MOST: <u>M</u>emory <u>O</u>versubscription-aware <u>S</u>cheduling for <u>T</u>ensor Migration on GPU Unified Storage

Junsu Kim, Jaebeom Jeon, Jaeyong Park, Sangun Choi, Minseong Gil, Seokin Hong, *Member, IEEE,* Gunjae Koo, *Member, IEEE,* Myung Kuk Yoon, *Member, IEEE,* Yunho Oh, *Senior Member, IEEE,*

***Abstract*—Deep Neural Network (DNN) training demands large memory capacities that exceed the limits of current GPU onboard memory. Expanding GPU memory with SSDs is a cost-effective approach. However, the low bandwidth of SSDs introduces severe performance bottlenecks in data management, particularly for Unified Virtual Memory (UVM)-based systems. The default on-demand migration mechanism in UVM causes frequent page faults and stalls, exacerbated by memory oversubscription and eviction processes along the critical path. To address these challenges, this paper proposes Memory Oversubscription-aware Scheduling for Tensor Migration (MOST), a software framework designed to improve data migration in UVM environments. MOST profiles memory access behavior and quantifies the impact of memory oversubscription stalls and schedules tensor migrations to minimize overall training time. With the profiling results, MOST executes newly designed pre-eviction and prefetching instructions within DNN kernel code. MOST effectively selects and migrates tensors that can mitigate memory oversubscription stalls, thus reducing training time. Our evaluation shows that MOST achieves an average speedup of 22.9% and 12.8% over state-of-the-art techniques, DeepUM and G10, respectively.**

I. Introduction

Deep Neural Network (DNN) training continues to push the limits of GPU off-chip memory capacity due to its exponential scaling [1], [2]. Prior work has explored expanding GPU memory with host DRAM, but this approach is less viable due to the end of DRAM scaling [3], [4]. A promising alternative is to leverage Solid State Drives (SSDs) to expand limited GPU off-chip memory. SSDs offer scalable capacity at a lower cost than DRAM.

Unified Virtual Memory (UVM) simplifies memory management by enabling user-transparent data migration [4], [5]. While UVM improves programmability, the use of UVM causes substantial delays in training time [3], [4]. The on-demand migration of UVM often incurs page faults and high latency, primarily due to limited SSD bandwidth. If GPU off-chip memory becomes fully utilized, memory oversubscription occurs, prompting UVM to evict existing GPU memory pages before loading new data [6]. Such evictions lie on the critical path of kernel execution. Since the memory footprints of large DNNs often exceed the capacity of GPU memory, training large models suffers from frequent stalls caused by memory oversubscription [6].

Prior work has introduced prefetching and pre-eviction techniques to hide data transfer latency over limited SSD bandwidth and to alleviate memory oversubscription stalls [3], [4], [7]. Prior work has migrated a tensor only if its transfer time is shorter than the interval before the data is reused (i.e., the reuse interval), so that prefetching completes in time [4]. However, under constrained SSD bandwidth, this scheduling policy fails to migrate large tensors due to their long transfer times. As such, large tensors remain in GPU off-chip memory, rapidly exhausting available capacity and causing memory oversubscription, which in turn slows down training. Therefore, despite the benefits of prefetching and pre-eviction, memory oversubscription stalls remain a significant barrier to the efficient training of emerging DNNs.

In this paper, we propose the Memory Oversubscription-aware Scheduling for Tensor Migration (MOST) software framework. The key idea behind MOST is to reduce memory oversubscription stalls by scheduling pre-eviction and prefetching of large tensors based on a cost-benefit analysis of stall reduction versus prefetch delay. Unlike prior work, MOST enables pre-eviction of large tensors even if their transfer time exceeds the reuse interval. However, in such cases, prefetching may not complete in time, potentially delaying the corresponding kernel execution. To balance this trade-off, MOST estimates the reduction in memory oversubscription stalls achieved by pre-evicting a tensor, as well as the potential stalls caused by late prefetching. If the former outweighs the latter, MOST schedules the tensor for pre-eviction and prefetching.

We compare the performance of MOST against the baseline UVM and state-of-the-art scheduling techniques with three transformer-based models [8]–[10]. In our evaluation, MOST reduces memory oversubscription stalls by 23.7% on average over the state-of-the-art technique, DeepUM and G10 [4], [7]. MOST shows performance improvement of 22.9% over DeepUM and 12.8% over G10, which results from this behavior.

II. Why MOST?

Despite advancements in GPU performance, the available on-board memory remains constrained by technological barriers [2]. Although prior work has utilized host DRAM to scale GPU memory, DRAM is insufficient to accommodate the memory footprints of emerging DNNs due to the end of DRAM scaling [3]–[5], [7], [11].

Leveraging SSDs to expand off-chip memory shows promise due to the scalability of SSDs. However, expanding GPU memory with SSDs using UVM poses significant challenges due to low SSD bandwidth, particularly in DNNs. UVM relies on on-demand migration as its default data transfer mechanism [5]. If a GPU accesses a page on an SSD, the system waits until the data is fetched, which is time-consuming due to the low SSD bandwidth.

Moreover, if GPU off-chip memory is full, UVM evicts pages before fetching new data [5]. On-demand paging and

TABLE I
SYSTEM CONFIGURATIONS

Platform	NVIDIA Geforce RTX 4090		
GPU Memory	24 GB GDDR6X		
Page Size	4 KB		
Page Fault Handling Latency	45 μs		
Page Fault Batch	256		
Interconnect	PCIe Gen 5		
SSD Capacity	2 TB		
DNN Models	BERT-Base	ViT-Base	OPT-1.3B
Number of Parameters	110M	86M	1.3B
Batch Size	512, 1024	2048, 4096	128, 256

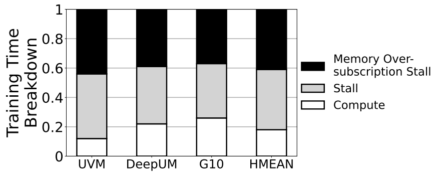


Fig. 1. Training time breakdown for the three architectures. The proportion of memory oversubscription stalls accounts for 43.7% of the total execution time in UVM, 38.5% in DeepUM, 36.7% in G10, respectively.

eviction mechanisms triggered by memory oversubscription share the page fault handling process managed by the UVM driver. This process operates at the granularity of a 2 MB Virtual Address Block (VABlock), which serves as the memory allocation unit within UVM, allowing batched page fault handling [6], [12]. If a page fault occurs, Streaming Multiprocessors (SMs) and micro Translation Lookaside Buffers (TLBs) generate a fault signal. GPU Memory Management Unit (GMMU) interrupts UVM driver and Direct Memory Access (DMA) engine migrates pages. The page fault handling for a 256-page VABlock takes at least 150µs, causing increased eviction stalls [6].

Prior work has introduced prefetching and pre-eviction algorithms to reduce stalls caused by data transfers [3], [4], [7]. Prefetching hides the long latency of SSD accesses by proactively fetching data for upcoming kernels, while pre-eviction alleviates memory oversubscription by freeing up space in the GPU off-chip memory for subsequent kernels. Prior work has proposed G10, an advanced pre-eviction technique that only evicts tensors whose transfer times are shorter than their inactive intervals to mitigate transfer delays [4]. Although the prior work has improved training performance, the memory oversubscription stalls remain a bottleneck.

We analyze the performance impact of memory oversubscription during the training of BERT, ViT and OPT-1.3B by varying batch sizes. Table I describes the detailed configurations. We use the UVM simulator proposed by G10 [4], [8], [13]. We consider UVM as a baseline, DeepUM and G10 as the state-of-the-art solutions employing pre-eviction and prefetching scheduling [4], [5], [7]. We define memory oversubscription stalls as the proportion of cumulative stall cycles resulting from eviction due to memory oversubscription.

Figure 1 shows that stalls caused by memory oversubscription significantly delay training time. In this figure, 'stall' refers to those not caused by memory oversubscription, such as latency from on-demand migrations. The term 'compute' denotes the kernel execution time during the workload. Our analysis shows that the UVM baseline suffers from memory oversubscription stalls, which account for 43.7% of the total execution time. DeepUM and G10 reduce both the overall training time and the severity of memory oversubscription

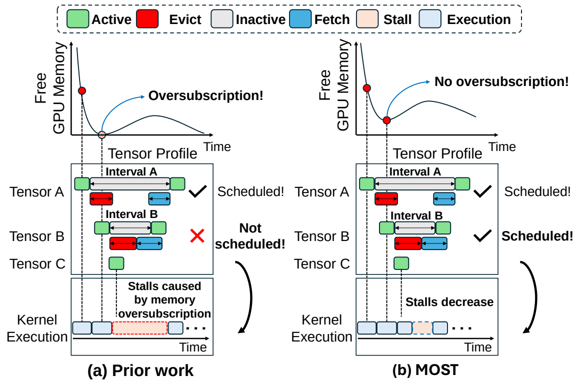


Fig. 2. Key idea of MOST. MOST schedules pre-eviction and prefetching for the tensors, if the eviction can reduce memory oversubscription stalls.

stalls through prefetching and pre-eviction scheduling. However, the proportion of execution time still spent on memory oversubscription stalls 38.5% for DeepUM and 36.7% for G10.

Even with an advanced pre-eviction policy, G10 migrates only small tensors under low SSD bandwidth. This behavior results in memory oversubscription. G10 schedules a tensor for migration only if its transfer time is shorter than the reuse interval to ensure timely prefetching. Under constrained SSD bandwidth, large tensors typically require longer transfer times than their reuse intervals, so G10 does not migrate them. As a result, large tensors stay in GPU off-chip memory, exhausting available capacity and causing memory oversubscription stalls that significantly degrade system performance.

III. MOST

To address the challenges explained in Section II, we propose the Memory Oversubscription-aware Scheduling for Tensor Migration (MOST) framework. Figure 2 depicts the key idea of MOST. MOST incorporates memory oversubscription stall estimation into its migration scheduling policy, evaluating the impact of each candidate migration on system-level stall reduction. Unlike prior work, which skips the migration of large tensors whose transfer time exceeds their reuse interval as shown in Figure 2a, MOST selectively schedules such migrations if they help reduce overall memory oversubscription stalls (Figure 2b). To make this decision, MOST estimates both the benefit of reducing oversubscription stalls through pre-eviction and the potential penalty from delayed prefetching. MOST uses the reuse interval of each tensor, which is defined as the time between successive uses, as a reference metric for scheduling. By comparing the estimated reduction in oversubscription stalls against the cost of late prefetching, MOST prioritizes migration decisions that reduce overall training time under limited SSD bandwidth.

A. *MOST Scheduling Algorithm*

Figure 3 describes the flowchart of MOST scheduling algorithm. MOST schedules pre-eviction and prefetching by quantifying the memory oversubscription stalls. The MOST scheduling algorithm first selects tensors based on the eviction benefit, B, defined as follows:

$$B = S \times I \tag{1}$$

where S and I are the tensor size and the inactive interval of the tensor, respectively.

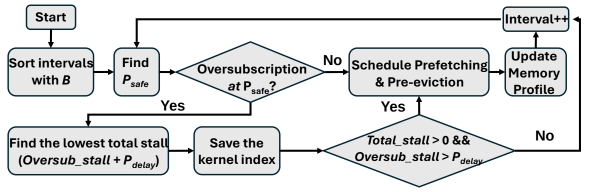
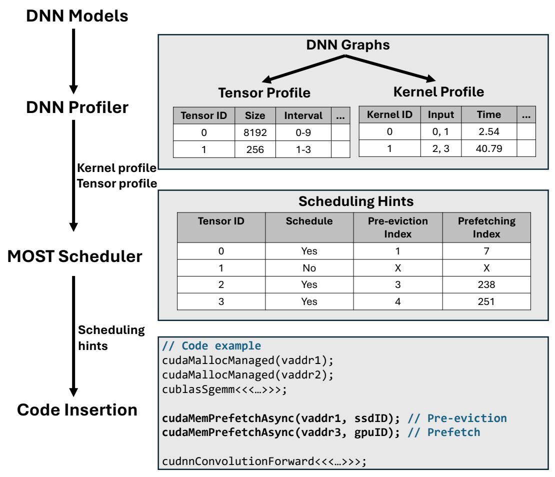

Start
Sort intervals with B
Find P_{safe}
Oversubscription at P_{safe}?
No
Yes
Schedule Prefetching & Pre-eviction
Update Memory Profile
Interval++
Find the lowest total stall ($Oversub\_stall + P_{delay}$)
Save the kernel index
$Total\_stall > 0$ && $Oversub\_stall > P_{delay}$
Yes
No

Fig. 3. Flowchart of the MOST scheduling algorithm.



Tensor Profile

Tensor ID	Size	Interval	...
0	8192	0-9	
1	256	1-3	

Kernel Profile

Kernel ID	Input	Time	...
0	0, 1	2.54	
1	2, 3	40.79	

Scheduling Hints

Tensor ID	Schedule	Pre-eviction Index	Prefetching Index
0	Yes	1	7
1	No	X	X
2	Yes	3	238
3	Yes	4	251

```
// Code example
cudaMallocManaged(vaddr1);
cudaMallocManaged(vaddr2);
cublasSgemm<<<…>>>;

cudaMemPrefetchAsync(vaddr1, ssdID); // Pre-eviction
cudaMemPrefetchAsync(vaddr3, gpuID); // Prefetch

cudnnConvolutionForward<<<…>>>;
```

Fig. 4. Block diagram of the MOST framework.

For timely prefetching, we define P_{safe}, the prefetching safe time. P_{safe} indicates the time at which prefetching should occur so that the data is available before kernel execution. P_{safe} is calculated by subtracting the fetching latency from the point at which the tensor is used. Based on the order of tensors with the highest eviction benefit, the scheduler in MOST determines P_{safe} to accurately compute prefetching delay (P_{delay}), the additional time consumed due to late prefetching beyond P_{safe}.

If memory oversubscription does not occur at P_{safe}, the scheduler performs pre-eviction immediately after tensor usage and schedules prefetching at P_{safe}. In this case, GPU memory gains the eviction benefit corresponding to the tensor without causing memory oversubscription. Otherwise, the scheduler calculates the oversubscription stalls, $Oversub\_stall$. We define $Oversub\_stall$ as the sum of eviction latency incurred by memory oversubscription for executing subsequent kernels until a prefetching performs. Also, MOST calculates P_{delay} if prefetching starts later than P_{safe}. MOST then identifies the kernel index that shows the lowest total stall, which is the sum of $Oversub\_stall$ and P_{delay}. If the total stall is greater than zero and $Oversub\_stall$ exceeds P_{delay}, MOST schedules pre-eviction after accessing the tensor and prefetching at the corresponding kernel index.

B. Framework Implementation

Figure 4 illustrates the operation of the MOST framework. The DNN profiler takes the DNN graph extracted from AI frameworks (e.g., PyTorch, HuggingFace) as input. The DNN graph contains information about the operations (e.g., addition, GEMM, GELU, etc.) required for the training workload, including the input and output tensor sizes for each operation. The execution time of each kernel can be determined by profiling each kernel individually. The DNN graph enables the framework to identify the kernels in the workload, the input and output tensor sizes for each kernel, the execution time of each kernel, and the history of GPU off-chip memory usage during kernel execution. As a result, the DNN profiler generates a kernel profile that includes detailed information about each kernel. Additionally, the DNN profiler creates a tensor profile, which captures information about the tensors used by each kernel, such as their sizes and inactive intervals.

Based on the profiling results, MOST schedules pre-eviction and prefetching for tensors. The scheduling algorithm iterates through all tensors and estimates the benefit of eviction. The scheduling algorithm then schedules evictions in descending order of benefit, while accounting for the latency caused by memory oversubscription and prefetching. After scheduling, MOST modifies the target workload code by inserting pre-eviction and prefetching instructions. Since the CUDA runtime API (cudaMemPrefetchAsync) supports only host-to-GPU transfers, we implement custom pre-eviction and prefetching APIs using NVIDIA GPUDirect Storage (GDS) APIs (e.g., cuFileRead, cuFileWrite). The customized API hooks into cudaMemPrefetchAsync and transparently redirects data transfers between SSD and GPU through GDS.

C. Overhead Analysis

MOST profiling requires executing a single training iteration on each workload to profile kernel execution time and tensor lifetime. Also, MOST scheduling consumes less time than profiling. Thus, MOST introduces negligible overhead relative to the overall training time, since training typically involves hundreds of thousands of iterations [9]. As an example, training ViT-Base on ImageNet 21K (14 million images) requires 90 epochs, with each epoch consisting of 3,467 iterations at a batch size of 4,096, totaling approximately 311,949 training iterations (90 × 3,467) [9]. Therefore, the combined overhead of profiling and scheduling accounts for less than 0.001% of the total workload execution time.

IV. Evaluation

A. Experimental Setup

We evaluate MOST with three transformer-based DNN models, BERT, ViT, and OPT-1.3B [8]–[10]. We compare MOST against the default UVM (used as a baseline) and the state-of-the-art scheduling algorithms DeepUM and G10 [4], [5], [7]. Since both DeepUM and G10 utilize host DRAM, we modify them to use only SSD storage. We use the UVM simulator proposed by G10, which is based on UVMSmart and GPGPU-Sim [4], [13], [14]. The simulator models GPU page fault handling, data migration, and address translation. The system specification for simulation is identical to those in Table I.

B. Results

Speedup: Figure 5 illustrates the speedup comparison between MOST and prior work. MOST achieves an average speedup of 2.6× over the baseline UVM, with speedup gains of 22.9% and 12.8% over DeepUM and G10, respectively. The correlated prefetching of DeepUM often struggles to predict accurate data to be used and accurate time for prefetching

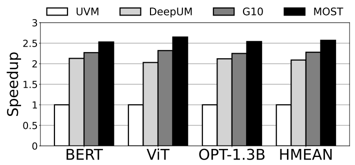


Fig. 5. Speedup comparison between prior work and MOST. MOST achieves 22.9% and 12.8% speedup over DeepUM and G10, respectively.

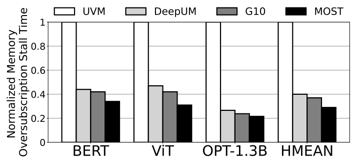


Fig. 6. The comparison of the memory oversubscription stalls. MOST reduces memory oversubscription stalls by 70.8% compared to UVM.

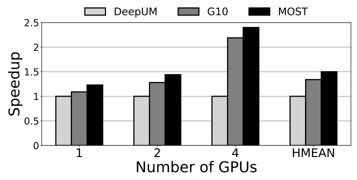


Fig. 7. Speedup comparison between MOST and prior work while varying number of GPUs. All the experimental results are normalized to the performance of DeepUM.

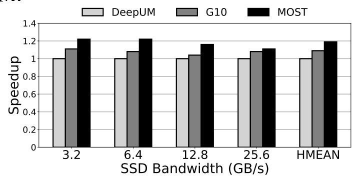


Fig. 8. Speedup comparison between MOST and prior work at varying SSD bandwidths. All the experimental results are normalized to the performance of DeepUM.

while G10 prefetches an accurate tensor in time by leveraging the workload profile. However, G10 incurs memory oversubscription as it does not consider explicit memory oversubscription stalls for scheduling. On the contrary, MOST reduces memory oversubscription stalls and achieves speedup over prior work.

Memory oversubscription stalls: As shown in Figure 6, MOST reduces memory oversubscription stalls by 70.8% compared to the baseline. Moreover, MOST achieves a reduction of the stalls by 27.5% and 20.8% on average over DeepUM and G10, respectively. By quantifying the benefit from reducing memory oversubscription stalls, MOST successfully migrates tensors that can reduce memory oversubscription stalls. Especially, pre-evicting large tensors helps alleviating memory oversubscription stalls. According to our analysis, MOST pre-evicts tensors that are on average 4.3$\times$ larger than those of G10, which contributes to a substantial reduction in memory oversubscription stalls.

C. Discussion

Impact of GPU Count: We conduct experiments in multi-GPU settings with a 900 GB/s NVLink bandwidth. Figure 7 shows the experimental results. MOST achieves average speedups of 50.3% and 12.0% over DeepUM and G10, respectively. In multi-GPU training using UVM with SSDs, tensor fetching begins from the SSDs. The GPU that initially receives a tensor from the SSD scatters it to other GPUs via NVLink. After kernel execution, a reduction operation merges the scattered tensors into a single GPU for storage. This behavior reduces kernel execution time through parallelism, while increasing data transfer time due to inter-GPU communication. As data transfer becomes a more significant bottleneck in multi-GPU training, the performance benefits of pre-eviction and prefetching become more substantial than single-GPU training.

Effect of SSD Bandwidth on Performance: MOST effectively reduces memory oversubscription stalls, particularly under realistic SSD bandwidths. Figure 8 presents the speedup of MOST compared to DeepUM and G10 across a range of realistic SSD bandwidths (3.2$\sim$25.6 GB/s). On average, MOST outperforms DeepUM and G10 by 20.0% and 9.9%, respectively. Even under high SSD bandwidth, G10 fails to schedule the migration of large tensors, whereas MOST pre-evicts large tensors and effectively reduces training time.

Performance Sensitivity to SSD Latency: We conduct experiments by varying SSD latency from 5 μs to 20 μs and observe that MOST achieves a 10$\sim$12% speedup over G10 across all latency configurations. During DNN training, large tensors composed of sequential pages are migrated. This sequential access pattern enables effective pipelining and overlapping of SSD latency, including FTL translation. Thus, SSD latency has a smaller impact on performance than SSD bandwidth.

V. Conclusion

We introduce the MOST framework that schedules pre-eviction and prefetching for tensors in UVM-based GPU systems using SSDs. MOST utilizes the memory oversubscription stalls for migration scheduling. We show that MOST outperforms the state-of-the-art migration scheduling methods by an average of 16.2%.

References

[1] A. Dubey *et al.*, “The llama 3 herd of models,” *arXiv 2024*.
[2] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, “Ai and memory wall,” *IEEE Micro*, 2024.
[3] J. Bae *et al.*, “Flashneuron:ssd-enabled large-batch training of very deep neural networks,” in *USENIX FAST 21*, 2021.
[4] H. Zhang, Y. Zhou, Y. Xue, Y. Liu, and J. Huang, “G10: Enabling an efficient unified gpu memory and storage architecture with smart tensor migrations,” in *MICRO 2023*.
[5] NVIDIA, “Unified memory for cuda beginners,” 2024.
[6] T. Allen and R. Ge, “In-depth analyses of unified virtual memory system for gpu accelerated computing,” in *SC 2021*.
[7] J. Jung, J. Kim, and J. Lee, “Deepum: Tensor migration and prefetching in unified memory,” in *ASPLOS 2023*.
[8] J. Devlin, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv 2018*.
[9] A. Dosovitskiy, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv 2020*.
[10] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen *et al.*, “Opt: Open pre-trained transformer language models,” *arXiv*, 2022.
[11] J. Choe, “Memory technology 2021: Trends & challenges,” in *IEEE SISPAD 2021*.
[12] H. Kim, J. Sim, P. Gera, R. Hadidi, and H. Kim, “Batch-aware unified memory management in gpus for irregular workloads,” in *ASPLOS 2020*.
[13] D. Ganguly, Z. Zhang, J. Yang, and R. Melhem, “Interplay between hardware prefetcher and page eviction policy in cpu-gpu unified virtual memory,” in *ISCA 2019*.
[14] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, “Analyzing cuda workloads using a detailed gpu simulator,” in *ISPASS 2009*.