Cost-Efficient LLM Training with Lifetime-Aware Tensor Offloading via GPUDirect Storage

Ziqi Yuan[1], Haoyang Zhang[1], Yirui Eric Zhou[1],
Apoorve Mohan[2], I-Hsin Chung[2], Seetharami Seelam[2], Jian Huang[1]
[1]University of Illinois Urbana-Champaign [2]IBM Research

I. Large LLM Workloads Are Hungry for Memory

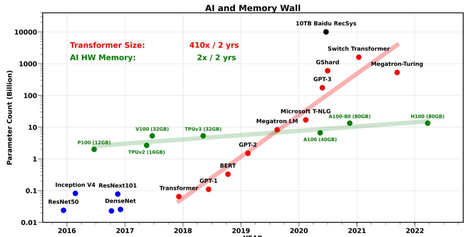


Photo credit: https://github.com/amirgholami/ai_and_memory_wall

GPU On-board Memory: Tens of GBs

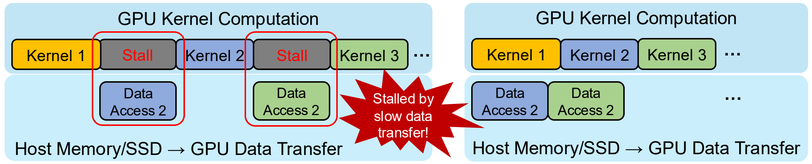
----------vs.----------

LLM Model Size: Tens of TBs

II. Expanding GPU Memory with Flash Memory

GPU On-board Memory: Tens of GBs + Flash Memory: Tens of TBs

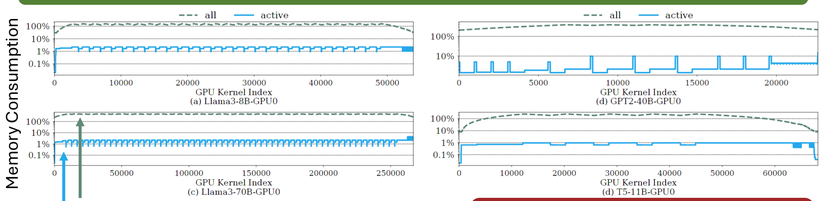
- Low Cost
- Large Capacity



Overlap Data I/O and Computation → Achieve Near-Ideal Performance with Slow Flash Memory

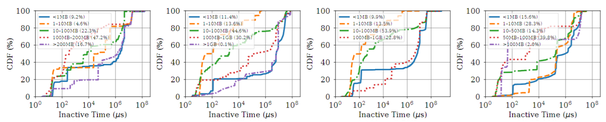
III. LLM Training Memory Characterization Study

Observation 1: Active Tensors Require Only A Small Portion of GPU Memory



Most tensors are inactive and can be swapped out during training

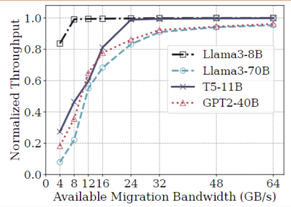
Observation 2: Many Tensors Are Unused For A Long Time



Many tensors can be safely swapped out to slow memory

Distribution (CDF) of tensor inactive period lengths for different tensor sizes

IV. Opportunities for Tensor Migration



Roofline Analysis Normalized training performance for different migration bandwidths

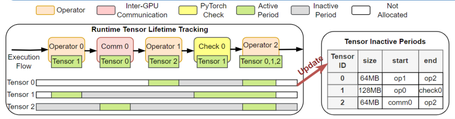
A bandwidth of 32 to 48 GB/s is sufficient to achieve near-ideal performance for large LLMs.

TeraIO

- **Extract the Semantic Knowledge of Tensors in LLM Models** → Tensor Liveness Analysis
- **Schedule Tensor Migrations using Semantic Knowledge** → Smart Tensor Placement and Migration
- **Integrating Automatic Migration Management in PyTorch** → Transparent Tensor Migration Engine (User Program ↔ PyTorch Migration Engine ↔ Host Memory / GPU Memory / SSD)

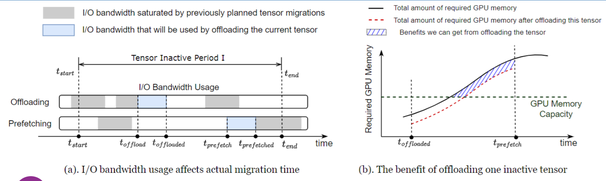
V. Enabling Smart Tensor Migrations with Rich Semantic Knowledge

Tensor Lifetime Profiler



Run the model a few times and record op runtime and tensor lifetime information

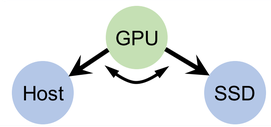
I/O Bandwidth-Aware Migration Planning



(a). I/O bandwidth usage affects actual migration time (b). The benefit of offloading one inactive tensor

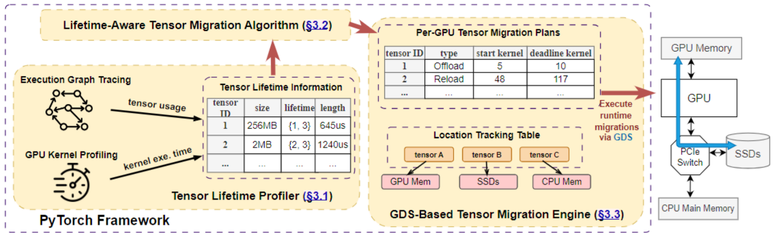
1. Estimate the impact of evicting an inactive tensor
2. Choose the one which will reduce the highest memory pressure (ⓑ), and cause the least I/O bandwidth pressure (ⓐ)
3. Update the "effects" of this decision

Decide Eviction Destination



1. Try to evict to SSD first
2. If SSD's current traffic is saturated, try host memory
3. If both SSD and Host are busy, go for SSD

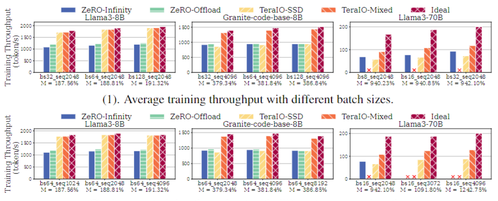
VI. Integrated Migration Engine in PyTorch



1. We place hooks before and after each operator to allow profiling and migration instrumentation
2. The migration schedule will be automatically executed before each operator is executed
3. We use GPUDirect Storage (GDS) when moving tensors between the GPU and SSDs

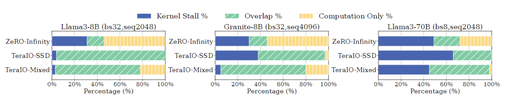
We use GDS because it provides significant scalability advantages by eliminating host resource contention

VII. Evaluation



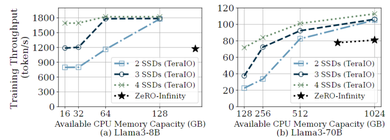
bs: Batch Size **seq**: Sequence Length **M**: Model Size / GPU Mem Capacity

With limited GPU memory, TeraIO achieves 80.7% of the ideal performance.

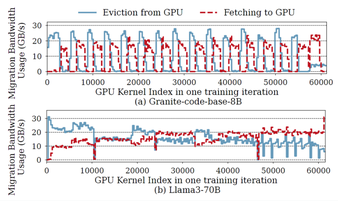


Latency breakdown of training iterations

TeraIO incurs the least stall time, as it achieves better I/O and computation overlapping.



TeraIO can achieve better performance with lower CPU memory requirements.



TeraIO maintains high I/O bandwidth utilization with our proposed I/O-aware migration algorithm.