Department of Computing
電子計算學系

Faculty of
Computer and Mathematical Sciences
計算機及數學科學學院

Compress-Transfer-Decompress for LLM Serving: cuSZp-Enabled CPU–GPU Data Pipeline in vLLM

Student Name: KONG Zirui
Student ID: 22103493D
Supervisor Name: ZHANG Zhaorui

A dissertation proposal submitted in partial fulfillment of the graduation requirements for the MSc degree of 61435-FCS

The Hong Kong Polytechnic University
October 24^{th}, 2025

Contents

Chapter 1

Background

1.1 The Rise of Large Language Models (LLMs)

In recent years, Large Language Models (LLMs) such as GPT-3, Llama, and their variants have demonstrated remarkable capabilities in natural language understanding and generation. Their success has driven a trend towards building even larger models. However, the sheer size of these models presents significant challenges for training and, critically, for inference. Serving these models efficiently—with low latency and high throughput—is a major engineering and research problem.

1.2 Advanced Memory Management in LLM Serving: PagedAttention

To tackle the memory capacity challenges of serving LLMs, frameworks like vLLM [2] have introduced sophisticated memory management techniques. The most notable of these is **PagedAttention**, which draws inspiration from virtual memory and paging in operating systems. Instead of allocating a single contiguous block of memory for the Key-Value (KV) cache of each sequence, PagedAttention partitions the KV cache into smaller, fixed-size blocks or ”pages.”

This design allows for non-contiguous storage of the KV cache in GPU memory, drastically reducing internal fragmentation and improving memory utilization. More importantly, it enables a crucial feature for handling large-scale workloads: **swapping**. When GPU memory is exhausted, idle or low-priority KV cache pages can be offloaded (swapped out) to the host CPU’s main memory (DRAM). When needed again, they are swapped back in. This mechanism is the primary source of frequent data traffic between the CPU and GPU.

1.3 The CPU-GPU Communication Bottleneck in LLM Serving

While PagedAttention solves many memory capacity issues, it intensifies another problem: the **CPU-GPU communication bottleneck**. Data transfer between the host system and the GPU device occurs primarily over the PCIe bus, which has significantly lower bandwidth compared to on-chip or inter-GPU interconnects (e.g., 64 GB/s for PCIe 5.0 vs. 900 GB/s for NVLink on H100).

In high-throughput serving scenarios, two main data paths contribute to this bottleneck:

1. **Host-to-Device (H2D) Transfers:** This includes new input request data (token embeddings) and, most significantly, KV cache pages being swapped in from CPU memory to GPU memory to serve ongoing requests.
2. **Device-to-Host (D2H) Transfers:** This primarily consists of KV cache pages being swapped out from the GPU to the CPU to free up VRAM for higher-priority requests.

During periods of high concurrency or when processing very long sequences, the volume of data being swapped can saturate the PCIe bus. The time spent on these H2D/D2H transfers becomes a substantial, and often dominant, part of the end-to-end inference latency, limiting the overall throughput of the serving system.

1.4 Problem Statement

The core problem this project addresses is the **CPU-GPU data transfer bottleneck in high-throughput LLM serving, particularly within frameworks like vLLM that rely on CPU-GPU swapping mechanisms**. The frequent transfer of tensors—especially KV cache pages—across the limited-bandwidth PCIe interconnect leads to increased latency and reduced token throughput.

This project proposes to solve this problem by integrating a high-speed, GPU-native lossy compression algorithm, **cuSZp** [1], into the **vLLM** serving framework's H2D/D2H data channels. The solution is to apply a **compress-transfer-decompress** workflow: tensors are compressed before being sent over the PCIe bus and decompressed upon arrival. By reducing the number of bytes transferred, this approach aims to decrease the time spent on communication.

Chapter 2

Project Methodology

2.1 Literature Review

A thorough review of existing work is essential. This will focus on three key areas:

2.1.1 LLM Serving Frameworks: vLLM

We will study the architecture of vLLM, a state-of-the-art LLM serving engine. Particular focus will be on its memory management system, PagedAttention [2], and its implementation of CPU-GPU page swapping. Understanding the exact code paths for H2D and D2H transfers within its `CacheEngine` is critical for the integration. The official vLLM source code will be the primary resource.

2.1.2 GPU-Accelerated Data Compression: cuSZp

We will investigate the SZ family of lossy compressors, designed for scientific floating-point data. We will focus on `cuSZp` [**?**], a GPU-native implementation known for its high speed. The review will cover its error-bounded compression modes (absolute or relative error bounds), its performance characteristics, and its C++/CUDA API. A deep understanding of its API and kernel launch semantics is crucial for achieving asynchronous, overlapping execution.

2.2 Proposed Approach

The project will be executed in four distinct phases:

Phase 1: Baseline Profiling and Environment Setup (Weeks 1-3) The first step is to establish a baseline to measure against.

- **Tasks:** Set up a multi-GPU development environment, deploy a standard LLM on vLLM, and use profiling tools like NVIDIA Nsight Systems to capture baseline performance data under workloads that trigger frequent page swapping.
- **Objective:** To precisely measure end-to-end latency and quantify the time spent in H2D/D2H transfers, confirming it as a significant bottleneck.

Phase 2: Integration of cuSZp into the vLLM Codebase (Weeks 4-6) This phase involves the core software engineering task of linking the two codebases.

- **Tasks:** Compile `cuSZp` as a library, modify vLLM's build system (CMake/setup.py) to link against it, and create a C++/CUDA wrapper within vLLM to expose `cuSZp`'s functions in an accessible way.
- **Objective:** To have a version of vLLM that can successfully call `cuSZp` functions on a given GPU tensor from within its Python/C++ backend.

Phase 3: Implementing the Compressed Data Pipeline (Weeks 7-11) This is the most critical phase, where the full end-to-end logic is implemented.

- **Tasks:** Implement the complete closed-loop data flow: for D2H, intercept the tensor, compress it on a dedicated CUDA stream, and initiate an asynchronous D2H copy of the compressed buffer. For H2D, initiate an asynchronous H2D copy of a compressed block from host memory, and upon completion, launch the `cuSZp` decompression kernel on a separate stream. This involves managing memory pools for compressed/decompressed buffers and using CUDA events for synchronization.
- **Objective:** To create a fully functional, end-to-end compressed communication pipeline within vLLM, complete with multi-stream overlap and memory management.

Phase 4: Evaluation, Ablation, and Analysis (Weeks 12-15) In this final phase, we will rigorously evaluate the modified system.

- **Tasks:** Re-run benchmarks on the modified vLLM, systematically varying parameters like sequence length, concurrency, and compression error bound. Collect and analyze the new performance metrics (latency, throughput, transfer time, effective bandwidth, host memory usage, overlap efficiency). Conduct ablation studies to isolate the benefits of H2D vs. D2H compression and different error settings. Evaluate the impact on model accuracy.

- **Objective:** To produce a comprehensive set of results, tables, and graphs that validate the project's hypothesis and quantify the performance gains and trade-offs.

Chapter 3

Project Schedule

A detailed timeline is proposed to manage the project effectively. The milestones are defined with respect to the project phases.

Table 3.1: Proposed Project Schedule

Phase	Tasks	Weeks	Milestone / Deliverable
1. Setup & Profiling	Environment setup, vLLM deployment, baseline benchmarking to identify H2D/D2H overhead.	1-3	A report with baseline performance data, confirming the CPU-GPU communication bottleneck.
2. `cuSZp` Integration	Compiling `cuSZp`, modifying vLLM build system, creating C++ wrappers.	4-6	A functional vLLM build that can invoke `cuSZp` functions from its backend.
3. Pipeline Implementation	Modifying page in/out logic, implementing the compress-transfer-decompress pipeline with stream overlap.	7-11	A working prototype of vLLM with a compressed CPU-GPU data transfer pipeline.
4. Evaluation & Analysis	Running comprehensive benchmarks, ablation studies, and analyzing performance, resource, and accuracy trade-offs.	12-14	A complete set of experimental results, graphs, and analysis, including new metrics.
5. Final Report & Presentation	Writing the final project report, documenting code, and preparing the final presentation.	15-16	Submission of the final report, source code, and project presentation.

Chapter 4

Resources Estimation

4.1 Hardware Resources

- **Primary Requirement:** A server or cloud instance with one or more high-end NVIDIA GPUs (e.g., A100, H100, or RTX 4090). The system should have a modern PCIe interface (Gen 4 or 5) to accurately represent the target bottleneck.
- Sufficient Host RAM ($>= 128$ GB) to accommodate swapped-out KV cache pages and SSD storage ($>= 1$ TB) for the OS, datasets, and large model weights.

4.2 Software Resources

- **Operating System:** Linux (e.g., Ubuntu 20.04 or 22.04).
- **Development Tools:** NVIDIA CUDA Toolkit (11.8+), cuDNN, NCCL, GCC/G++, Python 3.9+.
- **Core Libraries:** PyTorch, source code for vLLM [**?**], source code for `cuSZp` [**?**].
- **Profiling Tools:** NVIDIA Nsight Systems for system-wide performance analysis, and Nsight Compute for kernel-level analysis.

References

[1] Yafan Huang, Sheng Di, Xiaodong Yu, Guanpeng Li, and Franck Cappello. cuszp: An ultra-fast gpu error-bounded lossy compression framework with optimized end-to-end performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2023.

[2] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.