PICLE: TLB Prefetcher for Inter-CU Locality Exploitation

Lipika Garg*, Rutwik Jain†, Rajesh Shashi Kumar* and Vishnu Ramadas*
*Electrical and Computer Engineering, University of Wisconsin-Madison
†Computer Sciences, University of Wisconsin-Madison
Email: {lgarg3, rnjain, rajesh.shashikumar, vramadas}@wisc.edu

***Abstract*—Systems today are embracing heterogeneous compute platforms such as integrated CPU/GPU systems. In such systems, virtual memory is useful to support a unified address space with CPUs and have a simpler programming model. However, address translation bottlenecks can significantly degrade GPU application performance - L1-TLBs of GPU cores have miss rates as high as 99% for some GPU applications. Therefore, we explored virtual memory architecture in GPUs and examined potential avenues for improvement. Building on prior work that proposes mechanisms for improving TLB performance, we implemented modest changes in the architecture by introducing a prefetcher called PICLE, which exploits inter-TLB locality to improve GPU virtual memory performance. Finally, we implemented the proposed prefetcher using gem5 and achieved 2.4% performance improvement.**

***Index Terms*—GPUs, virtual memory, TLBs, prefetching**

I. Introduction

GPUs have emerged as first class computing platforms, repurposed from their original use for accelerating graphics to more general-purpose workloads, such as deep learning, graphic analytics, cryptocurrency mining, weather modeling etc. All these applications have high data-level parallelism, which makes computing via GPUs affordable as they employ thread-level parallelism to hide memory access latency. To lower the burden of programming on application developers, AMD [4] and Nvidia [5] released GPU architectures with Unified Memory (UM) or Shared Virtual Memory (SVM).

Unified virtual memory exposes a common address space to both the host and the GPU. It enables addresses to be meaningful between the host and all of the devices within a context and therefore supports the use of pointer based data structures [4]. This is useful because it eliminates the need to perform explicit memory copies and manages data transfer in a CPU-GPU or multi-GPU system [1]. It also allows oversubscription of memory resources. Support for virtual memory includes Memory Management Units (MMUs), page table walkers and TLBs on the hardware side and API calls and GPU driver support on the software side.

However, enabling virtual memory comes with it's own sets of bottlenecks. Every memory access can require up to four lookups, and features like demand paging and page evictions can significantly degrade performance. Additionally, GPU applications often have low spatial locality. Such irregular memory accesses can slow down GPU applications due to virtual-to-physical memory translation overheads. TLB hierarchies are critical to virtual memory performance, which is often limited by the size of L1-TLBs per-CU (Compute Unit, or Streaming Multiprocessor). This creates additional pressure on the shared L2-TLB. In this work, we explore ways to reduce overheads due to TLB misses. To improve L1-TLB hit rates, prior work [1] proposed TLB prefetching mechanisms detailed in Section II-A. In this work, we first studied the gem5 [6] Memory Management Unit (MMU) to understand how the simulator carries out virtual memory performance modelling. We then replicated the Valkyrie prefetcher which takes advantage of temporal inter-core page sharing behavior [1]. Finally, we validated the functionality of our implementation and evaluated it by running a subset of applications from the Rodinia benchmark suite [2].

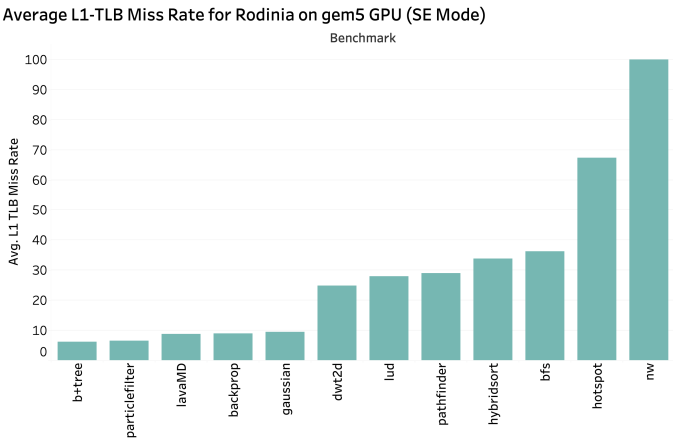

Fig. 1. Average L1-TLB Miss Rates using the gem5 GCN3 GPU model for a subset of applications from the Rodinia benchmark suite.

II. Picle

A. Motivation

GPUs have thousands of threads concurrently in-flight which generate a lot of memory traffic, consequently pressuring TLBs and caches severely. Baruah et. al carried out workload characterization to show extremely high miss rates in per-CU L1-TLBs [1]. We ran a subset of benchmarks from Rodinia using the gem5 GCN3 GPU model in System Emulation (SE) mode to (i) corroborate their findings, (ii) validate functional correctness of the virtual memory model in gem5 SE mode and (iii) get baseline results that we could

compare with later. Figure 1 shows the results from this characterization. NW has a near 100% miss rate, while 9 out of the 12 benchmarks have $> 20\%$ miss rates.

Based on observations of page sharing and reuse across CUs in GPU applications, Baruah et. al proposed *Valkyrie* [1], a set of hardware mechanisms to improve L1-TLB hit rates. They recognized that the same page translations or Page Table Entries (PTEs) are requested by different cores over time. By tracking which cores share PTEs, there is a potential to prefetch these entries ahead of time to improve performance.

B. Background: Valkyrie

Valkyrie proposed an integrated cooperative TLB prefetching mechanism, shown in Figure 2. It consists of a hardware structure called the Locality Detection Table (LDT), which stores a hashed tag value per PTE and a bitmap to indicate which CUs have hit in the L1-TLB for this page before. On every L2-TLB eviction, the LDT is looked-up for the corresponding tag and using the bitmap, the PTE is prefetched into the L1-TLBs of the appropriate CUs. While this improves L1-TLB hit rates if accesses have temporal locality, prefetching these entries into L1-TLBs that are already full might cause eviction of useful entries and degrade hit rates. Thus, the effectiveness of PICLE depends on the amount of temporal locality the application has, the memory footprint of the application and the sizes of the TLBs.

PICLE is our attempt to replicate the Valkyrie prefetcher using gem5's GCN3 GPU model.

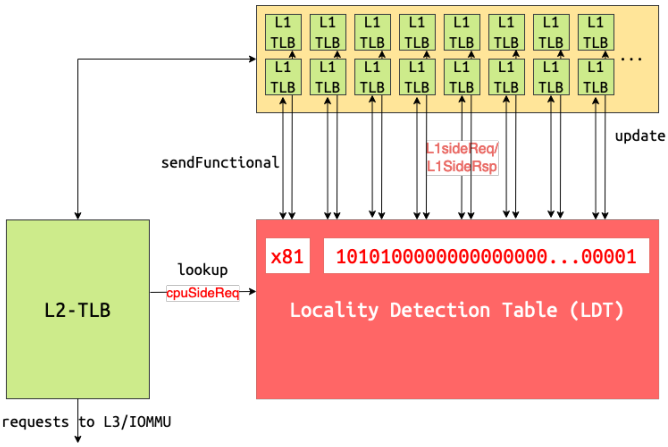

Fig. 2. Block diagram of the Valkyrie/PICLE prefetching mechanism. Newly added hardware structures and interfaces are marked in red.

III. Implementation

A. gem5 GPU Model

The gem5 GPU model supports AMD's GCN and VEGA ISAs. We used the GCN3 ISA for our experiments. gem5 also uses different TLB models for each operating mode. Full system (FS) mode uses a different model as compared to the Syscall emulation (SE) mode [3]. We chose to use the SE mode because of two advantages it has over FS. First, the runtimes are much faster in SE since it does not run a software stack as elaborate as the one in FS mode. Second, SE is much more widely used over FS and contains significantly more online resources. We also observed that the FS model results in many more TLB accesses and misses because it runs an operating system image.

The Memory Management Unit (MMU) in gem5 has two modes of operation in SE mode. It can be used either as a purely functional unit with zero latency or with access and miss latencies accounted for [3]. We chose the latter and found that it's structure to be as follows:

B. Locality Detection Table

Since the Locality Detection Table (LDT) is at the core of the prefetcher, we model it as a class that can interface with the different TLB levels. The LDT consisists of a list that keeps track of which virtual address was accessed by which CUs. This list is used to send the translation requests that get evicted by the TLB to the CUs that used them in the past. The LDT contains request port to receive evictions from L2 and a pair of request and response ports per CU to receive updates/send translation entries. The size of the LDT and latencies for lookup and update operations are derived from parameters exposed to the architect.

C. Transaction Lookaside Buffer

The SE mode uses a generic TLB class to store page translation entries. The same class is used across all levels of the MMU hierarchy (which are three in this case). We added two request ports and a response port to the TLB class to facilitate interactions with the LDT. While one request port is used only at the L2 level, the other two ports are used at the L1. Whenever a page translation request enters the L1 TLB, an update is issued to the LDT in parallel to servicing the request. Similarly, whenever the L2 TLB evicts a page table entry on a capacity miss, it sends the information to the LDT. This spawns a series of actions that culminate in L1 TLBs receiving translation entries from the LDT. These entries are inserted into the TLB.

D. Integration and Verification

gem5 MMU consists of one L1 TLB per CU, one global L2 and L3 TLB each. Since the number of evictions in L3 TLB will be less than in the L2, we interfaced our LDT with the L2 TLB. Once LDT was integrated with the MMU hierarchy, we functionally verified it using square from gem5-resources [6]. We observed that LDT works as expected but running `square` results in a performance degradation by suffering 1% extra misses. These misses can be attributed to TLB pollution and the sequential access nature of square leads to more misses with the prefetcher. The TLB class update, along with the new LDT class and the top-level connections made to gem5 are captured below in Figure-3.

IV. Evaluation and Results

In this section, we discuss the results from the evaluation of the LDT implementation and some important considerations in the process.

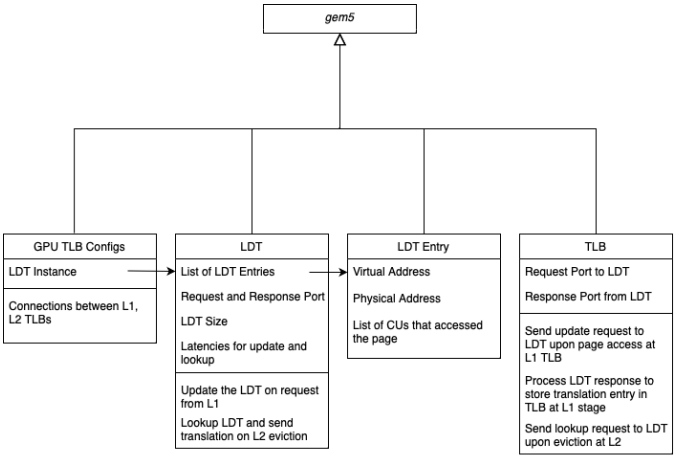

Fig. 3. gem5 Model Changes.

Simulator: As we know, the operating system is responsible for virtual memory and page table management. In gem5, the GPU FS mode boots an image and models memory management with full fidelity but is slower. However, the SE mode uses less accurate modeling and relies on delay annotations to mimic TLB misses and page table walk. In our evaluation, we ensured that the TLB miss latency is modeled to be equivalent between FS and SE. To do so, we relied on bringing up both FS and SE simulations to valid our assumptions. In the process, we made upstream contributions to gem5 in the form or a merged pull request, bug report and mailing list exchanges. Figure 4 shows square runtime being 16 times faster in SE mode enabling design space exploration and debugging.

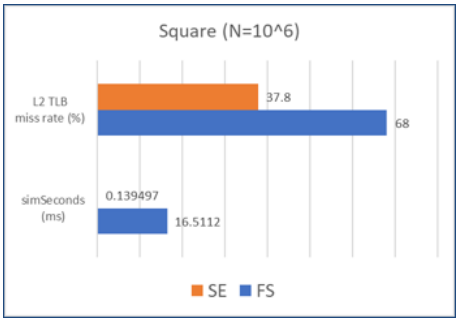

Fig. 4. FS vs SE: Runtime and L2 TLB miss rate comparison for square.

GPU system: We used the configuration listed in Table I. The sizing of TLBs was intentionally reduced so that the effect of prefetching and pollution of TLB would be prominent.

Benchmarks: We used square and the Rodinia suite [2] for evaluation. Rodinia comprises of a representative set of parallel heterogeneous computing workloads that would capture the diverse TLB access patterns. We excluded some applications that either had a long tuntime (huffman, hotspot3D, streamcluster, cfd) or lack of texture support in gem5 (kmeans, leukocyte, heartwall).

TABLE I
GPU VIRTUAL MEMORY CONFIGURATION USED IN EVALUATION.

Component	Configuration
Compute Unit	64
Shader Engines	4
L1 TLB	64
L1 TLB size	4 sets, 1 way
L2 TLB	1
L2 TLB size	16 sets, 1 way
L1 TLB miss latency	150 cycles
Page walk latency	750 cycles

Discussion: Figure 5 captures the average L1 TLB miss rate across baseline and LDT simulations on the reference system. Originally, Valkyrie implemented the LDT at the last level of cache. While they use a 2 level TLB structure, gem5 models an addition level of L3 TLB. To rule out any effects of this difference, we added a second configuration represented by the red bars, that removed the L3 TLB. With LDT, we observed a 2.4% and 0.8% reduction in L1-TLB miss rate compared to the baseline for the LUD and hotspot applications respectively. Also, there was a 2.3% increase in miss rate observed with backprop. The quantitative numbers may seem low at first for a prefetcher at first, but there are three important aspects to remember here. First, unlike cache prefetchers, LDT is only ever active when there is an eviction at the L2 TLB. Second, the prefetch at page granularity is much larger than the cache line granularity accounting for fewer misses especially in applications with regular memory access patterns or low-memory footprint. Furthermore, Valkyrie claims significant performance benefits with the introduction of probing which remains unimplemented in our work. Third, TLB lookups unlike cache happen in parallel and are not on the critical path so the effect of miss latency may not show up prominently with changes to how L2 TLB evictions are handled. These considerations can explain the same performance as baseline in most applications or a marginally better performance in some applications such as LUD and Hotspot. To explain the degradation in case of backprop, we resorted to experiments with the square application for varying input sizes. As shown in Figure 6, the degradation was reproducible and we attribute this to TLB pollution as a result of re-insertion of evicted entries from L2 that may no longer be needed. Avoiding this degradation would require the probing mechanism from Valkyrie to be implemented.

V. Conclusion and Future Work

With chiplet-based GPU's around the horizon, non-uniform TLB access is a growing concern. In this work, we investigate a monolotithic GPU TLB implementation to see how we can exploit locality and prefetching to improve TLB lookup performance. Our study reveals that there is scope for hardware optimizations in GPU virtual memory managements, but the benefits are heavily dependent on application memory access patterns (where locality comes into play) and system configuration for the sizing of various TLB structures. Another

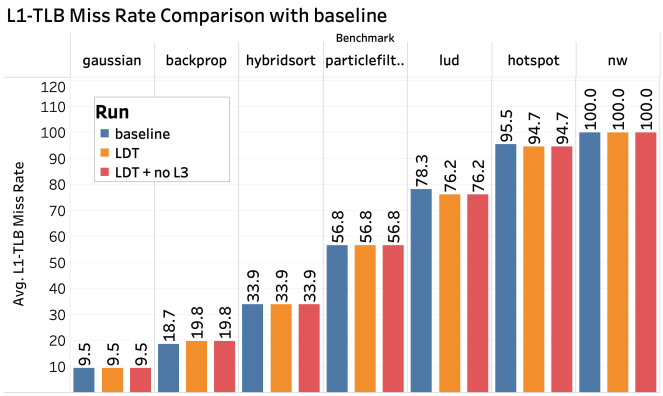

Fig. 5. Results: Average L1-TLB Miss Rate comparison for three configurations - baseline is in blue, while yellow and red denote PICLE and PICLE with L3-TLB size reduced to 1 entry respectively.

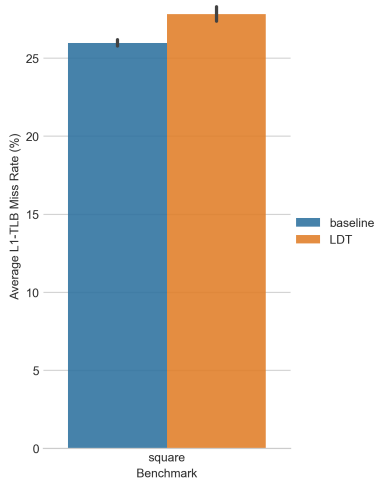

Fig. 6. Avg L1-TLB Miss Rate for `square` with and without the LDT.

key learning from this work was highlighting the challenges of modeling GPU virtual memory in a simulator, specifically gem5 and understanding the trade-offs of the different modes of simulation. We also came to the realization that the choice of benchmarks or microbenchmarks is key to the evaluation of virtual memory performance since it is not on the critical path.

For future work, we want to extend our design to include the prober proposed in Valkyrie by implementing the interconnect infrastructure for communicating between the TLBs. There is also scope for further design space exploration with the LDT to improve performance. Finally, evaluating the implementation on multi-chip module GPUs when gem5 support becomes available would be imperative to analyze the reduction in virtual memory overheads with the proposed design.

REFERENCES

[1] Trinayan Baruah, Yifan Sun, Saiful A. Mojumder, José L. Abellán, Yash Ukidave, Ajay Joshi, Norman Rubin, John Kim, and David Kaeli. 2020. Valkyrie: Leveraging Inter-TLB Locality to Enhance GPU Performance. In Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques (PACT '20).

[2] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," 2009 IEEE International Symposium on Workload Characterization (IISWC), 2009, pp. 44-54, doi: 10.1109/IISWC.2009.5306797.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti et al. 2011. The gem5 simulator. ACM SIGARCH Comput. Archit. News 39, 2 (2011), 1–7.

[4] AMD. 2017. Radeons Next-generation Vega Architecture. https://radeon.com/_downloads/vega-whitepaper-11.6.17.pdf. (2017).

[5] NVIDIA. 2018. NVIDIA Pascal Architecture. https://www.nvidia.com/en-us/data-center/pascal-gpu-architecture/. (2018).

[6] gem5 - gem5 Resources (stable v22.0.0.2). Available online at: https://gem5.googlesource.com/public/gem5/.