Accessing Serialized Data Fromats with GPU-Initiated I/O

Luke Logan
Illinois Tech
United States of America
llogan@hawk.illinoistech.edu

Anthony Kougkas
Illinois Tech
United States of America
akougkas@illinoistech.edu

Xian-He Sun
Illinois Tech
United States of America
sun@illinoistech.edu

1 Introduction

GPUs have increasingly become the primary processors for scientific data exploration and analysis through general purpose programming models such as CUDA. However, GPU kernels continue to lack I/O interfaces and typically assume the CPU serves as the primary controller for program logic and data movement. In conventional GPU computing paradigms, the CPU is responsible for reading data from storage and orchestrating transfers to GPU memory. This CPU-initiated I/O introduces several critical limitations: increased development complexity due to explicit memory management, reduced scalability as CPU resources become bottlenecks, and suboptimal resource utilization due to the use of DRAM for transferring data between CPU and GPU [1, 2]. Ideally, GPU kernels should also have I/O interfaces that allow them to access data from storage.

Recent research efforts have began addressing these limitations. NVIDIA'S Magnum I/O offers a data path that allows CPUs to transfer data directly from NVMe to GPU memory, bypassing DRAM bounce buffers. Many systems and I/O libraries support Magnum I/O, such as HDF5. However, these approaches cannot be called directly within GPU code. To address this, various works have began exploring GPU-initiated I/O, where GPU kernels are extended to have I/O interfaces. BaM [2] provides an interface that allows GPUs to submit I/O commands directly to NVMes using a custom kernel module. GeminiFS [1] expands this to a specialized filesystem for NVMe SSDs designed to leverage GPU parallelism.

While existing GPU-initiated approaches can offer performance and programmability improvements in certain cases, they all face a fundamental challenge: they do not directly support access to complex serialized data formats within GPU kernels. Scientific applications typically store data in serialized formats such as HDF5, NetCDF, ADIOS, and Parquet to ensure data longevity and portability. Current GPU-driven storage systems assume that GPU kernels will handle raw data interpretation, which would require substantial reimplementation of existing I/O libraries and data format parsers—a prohibitively complex undertaking for formats like HDF5 that involve intricate metadata structures, compression schemes, and hierarchical organization.

This work explores the viability of accessing serialized data formats through GPU-initiated I/O transfers to a specialized, multi-threaded runtime executing on the CPU. We study the overhead of the GPU-to-CPU communication process and assess the initial scalability properties of this hybrid approach. While this design requires CPU involvement, it enables GPU kernels to seamlessly access complex serialized data formats without requiring extensive library reimplementation or sacrificing the rich functionality of existing data format ecosystems. This also enables support for storage devices where direct access by GPU is not always possible, such as SATA SSDs, HDDs, and many parallel filesystems. As future work, we are exploring reimplementations of popular I/O libraries to enable GPUs to directly access serialized data formats natively and integrate with programming frameworks such as kokkos.

2 Initial Results

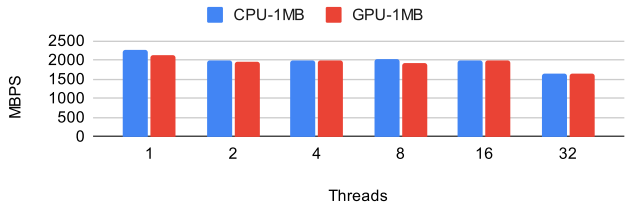

Figure 1: Performance comparison of GPU-Initiated I/O to CPU-driven for 1MB transfer size

To evaluate the viability of our approach, we conducted a preliminary performance analysis comparing GPU-initiated I/O transfers against traditional CPU-initiated data movement for sequential write workloads to HDF5 files. Our experimental testbed consists of an NVIDIA GeForce RTX 4070 GPU paired with a 16-core CPU system. We vary the CPU-side I/O runtime threads (1-32) to cover typical scientific computing access patterns. We keep the number of GPU threads equal to the number of CPU threads. We use a 1MB transfer size for the I/O. Each benchmark configuration runs for 25 seconds to ensure statistical significance and account for system warm-up effects. From Figure 1, our preliminary results demonstrate that the GPU-initiated I/O approach maintains competitive performance with traditional CPU-initiated transfers, exhibiting overhead less than 5% across all tested configurations. The main overhead is only the PCIe bus latency for the task transfer from GPU to CPU, which is minimal compared to the I/O cost. This suggests that the hybrid architecture successfully bridges the gap between GPU computational capabilities and complex serialized data access without introducing prohibitive performance penalties. These initial findings indicate that GPU kernels can effectively coordinate with CPU-based I/O runtimes for accessing serialized data, providing a foundation for more comprehensive evaluation and optimization in future work. Note that HDF5 decreases in performance with thread counts due to a global lock required for synchronization, which is expected.

References

[1] Shi Qiu, Weinan Liu, Yifan Hu, Jianqin Yan, Zhirong Shen, Xin Yao, Renhai Chen, Gong Zhang, and Yiming Zhang. 2025. GeminiFS: A Companion File System for GPUs. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. USENIX Association, Santa Clara, CA, 221–236.

[2] Zaid Qureshi, Vikram Sharma Mailthody, Isaac Gelado, Seungwon Min, Amna Masood, Jeongmin Park, Jinjun Xiong, Chris J Newburn, Dmitri Vainbrand, I-Hsin

Chung, et al. 2023. GPU-initiated on-demand high-throughput storage access in the BaM system architecture. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ACM, 325–339.