Page Reuse in Cyclic Thrashing of GPU Under Oversubscription: Work-in-Progress

1st Dojin Park
Sungkyunkwan University
Suwon, Republic of Korea
djpark@arcs.skku.edu

2nd Hyunjun Kim
Sungkyunkwan University
Suwon, Republic of Korea
hjunkim@skku.edu

3rd Hwansoo Han
Sungkyunkwan University
Suwon, Republic of Korea
hhan@skku.edu

***Abstract*—Nvidia GPUs can oversubscribe CPU-side RAM, elevating GPU programmability and enabling GPU to use data beyond the size of GPU memory. However, oversubsrciption accompanies overhead of constant exchange of pages between the GPU and the CPU. To this problem, while previous works implemented and experimented solutions overhead in GPU simulators, we propose a new method implemented in the driver, which runs actual GPU hardware. We developed algorithms that detect repeated page replacement of the pages and pin pages in GPU memory to be reused in the future without replacement.**

***Index Terms*—GPU, unified memory, eviction policy, page reuse**

I. Introduction

In running CUDA Unified Memory applications, it hugely depresses the execution time when GPU references data that resides in CPU-side memory("oversubscribes"), due to the overhead of page replacement. Compared to applications that access each page only once, such as vector addition, for instance FDTD, NW, RA, which access pages multiple times, undergoes even more dramatic slow down due to repetitive replacement of the pages.

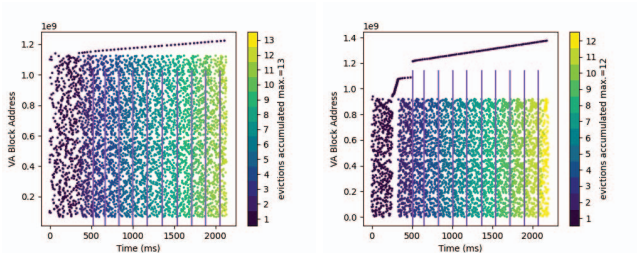
We conducted profiling on the NVIDIA GPU driver, to collect history of eviction of pages of such applications. In our analysis, we have discovered a common behavior, cyclic thrashing, which states a tendency of replacing same pages repeatedly. To speed up applications under cyclic thrashing, we have designed and implemented in the driver, a set of new functions that detects cyclic thrashing and alleviate thrashing by preserving pages in the GPU. By pinning pages in the GPU and eviction order, GPU can preserve pages from being evicted and possibly reuse in the future. Unlike previous works [1], [2], we implemented our algorithms purely on the driver and tested on the real hardware with large memory footprint, without using a GPU simulator [3].

II. Driver Design

A. Cyclic Thrashing Detector

The cyclic thrashing detector counts the number of evictions for all pages. Then, it groups pages by the same number of evictions, as a page eviction peer group. Then it declares a state of cyclic thrashing, if the size of peer group reaches at certain ratio of the size of peer group with one less eviction count. Cyclic thrashing detector ignores pages deviates eviction count from the mainstream peer groups. In Figure 1, the violet line indicates detection of cyclic thrashing tuned to raise at the peer group ratio of 75%.

This research is based upon work supported by the National Research Foundation in Korea under PF Class Heterogeneous High Performance Computer Development (NRF2016M3C4A7952587).



(a) 1,500MB data size (b) 1,300MB data size

Fig. 1: RA eviction of pages shown by VA block address on 1,000MB GPU, applied 25% page pinning.

B. Page Pinning

The first way to reduce thrashing is, at the point of detecting cyclic thrashing, to pin some of the pages in the GPU memory. By this way, the GPU reduces the total number of evictions by reusing these pages, thus reducing the kernel execution time. Page pinning continues as long as cycling thrashing is being detected.

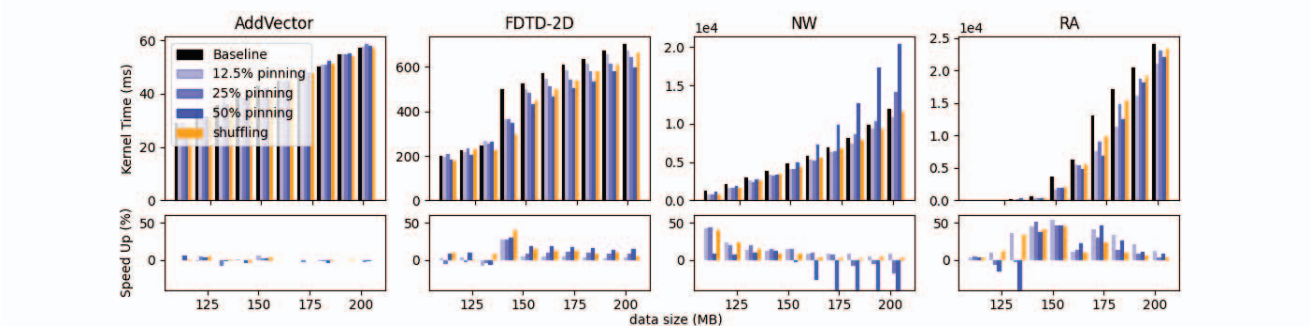
C. Eviction Order Shuffling

The second way is to shuffle the order of the LRU list of page eviction, to randomly preserve pages. As we will discuss in the evaluation, eviction order shuffling performs more stable in some of settings. To perform shuffling, peer group size ratio of cyclic thrashing detector is set low, with lower bound, to work with fragmented peer groups.

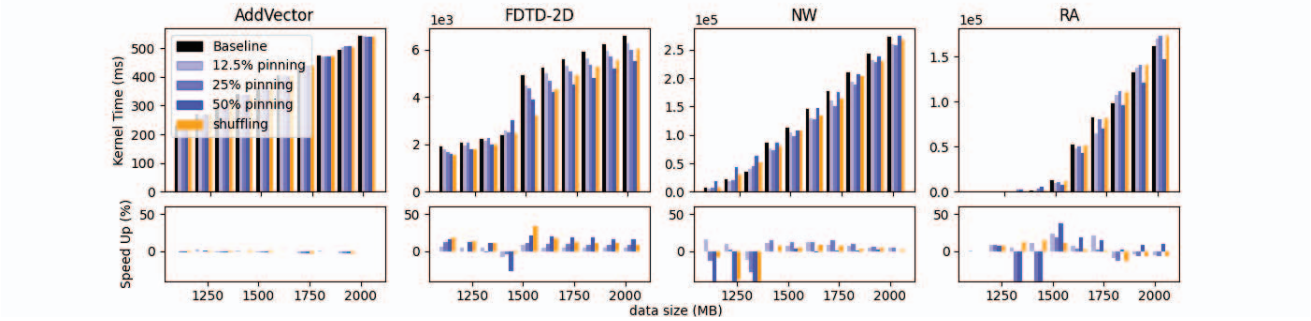
III. Evaluation

Figure 2. shows execution time of the baseline, page pinning with varying pinned sizes and eviction order shuffling on 100MB and 1,000MB GPU memory. The benchmarks AddVector, FDTD-2D, NW and RA are taken from Rodinia Suite [4] and Polybench [5].

978-1-7281-9192-8/20/$31.00 ©2020 IEEE



(a) 100MB GPU memory.



(b) 1,000MB GPU memory.

Fig. 2: Kernel execution time of the algorithms with different page pinning ratio from GPU memory size

The hardware used in these experiments are Intel i5-9400F, 16GB RAM and Nvidia GTX 1660 with 6GB memory.

The first column in the figure shows the AddVector kernel time. From our analysis, AddVector has shown to access all pages only once. It shows that there is no significant change in kernel time as long as cyclic thrashing detection is not raised.

Other benchmarks has shown speed up reaching 40% to 55% at maximum. However, NW and RA has shown severe anomalies on certain data size, slowdown reaching maximum at x2.1, with RA on 1,300MB data size on 1,000 GPU memory as in Figure 1b. We have found in our analysis, that these anomalies are caused by pinning pages that are not reused in the future. While Figure 1a. successively pinned pages that are reused in the future, achieving 38% speedup, Figure 1b. pinned pages in the upper region of VA block address, where pages are not reused in the future. This explains why the benchmarks experience slowdown anomalies in certain range of data size, because certain combination of data size, cycle thrashing detection sensitivity and page pinning ratio match together to pin pages that cause slowdown. Because of this, eviction order shuffling experiences less or no slowdown than page pinning algorithms.

IV. Conclusion and Future Work

From our evaluation benchmarks on our algorithms achieved speedup, but not stable. There is slowdown when in combination of certain data size, memory size, cyclic thrashing detection sensitivity and page pinning size which causes to pin pages that will not be reused in the future. Eviction order shuffling can avoid pinning some of the non-reused pages, but not completely and the speedup by this algorithm is limited.

A solution expected to address this problem is to release pinned pages that are not reused in the next cyclic thrashing detection and pin other candidate pages. One way to implement this is to keep track of page access in the driver. But since this could lead to higher underlying overhead, we can release pages whose neighboring pages do not experience eviction count increase, since page eviction peer groups lie in the contiguous VA block address because they are allocated by the same cudaMallocManaged API function. The maximum speedup by page pinning, approximately 40% to 50% in our evaluation, which we obtained when there was no non-reused pages pinned, is the expected performance of the best solution.

References

[1] G. et al., "Interplay between hardware prefetcher and page eviction policy in cpu-gpu unified virtual memory," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19, 2019.

[2] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Towards high performance paged memory for gpus," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 345–357.

[3] A. B. et al, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163–174.

[4] S. C. et al., "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009.

[5] S. G. et al., "Auto-tuning a high-level language targeted to gpu codes," in *2012 Innovative Parallel Computing (InPar)*, 2012.