Divyam Sharma (divyams), Peter Cao (petepc), Harsh Sinha (hasinha), Shmeelok Chakraborty (shmeelok)

Summary of “Efficient Memory Management for Large Language Model Serving with PagedAttention”

Problem and Motivation

LLM Serving is a major task in the LLM pipeline, especially when running inference with the hosted applications, taking a lot of GPUs. This serving process is memory-bound, using up the memory while not using the full GPU compute power and also the performance throughput. To do this we want to improve the memory performance of managing the KV cache. Most current work stores the KV cache in contiguous memory like other tensors in traditional deep learning. However, the KV cache is dynamically allocated so traditional methods may not be the best and these traditional methods do not take advantage of the algorithms used. Thus, PagedAttention and vLLM are proposed in order to take advantage of the algorithms used and also have better memory use via virtual memory with paging.

Related Works

- General Model Serving: Earlier model serving systems like Clipper, TensorFlow Serving, Nexus, InferLine, and Clockwork use batching, caching, placement, and scheduling for serving models. DVABatch uses multi-entry multi-exit batching. REEF and Shepherd use preemption, and AlpaServe uses model parallelism and multiplexing.
- Transformer Serving: There are various different methods that use GPU kernel optimizations, advanced batching, model parallelism, and parameter sharing. Orca is a complementary solution to the serving problem but instead of using virtual memory, Orca uses interleaving requests to increase memory utilization.
- Memory Optimizations: FlexGen tries to swap weights and token states in the case of limited GPU memory while OLLA optimizes tensor lifetime and location to limit fragmentation. FlashAttention uses tiling and kernel optimization to reduce memory usage and I/O costs.

Solution Overview

1. PagedAttention Algorithm:

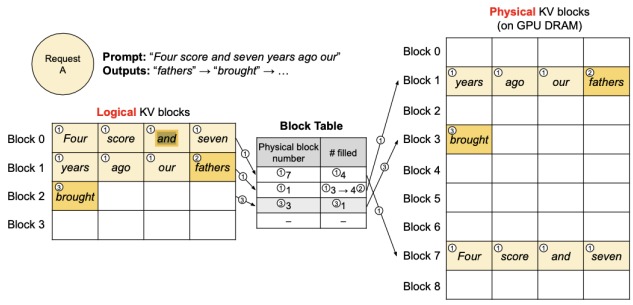

Figure 6. Block table translation in vLLM.

    - Memory Management: PagedAttention allows the KV cache memory to be stored in non-contiguous physical blocks rather than requiring contiguous physical memory allocation. This approach significantly reduces internal and external fragmentation, common problems in existing systems where memory is often wasted due to inefficient pre-allocation strategies.
    - Block-Level Flexibility: By partitioning the KV cache into smaller, fixed-size blocks (similar to paging in virtual memory), the algorithm allows more efficient use of GPU memory. It also enables the sharing of KV cache blocks across different requests, further reducing memory overhead.
2. vLLM Serving System:
    - High-Throughput Serving: Built on top of PagedAttention, the vLLM system manages KV cache memory dynamically and efficiently, allowing it to serve large batches of requests without the typical memory-related bottlenecks. It achieves near-zero waste in KV cache memory, which is crucial for scaling the serving of large models.
    - Performance Gains: The system demonstrates significant improvements in throughput—2x to 4x—compared to state-of-the-art systems like FasterTransformer and Orca, particularly for larger models and more complex decoding tasks.
3. Evaluation:
    - Scalability: The system's effectiveness was validated across various LLMs, including GPT, OPT, and LLaMA, on different workloads. The results showed consistent throughput gains, especially with longer sequences and larger models.
    - Memory Sharing: The system also supports complex decoding scenarios like beam search and parallel sampling, where the KV cache can be shared across multiple outputs from the same input, further optimizing memory usage.

Limitations

This solution only works well if the application is memory-bound. If the application is compute-bound, then we would have the issues of overhead from memory indirection and non-contiguous blocks. Also for recovery, vLLM uses both recomputation and swapping as recovery mechanisms. Recomputation is limited when block size is large while swapping is worse when block size is small. Another drawback of this method is that we also have to re-implement OS memory management in this solution as well.

Future Research Directions

1. Applying these techniques to other GPU Workloads
    - LLM serving has a workload with dynamic memory allocation and also a GPU memory-bounded performance. There are some LLM workloads that do not follow this where the paper’s methods will actually make the performance worse. However, there may be other LLM workloads with the same properties as LLM serving which may work well with the virtual memory and paging.
2. Optimizations in applying vLLM

- vLLM already takes advantage of the application-specific uses of LLM serving, but there are other undiscovered optimizations that one may be able to take advantage of.

Summary of Class Discussion

Q: We see that there are tokens computed over a period of time instead of just a snapshot, how was this calculated?
A: In the evaluation section, they observed the system for 1 hour on simulated data.

Q: Is block size fixed or changing?
A: The block size is currently fixed.

Q: Why not dynamically allocate different block sizes based on request type?
A: It's very hard to predict the size ahead of time so this leads to the utilization problems that are caused earlier.

Q: Right now they use a First Come First Serve scheduler, is there a way to do it differently? [Explanation: Bigger sequences together and smaller together]
A: Lots of different block sizes will be there with rather increased fragmentation. Keeps management of memory simpler.

Q: How does eviction work, do you evict from virtual or physical? Who gets to be removed when eviction happens?
A: Paged Attention does First Come First Serve scheduling, but it evicts those who have been there for a long time and does this in physical memory and the block table data just removes the entry.

Q: When does eviction start?
A: [Implementation Nuance] Before taking the next sequence it clears up memory from the previous ones. And hence, eviction happens every time before taking a new sequence.

Q: What is shared memory in vLLM?
A: Shared KV cache via Logical Block memory among different requests.

Summary of “vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention”

Problem and Motivation

Similar to the last paper, vAttention aims to improve upon the inefficiency in GPU memory management for Large Language Model (LLM) inference tasks. Researchers have proposed batching to amortize the cost of transferring model weights into SRAM during the decode phase of LLM inference. While batching increases throughput, inefficient GPU memory management hinders the throughput gain from batching. As our models grow larger and larger, any inefficiency has a massive impact on throughput, cost, and usability of a LLM for users and businesses. Thus, addressing the inefficiency in GPU memory management allows us to improve performance and scalability of LLMs.

Specifically, vAttention is designed to efficiently allocate GPU physical memory for the KV-Cache, which accounts for the majority of the memory requirements during inference. Efficiently allocating memory for the KV-Cache is challenging because the actual amount of memory needed per request is not known ahead of time, and the cache size grows over time as tokens are added each iteration. Allocating the max memory needed for the cache leads to internal fragmentation and poor memory usage across all GPUs. Furthermore, PagedAttention is also flawed because it requires changes to attention kernels which are continuously optimized, adds software complexity because of its extra memory translation layer (think extra translation when using a hypervisor), and adds cpu overhead related to managing the extra translation (think managed vs unmanaged programming language).

Related Works

Prior research, such as Orca and FasterTransformer, allocated physical memory based on the maximum context length a model could handle (e.g. Yi-34B had a limit of 200k tokens). However, in most cases the actual memory usage would fall short of the amount allocated and would cause fragmentation. vLLM introduced PagedAttention, based on demand paging in OS kernels, to address the fragmentation issue. PagedAttention dynamically allocates smaller blocks of GPU memory whenever necessary to reduce fragmentation; however, this requires massive modifications to user code. This is quite infeasible because it requires that users are constantly porting their optimized attention kernels to work with PagedAttention. Despite this drawback, PagedAttention is the standard approach for dynamic memory allocation in LLM serving frameworks such as TensorRT-LLM, HuggingFace TGI, and LightLLM.

GMLake demonstrated that by using CUDA’s virtual memory support they can help reduce memory fragmentation in deep neural network (DNN) training. This allows for larger training batch sizes because they combine smaller physical memory pages into one virtual contiguous object. GMLake is more geared towards training, whereas vAttention is designed for LLM inference tasks which requires more latency-sensitive and finer-granularity allocations. Furthermore, PyTorch implemented support for expandable segments that dynamically attach physical memory pages to a virtual buffer using synchronous allocation with fixed page size: vAttention neither uses asynchronous allocations nor requires fixed page sizes.

Solution Overview

The authors experimented and benchmarked KV-Cache memory footprint during inference and designed vAttention based on two observations. First, their experiments showed that as long as

a request is active, the KV-Cache grows uniformly by one token. This makes it such that the memory footprint is known on a per iteration basis. Second, their benchmarks showcased that KV-Caches across different model sizes do not have a high memory allocation bandwidth because the number of tokens generated will reach steady state over time.

vAttentions exploits virtual memory limits by pre-allocating at least as much memory to hold the maximum KV-Cache. This allows the entirety of the cache to be laid out in continuous virtual memory which is an improvement over PagedAttention which forces attention kernel modifications due to the cache being non-continuous in virtual memory. Furthermore, vAttention will allocate a configurable fixed amount of physical memory so that it is readily available. Similar to an OS kernel, as virtual memory mapped to an invalid page gets used, vAttention will map a physical page to the virtual page. Despite the attention kernel believing it has unlimited virtual memory the physical memory demand scales linearly with the KV-Cache memory requirement for each iteration per inference request. When a request finishes, vAttention will either immediately free the associated physical memory or defer them to be freed later based on the memory pressure of the GPU.

The authors implemented vAttention as a python library using CUDA/C++ extensions to interact with the CUDA drivers, and provided a simple API to interact with: consisting of, at a high level, an initialize, allocate, free, and manage physical memory API calls. The authors implemented two optimizations to solve challenges they encountered while developing vAttention: CUDA APIs requiring a minimum allocation of a 2MB physical memory page, which can lead to internal fragmentation, and the high latency associated with CUDA API calls related to mapping and unmapping physical memory pages. Their first optimization was implementing new API functions within the open-source sections of NVIDIA drivers which enabled smaller page sizes allocation to help with internal fragmentation (similar to Linux's mmap syscall). Moreover, they masked the physical memory allocation latency by overlapping it with computation. They made use of background threads to allocate physical memory in a previous computation iteration by anticipating the memory demands for the next iteration (based on the observations made earlier). Furthermore, they also deferred reclamation of physical memory pages and employed eager allocation to optimize the reuse of memory pages between requests (e.g. kernel memory pager lazy deallocation policy).

Limitations

When vAttention is unable to map all the physical pages required, it will throw an error and allow the higher level framework to preempt requests to allow forward progress of the system rather than grinding it to a halt. vAttention does not have a policy on what physical memory page it can evict to CPU memory to continue servicing the current request.

The authors were forced to implement support for multiple page sizes using open source NVIDIA drivers for unified memory management instead of directly within CUDA. Furthermore, they made use of CUDA API calls to implement this functionality which will add latency for every call they make. Being able to implement support for multiple page sizes in the CUDA APIs will help to reduce the latency by being able to directly interact with the hardware rather than having to go through an additional layer multiple times.

Future Research Directions

While vAttention requires minimal user level code changes to work, for larger code bases it can be quite the challenge to port everything to work with it. Researchers could implement an

optimization pass in an ML compiler to transform kernel code to work with vAttention. Researchers could implement and benchmark strategies on evicting pages from GPU to CPU memory and back to better make use of GPU/CPU resources when the system is under high memory pressure.

Summary of Class Discussion

Q: Difference between pre-allocation in paged and v attention?
A: Since the memory buffer is small, the allocation goes on at that level. However, the actual physical memory is not allocated at this point. This is the difference between paged and v attention. The paged attention has the physical memory allocated at this point.

Q: What do the authors mean by non-contiguous virtual memory in PagedAttention?
A: The virtual memory in the Paged Attention paper is a layer over the virtual memory of the OS. So, it is possible to have a non-contiguous virtual memory. vAttention goes towards a contiguous virtual memory.

Q: They abstracted over OS, but what is being used? Are they relying on NVIDIA's driver?
A: There is not much detail about this part. NVIDIA driver part is not accessible to them. It is running on whatever CUDA exposes. In some way that is OS and some driver API.

Q: When does eviction happen?
A: This would happen when not enough space is available. If a big one is taken out, then a big chunk of memory is free. The paper uses various heuristics to decide which one to evict. But not much details are provided. They do compare swapping and recomputation. Recomputation is simply re-computing. Swapping overhead increases with small block sizes.

Q: Why would smaller blocks affect? The number of bytes is roughly the same?
A: The overhead comes from cleaning, due to high movement as they can be sparsely located.

Q: What is the notion of shared memory in vLLM paper?
A: Share key and value for the sequences generated for the same prompt. But vAttention will not be as good as vLLM.