Uncover the Overhead and Resource Usage for Handling KV Cache Overflow in LLM Inference

Jie Ye[1] Bogdan Nicolae[2] Anthony Kougkas[1] Xian-He Sun[1]

[1]Illinois Institute of Technology [2]Argonne National Laboratory

Gnosis Research Center @ ILLINOIS TECH

Argonne NATIONAL LABORATORY

Context

- Speeding up LLM and transformer inference requests is instrumental in achieving **high throughput** and **low latency** at scale, especially when considering the need to serve a large number of users under concurrency
- At the core of each LLM inference request are **two phases**:
  - **Prefill phase:** the entire prompt (input tokens) is processed by the attention mechanism in parallel, typically during a single forward pass
  - **Decode phase:** an iterative prediction of the next most likely token, which is the appended to the prompt, and the process is repeated until a special termination token or a maximum number of tokens is reached
- Intermediate **KV pairs** computed by the attention layers during inference can be **cached** in order to **avoid recomputations** and significantly speed up the decode efficiency

Limited KV Cache Space

- When running out of space on the GPU memory for the KV cache (likely for complex inferences with many tokes that are batched together) , state of art approaches enable two alternative strategies: (1) drop intermediate results from the cache and **recompute** them later, or, (2) **swap** intermediate results to a secondary host memory cache
- **Challenges:** the problem of how to choose between recomputations (compute overheads) vs. swapping (I/O overheads due to GPU-host memory transfers) has not been thoroughly explored by state of art
- **Contributions:** we quantify the overhead of these methods and analyzing the resource utilization of the KV cache during LLM inferences to derive new insights and observations.

Recompute vs. Swap Strategies

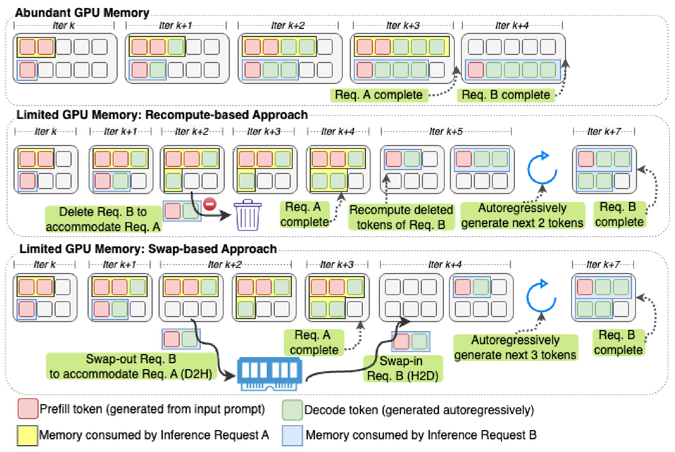

Figure 1: Recompute vs. Swap Strategies for Handling KV Cache overflow

Evaluation Methodology

- **Platform:** ALCF's Polaris. Each Node: 4×A100 GPUs with 40 GB HBM2 on each GPU
- **Inference System:** vLLM [1]
- **Model:** Yarn-Llama-2-7B-64k [2] (#Layers=32, #Heads=32, Hidden Size=4096, context length=64K);
- **Workloads:** Synthesized workloads by generating prompts for each request using dummy data

Impact of Various Factors on Overhead

- Many factors can influence the overhead of recomputing and swapping (e.g., block size (the number of tokens in a block), the number of recomputed/swapped tokens, and available GPU Mem. for KV Cache)
- **Goal:** Evaluate how different factors affect their overhead.

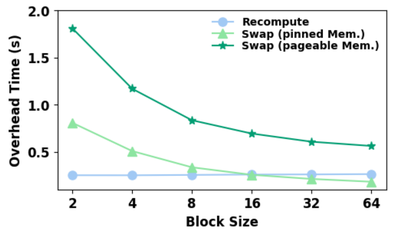

Figure 2: Overhead of recompute/swap by varying block size (i.e., the number of tokens in a block)

- **Recompute:** block size has minimal effect on its overhead
- **Swap:** 1) the overhead decreases with increasing block size; 2) more efficient for large block sizes with pinned memory

- The overhead increases with increasing the number of recomputed/swapped tokens
- Swap has lower overhead when more tokens are evicted in an overflow

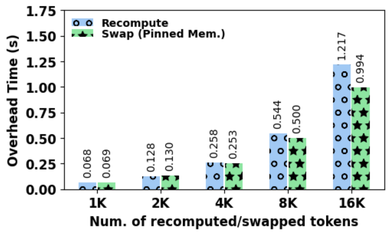

Figure 3: The Num. of recomputed/swapped tokens impact on overhead

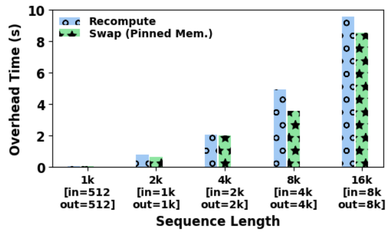

Figure 4: Overhead of recompute/swap by varying sequence length (batch size: 16)

- The overhead increases with increasing the sequence length
- Swap performs better than recompute for long sequences (lower overhead)

Impact of Various Factors on Overhead

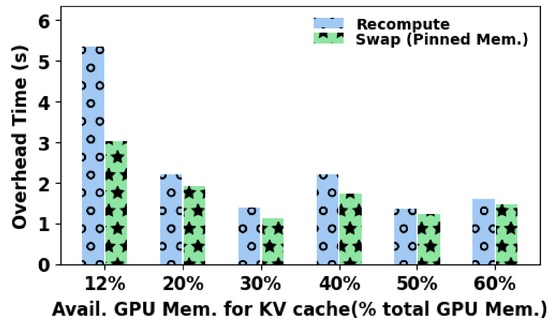

Figure 5: Overhead of recompute/swap by varying avail. GPU Mem. for KV cache

- Sequence (input=1500, output=1024); requests (batch size)=32; all requests are issued at the same time
- Overhead decreases with increasing the avail. GPU Mem. (<=40%)
- Recompute/Swap get the lowest overhead when avail. GPU Mem. is 30% or 50%

KV Cache Resource Utilization

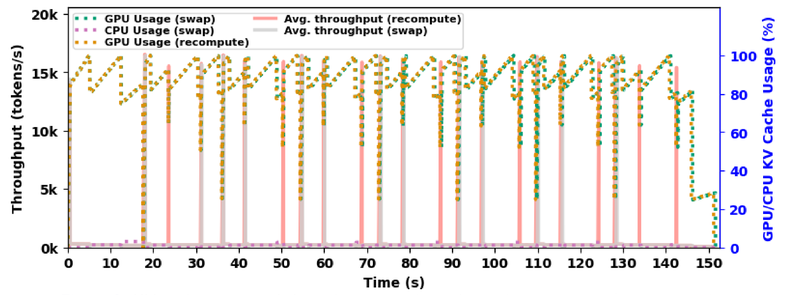

Figure 6: KV cache usage and throughput over time when using recompute/swap strategies

- Shows the average throughput and KV cache usage over time when avail. GPU Mem. is 12%
- **Recompute:** Throughput increases periodically over time due to the processing of waiting and resumed requests; GPU usage decreases when eviction happens and increases when evicted requests are resumed.
- **Swap:** The periodically sharp increase of throughput is caused by the processing of waiting requests (prompt phase); GPU usage decreases when eviction happens and increases when evicted requests are resumed.

References

[1] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

[2] "Yarn-llama-2-7b-64k." `https://huggingface.co/NousResearch/Yarn-Llama-2-7b-64k`, 2023.

jye20@hawk.iit.edu