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

Jie Ye*, Bogdan Nicolae†, Anthony Kougkas*, Xian-He Sun*
*Illinois Institute of Technology, Chicago, IL, USA
jye20@hawk.iit.edu akougkas@iit.edu sun@iit.edu
†Argonne National Laboratory, Lemont, IL, USA
bnicolae@anl.gov

I. Extended Abstract

Pre-training of LLMs and transformers is known to take weeks if not months even of powerful HPC systems. For this reason, a large part of related work studies how to scale pre-training. However, inferences are an equally important problem: once pre-trained, the model needs to serve a large number of inferences submitted under concurrency by multiple users. Thus, speeding up each inference request is instrumental in achieving high throughput and latency at scale.

At the core of each LLM inference request are two phases: (1) a prefill phase, during which the entire prompt is processed by the attention mechanism in parallel, typically during a single forward pass; (2) a decode phase that involves 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 ((<EOS>)) or a maximum predefined number of tokens is reached.

To avoid redundant recomputation in each decode iteration, a Key-Value (KV) cache is used to store previously computed keys (K) and values (V), speeding up token generation. GPU memory is primarily consumed by model weights and KV cache during LLM inference serving. While model weights use a fixed amount of memory, KV cache memory grows linearly with context length and batch sizes due to the auto-regressive nature of LLMs. Consequently, KV cache memory can easily exceed the GPU memory limits of a single instance. To handle KV cache overflow, state of art LLM inference systems (such as vLLM [1]) often use to alternative strategies, recomputation or swap to host memory, each with its own advantages and disadvantages.

Contribution: given the limited studies on the nature of recomputations vs. swapping, we characterize their behavior. To this end, we study the overheads and resource utilization under several configurations that involve different cache block sizes and inference request sizes. Then, we identify interesting patterns and correlations that can be exploited by future work to improve the latency and throughput of inference requests.

II. Recompute vs. Swap Strategies

Figure 1 describes how recompute/swap functions in LLM inference with limited GPU memory.

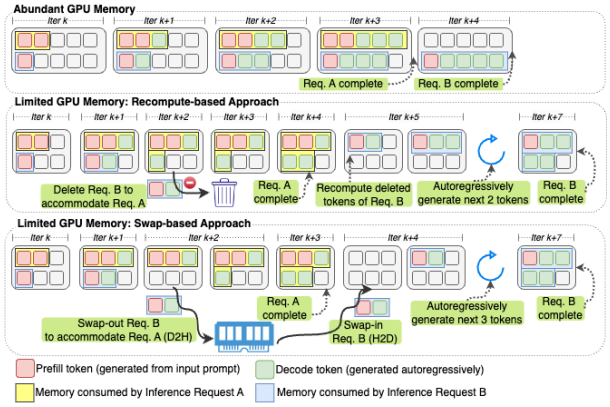


Fig. 1. Recompute vs. Swap Strategies for handling the KV cache overflow

Recompute: When GPU memory is insufficient, a set of lower-priority running inference requests are selected and their KV cache is discarded. Upon resumption later, their KV cache is recomputed rather than retrieved from GPU memory.

Swap: This method leverages host memory. When GPU memory is insufficient, the selected lower-priority inference requests have their KV cache moved to host memory. Upon resumption later, their KV cache is reloaded back to GPU memory before continuing.

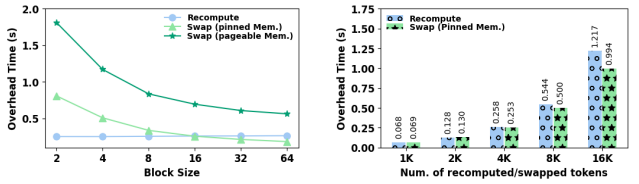
III. Study of Recompute and Swap Overhead in Handling KV Cache Overflow and Resource Usage

A. Experimental Setup

All the experiments on the ALCF's Polaris platform. Each node has 4×A100 GPUs with 40 GB HBM2 on each, and 1×512 GB DDR4 RAM. LLM inferences were conducted with vLLM 0.4.2 [2] on a single GPU. We use the Yarn-Llama-2-7B-64k [3] model since it is built on a popular LLM architecture, fits on a single GPU, and supports long contexts. Our experiments employ synthetic workloads because inference accuracy is not our focus.

B. Analyze the influence of different factors on the overhead of Recompute and Swap strategies

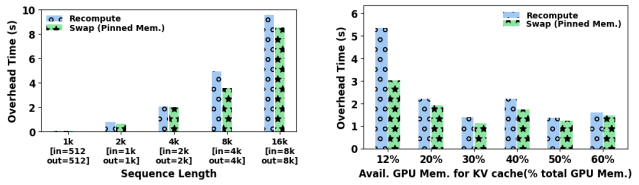
Various factors can influence the recompute and swap overhead, including block size, the number of recomputed/swapped tokens in an overflow operation, sequence length, and the available GPU memory for KV cache. This subsection explores the effects of these factors on their overhead.



(a) Overhead of recompute/swap by varying the block size

(b) Impact of the number of recomputed/swapped tokens in an overflow

Fig. 2. Impact of block size and the number of recomputed/swapped tokens in an overflow on the recompute and swap overhead time (running with 2 inference requests)



(a) Overhead of recompute/swap by varying the sequence length of a request (requests=16)

(b) Overhead of recompute/swap by varying available GPU memory for KV cache (requests=32)

Fig. 3. Impact of sequence length and available GPU memory for KV cache on the recompute and swap overhead time

Block Size: Block size refers to the number of tokens held in a block. We evaluated both pinned memory and pageable memory for swapping. As shown in figure 2(a): (1) block size has minimal effect on recomputation since it depends only on the number of recomputed tokens; (2) swap overhead decreases with larger block sizes, as smaller blocks trigger more small data movements; (3) swap performs better for the large block size with pinned memory.

Num. of evicted tokens in an overflow: As figure 2(b) shows: (1) overhead increases with increasing the number of recomputed/swapped tokens due to higher computation and data movement; (2) Swap has lower overhead when more tokens are evicted in an overflow.

Sequence Length: Figure 3(a) demonstrates that: (1) overhead of recompute and swap increases with longer sequences; (2) swap performs better than recompute for long sequences (lower overhead).

Available GPU Memory for KV Cache: We test the impact of available GPU memory for KV cache by issuing 32 requests to vLLM, each with 1500 prompt tokens and generating 1024 tokens. Figure 3(b) shows: (1) overhead decreases as available GPU memory increases up to 40%; (2) the lowest overhead occurs when available GPU memory is 30% or 50%.

C. Resource Utilization During LLM inference

This experiment describes the KV cache utilization and throughput over time with 12% available GPU memory. Figure 4 shows: (1) with recompute strategy, the throughput periodically spikes due to processing waiting and resumed requests (red solid line), while GPU KV cache utilization fluctuates (decreasing and then increasing) due to eviction/completion and processing of these requests (brown dotted line); (2) with swapping, throughput also periodically increases with the processing of waiting requests, and GPU KV cache utilization similarly fluctuates, with CPU KV cache increasing as GPU cache decreases due to data movement.

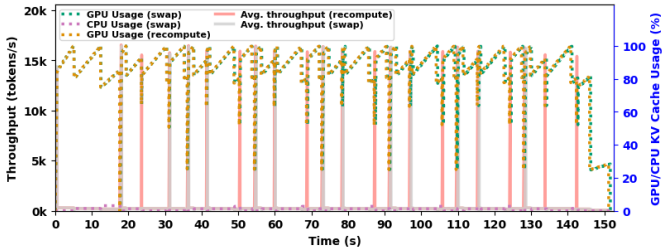


Fig. 4. KV cache usage and throughput over time when using recompute/swap strategies

IV. Conclusions

This study explores the impact of various factors on recompute and swap overhead during KV cache overflow and resource utilization. Initial results show: 1) swap is more efficient with large block sizes and long sequences; 2) vLLM's scheduling strategy batches only pure prefill or decode requests in one forward pass, causing variations in throughput and KV cache usage. Future work will investigate KV cache I/O patterns to gain insights and identify management bottlenecks.

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contracts DE-AC02-06CH11357. Additionally, this work is partially supported by the National Science Foundation (NSF), Office of Advanced Cyberinfrastructure, under Grants CSSI-2104013 and Core-2313154.

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*, 2023, pp. 611–626.

[2] vllm-v0.4.2. [Online]. Available: https://github.com/vllm-project/vllm/tree/v0.4.2

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