Remote Access Tag Array for Efficient GPU Intra-Cluster Data Sharing

Bo-Wun Cheng† En-Ming Huang† Chen-Hao Chao† Wei-Fang Sun†
Tsung-Tai Yeh‡ Chun-Yi Lee†*
† ELSA Lab, Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
‡ Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan

Abstract— In this work, we aim to address the memory congestion problem of modern GPUs by incorporating a remote access tag array (RATA) into the baseline architecture. With the assistance of RATA, GPUs are able to service replicated cache requests within stream multiprocessor (SM) clusters without resorting to the level-two (L2) cache. Our experimental results show that the adoption of RATA has the potential to alleviate the memory congestion problem and enhance the overall system throughput.

I. Introduction

Contemporary graphics processing unit (GPU) architectures feature a large number of stream multiprocessors (SMs) grouped into multiple SM clusters [1]. To conserve memory bandwidth, each SM is equipped with its own private level-one (L1) cache, which is backed up by a unified level-two (L2) cache connected via a network-on-chip (NoC) fabric, as depicted in Fig. 1. As the number of SMs increases, memory bandwidth often becomes the performance bottleneck due to congestion [2]. Moreover, the clustering of SMs further exacerbates this problem as all SMs within a cluster share a common link to the NoC.

The bandwidth inefficiency of the current cache configuration stems from the private nature of the L1 caches. Due to the lack of communication among them, cache lines may be fetched repeatedly from the L2 cache by different SMs [2, 5]. Fig. 2 demonstrates such a scenario in terms of the replication rate, which is defined as the percentage of cache miss requests that can be found in other L1 caches within the same cluster, for a number of benchmarks [3, 4]. It is observed that for the replication sensitive benchmarks, such as *cfd*, *lud*, *AlexNet*, and *ResNet*, 68% of the global memory reads are dedicated to fetching cache lines that are already cached within the same cluster, leading to inefficiency in bandwidth usage.

In light of the above issue, an array of works have been proposed to reduce the number of replicated cache requests by either employing a shared L1 cache [5] or introducing extra inter-SM links [2]. In this work, we explore the use of a per-cluster remote access tag array (RATA), a cache-like structure similar to that used in [6], to efficiently service replicated requests within clusters with an aim to alleviate the memory congestion problem. To validate our idea, we perform two motivational experiments and show that the adoption of RATA is potentially beneficial in boosting the throughput of the baseline GPU.

*Corresponding author. E-mail: cylee@cs.nthu.edu.tw

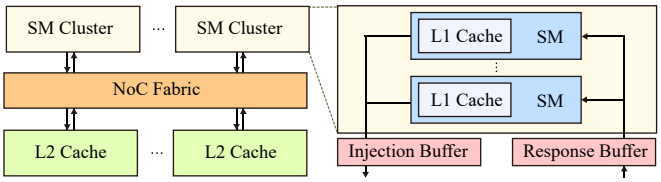

Fig. 1. The baseline GPU architecture containing SM clusters.

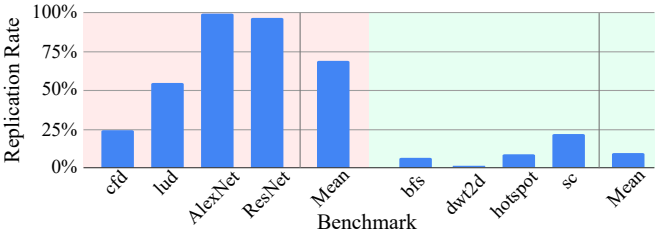

Fig. 2. The replication rate of the replication-sensitive (left-hand side) and the replication-insensitive benchmarks (right-hand side).

II. Methodology

In this section, we introduce the detailed organization and workflow of RATA, a per-cluster component that manages the inter-SM access traffic within an SM cluster. Note that in the following sections, RATA is incorporated into a baseline GPU featuring ten 8-SM clusters, 48KiB of per-SM L1 cache, and a 40-bit global address space.

In the proposed architecture, RATA is employed to keep track of the SMs that cache a copy of a specific cache line. In order to enable fast lookup and replacement, RATA is constructed as a typical set-associative data cache that employs least recently used replacement policy. Similar to a typical data cache, each entry within RATA comprises of a 1-bit valid indicator and a 27-bit cache tag to enable efficient lookup operation. However, instead of storing the full 128-byte data line, RATA only stores a 3-bit data field to indicate which SM within the cluster has lastly requested for a copy of that cache line from the L2 cache.

Fig. 3 shows the inter-SM access workflow of RATA. The process is elaborated as follows. ① As a memory response exits the response buffer, a RATA entry is allocated and set up to point to the SM that has requested for the cache line. ② On an L1 cache miss, RATA is accessed to check whether the missed line resides in one of the L1

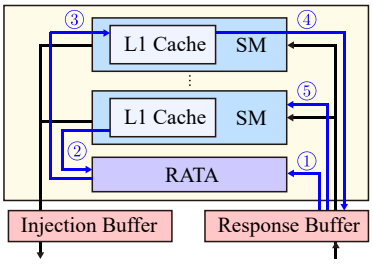

Fig. 3. The workflow of RATA consisting of five steps.

caches in the cluster. ③ If the RATA access results in a hit, the memory request is redirected to access the L1 cache of the SM pointed to by the RATA entry. ④ If the L1 cache access hits, the requested cache line is pushed to the tail of the response buffer to be send to the requesting SM. ⑤ Finally, as the request reaches the head of the response buffer, it is directed to the original requesting SM, completing the inter-SM access process. Please note that in ⑤, RATA is not updated and the requested cache line is not filled into the L1 cache of the requesting SM. Due to the shared nature of RATA, an entry within RATA can be replaced before its corresponding line is evicted from the L1 cache, leading to a RATA lookup miss. This is referred to as the *false-negative error* hereafter. In addition, the L1 cache access in ④ does not guarantee a hit as RATA is not informed of cache line evictions in the L1 caches. Thus, the information RATA holds can be out-of-date. This is referred to as the *false-positive error* hereafter.

III. Experimental Results

In this section, we present two motivational experiments to demonstrate the potential benefit of RATA. In these experiments, we model the baseline and our proposed architecture using the GPGPU-sim [7] simulator, and perform simulation on a number of benchmarks selected from the Rodinia [3] and the Tango [4] benchmark suites. Please note that in the following experiments, we assume that the accesses to RATA have zero latency.

In the first experiment, we assume that the per-cluster RATA can capture all potential replicated requests. In other words, in this ideal scenario, RATA does not incur any false positive or false negative error. Fig. 4 shows the performance gain of incorporating RATA in terms of instructions per cycle (IPC). It is observed that the performance of the baseline is improved by an average of 134% across the replication sensitive benchmarks. RATA allows part of the cache misses, which would have consumed precious bandwidth to the L2 cache, to be serviced within the cluster by the other SMs. As a result, the memory congestion is relieved, enhancing the overall system throughput.

In the second experiment, we simulate the actual replacement behavior of RATA to demonstrate its effectiveness in capturing replicated requests. Our experimental results suggest that even with the presence of the false-positive and false-negative errors, RATA is can still capture 94% of all potential replicated requests across the replication sensitive benchmarks, as shown in Fig. 5.

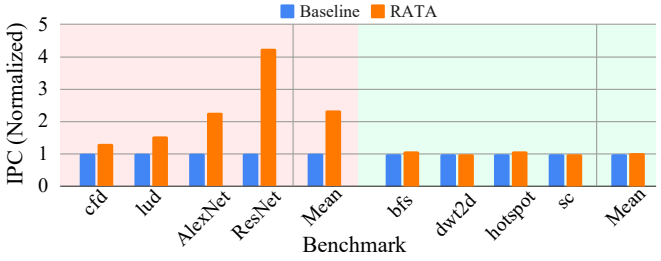

Fig. 4. The normalized IPC in under ideal scenario. The left (red) and the right (green) half of the image show the performance of the replication-sensitive and insensitive benchmarks, respectively.

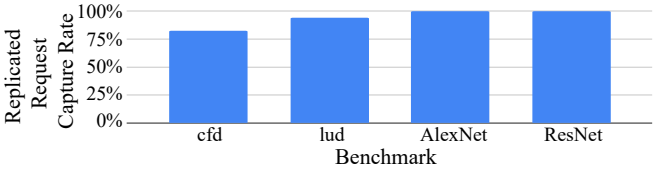

Fig. 5. The percentage of replicated requests captured by RATA.

IV. Conclusion

In this work, we propose to alleviate the memory congestion problem by introducing RATA. Our experimental results suggest that RATA is able to capture most of the replicated memory requests, and can potentially ease memory congestion and enhance GPU performance.

Acknowledgment

This work was supported by the National Science and Technology Council (NSTC) in Taiwan under grant number MOST 111-2638-E-007-010-. We would like to thank the effort of the reviewers for reviewing this paper.

References

[1] NVIDIA, "NVIDIA Amphere GA102 GPU Architecture", https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf

[2] Dublish *et al.*, "Cooperative caching for GPUs", *ACM Trans. Archit. Code Optim.*, vol. 13, no. 39, 2016.

[3] Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing", *IEEE Int. Symp. on Workload Characterization (IISWC)*, pp. 44-54, 2009.

[4] Karki *et al.*, "Detailed characterization of deep neural networks on GPUs and FPGAs", *Wksp. General Purpose Processing Using GPUs*, pp. 12-21, 2019.

[5] Ibrahim *et al.*, "Analyzing and leveraging decoupled L1 caches in GPUs" *Int. Symp. High-Performance Computer Architecture*, pp. 467-478, 2021.

[6] Tarjan *et al.*, "The sharing tracker: Using ideas from cache coherence hardware to reduce off-chip memory traffic with non-coherent caches", *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, pp. 1-10, 2010.

[7] Bakhoda *et al.*, "Analyzing CUDA workloads using a detailed GPU simulator", *Int. Symp. Performance Analysis of Systems and Software*, pp 163-174, 2009.