Accessing Serialized Data Formats with GPU-Initiated I/O

Luke Logan, Anthony Kougkas, Xian-He Sun

The Problem

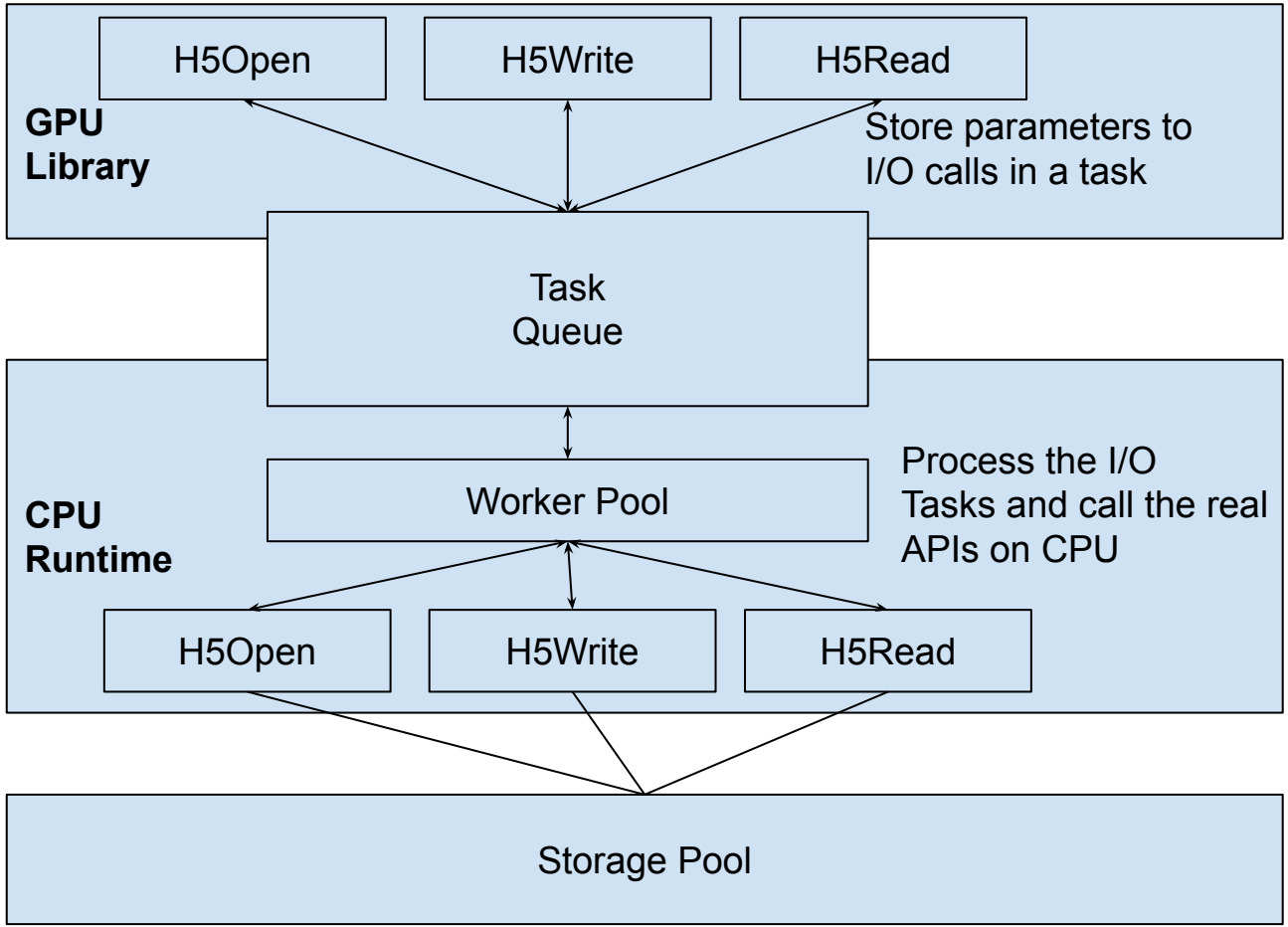
- GPUs are becoming the main processors of data
  - Simulation, AI Inference, etc.
- However, GPUs lack native I/O APIs
- CPU orchestrates reading data from storage → transferring to GPU memory
- **Creates critical bottlenecks:**
  - Increased development complexity (explicit memory management)
  - CPU becomes scalability bottleneck
  - Suboptimal resource utilization (data bounces through DRAM)

Current Approaches

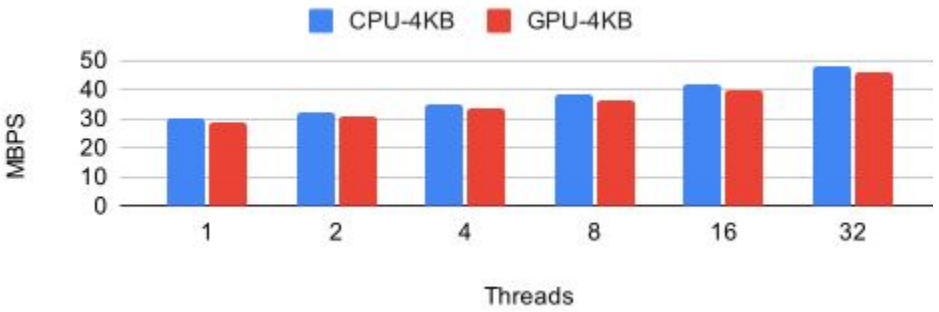
- **GPU-Initiated I/O Systems Emerging:**
  - BaM: GPU kernels submit I/O commands directly to NVMe
  - GeminiFS: Specialized filesystem leveraging GPU parallelism
  - NVIDIA SCADA: Similar to BaM, but distributed
- **Limitations:**
  - None support accessing serialized data formats within GPU kernels (e.g, HDF5)
  - Prohibitively complex for formats with intricate metadata, compression, hierarchical organization

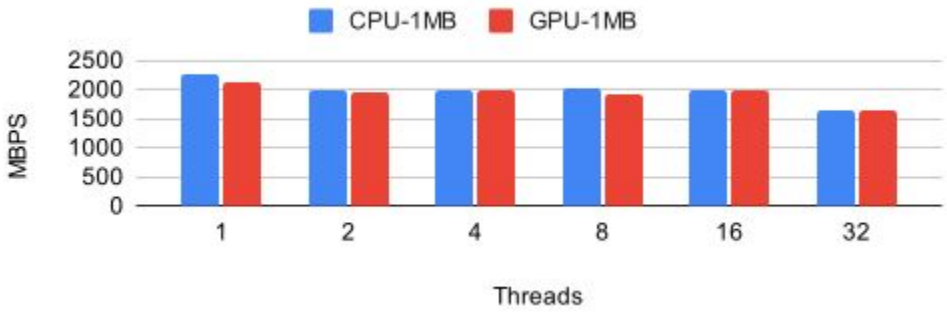
Our Approach

- GPU kernels initiate I/O transfers to the I/O libraries
- Challenge: How do we perform
- Prototype: Just bundle I/O operations and route to a specialized runtime on the CPU


H5Open
H5Write
H5Read
GPU
Library
Store parameters to
I/O calls in a task
Task
Queue
CPU
Runtime
Worker Pool
Process the I/O
Tasks and call the real
APIs on CPU
H5Open
H5Write
H5Read
Storage Pool

Results: HDF5





- Varied number of workers. We assign exactly 1 queue per worker.
- GPU threads matched to CPU threads
- Sequential write of 4KB and 1MB I/O for 25s to chunked HDF5
- For both 1MB and 4K, the queuing did not add major overhead (<6%)
- HDF5 synchronized with global lock and does not scale in terms of threads

Conclusion and Future Work

- We demonstrated it is feasible to call I/O APIs on the GPU
- In the future, we plan to explore optimizations to I/O libraries themselves to be optimized for GPU and avoid the CPU entirely