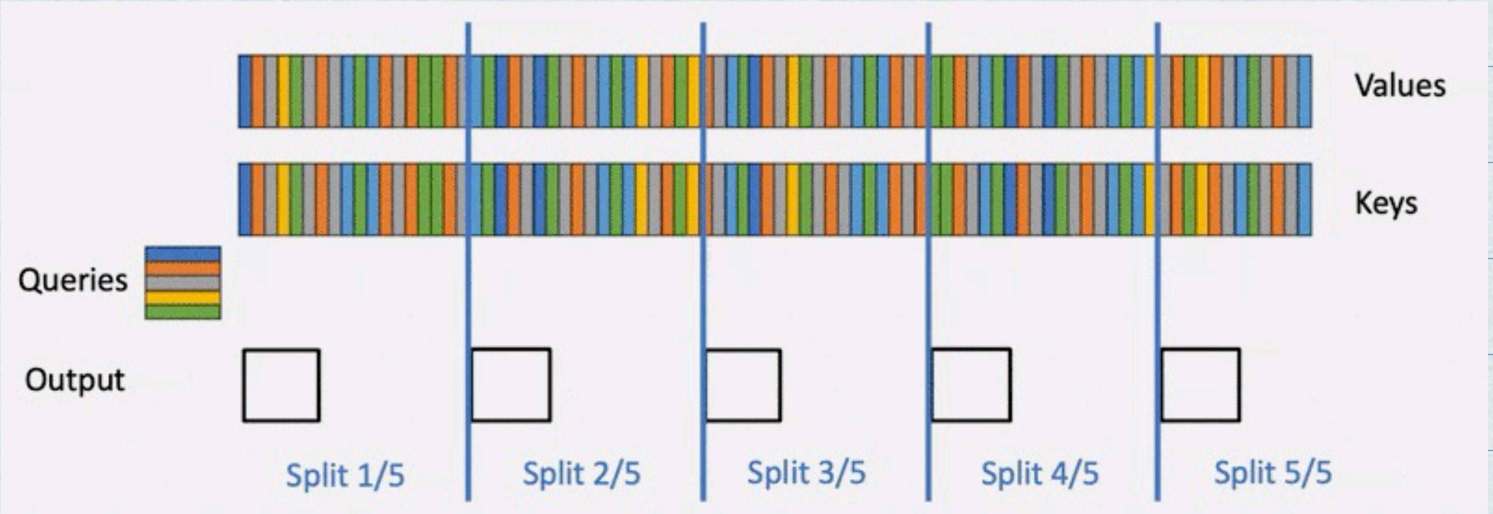
→ problem: to compute rescaling factors

-FA, within a single query, goes through K/V sequentially!

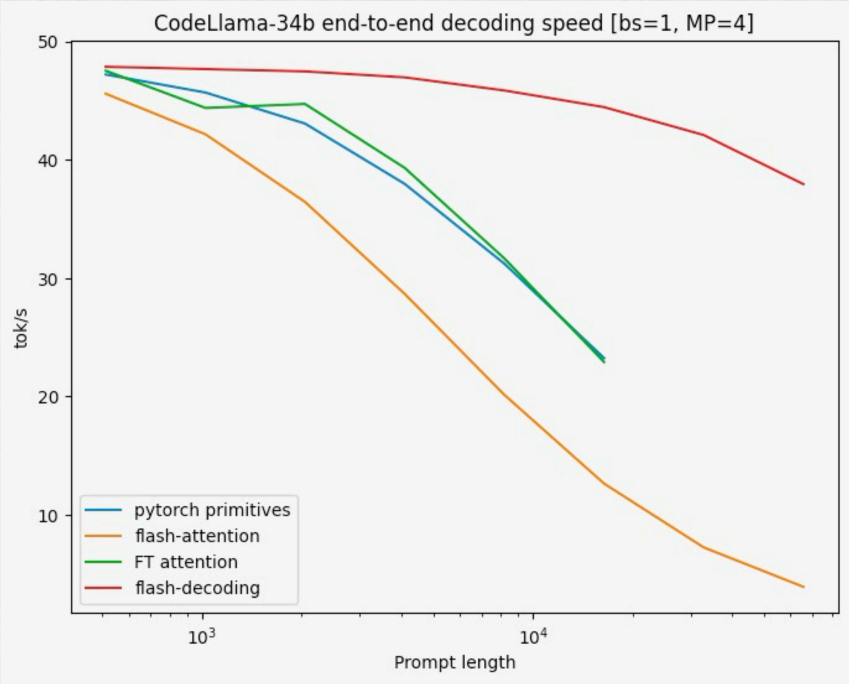
* flash decoding: solves this and lets

us parallelize over keys /values:

https://crfm.stanford.edu/2023/10/12/flashdecoding.html



1. splits K/V into smaller chunks
2. compute attn of query w/ each split w/ FA.
   write one extra scalar per row: log-sum-exp of attn values (scaling factor)
3. compute output by rescaling over splits w/ rescaling factor



Result:
as prompt len increases, flash decoding has better perf!

PART 3 OF LECTURE : speculative decoding

-problem: generally, decoding (ignoring FD) has

LOW arithmetic intensity

↳ load KV cache

↳ load weights W_Q, W_K, W_V

↳ load weights for MLP (multilayer perceptron)

→ run attn / feedforward to sample ONE

TOKEN

* Dominant cost here is these memory

options

- intuition: let's TRY to guess al output using

a small model

cmu slides

# Parameters	175B	13B	2.7B	760M	125M
TriviaQA	**71.2**	57.5	42.3	26.5	6.96
PIQA	**82.3**	79.9	75.4	72.0	64.3
SQuAD	**64.9**	62.6	50.0	39.2	27.5
latency	20 s	7.6s	2.7s	1.1s	**0.3s**
# A100s	10	1	1	1	**1**

Comparing multiple GPT-3 models*

- Large models are generally slower (and require

more resources), but have better accuracy!

- Small models are cheap / fast - but have

lower accuracy

- how to use small model?

1. Predict LM's output w/ SSM (small spec-

ulative model).

2. use large LM to VERIFY output of smaller model

↳ this can be done in parallel/bulk on entire sequence so far + # of speculated tokens

- after we have outputs on seq so far + speculated tokens:

- verify, at each step, if output matches NEXT speculative token

- if some of the speculative tokens are correct - we've done 2x or 3x more work at each timestep in the time for 1 token

* 2-3x speedup on chinchilla (70B), T5 (11B), LaMDA (137B)

HOW DO WE GUESS/SPECULATE on next few tokens?

→ guessing should be fast

→ speedup ONLY if model agrees

* Option 1: smaller draft model (described above):

- trained on same data (or distilled)

- can tune size

- But:
  - managing another model could complicate infra
  - additional mem. consumption

* option 2: medusa heads
  - Train separate heads that don't predict just next token, but next-next or next-next-next
  * con: getting back to this "joint" distr. idea - could become really bad really fast

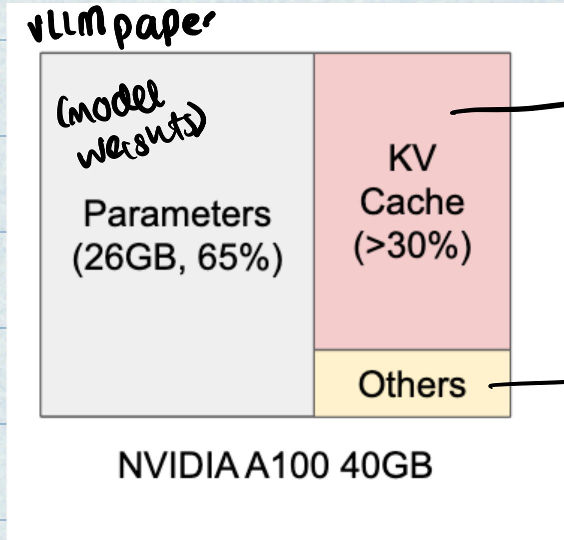
* option 3: Lossy optimization
  - techniques such as int4 quantization, skip layers, skip attn heads, stop early
  - Pros:
    - guesses can be extremely correlated w/ non-lossy model
    - no additional models to manage

* But:
  - code can be complicated

* you will explore speculative decoding in the project!

LAST PART OF LECTURE: PAGED ATTENTION

- vLLM's original main technique
- core question: how do we MANAGE GPU memory (for KV cache, specifically)

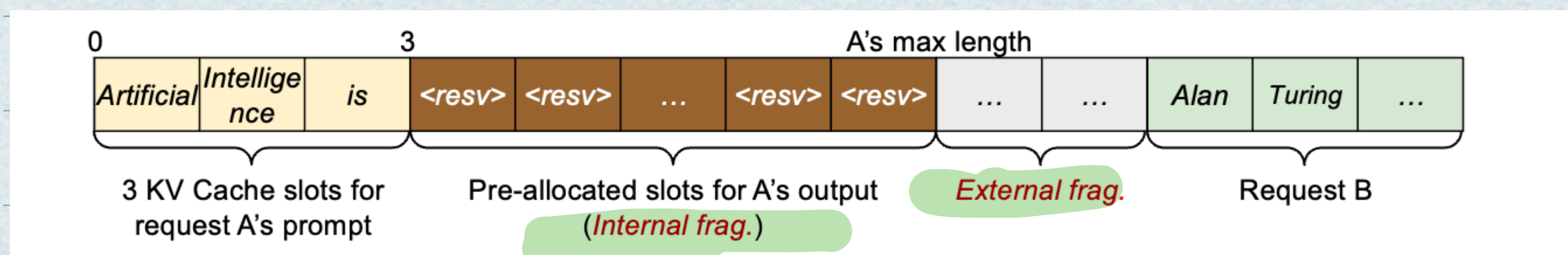


stored K/V vectors of previous tokens

intermediate activations

- a few things make memory management hard:
  1. For each request doing autoregressive decoding, we **DON'T KNOW** how long the generation will be
  2. Across requests, users can provide dif. max request lengths

* previous engines allocated contiguous spaces of memory in KV cache for each request's max memory:



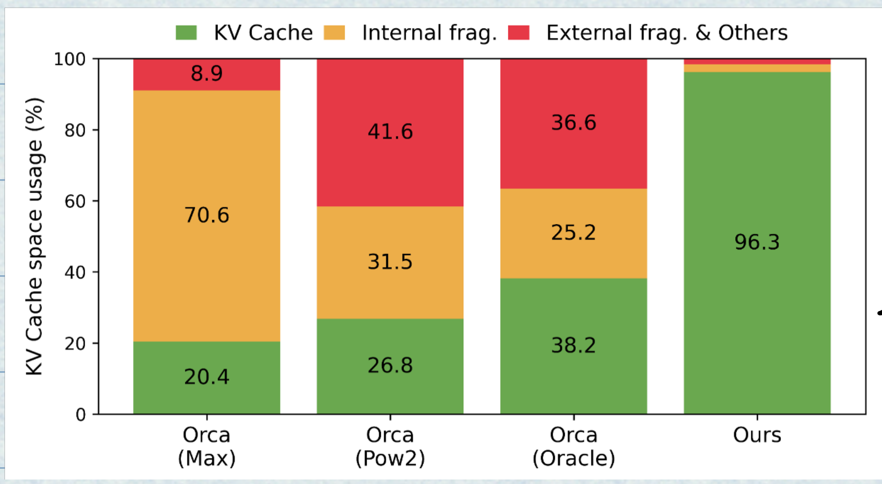
* some observations:

- A and B have DIF max request lengths
- engine pre-allocates slots up to max length

- this "static memory management":
  - leads to 2 types of fragmentation
    1) Internal frag: LLm does not predict up to max request length. wasted space
    2) External frag: when allocating memory requests of dif. sizes, there will always be some unused space

+go study memory allocation schemes for more info

FRAGMENTATION IS SIGNIFICANT:

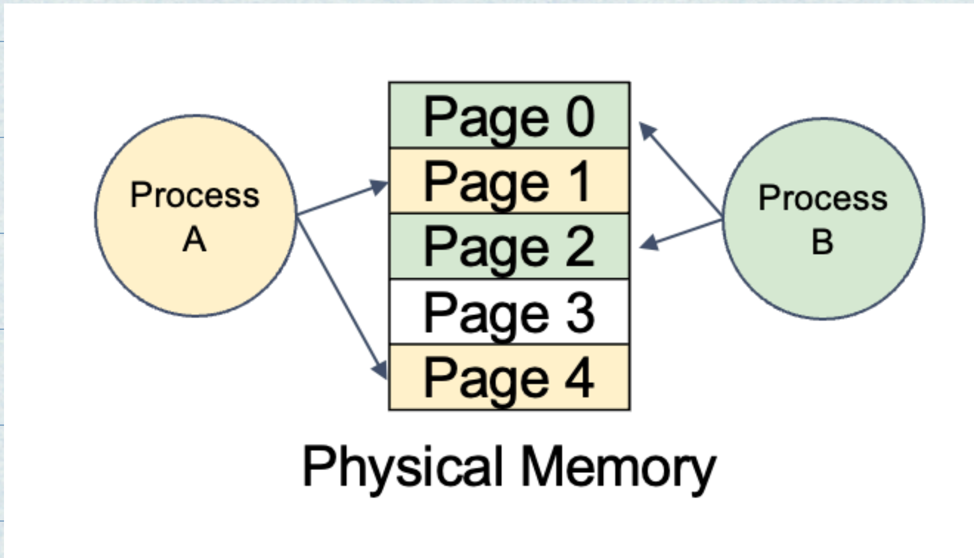


(from vLLM paper):
- in previous baselines - only 20-40% of available memory for KV cache is used for useful data

- GPU device memory is the bottleneck - we really shouldn't be wasting any!

KEY INSIGHT OF VLLM WORK:

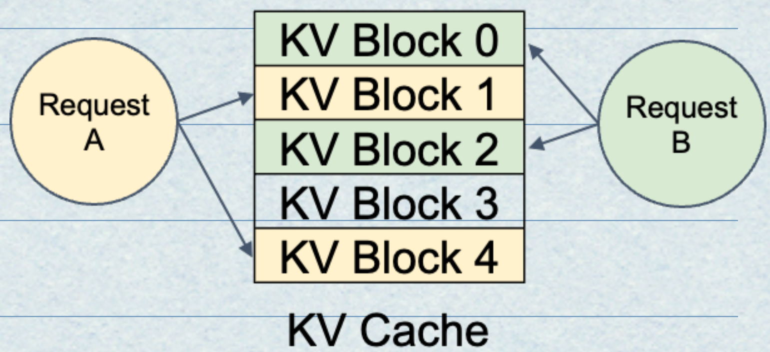
- we've already solved a similar problem w/ OS-level paging and virtual memory:
  - OS divides up memory into "pages"
  - will map program LOGICAL pages to physical pages
    - ↳ so logical pages need not be physically contiguous:



But each process THINKS it has logically contiguous memory

Let's do something similar for the KV cache!

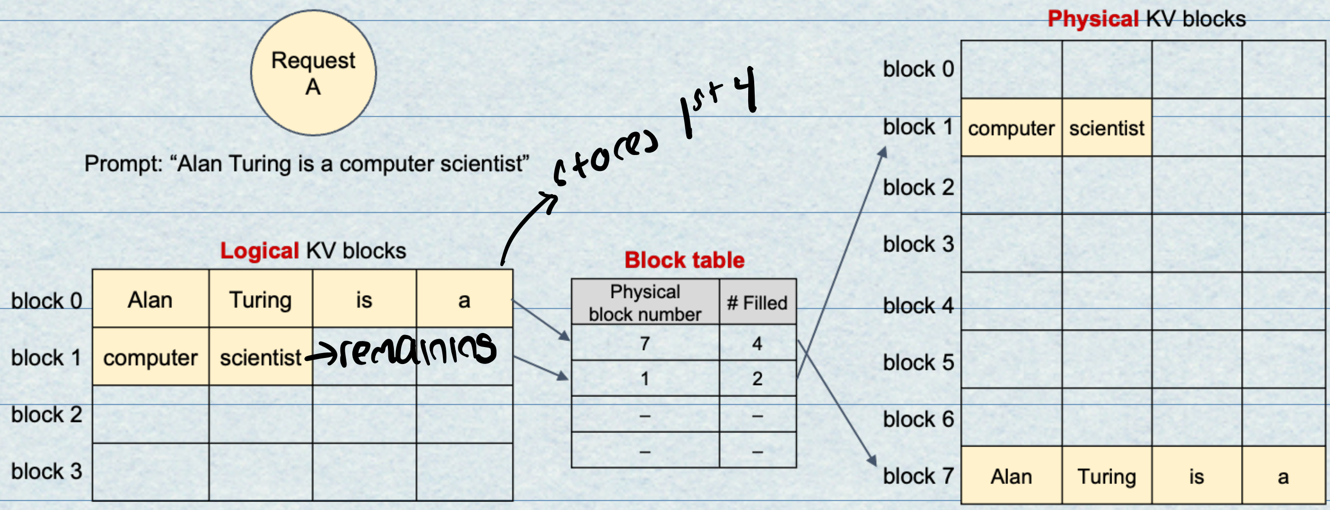
- partition KV cache into "blocks"
- to minimize fragmentation, blocks for dif. requests do NOT have to live next to each other:



KV-block = fixed-size contiguous memory chunk that stores KV states from left to right.

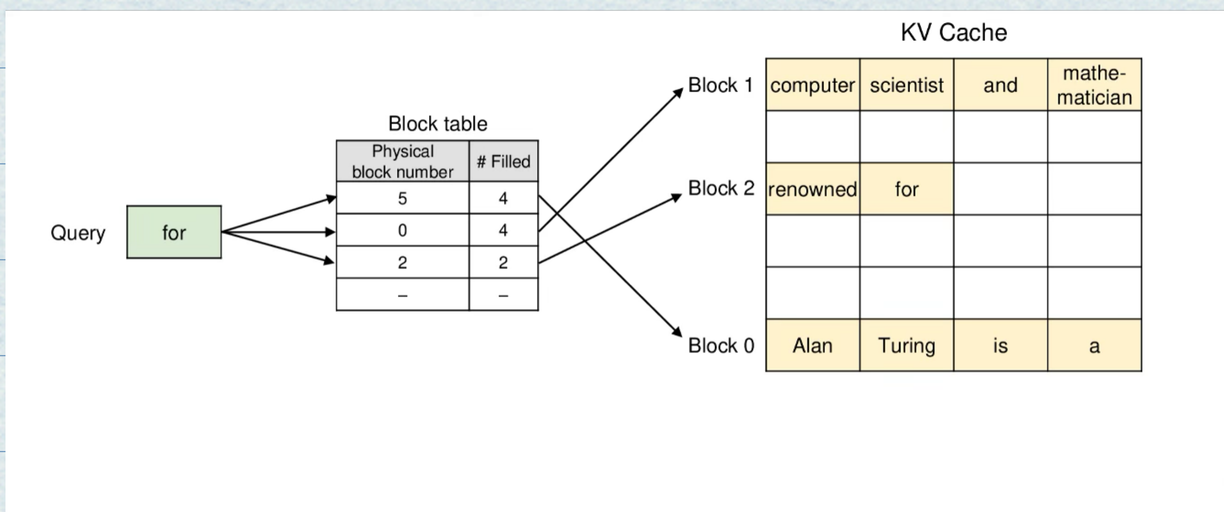
↳ block-size of 4 would be able to store k/v projections for 4 tokens

* now we can VIRTUALIZE THE KV CACHE:



↳ request "thinks" B0 and B1 are contiguous, but they are physically not: (w/ block/page table in between)

* How do we compute attention?



- we need to compute attention for query "for" - w/ respect to each token before; but KV blocks cannot be fetched all at once!

- iterate over blocks containing KV cache for this request:

    - for each logical block, set row of block table

    - look up physical block location

    - calculate attn of query w.r.t those keys/values:

        - we are trying to calculate an attn score for each token so far:

    - consider block $\underline{j}$ and we are trying to compute attn score of token i (i is query)

contains keys/values for token index:

$B\cdot(j-1)+1 \ldots B\cdot j$

↳ for $B=4$;

$K_1 \rightarrow$ keys for tokens $1 \ldots 4$

$K_2 \rightarrow$ keys for token $5 \ldots 8$

$V_1 \rightarrow$ vals for token $1 \ldots 4$

$V_2 \rightarrow$ vals for token $5-8$

$1 = 4(1-1)+1$

$5 = 4(2-1)+1$

$1\times D \quad D\times B$

$$\underbrace{A_{ij}}_{} = \frac{e^{\frac{q_i \cdot K_j^T}{\sqrt{D}}}}{\sum_{t=1}^{i/B} e^{(q_i \cdot K_t^T / \sqrt{D})}}$$

attn scores of token i, block j

$$O_i = \sum_{j=1}^{i/B} V_j A_{ij}^T$$

→ iteration for $t=1 \ldots i/B$ → all blocks so far! → gets us proper softmax sum

→ in O_i: again iterate from $j=1$ to i/B to get all values

* One more note: how do we store KV for new tokens?

1. Try to fill in block of where prefix ended
2. otherwise can always allocate a new block!

Further Resources / references:

Cs 229s slides with speculative decoding:https://docs.google.com/presentation/d/1PV0cKnzcHRCAbJy3OtER1UdJME7T7WBEqLLYmKWJR4g/edit#slide=id.g289e2a5aa10_1_144

Flash attention 2 blog post: https://hazyresearch.stanford.edu/blog/2023-07-17-flash2

Flash decoding blog post: https://crfm.stanford.edu/2023/10/12/flashdecoding.html

VLLM sosp paper: https://arxiv.org/pdf/2309.06180

CMU CS442 slides: https://arxiv.org/pdf/2309.06180