COLAB: Collaborative and Efficient Processing of Replicated Cache Requests in GPU

ABSTRACT
In this work, we aim to capture replicated cache requests between Stream Multiprocessors (SMs) within an SM cluster to alleviate the Network-on-Chip (NoC) congestion problem of modern GPUs. To achieve this objective, we incorporate a per-cluster Cache line Ownership lookup tABle (COLAB) that keeps track of which SM within a cluster holds a copy of a specific cache line. With the assistance of COLAB, SMs can collaboratively and efficiently process replicated cache requests within SM clusters by redirecting them according to the ownership information stored in COLAB. By servicing replicated cache requests within SM clusters that would otherwise consume precious NoC bandwidth, the heavy pressure on the NoC interconnection can be eased. Our experimental results demonstrate that the adoption of COLAB can indeed alleviate the excessive NoC pressure caused by replicated cache requests, and improve the overall system throughput of the baseline GPU, while incurring minimal overhead. On average, COLAB can reduce 38% of the NoC traffic and improve instructions per cycle (IPC) by 43%.

1 INTRODUCTION
Modern graphics processing unit (GPU) architectures feature a number of Stream Multiprocessor (SM) clusters, each of which consists of multiple SMs [17]. To conserve memory bandwidth, each SM utilizes a private level-one (L1) cache, which connects to a unified level-two (L2) cache via an Network-on-Chip (NoC) interconnection, as depicted in Fig. 1. As L1 caches are unable to communicate with each other due to their private nature, all cache requests caused by L1 cache misses are sent across the NoC fabric to the L2 cache. However, it has been pointed out by previous studies [3, 4, 6–8, 22] that a number of cache requests are issued to fetch lines that have already been cached by other SMs, leading to inefficient use of the NoC bandwidth and incurring burden on it.

Fig. 2 quantifies the inefficiency across several benchmarks in terms of replication rate, which is defined as the percentage of cache misses that can be found in the other L1 caches within the same cluster. Please note that in this work, a benchmark that induces 25% or a higher replication rate is defined as a replication sensitive benchmark. Otherwise, it is defined as a replication insensitive benchmark. It is observed that among the replication sensitive benchmarks, an average of 52% of the cache misses are issued to fetch cache lines that have already resided in at least one of the other L1 caches within the same SM cluster. In other words, more than half of the NoC bandwidth in the baseline architecture is consumed by replicated cache requests that could have been serviced within SM clusters. By enabling inter-SM communications, the redundant NoC traffic caused by replicated cache requests can be avoided, which eases NoC pressure and enhances performance.

Several methods have been proposed to address the problem of replicated cache requests in GPUs. Some of them utilize consolidated L1 caches [3, 6, 7, 22] or probing-based inter-SM communication schemes [4, 8] to reduce the number of replicated cache requests sent to the NoC interconnection. Nonetheless, these methods either introduce new performance bottlenecks [3, 6, 7, 22], or incur latency and energy overhead to cache requests that cannot be serviced by other SMs [4, 8]. Meanwhile, another work [21] has proposed to utilize a directory-like structure to capture inter-SM replications across the entire GPU to reduce its off-chip memory traffic. However, it has been pointed out by [4] that the architecture proposed in [21] does not bring positive benefits to modern GPUs.

In this work, we propose Cache line Ownership lookup tABle (COLAB) to deal with the problem of replicated cache requests by enabling SMs to collaboratively process such requests locally within a cluster. Specifically, within an SM cluster, COLAB records cache line ownership information and redirects replicated cache requests to their corresponding owners accordingly. By incorporating COLAB into the baseline architecture in Fig. 1, replicated cache requests can then readily be serviced within a cluster, rather than resorting to the L2 cache through the NoC interconnection. This allows the NoC traffic to be alleviated, and in turn, mitigates the stalls in the SMs and improves the throughput of the entire GPU.

To demonstrate the effectiveness and validate the design of COLAB, we faithfully model the proposed architecture and the baseline GPU using the famous and widely adopted GPGPU-Sim [12] simulator, and conduct a series of experiments on both replication sensitive benchmarks and replication insensitive benchmarks. Our experimental results demonstrate that the incorporation of COLAB is indeed able to capture and service replicated requests within SM.
clusters, leading to noticeable improvements in NoC traffic, execution throughput, as well as stalls in SMs for the replication sensitive benchmarks. Moreover, the experimental evidence indicates that COLAB brings negligible negative performance impact when it is evaluated on the replication insensitive benchmarks. Furthermore, the results also reveal that COLAB can assist in decreasing the energy consumption by reducing the traffic that flows through the NoC fabric as well as the number of L2 cache accesses for replication sensitive benchmarks. In addition to the above insights, we further offer ablation studies to examine the design choices of COLAB. The main contributions of this paper can be summarized as follows:

- We highlight the issue of replicated cache requests in GPUs.
- We propose COLAB, an architecture that allows replicated cache requests to be redirected and serviced efficiently within a cluster by utilizing the cache line ownership information.
- We experimentally validate that the incorporation of COLAB is able to reduce the NoC read traffic by an average of 38% and improve the overall throughput by an average of 43%.
- We offer a set of ablation studies to justify the design choices.
- We investigate and analyze the energy reduction of different GPU components achieved by the incorporation of COLAB.

The remainder of this paper is organized as follows. Section 2 presents the designs and the workflow of COLAB. Section 3 reports the experimental results and examines their implications. Section 4 discusses the works related to this paper. Section 5 concludes.

2 METHODOLOGY

In this section, introduces the COLAB and present its design details. In Section 2.1 we provide an overview on the functionality and objective of COLAB. Section 2.2 presents the workflow involving COLAB, and show how it is incorporated into the baseline GPU architecture. In Section 2.3, we describe the organization of COLAB.

2.1 Overview

As discussed in Section 1, NoC congestion has become a major performance bottleneck for modern GPUs. Moreover, as shown in Fig. 2, a significant portion of cache requests is issued to fetch cache lines that have already been stored within the cluster, wasting precious NoC bandwidth. In light of this problem, we propose COLAB, a per-cluster structure used to capture and redirect replicated cache requests within the cluster, to alleviate pressure on the NoC network. Specifically, COLAB keeps track of which SM within the cluster holds a copy of a specific cache line. By consulting COLAB, subsequent replicated cache requests can then be redirected and be serviced within the cluster. Fig. 3 illustrates an example of such a process within an SM cluster. In this example, both SM0 and SM7 need to fetch cache line A. As SM7 issues its request for A, a copy of A has already been cached in SM0 and has been recorded by COLAB to reflect this situation. By the time SM7 issues its request for A, instead of sending it to the L2 cache, SM7 consults COLAB which redirects the cache request to SM0. By redirecting and servicing cache requests like the one depicted in this example within a cluster, unnecessary traffic to the L2 cache via the NoC fabric can be avoided, and hence, the traffic burden on NoC can be mitigated.

Figure 3: An illustration of how a repeated request for a cache line can be serviced with the assistance of COLAB.

2.2 Workflow of COLAB

In this section, we walk through the workflow and explain the design details along with the arbitration policy employed by COLAB.

2.2.1 Workflow. Fig. 4 illustrates the workflow of an SM cluster augmented with COLAB. The detailed procedure of the workflow is described as follows. ① As a response of a cache request exits the response buffer, COLAB is updated to associate the cache line with the requesting SM. ② On an L1 cache miss, the cache request is sent to the input queue of COLAB, which is employed to serialize simultaneous requests from the SMs. However, if the input queue is full, subsequent miss requests are sent to the L2 cache until there is a slot available. As the cache request reaches the head of the input queue, COLAB is accessed to check whether the missed line resides in any of the L1 caches within the cluster. ③ If the COLAB access results in a hit, the cache request is redirected to the access queue of the SM indicated by COLAB. As the requests reaches the head of the access queue, a normal L1 cache access is performed. ④ If the L1 lookup results in a hit, the requested line is retrieved from the L1 cache and pushed to the tail of the response buffer to be sent to the requesting SM. ⑤ Subsequently, as the requested line reaches the head of the response buffer, it is sent to the requesting SM, completing the access process. ⑥ On the other hand, if the L1 lookup results in a miss due to outdated information in COLAB, the request is directed to the L2 cache. ⑦ Similarly, if the COLAB access results in a miss or if the access queue of the targeted SM is full, the request is also sent to the L2 cache. Please note that in ⑤, COLAB is not updated and the cache line is not filled into the L1 cache of the requesting SM to avoid storing replicated cache lines. Please also note that COLAB only processes read requests, while write requests follow the default mechanism of the baseline GPU.

2.2.2 False-Positive and False Negative Errors. Due to the shared nature of COLAB, an entry within COLAB can be overwritten before its corresponding line is evicted from the L1 cache, leading to a COLAB lookup miss. This is referred to as the false-negative error hereafter. Furthermore, since COLAB is not informed of cache line evictions from the L1 caches, the information COLAB holds can become stale. This in turn leads to L1 lookup misses as cache requests are incorrectly redirected due to the outdated information. Such an error is referred to as the false-positive error hereafter.

2.2.3 Arbitration Policy between COLAB and Local L1 Requests. As L1 cache bandwidth is shared between requests from COLAB and local cache accesses, a mechanism is required to arbitrate these accesses. In this work, COLAB requests are granted priority over
local L1 accesses. By prioritizing requests from COLAB, they are less likely to be suffered from starvation that might cause excessive stalling of COLAB. The arbitration policy greatly reduces the number of requests redirected to the L2 cache (i.e., 2 and 3), allowing more replicated cache requests to be serviced within the cluster.

2.3 Organization of COLAB

In this section, we present the detailed architectural organization of COLAB along with a discussion on its estimated hardware overhead.

2.3.1 Detailed Architecture of COLAB. In order to support fast lookup of cache line ownership information, COLAB is implemented as a typical set-associative tag array. Fig. 5 depicts the organization of COLAB. Each entry within COLAB consists of a 1-bit valid indicator and a 27-bit tag field, and the entries in COLAB are replaced using the least recently used (LRU) replacement policy. Unlike normal data caches that store complete 128-byte lines, each entry in COLAB only keeps a 3-bit SM pointer (assuming 8-SM clusters) to indicate the SM that has previously requested for the cache line. To redirect a cache request, COLAB utilizes a demultiplexer to forward the request to the SM pointed by the SM pointer.

2.3.2 Estimated Hardware Overhead. The hardware overhead of incorporating COLAB is mainly contributed by the storage capacity required by COLAB and its associated queues mentioned in Section 2.2.1. In this work, the number of entries in COLAB is configured to match the total number of the L1 cache entries within a cluster (i.e., 8-SM clusters and per-SM L1 cache of 64KB). Additionally, COLAB is equipped with a 32-entry input queue, and each SM is augmented with an 8-entry access queue. As shown in Table 1, the total overhead of incorporating COLAB is an extra storage of 12.375KB per SM cluster, which is only 2% of the total L1 cache capacity of a cluster. Furthermore, since the L1 caches only consume a portion of the overall die area of a GPU, we conclude that the incorporation of COLAB only incurs minimal overhead.

3 EXPERIMENTAL RESULTS

In this section, we present our experimental results to validate the effectiveness of COLAB. First, we provide the detailed simulation configuration in Section 3.1. Next, we show the performance improvement and the NoC traffic reduction achieved by incorporating COLAB in Section 3.2. Subsequently, we compare the energy consumption of the proposed architecture to that of the baseline in Section 3.3. Finally, in Section 3.4, we perform ablation analysis to justify the design choices made in the proposed architecture.

3.1 Experimental Setup

In this work, we utilize GPGPU-Sim [12], a cycle-accurate GPU simulator, to model the proposed architecture and the baselines. Table 2 presents the detailed configuration employed in the experiments of this work. The baseline GPU assumed is similar to the one depicted in Fig. 1 and described in Section 1. Please note that the terminologies used in Table 2 correspond to those used by NVIDIA and the latency of COLAB is derived conservatively using CACTI [16]. For energy simulation, we estimate the energy consumption of COLAB conservatively using CACTI and model the other components within the GPU using GPUWatch [14]. We evaluate our architecture and the baselines on 14 benchmarks from the Rodinia [1], Tango [10], and Polybench [5] benchmark suites.

Table 1: Hardware required per cluster to include COLAB.

<table>
<thead>
<tr>
<th>Module</th>
<th># of entries</th>
<th>Size/Entry</th>
<th>Total size</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLAB</td>
<td>64 (set) × 6 (way) × 8 (SM) = 3,072</td>
<td>31 bits</td>
<td>11.625KB</td>
</tr>
<tr>
<td>Queues</td>
<td>32 (input) + 8 (access) × 8 (SM) = 96</td>
<td>8 bytes</td>
<td>0.75KB</td>
</tr>
</tbody>
</table>

Table 2: The configuration of the baseline GPU and COLAB.

| # of SMs       | 80, 8 per cluster @1,481MHz           | 96KB shared memory, |
| Resources/SM   | Max. 2048 thread                       | Max. 32 thread blocks. |
| L1 Caches/SM   | Data: 64KB, 6-way, latency=28 cycles, | Texture: 48KB, 24-way, |
|                | Constant: 12KB, 2-way.                | Instruction: 4KB, 4-way. |
| L2 Cache       | 128KB 16-way/Channel (2.75MB total),  | 128 bytes per line, Latency=120 cycles. |
| NoC            | 10 × 22 crossbar @2.962MHz            | Hynix GDDR5 timing. |
| DRAM           | 11 partitions of GDDR5 @2.750MHz.     | Hynix GDDR5 timing. |
| COLAB          | Per-cluster 64-set 48-way lookup table | latency=8 cycles   |
Overall Others Normalized NoC

Normalized Throughput

Normalized # of SM Stalls

The reduction in stalls cycles in turn enhances the utilization of the SMs, and boosts the overall throughput of the GPU. The improvement in execution throughput is illustrated in Fig. 7, which shows that the employment of COLAB is able to improve the IPC of the baseline GPU by an average of 43%. However, it is noticed that the NoC traffic and IPC improvements of cfd and gaussian are rather limited as compared to the other replication sensitive benchmarks. This is mainly due to their relatively lower replication rates, as indicated in Fig. 2. On the other hand, since the replication insensitive benchmarks induce few replicated cache requests, the performance benefits offered by COLAB becomes minimal. Nevertheless, it can still be observed that the NoC read traffic and the IPC of the replication insensitive benchmarks are not negatively impacted by the employment of the proposed COLAB architecture.

### 3.3 Energy Evaluation

In this experiment, we analyze how the incorporation of COLAB impacts the energy consumption of different components in a GPU. As demonstrated in Section 3.2, the incorporation of COLAB can significantly reduce the NoC traffic by capturing replicated cache requests. Owing to the reduced NoC traffic, the energy consumed by the NoC network can also be lowered. As presented in Fig. 9, the energy consumption of the NoC interconnection is reduced by an average of 25%. Furthermore, by servicing replicated requests within SM clusters instead of resorting to the L2 cache, COLAB is also able to decrease the number of accesses to the L2 cache, and offers benefits in reducing the energy consumption. On average, a reduction of 21% in energy consumed by the L2 cache is observed across the replication sensitive benchmarks. Our experimental results suggest that energy consumed by the remainder of the GPU (i.e., including COLAB the other components in the GPU) is reduced by 5%. After accounting for the aforementioned energy benefits contributed by different components in the GPU, it is observed that the overall energy consumption of the GPU has reduced by 12% across the replication sensitive benchmarks by employing COLAB. On the other hand, it can be observed from Fig. 9 that on average, COLAB offers a slight overall energy benefit of 2% over the replication insensitive benchmarks for the entire GPU, as neither the NoC read traffic nor the stall cycles in SMs are significantly improved.

### 3.2 NoC Traffic and Execution Throughput

Figs. 6 and 7 show the the normalized NoC read traffic and the normalized execution throughput in terms of instructions per cycle (IPC) achieved by incorporating COLAB. It is observed from Fig. 6 that for the replication sensitive benchmarks, COLAB is able to reduce an average of 38% of the NoC read traffic by servicing replicated cache requests within a cluster. Owing to the reduced NoC read traffic load, the cycles of stalls in SMs across the entire GPU are also alleviated. This is supported by Fig. 8, which shows that the adoption of COLAB indeed reduces the number of stalled cycles of the GPU by an average of 40% for replication sensitive benchmarks. The reduction in stalls cycles in turn enhances the utilization of the SMs, and boosts the overall throughput of the GPU. The improvement in execution throughput is illustrated in Fig. 7, which shows that the employment of COLAB is able to improve the IPC of the baseline GPU by an average of 43%. However, it is noticed that the NoC traffic and IPC improvements of cfd and gaussian are rather limited as compared to the other replication sensitive benchmarks. This is mainly due to their relatively lower replication rates, as indicated in Fig. 2. On the other hand, since the replication insensitive benchmarks induce few replicated cache requests, the performance benefits offered by COLAB becomes minimal. Nevertheless, it can still be observed that the NoC read traffic and the IPC of the replication insensitive benchmarks are not negatively impacted by the employment of the proposed COLAB architecture.
3.4 Ablation Studies

We next present a set of studies to validate the designs of COLAB.

3.4.1 Analysis on the Arbitration Policy. In this experiment, we investigate how different arbitration policies affect COLAB’s ability of capturing replicated cache requests, and demonstrate that the proposed arbitration policy described in Section 2.2.3 delivers the maximum performance. Fig. 10 plots the replicated cache request capture rates (referred to as the capture rate hereafter), which is defined as the ratio of the number of replicated requests captured by COLAB to the total number of replicated cache requests, across the replication sensitive benchmarks under different arbitration policies. In this experiment, Ours corresponds to the proposed arbitration policy that always prioritizes COLAB requests. Meanwhile, AP1/2, AP1/4, and AP1/8 represent the arbitration policies that grant priority to COLAB requests once every 2, 4, and 8 cycles, respectively. Lastly, Local is a policy that always gives local accesses priority over those from COLAB. It is observed that the capture rates of COLAB are degraded if the arbitration policies favor local requests. The reasons behind the degradation in capture rates are twofold. First, as requests from COLAB become less prioritized, the per-SM access queues and the COLAB input queue are more likely to become full due to the increase in queuing time. This in turn leads to excessive queuing time, causing a large number of requests to be dropped, as described in Section 3.4.1. In short, by employing a properly sized SM cluster, we are able to maximize COLAB’s ability to enhance the overall system throughput.

4 RELATED WORK

In order to eliminate duplicated cache requests to reduce the burden of NoC, many previous endeavors have been dedicated to optimizing or modifying the L1 caches in GPUs. An array of works have proposed cache management schemes [2, 13, 15, 18], warp schedulers [9, 19, 20], or thread block scheduler [11], to optimize L1 caches’ ability of capturing intra-SM data locality. By maximizing cache line reuse in an SM, these works were able to avoid cache contention and reduce the number of repeated requests sent to fetch the same cache line. Meanwhile, other researches discovered that data locality exists beyond a single SM, and a significant amount of NoC traffic are dedicated to fetching cache lines that have been cached by other SMs in a GPU. A branch of these works proposed to capture this inter-SM locality by consolidating all or part of the private L1 caches into a shared L1 cache accessible by multiple or all of the SMs in a GPU [3, 6, 7, 22]. With the help of shared caches, these methods were able to capture requests across SMs and prevent caching replicated lines, boosting the overall cache efficiency. However, the consolidation of private L1 caches could also lead to a reduction in peak L1 bandwidth [7, 22], introducing a new bottleneck to the memory hierarchy. On the other hand, another line of works proposed to utilize such a locality by employing inter-SM communication mechanisms. These works attempted...
to reduce the NoC pressure by servicing replicated cache requests between SMs via a ring interconnection [4], or a parallel cache probing scheme [8]. Nonetheless, these methods require cache requests to probe multiple L1 caches before locating the desired cache line or resorting to the L2 cache, introducing extensive latency to L1 cache misses. As the number of SMs continues to grow in modern GPUs, such a latency overhead can be intensified, and may hinder their performance. The authors in [21] proposed to enable inter-SM communications across the NoC by employing a directory-like structure to reduce off-chip memory traffic. However, as pointed out in [4], exploiting inter-SM locality does not necessarily lead to a reduction in off-chip memory traffic in modern GPUs. Furthermore, the authors did not model the NoC traffic in detail and failed to account for the additional NoC traffic incurred by the inter-SM communications that may exacerbate the congestion of the NoC network. In our work, we implement an efficient scheme that makes use of cache line ownership information stored in COLAB to redirect and service replicated cache requests without the need of extensive searching. Moreover, by limiting inter-SM communications to be within a cluster, we are able to address the NoC congestion problem without introducing additional NoC traffic between clusters and L2, improving the overall throughput (i.e., IPC) of the baseline GPU.

5 CONCLUSION

This paper proposed COLAB to reduce replicated cache requests within an SM cluster without resorting to the L2 cache via the NoC interconnection. COLAB records the ownership of the cache lines which are already cached in an SM cluster, and redirects replicated cache requests to their corresponding owners accordingly. Our experimental results show that on average, employing COLAB can reduce 38% of the NoC read traffic, and improve the throughput in terms of IPC by 43%. In addition, our results also indicated that COLAB can assist in energy reduction by 12% due to the decreased NoC read traffic and the reduced number of L2 cache accesses. Furthermore, COLAB induces a negligible hardware overhead of 2% as compared to the total L1 caches within an SM cluster. Based on our analyses and experimental evidence, COLAB is indeed an efficient and cost effective approach for enhancing the GPU performance.

REFERENCES