GPU Performance Optimization via Inter-group Cache Cooperation

Guosheng Wang, Yajuan Du, Weiming Huang

***Abstract*—Modern GPUs have integrated multi-level cache hierarchy to provide high bandwidth and mitigate the memory wall problem. However, the benefit of on-chip cache is far from achieving optimal performance. In this paper, we investigate existing cache architecture and find that the cache utilization is imbalanced and there exists serious data duplication among L1 cache groups.**

In order to exploit the duplicate data, we propose an inter-group cache cooperation method (ICC) to establish the cooperation across L1 cache groups. According the the cooperation scope, we design two schemes of the adjacent cache cooperation (ICC-AGC) and the multiple cache cooperation (ICC-MGC). In ICC-AGC, we design an adjacent cooperative directory table to realize the perception of duplicate data and integrate a lightweight network for communication. In ICC-MGC, a ring bi-directional network is designed to realize the connection among multiple groups. And we present a two-way sending mechanism and a dynamic sending mechanism to balance the overhead and efficiency involved in request probing and sending.

Evaluation results show that the proposed two ICC methods can reduce the average traffic to L2 cache by 10% and 20%, respectively and improve overall GPU performance by 19% and 49% on average, respectively, compared with the existing work.

***Index Terms*—GPU, Cache, Cooperation, Hit Ratio**

I. Introduction

NOWADAYS , GPUs have been widely used in various fields such as deep learning, data mining, and graphic processing [1]–[5]. The powerful parallel processing capabilities promote continuous GPU architecture innovations, aiming at higher performance [6]–[9]. GPU have integrated configurable multi-level on-chip cache hierarchy to provide high bandwidth and mitigate the memory wall problem [10], [11]. Each GPU streaming multiprocessor (SM) in NVIDIA has a private L1 cache and access the shared L2 cache through a Network-on-Chip (NoC) [12], [13]. L1 cache plays an important role in reducing memory bandwidth and latency [14], [15]. Many cache management schemes have been developed to make full use of finite cache capacity [16]–[22]. However, its utilization is still far from satisfactory.

Many existing works focused on cache sharing to improve GPU performance. In order to benefit from the duplication among L1 caches, Dublish et al. [23] design a cooperative caching network among L1 caches to leverage remote bandwidth. Ibrahim et al. [24] design a data sharing prediction and parallel probing mechanism to optimize inter-core communication. Besides to exploiting locality, some works focus on reducing or eliminating data replication among caches. Ibrahim et al. [25] introduce a shared L1 cache organization, in which each core only cache a non-overlapping slice of the entire address range. Meanwhile, they propose a Clustered Shared Decoupled L1 cache (CSD-L1), which decouples the L1 caches from cores and aggregates them into groups [26].Thus, data duplication is eliminated within each group and the cache capacity is increased. Xu et al. [27] observe that there exists high resource contentions in CSD-L1, and propose ATA-Cache to decouple and aggregate the tag arrays of multiple L1 caches and co-design a two-level thread-block scheduling scheme to maximize locality.

In order to optimize GPU performance, this paper first conducts a series of preliminary studies on existing CSD-L1 architecture. First, we collect the results of L1 and L2 miss rate, which shows that the L1 cache is vastly under-utilized and there exists severe bandwidth pressure on the shared L2 cache. Second, we quantify the distribution of duplicate cache lines among L1 cache groups, which shows that there exists serious data duplication.

By exploiting the duplication among L1 cache groups, we propose an inter-group cache cooperation method (ICC). In ICC, two schemes of ICC-AGC and ICC-MGC are designed to realize the cooperation in different scopes of cache groups. In ICC-AGC, the adjacent cache groups cooperation is implemented by integrating an adjacent cooperative directory table (ACD-Table) and a lightweight interconnection network. In ICC-MGC, a ring bidirectional interconnection network is utilized to achieve multiple group cooperation. Besides, we design the two-way sending mechanism and the dynamic sending mechanism to balance the probing overhead and efficiency in ICC-MGC.

The proposed ICC method is evaluated in GPGPUSim-3.2 using workloads from the *Rodinia*, *Polybench* and *Mars* benchmark suites. Experimental results show that both of ICC-AGC and ICC-MGC can increase cache hit rate and improve GPU performance, significantly. The main contributions of this paper are summarized as follows.

- We conduct a series of experimental preliminary studies to analyze the cache utilization and L1 cache duplication characteristic.
- We propose ICC to exploit the duplication characteristic and we first design ICC-AGC to leverage adjacent cache groups duplication. In ICC-AGC, an ACD-Table is designed to perceive the duplication and a lightweight connection network is integrated for communication.

Yajuan Du is with the School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China, and also with the Shenzhen Research Institute, Wuhan University of Technology, Shenzhen 518000, China. Yajuan Du is the corresponding author (e-mail: dyj@whut.edu.cn).

Guosheng Wang and Weiming Huang are with the School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China. Email: wgsheng@whut.edu.cn, hwm0726@whut.edu.cn

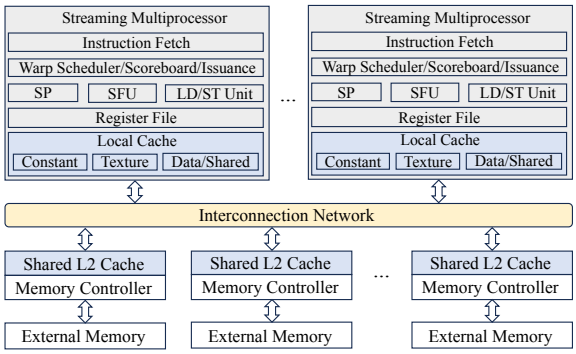


Fig. 1. GPU architecture.

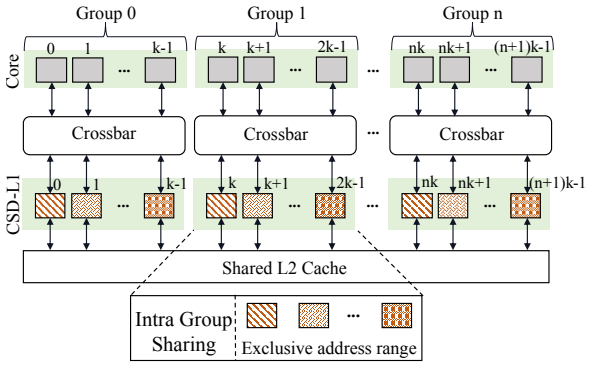


Fig. 2. The architecture of CSD-L1.

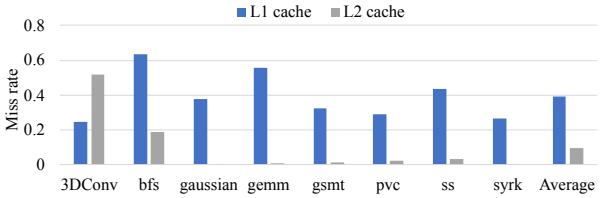


Fig. 3. L1 and L2 cache miss rate in existing CSD-L1 architecture.

- We further design ICC-MGC to exploit the duplicate data among multiple cache groups. In ICC-MGC, we utilize a ring bidirectional interconnection network and design the two-way sending mechanism and the dynamic sending mechanism to balance the probing latency and efficiency.
- We conduct comprehensive experiments to evaluate the proposed ICC method in GPGPU-Sim and experimental results show that ICC-AGC and ICC-MGC can speed up the overall GPU performance by 19% and 49% on average, respectively, compared with the existing work.

The rest of this paper is organized as follows. In Section II, we review the background of existing GPU cache hierarchy and the architecture of existing CSD-L1, and we illustrate the preliminary study and the results of cache hit rates and data duplication, which motivates the design of the cooperation scheme. Section III presents the details of the proposed ICC method and Section IV demonstrates the experimental setup and evaluation results. Section V presents the related works and Section VI concludes this paper.

II. Background And Motivation

In this section, we first review the background of GPU architecture and then illustrates the existing cache architecture of CSD-L1. Finally, we demonstrate the preliminary study details which motivated our work.

A. GPU Architecture and Cache Hierarchy

The NVIDIA GPU architecture is composed of Streaming Multiprocessors (SMs), as shown in Fig. 1. All SMs are connected to the shared L2 cache via the on-chip interconnection network. Each SM contains many CUDA cores (i.e., Streaming Processors or SPs), which are the functional units and can execute one instruction per cycle. All SPs within a SM share an instruction fetch/decode unit, a L1 cache and a large register file that can be scheduled flexibly. Each thread in a SM can be allocated a specific number of registers depending on the number of executed threads.

GPU has been integrated conventional multi-level cache hierarchy to alleviate memory wall problem in the Fermi architecture [6]. The local caches within a SM consist of constant cache, texture cache, shared cache and data cache. All SMs share the banked L2 cache through a NoC, which have a higher latency compared to local caches. The L2 cache is divided into multiple storage partitions, also called cache banks, and communicate with DRAM through the memory controller. The private L1 caches and common L2 cache are responsible for reducing traffic to the interconnection network.

B. Clustered Shared Decoupled L1 Cache

To efficiently utilize the valuable L1 caches, Ibrahim et al. [26] propose CSD-L1, details of which is shown in Fig. 2. In CSD-L1, the L1 caches are decoupled from SMs and aggregated into multiple independent groups, i.e., there is no communication among these cache groups. The decoupled L1 caches are shared and accessed by the SMs through a NoC. Therefore, the grouped L1 caches can share a wider address range and sufficiently cache more blocks by eliminating duplicate data blocks among previous private L1 caches. Specifically, for a coalesced L1 data cache read request generated by an SM, the request is routed to the responding L1 cache through the NoC according to the missed address.

C. Preliminary Study

In order to study GPU efficiency under existing cache architecture of CSD-L1, we perform a preliminary study and run eight workloads using GPGPU-Sim to study the cache behaviors and characteristics. We mainly collect statistics about cache miss rate, L1 cache utilization and the proportion and distribution of duplicate data among cache groups.

1) Cache Miss Rate.: Fig. 3 depicts the miss rate of L1 and L2 cache in CSD-L1. The x-axis represents the workload, and the last term is the mean of all workloads. We can obtain two critical observations.

First, the L1 miss rate is high for some workloads, for instance, the miss rate reaches as high as 64% for *bfs* which

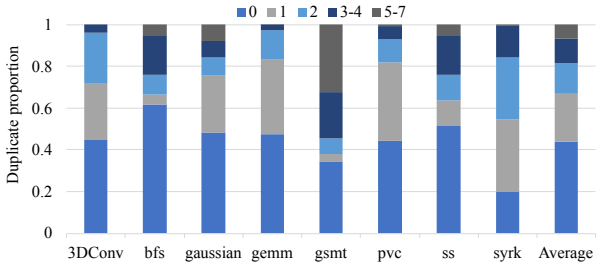


Fig. 4. Proportion of duplicate data among L1 cache groups.

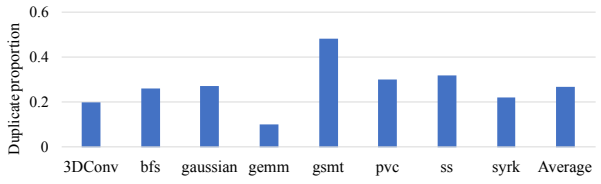


Fig. 5. The adjacent groups duplicate data proportion.

with poor temporal locality. The average L1 miss rate is 39% for all workloads, which shows that CSD-L1 cannot make full use of L1 cache optimally.

Second, the average L2 cache miss rate for all workloads reaches 9.8% which is much lower than that of L1 cache. We observe that the L2 miss rate for many workloads is less than 4% , and for *gaussian* and *syrk*, the L2 miss rate is even less than 1%. Combined with the large capacity and shared characteristics of L2 cache, we speculate that there may be many common cache blocks among L1 caches.

2) Duplicate Data Characteristics: In order to uncover the characteristics of duplicate data among L1 cache groups, we collect the duplication proportion and distribution of data between adjacent caches and all caches.

Duplicate data proportion. We calculate the proportion of duplicate cache lines among L1 cache groups to analyze the sharing behavior. The results shown in Fig. 4 depict the duplicate proportion of cache lines with different levels. The x-axis represents the individual workloads and the y-axis represents the duplicate proportion of multiple series. There exist five possible intervals for y-axis: ”0”, ”1”, ”2”, ”3-4” and ”5-7”. We merge the adjacent ranges with relatively small overall proportions, such as ”3-4” represents the cache lines with 3 or 4 duplicates in remote grouped L1 cache. Series ”0” indicates that current missed L1 cache line is not duplicated in remote L1 cache groups.

Fig. 4 demonstrates that workloads other than *bfs* have more than 50% duplicate cache lines and the average duplication is 56%. The proportions of the four duplication intervals are 22%, 14%, 12%, and 6%, respectively, which indicate that there are a lot of duplicate cache lines among L1 cache groups.

Duplication between adjacent cache groups. We quantify the proportion of duplicate data between adjacent groups, by mainly calculating the requests that can be serviced by the left and right cache groups. The proportion results are shown in Fig. 5, and the x-axis represents the workload and the last item is the average. Specifically, *gsmt* has the highest duplication between adjacent groups, with a ratio of 48%, indicating that nearly half of the read miss requests in current cache group can be serviced in the adjacent cache groups. The lowest duplication proportion is 10% for *gemm*. The others workloads exhibit similar duplication proportion, and the duplication ratio of all workloads is 27%. Analyzing the distribution of duplicate cache blocks in a cache group can provide novel insights for us to optimize CSD-L1 and explain the performance changes of different workloads.

Duplication among all cache groups. We further study the distribution of the duplicate data with detailed experiments to break down and quantify the distribution of duplicate data among L1 cache groups. According to the remote group duplication proportion and distribution of the eight workloads, we summarize three duplication patterns of weak duplication, moderate duplication and strong duplication and demonstrate four typical workloads detail in Fig. 6. For each sub-graph, the x-axis represents the groups that incur an L1 cache read miss request, and y-axis indicates the remote L1 cache groups. At the (x, y) coordinate, the darker the color in the square, the higher proportion of group x and y are duplicated. From Fig. 6, we make the following observations.

For *gsmt* and *3DConv*, they exhibit weak duplication pattern as shown in Fig. 6(a). The duplication among L1 cache groups is poor or even non-existent duplication for some groups.

Fig. 6(b) demonstrate moderate duplication pattern corresponding to *syrt*, due to the high duplication among even-numbered groups, the overall duplication is much higher than weak duplication pattern. *gsmt* and *pvc* both present moderate duplication pattern as shown in Fig. 6(c). Different from the pattern details in Fig. 6(b), the cache groups that close to the current cache group show a relatively high duplication ratio in *gsmt* and *pvc*, and the color becomes lighter as the group distance increases, indicating that the duplication ratio is getting lower. The duplication distribution is relatively uniform and high among the groups for *bfs*, *gaussion* and *ss*, and they all demonstrate strong duplication pattern as shown in Fig. 6.

In summary, we can obtain two common observations. First, the utilization of on-chip bandwidth is imbalanced. The miss rate of L1 cache groups is high while the miss rate of L2 cache is poor. Second, there exists many duplicate cache lines among L1 cache groups.

The duplication characteristics inspire us to explore and design an efficient inter-group cache cooperation schemes to optimize GPU performance. There are two challenges. The first is how to perceive the valid duplication among L1 cache groups. And the second is how to design the interconnection network to provide low overhead and latency.

By exploiting above observations, we optimize the CSD-L1 architecture to leverage duplicate data among L1 cache groups and propose inter-group cache cooperation (ICC) methods to improve GPU performance. Considering the high complexity and overhead of cache groups connection and the duplication ratio between adjacent groups, we firstly propose the Adjacent Group Cooperation, named ICC-AGC (details in Section III-B). Since the duplication characteristics of each workloads are different, we expand the cooperation scope to fully utilize the bandwidth of L1 cache and duplication in more cache groups, we further propose Multiple Group Cooperation,

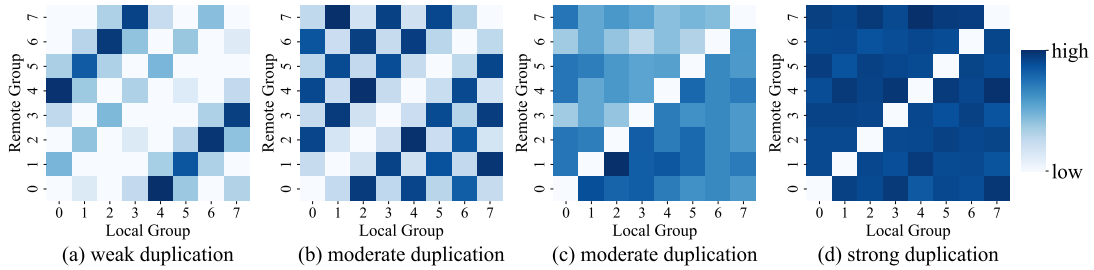


Fig. 6. Distribution of duplicate data among L1 cache groups.

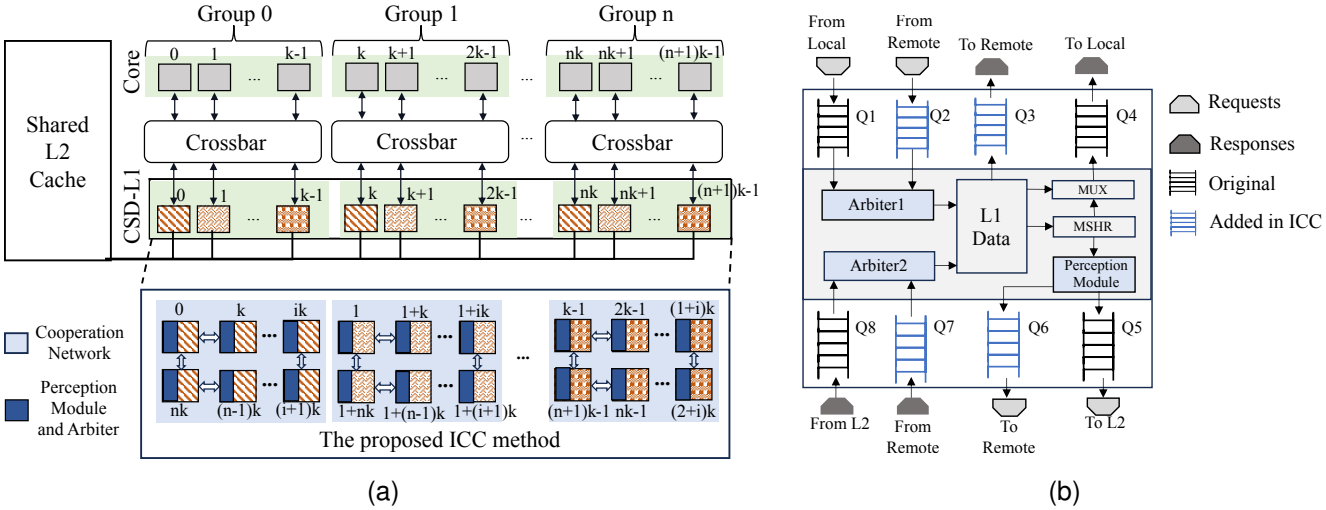


Fig. 7. The overview of the proposed ICC method, the key components of which are denoted in blue. (a) The architectural overview of ICC. (b) The L1 cache node in ICC.

named ICC-MGC (details in Section III-C).

III. The Proposed ICC Method

In this section, we first describe the overview architecture of our proposed ICC method. Next, we demonstrate the design and implementation details of ICC-AGC and ICC-MGC, respectively. Then we show the workflow of ICC-AGC and ICC-MGC. Finally, we analyze the hardware overhead and latency of our proposed ICC schemes.

A. Overview of ICC Architecture

Fig. 7a shows the whole architecture of ICC, which mainly contains three components: the cooperation interconnection network, the perception module and the arbiters. The network enables the data transfer among cooperative L1 cache groups. The perception module is used to probe states of the current L1 cache while the two arbiters are responsible to decide how to implement inter-group cooperation in detailed situations. Besides, four buffer queues are integrated in each L1 cache to temporarily store requests or responses from the interconnection network, as shown in Fig. 7b. With these components, ICC can effectively leverage duplicate data among L1 cache groups to improve GPU performance.

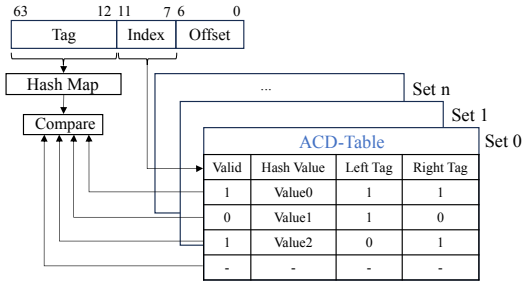


Fig. 8. The design of ACD-Table.

B. The Design of ICC-AGC

ICC-AGC mainly contains three components of an Adjacent-group Cooperation Directory Table (ACD-Table) as the perception module, the adjacent cooperation interconnection network (AdjNet) to be responsible for the adjacent L1 cache groups communication, and priority arbiters to determine the priority of requests responses.

1) ACD-Table: As shown in Fig. 8, ACD-Table is organized as a multi-way set-associative structure. Each entry in ACD-Table records the information of a cache block from an adjacent L1 cache group. The entry is composed of a hashed tag and a bitmap. We hash the 52 bits virtual page tags to an 20-bit tag for a 64-bit virtual address using 4KB page. The

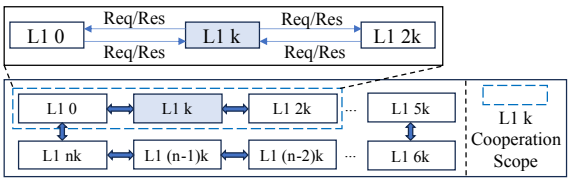


Fig. 9. The design of AdjNet.

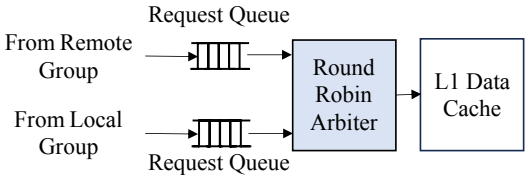


Fig. 10. The design of priority arbiter.

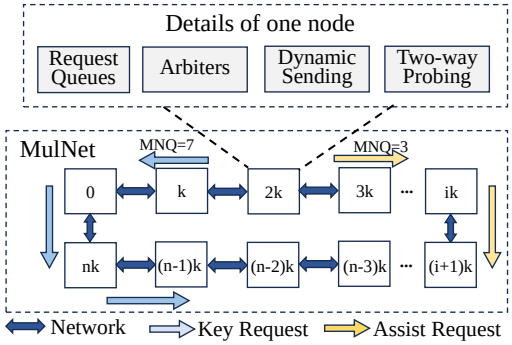


Fig. 11. An example for MulNet.

20-bit hash map is enough to eliminate hash conflict and save space efficiently. The bitmap contains 3 bits, corresponding to the valid tag of local, left and right L1 cache group. Left adjacency cache group is set to have a higher priority and the first L1 cache's left L1 cache is the last L1 cache of the same cluster. The valid tag of '1' indicates that the corresponding cache line is stored. To balance the trade-off of searching overhead and hardware requirements, we set the same number of sets and ways as L1 cache. The update of ACD-Table is done by its adjacent cache groups. When a cache block in the adjacent cache groups is modified or inserted, its tag information in ACD-Table is simultaneously updated .

2) AdjNet: Fig. 9 shows the structure of the interconnection network. The overall structure of AdjNet is similar with a bi-directional ring, but the scope is limited to the adjacent L1 cache groups. For example, the adjacent cache groups of group *1* is group *0* and group *2*, and the cache L1 *k* can communicate with cache L1 *0* and cache L1 *2k* through AdjNet. ICC-AGC can benefit from AdjNet in the two following reasons. On the one hand, adjacent cooperation does not require much bandwidth from the interconnected network, which controls implementation overhead. On the other hand, L1 caches only need to process requests from adjacent cache groups, which controls the request traffic to the interconnection network, thus ensuring efficient data transmission. In the following article, we define a request that is sent to remote groups through cooperation network as a cooperative request.

3) Priority Arbiter: In ICC, we commonly integrate 4 buffer queues in the L1 cache to complete request processing (details in Fig. 7). The request queues, Q2 and Q6, are responsible for buffering cache access requests from remote groups and to remote groups, respectively. The response queues, Q3 and Q7, are used to buffer the replies from interconnection network. Specifically, the element in Q3 would be sent to the remote group, and that in Q7 would be received from the remote group.

Due to the added buffer queues for requests and responses, it is necessary to integrate two arbiters to determine the source of processing when L1 cache reads and writes data. Specifically, the L1 cache needs to handle the requests from the local and remote groups in an orderly manner. Since the L1 cache can only process one cache request in one cycle, it is necessary to integrate a priority arbiter to arbitrate the two types of requests, and the details of priority arbiter are shown in Fig. 10. In order to balance fairness and bandwidth allocation, the arbiter fetch requests from Q1 and Q2 in a round-robin manner. Similarly, a response priority arbiter is also integrated to process data from the L2 cache and remote groups in an orderly manner.

C. The Design of ICC-MGC

In order to benefit from more duplicate data, ICC-MGC is designed to increase the cooperation scope among L1 caches. In ICC-MGC, a Multiple Group Cooperation Network (MulNet) is developed to be responsible for the multi-group cache communication. Meanwhile, we design the two-way sending mechanism and the dynamic sending mechanism as the perception module to balance the trade-off of probing overhead and efficiency.

1) MulNet: Considering that each L1 cache is directly connected to the other L1 caches, the interconnection network would be extremely complex, and also incur expensive hardware overhead. In addition, it is necessary to design a flexible and scalable network to handle a large number of cooperative requests and responses. A bidirectional ring network topology with channel queues is suitable for the connection of multiple cache groups with consecutive physical location. Specifically, each L1 cache directly connect to adjacent groups, and the remote L1 caches can communicate with each other through the forwarding of adjacent cache groups. Each L1 cache integrates 2 ports for communication and 4 queues to buffer the cooperative data. Two queues with 5-entry are designed to house the cooperative requests and responses from remote groups, and the other two queues are responsible for buffer the requests that missed in local cache, and responses to remote groups. Furthermore, the number of cooperation groups is limited to 8 which can effectively mitigate network congestion and resource contention. Due to the independent transmission of cooperative requests and responses, it is enough to set 5-entry for each queue.

2) The Two-way Sending Mechanism: In ICC-MGC, as the groups cooperation scope has expanded, the limited capacity table structure of ACD-Table cannot be used to perceive duplicate cache lines of remote groups. Therefore, the two-

way sending mechanism was proposed to achieve duplication probing and perception.

Specifically, taking cache L1 *k* as an example, when a missed request was send to MulNet, the MulNet generate two cooperative requests, and forward requests in two directions, respectively. Two cooperative requests are set with a maximum number of queries (MNQ), which is used to limit the request probing range.

Assumes that the request in the counterclockwise direction is R_k, and the request in the clockwise direction is R_a. R_k is used to quickly return probing results, so its MNQ value is set to 2, that is, up to 2 remote caches can be queried. R_a traverses all cache groups in the entire MulNet, so its MNQ value is set to 7 (there are 8 cache groups).

If the requests hit the L1 cache of the remote group, the cooperative reply is immediately returned to the cache L1 *k* of the source group of the request. Otherwise, the MNQ value is decremented by 1. If MNQ is greater than 0, the cooperative requests is forwarded to the next remote cache group along the original direction until the request hits or the MNQ reaches 0.

When a cooperative request returns, in order to reduce the return time, we first calculate the distance among the remote cache group and the source cache group in the two return directions, and then choose the direction with the smaller distance to return. In most cases, L1 *k* receives two cooperative replies corresponding to R_k and R_a respectively. If both requests hit the remote cache, L1 *k* adopts the reply that arrives first in time and discards the reply that arrives later. If R_k misses within the MNQ range, L1 *k* would receive a cooperative reply carrying miss information, and then L1 *k* sends the read miss request to L2 to prevent R_a from also missing. Normally, the reply corresponding to R_k reaches L1 *k* earlier. Thus cache L1 *k* would receive the valuable preprocessing information.

3) The Dynamic Sending Mechanism: In ICC-MGC, the duplication ratio of remote cache groups cannot be predicated in advance. Therefore, if requests have poor duplication rate in remote cache groups, blindly sending missed requests to MulNet would hurt overall performance. On the one hand, a large number of requests miss waste valuable L1 cache bandwidth. Meanwhile, local cache read miss request that always missed in remote cache still need to go to L2 cache, resulting in a lot of meaningless additional overhead. Therefore, the dynamic sending mechanism is proposed to arbitrate whether to use ICC schemes based on the program execution status.

In ICC-MGC, the average data access time is monitored in real time to measure the efficiency of inter-group cache cooperation. We denote the average access time of the requests that send to the MulNet as T_{MGC}, and the average access time of the requests that directly go to L2 as T_{L2}. We calculate T_{MGC} by Formula 1:

$$T_{MGC} = R_h * T_h + R_m * (T_m + T_{m_{L2}}) \quad (1)$$

In Formula 1, R_h and R_m represent the hit rate and the miss rate respectively, and the subscript represents which structure they belong to, ICC-MGC or L2. T_h and T_m represent the average access latency on hits and misses respectively, and the subscript represents which structure they belong to, ICC-MGC or L2.

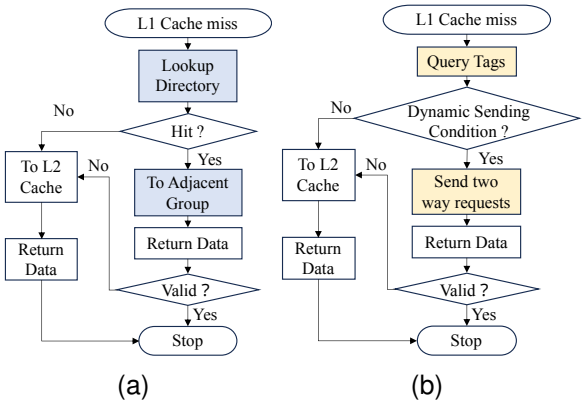


Fig. 12. The workflow of ICC schemes. (a) The workflow of ICC-AGC. (b) The workflow of ICC-MGC.

The dynamic sending mechanism is executed in a fixed period, and each period is divided into a sampling phase and an execution phase. The basic unit of the mechanism is the number of requests per period. The number of requests in the sampling phase is 256, and the number of requests in the execution phase is 4096. During the sampling phase of each period, a L1 cache read miss request would be sent to MulNet to access remote groups, and the perception module act as an arbiter to determine requests processing method in the next execution stage according to the response time. Specifically, if $T_{MGC} < T_{L2}$, it means that cache lines can be obtained efficiently through MulNet in next execution phase, and the L1 cache read miss requests would be sent to MulNet. Otherwise the requests would directly go to L2 cache.

D. The Workflow of ICC

In this part, we present the workflow details of ICC-AGC and ICC-MGC in Fig. 12. Next, we take cache L1 *k* in cache group 1 as an example to illustrate the workflow details in ICC-AGC and ICC-MGC, respectively.

1) The Workflow of ICC-AGC: Fig. 12(a) depicts the workflow in ICC-AGC. Assuming a read request, R_e, was fetched from Q1 or Q2 and missed in local L1 cache. Then R_e would be recorded in MSHR (Miss Status Holding Registers) which is responsible for recording and merging requests that L1 cache missed. In the baseline architecture, R_e would be buffered in Q5 then directly go to L2 cache. However, in AGC-L1, R_e would firstly check the ACD-Table in L1 *k* to determine next process. On a miss in ACD-Table, R_e would be forwarded to L2 cache as usual. On a hit in ACD-Table, it indicates that the missed cache block exists in the adjacent cache group on the left or right. In this case, R_e would be buffered in Q6 and wait to be sent to the corresponding L1 cache group according to the bitmap of the entry through AdjNet.

Suppose that the bit map is '001', this indicates that the L1 *2k* which is located in the right of L1 *k* can service R_e or has stored the requested cache block. R_e would buffered in Q1, which is located inside L1 *2k*. At this point, R_e needs to wait

until it is selected by the Arbiter1 in L1 *2k*, after which L1 *2k* would push the cache data corresponding to R_e into Q3. The cache block is passed back over the network and buffered at Q7 which is located in L1 *k*. In summary, a complete adjacency cache access process is realized. After that, check the validity of the cache block, as some data is invalid because it has been modified or evicted. If it is valid, wait for the response arbiter to be selected and written to the L1 *k*. Otherwise, the cache block would be discarded and R_e be pushed to Q5 waiting to go to L2 cache.

2) The Workflow of ICC-MGC: Fig. 12(b) depicts the processing workflow of cache requests in ICC-MGC. Different from AGC-L1, for an L1 cache miss request, R_e, L1 cache firstly checks the dynamic request flag to decide whether to send R_e to MulNet. The initial stage is set as the sampling stage, and the dynamic request flag is true. In other stages, the dynamic request flag is modified based on the real-time monitoring results. If the cooperation flag is false, it means that the latency in ICC-MGC is high and R_e should be sent to L2 directly. Otherwise, R_e would be sent to MulNet.

Assumes that L1 *3k* received R_e from MulNet, R_e would be pushed in Q2 and the MNQ value R_e decrease 1. The Arbiter1 in L1 *3k* fetch requests from Q1 and Q2 in a round robin. On R_e hit in cache *3k*, the response data block would be pushed in Q3, and send the reply to L1 *k*. Otherwise, R_e would be directly pushed into Q6 which would send forward through MulNet. In a case of MNQ is 0, a miss reply would be returned to the original L1 *k*. Similar to ICC-AGC, when receiving a data response from mote group, the L1 *k* also checks the the validity information to determine whether adopt the response.

E. Overhead Analysis

This section analyzes the overhead on hardware requirements and time delays involved in ICC.

1) The Overhead of ICC-AGC: The hardware requirement is mainly from the AdjNet and the ACD-Table. Two communication ports are integrated for each L1 cache to be responsible for receiving and sending cooperative data, respectively. There are 2 requests queues and 2 responses queues both with 5-entry to buffer the cooperative data from adjacent groups. The wide of each entry in request queues is 8 Bytes, while the wide of each entry in response queues is 128 Bytes. The total overhead of 4 buffer queues is 1288 Bytes.

The structure configuration of ACD-Table is the same as that of L1 cache, with 4-way and 32-set. Each entry requires 20 bits to store the hashed tag, and 3 bits to record the validity of two adjacent L1 cache and itself. Therefore, the hardware requirement by each ACD-Table is 368 Bytes. In summary, the total increased hardware overhead for each L1 cache is 1656 Bytes.

The time overhead is mainly increased by the two processes of ACD-Table lookup and L1 cache group communication. For the first issue, there is additional one cycle for the L1 cache read miss request that also missed in ACD-Table. Also, it will take L1 cache one cycle to service cooperative requests from adjacent groups. But on a hit of ACD-Table will speed up data access compared with directly go to L2, which will mitigate overall latency. The entries in ACD-Table are updated and inserted by the adjacent L1 cache, which is not in the critical path. For the second issue, the cooperative requests can be sent during one cycle and would not occupy extra ports. Thus, there is little time delay for communication.

2) The Overhead of ICC-MGC: For the hardware implementation overhead, we mainly integrate a MulNet and arbiters. Each L1 cache can process and forward the cooperative requests, so each L1 cache is required to be equipped with 2 communication ports, which are responsible for sending and receiving cooperation data (requests or responses, respectively). Similar to ICC-AGC, each L1 cache also requires 2 arbiters and 4 queues. The 4 buffer queues totally bring additional 1288 Bytes hardware overhead. It is worth noting that the dynamic request arbiter does not require additional hardware overhead, and it is enough to modify the relevant processing logic of L1 cache.

In terms of the time delay in ICC-MGC, extra time overhead is cause by the dynamic sending mechanism and remote requests delay. Specifically, during the execution phase, the L1 cache would check the dynamic request tag that calculated during the sampling phase to decide whether the request should be sent to MulNet in current phase, and this process would take one cycle. When the communication port is not occupied, it takes one cycle for the L1 cache to communicate with adjacent cache groups.

IV. Evaluation

In this section, we first demonstrate the experiment environment of GPGPU-Sim, the system configurations and benchmarks. Next, we present and analyze the experimental results. Finally, we show the sensitivity studies.

A. Experimental Setup

GPGPU-Sim [28] is a cycle-level GPU performance simulator and can simulate the performance of GPU architecture accurately. We have extended GPGPU-Sim-3.2 to implement our proposed two ICC schemes. We not only model the interconnection network and cache request mechanism but calculate the execution latency exactly.

The important components parameters of modeled GPU system for the experiment are shown in Table I and other parameters are the default values.

We select a various set of workloads from *Rodinia* [29], *Polybench* [30], *Mars* [31], *Tango* [32] and *SHOC* [33] benchmark suits. The workloads come from multiple fields such as graph algorithms, linear algebra, and data mining, and the source, description, and dataset size of each workload are shown in Table II.

In order to validate the effectiveness of our proposed optimization ICC schemes, we demonstrate the performance of four different architectures including Baseline (private L1 cache), CSD-L1, ICC-AGC and ICC-MGC.

- **Baseline**: In traditional GPU cache hierarchy, L1 cache is private to each SM and independent of each other.
- **CSD-L1** [26]: CSD-L1 is a state-of-the-art GPU cache optimization scheme based on cluster group sharing.

TABLE I
THE SIMULATED GPU SYSTEM CONFIGURATION

Components	Configuration
GPU System	1.4 GHz core clock, 32 cores (SMs), SIMD width = 32
Resources/SM	1536 threads, 32768 registers, 48 warps
L1 Data Cache	16KB, 4-way, 32 sets, 128B per block
L2 Cache	768 KB, 8-way, 64 sets, 128 bytes per block
DRAM	924 MHz memory clock, 16 DRAM-banks, 4 bank-groups/MC

TABLE II
WORKLOADS FROM DIFFERENT BENCHMARKS

Suit	Workloads	Suit	Workloads
Rodinia	bfs	Polybench	2mm
Rodinia	gaussion	Polybench	syrk
Rodinia	backprop (bp)	Polybench	gsmt
Rodinia	needle	Mars	ss
Rodinia	cfd	Mars	pvc
Rodinia	lud	Tango	AlexNet (AN)
Polybench	3DConv	SHOC	triad
Polybench	gemm	SHOC	reduction

- **ICC-AGC**: ICC-AGC is the proposed cache cooperation scheme between adjacent cache groups.
- **ICC-MGC**: ICC-MGC is the proposed cache cooperation scheme among multiple cache groups.

B. Experimental Results

We present the evaluation results of our optimized ICC schemes including the L1 miss rate, the traffic of L2 cache, the overall GPU performance, and the power consumption.

1) Cache Hit Rate: We present the hit rate results of cache read requests from two aspects, one is the local L1 cache hit rate, and the other is the hit rate of remote L1 cache. In ICC-AGC and ICC-MGC, a local L1 cache read miss request is usually sent to the network as a cooperative request to accelerate data access through inter-group cooperation. Similar to the definition of L1 local cache hit rate, we define the ratio of the number of cooperative requests hitting in remote L1 caches to the total number of local L1 cache read miss requests as the remote L1 cache hit rate.

a) Remote cache hit ratio: Fig. 13 shows the hit ratio of remote L1 cache, where x-axis represents the workload, y-axis represents the hit ratio, and the last item on x-axis represents average. For all workloads, the average remote cache hit ratio for ICC-AGC and ICC-MGC are 19% and 32%, respectively. Among them, *AN* demonstrates the highest hit ratio with 50% and 80% in ICC-AGC and ICC-MGC, respectively, indicating that most cooperative requests can be serviced by adjacent cache groups. For most workloads like *syrk, cfd, 3DConv, gauss, gsmt* and *ss* all present higher remote cache hit ratio. Therefore, most cooperative requests in these workloads can obtain the data from remote caches, which can effectively mitigate the L2 cache pressure. For *gsmt*, the data duplication ratio between adjacency groups is the highest (details in Section II-C), so the remote hit ratio in ICC-AGC is higher. But there is a large difference in remote hit ratio

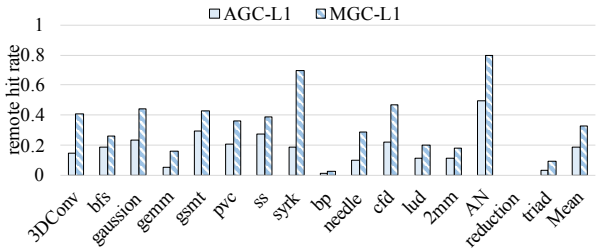


Fig. 13. The hit ratios of remote L1 cache in ICC. As existing methods do not support remote access, there are no compared results.

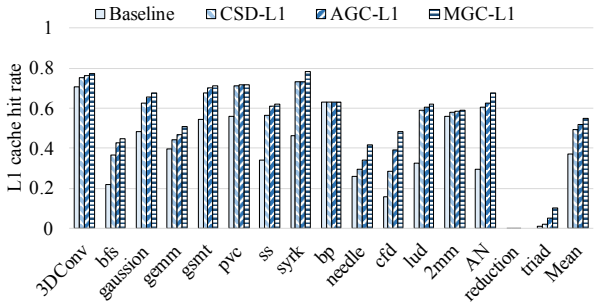


Fig. 14. The hit ratios of local L1 cache.

for *syrk* between ICC-AGC and ICC-MGC, which will affect the overall cache access latency. Both ICC-AGC and ICC-MGC have poor hit ratios in *gemm, bp, lud, 2mm and triad*, thus the overall performance improvement is limited. Also, the remote hit rate in *reduction* is zero, indicating that there is no duplication across different cache groups. Overall, the remote cache hit rate in ICC-MGC is higher than that in ICC-AGC, which translate into much higher improvement in overall performance (details in Section IV-B3).

b) Local cache hit rate: Fig. 14 depicts that ICC-AGC and ICC-MGC can effectively increase the local L1 cache hit rate and the average improvement is 15% and 18% compared with baseline, respectively. For most workloads, both ICC-AGC and ICC-MGC get a certain improvement compared with CSD-L1, and ICC-MGC has a higher improvement.

Specifically, for *syrk* and AN, Fig. 13 shows that the remote cache hit rate of ICC-MGC is closed to 70% and 80%, so ICC-MGC can greatly improve the L1 cache hit rate. Since ICC-AGC only focus on the adjacent groups cooperation, the remote cache hit rate in *syrk* is only 18%, so the overall hit rate of the L1 cache is not significantly improved.

For workloads with a high L1 cache hit rate in CSD-L1, such as *3DConv* , its local cache hit rate is 75%, and the remote cache hit rate in ICC-MGC is 41%. However, since the read miss requests in the above workload account for a small proportion of the total requests, the L1 cache hit rate is only improved by 2%. For workloads with a low L1 cache hit rate in CSD-L1, such as *bfs, cfd and needle*, there are much further local hit ratio increment than CSD-L1. Specifically, for *bfs*, the local cache hit rate is only 36%, indicating that cache miss occurs frequently, even if the hit rate of remote cache in ICC-AGC and ICC-MGC are poor(18% and 25%, respectively), but ICC methods can also significantly improve

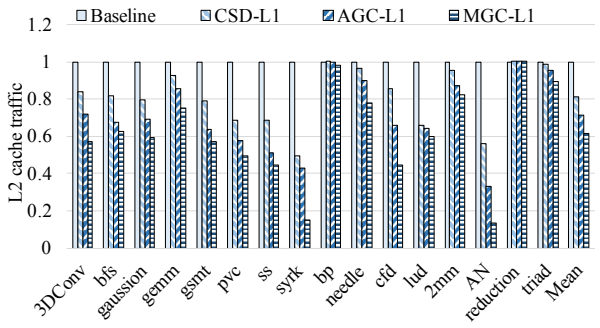


Fig. 15. The normalized results of L2 cache traffic.

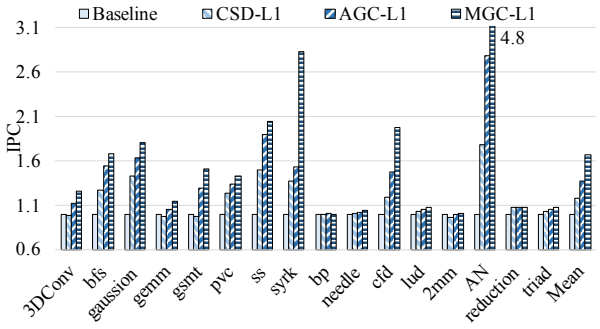


Fig. 16. The normalized results of overall performance compared to baseline.

the L1 cache hit rate. For *reduction*, there is no local cache hit ratio improvement due to the high miss ratio caused by specific cache access pattern.

2) L2 Cache Traffic: We collect statistics on the traffic of L1 cache groups to shared L2 cache. The traffic results are shown in Fig. 15, and all results are normalized based on Baseline. From Fig. 15, we obtain the following observations.

Overall, both ICC-AGC and ICC-MGC can effectively reduce the L2 cache traffic and the traffic is reduced by on average of 10% and 20% compared with CSD-L1, respectively. The L2 traffic reduction shows that both ICC-AGC and ICC-MGC can take advantage of the inter-group cache cooperation and enhance the L1 cache to handle more missing requests, thus reducing the requests sent to L2 cache.

There are considerable traffic reduction for workloads with high remote hit ratio, such as *syrk and AN* in ICC methods compared with CSD-L1, especially in ICC-MGC. Specifically, for *syrk*, the L2 traffic in ICC-MGC is significantly reduced, which is 35% lower than that in CSD-L1. The main reason is that the remote cache hit rate in ICC-MGC is high (nearly 70%), so that most missed requests can be obtained through inter-group cache cooperation. For *gemm*, due to the poor remote cache hit rate in ICC-AGC and ICC-MGC (only 5% and 16% respectively), the L2 traffic is only reduced by 7% and 18% compared to CSD-L1, respectively. However, there is little or non traffic reduction in reduction, because there are no duplication and remote group to service the cooperative requests.

3) Overall Performance (IPC): We use IPC (Instruction per Cycle) to represent the overall GPU performance. Fig. 16

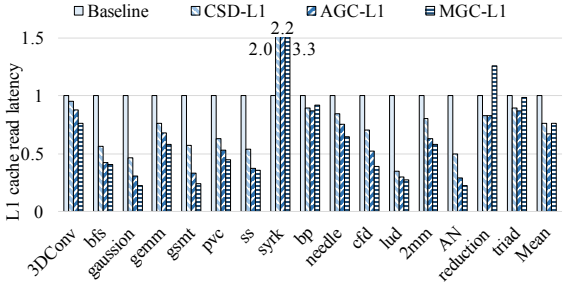


Fig. 17. The normalized results of L1 data cache read latency.

shows the normalized results of IPC based on Baseline. The horizontal axis represents each workload, and the last term is the average performance. Overall, ICC-AGC and ICC-MGC provide a modest improvement in performance by varying degree for each workload compared to Baseline and CSD-L1. Specifically, compared to CSD-L1, ICC-AGC and ICC-MGC can provide an additional 19% and 49% performance improvement on average, respectively. ICC-AGC and ICC-MGC perform differently for each workload and we observe several characteristics as following.

First, for *AN*, ICC-MGC has the greatest performance improvement, the performance is 4.8x more than that of baseline, and 3x more than that of CSD-L1. The hit ratio of remote groups is high (nearly 80% in ICC-MGC and 50% in ICC-AGC), which can significant reduce the traffic of L2 cache and fully utilize remote cache bandwidth. Also, *syrk, cfd and ss* all get significant performance improvement for the large duplication of remote cache groups.

Second, for *gmst*, CSD-L1 demonstrate performance degradation, while ICC-AGC and ICC-MGC can greatly improve the overall performance compared with CSD-L1 (31% and 53%, respectively). The main reason for the difference in *gsmt* is the high proportion of duplication between adjacent L1 cache groups (48%). In addition, there are a large number of identical cache lines among L1 cache groups (the duplication of ”3-4” and ”5-7” account for 22% and 32%, respectively), so most cooperative requests in ICC-MGC can be hit in the closed cache group, resulting in an overall performance improvement of 53% compared to CSD-L1.

Third, both ICC-AGC and ICC-MGC have relatively small performance improvements for *gemm* (8% and 16%, respectively). The low proportion and random distribution characteristics of duplicate data are the main reasons, which also result in ICC-MGC with a larger cooperation scope performing better than ICC-AGC. For *reduction*, there is no performance improvement in ICC-AGC and even little performance degradation in ICC-MGC compared with CSD-L1. On the one hand, the high miss ratio and irregular access pattern significantly influence cache groups cooperation efficiency. With the specific design of perception module in ICC-AGC and ICC-MGC, ICC methods can maintain the overall performance improvement of CSD-L1 for irregular and replication intensive workloads. For other workloads, both ICC-AGC and ICC-MGC deliver good performance improvements, and ICC-MGC outperforms ICC-AGC in many workloads.

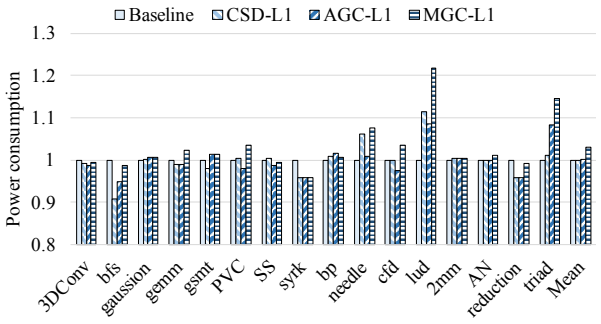


Fig. 18. The normalized results of power consumption.

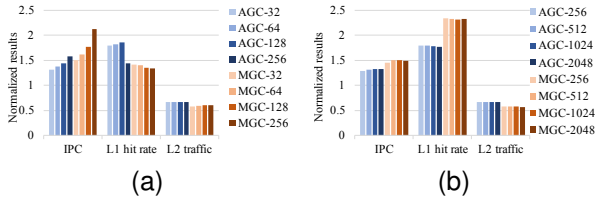


Fig. 19. Effect of GPU core count and L2 cache traffic on IPC, L1 hit rate and L2 traffic, respectively. (a) Effect of GPU core count. (b) Effect of L2 cache capacity.

4) L1 Cache Read Latency: To further analyze the performance details, we collect data on the L1 data cache average read latency, and the results are normalized compared to baseline (shown in Fig. 17). Compared with CSD-L1, ICC-AGC can effectively reduce the L1 cache read latency for most workloads, and bring 10% reduction on average, specifically. While there is little latency reduction in ICC-MGC on average. We observe that workloads with high duplication demonstrate a higher latency reduction in both ICC-AGC and ICC-MGC. Specifically, *gsmt* presents the highest improvement in ICC-AGC and ICC-MGC and can obtain 24% and 33% cache access latency reduction than that in CSD-L1. It is expect that *cfd, gaussian, AN* and *2mm* all demonstrate effectively cache access latency reduction in ICC-AGC and ICC-MGC, because there is a large number of repeatedly cache access and duplication across L1 caches.

Note that there are none latency reduction or even latency increment than CSD-L1 in *reduction* and *syrk*, especially in ICC-MGC. For *reduction*, the most reason is the poor duplication across L1 cache blocks (the remote hit ratio nearly 0) and the high local L1 cache miss rate. For *syrk*, there are great latency increment in ICC-MGC and the increase is nearly 120% than that in CSD-L1, which may caused by the large (nearly 70%) and normalized duplication proportion across L1 caches, shown in Fig. 5. Different with *AN*, the remote hit ratio difference between ICC-AGC and ICC-MGC in *syrk* is much larger than that in *AN*, which will take up more latency to service cooperative requests from remote groups.

5) Power Consumption: We use GPUWattch [34] to estimate the power consumption of our proposed ICC methods. The energy consumption results of the four structures are shown in Fig. 18, and all results are normalized based on Baseline. Compared with Baseline, the energy consumption in ICC-AGC changes very little, but there is 3.1% power consumption increment on average in ICC-MGC. Compared with CSD-L1, ICC-MGC increases energy consumption by 10% and 13% in *lud* and *triad* respectively, and ICC-AGC increases energy consumption by 7.3% in *triad*. With the poor remote duplication and low local hit rate, the cooperative requests in ICC methods cannot efficiently serviced by remote groups. Therefore, the additional overhead and power consumption of ICC methods have been exposed. But with the specific perception module, the power overhead of ICC-MGC is limited. Both optimization methods induce little energy consumption of other workloads. Specifically, for *ss*, the power consumption in ICC-AGC and ICC-MGC is lower than that in CSD-L1. This is mainly because the cooperation among L1 cache groups reduces the requests to shared L2 cache, so lower traffic to L2 cache leads to lower pressure and congestion to NoC, thereby, alleviating the power consumption caused by other components and reducing overall power consumption.

C. Sensitivity Studies

In this part we analyze the sensitivity of ICC schemes to the GPU core count and the L2 cache size.

1) Effect of GPU Core Count: In order to explore the possible impact of different GPU cores on the architecture of ICC-AGC and ICC-MGC, we set the number of GPU cores to 32, 64, 128 and 256 respectively for experiments. We mainly collect statistics on IPC, L1 cache hit rate and L2 cache traffic, and the experimental results are shown in Fig. 19a, and all results are normalized based on their respective baselines. The x-axis represents the specific indicator of IPC, L1 hit rate and L2 traffic. The y-axis represents the normalized results. Each indicator contains eight results, representing 2 architectures, ICC-AGC and ICC-MGC, with 4 numbers of 32, 64, 128, 256. The name of each result in the legend is consist of the specific ICC method and the number of GPU cores.

As shown in Fig. 19a, on increasing the number of GPU cores from 32 to 256, we observe that the average GPU performance also increased steadily, and most workloads exhibit similar performance improvement. The main reason for this improvement is that as the number of GPU cores increases, the proportion of duplicate data blocks among L1 cache group also increase, providing more opportunities for cooperative requests to hit in other cache groups, thereby increasing the cooperative requests hit rate. Therefore, a large number of missing requests can be serviced , which improves L1 bandwidth utilization and reduces L2 traffic, resulting in a higher performance improvement with larger cores.

In Fig. 19a, we observe that most workloads exhibit relatively stable improvements in L1 cache hit rates in ICC-AGC and ICC-MGC. On average, ICC-MGC has a higher L1 cache hit rate than that in ICC-AGC, mainly because ICC-MGC has a larger cooperation scope, and can utilize more duplicate data. Fig. 19a depicts that for most workloads, L2 cache traffic obtain significant reduction in ICC-AGC and ICC-MGC, and the reduction remains stable as the number of GPU cores increases. The average L2 cache traffic in ICC-MGC is much smaller than that in ICC-MGC, because the remote L1 cache

group can handle more L1 cache read miss requests in ICC-MGC.

2) Effect of L2 Cache Capacity: In this study, we also analyze the performance impact of L2 cache capacity in ICC-AGC and ICC-MGC, and Fig. 19b shows the performance variation on IPC, L1 hit rate and L2 traffic with different L2 cache capacity, and the number of L2 cache blocks is 256, 512, 1024, 2048 respectively. The organization and layout of the content in Fig. 19b is consistent with that in Fig. 19a, and all results are normalized based on the baselines respectively. Similarly, the name of each result in the legend is consist of the specific ICC method and the number of L2 cache blocks.

It is worth noting that for most workloads, the performance effect of L2 cache capacity is not significant, and the workloads exhibit similar behaviors under different configurations. The main reason is that our proposed inter-group cache cooperation method, ICC-AGC and ICC-MGC, mainly focus on and utilize duplicate data among L1 caches, so the L2 cache capacity would not directly impact the overall performance. On average, for different L2 cache capacity configurations, the performance obtain relatively stable improvement in both ICC-AGC and ICC-MGC, which verifies the effectiveness of our proposed ICC methods.

But for different L2 cache capacity configurations, the performance in ICC-AGC and ICC-MGC is better than that in Baselines. When the L2 cache capacity increases, the L1 cache hit rate decreases in ICC-AGC and ICC-MGC, but the overall performance only changes slightly. Similarly, L2 cache traffic can be significantly reduced in ICC-MGC compared to Baseline and ICC-AGC, indicating that a considerable part of cooperative requests can be served in ICC-MGC, thus leading to overall performance improvement.

V. Related work

In this section, we briefly summarize existing works that focus on duplicate data among L1 caches, and categorize them into two aspects of exploiting and removing duplication among L1 caches. Detailed works are presented as follows.

Many prior works attempt to design an efficient communication mechanism to exploit duplication among L1 caches to reduce shared L2 traffic. Specifically, Dublish et al. [23] designed a Cooperative Caching Network to connect all L1 caches to leverage duplication of remote SMs. Ibrahim et al. [24] design duplicate data prediction and parallel probing mechanisms to further optimize multiple SMs communication. Recently, Cheng et al. [35] proposed COLAB to perceive duplicate requests within a cluster to accelerate data servicing.

Other works focus on developing novel shared architecture to remove duplication among L1 caches, thus increasing overall cache capacity. Wang et al. [36] and Choo et al. [37] proposed to share an L1 Data cache across several SMs. Ibrahim et al. [25] introduced a shared L1 cache organization, where each SM only cache a non-overlapping slice of the entire address range to further eliminate duplication. Ibrahim et al. [26] recently proposed Clustered Shared Decoupled L1 cache (CSD-L1), which decoupled the L1 caches from SMs and aggregated to groups, then eliminate data duplication within each group and limit overall duplication. However, Xu et al. [27] observed that there exists high resource contention in CSD-L1, and they proposed ATA-Cache, which decoupled and aggregated the tag arrays of multiple L1 caches and co-design a two-level thread-block scheduling to maximize locality.

VI. Conclusion

This paper studies the duplication characteristics among L1 cache groups in existing cache architecture, and propose a inter-group cache cooperation method ICC. ICC is composed of two cooperation schemes of ICC-AGC and ICC-MGC. Experimental results on GPGPU-Sim with comprehensive benchmarks have shown that ICC can achieve a higher cache hit ratio and GPU performance.

References

[1] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: dynamic gpu memory management for training deep neural networks," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–53. [Online]. Available: https://doi.org/10.1145/3178487.3178491

[2] J. Ryoo, M. Fan, X. Tang, H. Jiang, M. Arunachalam, S. Naveen, and M. T. Kandemir, "Architecture-centric bottleneck analysis for deep neural network applications," in *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2019, pp. 205–214.

[3] E. Choukse, M. B. Sullivan, M. O'Connor, M. Erez, J. Pool, D. Nellans, and S. W. Keckler, "Buddy compression: Enabling larger memory for deep learning and hpc workloads on gpus," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 926–939.

[4] Q. Sun, X. Zhang, H. Geng, Y. Zhao, Y. Bai, H. Zheng, and B. Yu, "Gtuner: tuning dnn computations on gpu via graph attention network," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1045–1050. [Online]. Available: https://doi.org/10.1145/3489517.3530584

[5] M. Risso, A. Burrello, F. Conti, L. Lamberti, Y. Chen, L. Benini, E. Macii, M. Poncino, and D. J. Pagliari, "Lightweight neural architecture search for temporal convolutional networks at the edge," *IEEE Transactions on Computers*, vol. 72, no. 3, pp. 744–758, 2023.

[6] NVIDIA, "Nvidia's next generation cuda compute architecture: Fermi," Website, 2009, https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.

[7] NVIDIA, "Nvidia tesla p100." Website, 2016, https://images.nvidia.cn/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[8] NVIDIA, "Nvidia telsa v100 gpu architecture." Website, 2017, https://images.nvidia.cn/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[9] NVIDIA, "Nvidia turing gpu architecture," Website, 2018, https://images.nvidia.cn/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[10] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, p. 20–24, mar 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[11] S. H. Fuller and L. I. Millett, "Computing performance: Game over or next level?" *Computer*, vol. 44, no. 1, pp. 31–38, 2011.

[12] N. D. E. Jerger and L.-S. Peh, "On-chip networks," *Synthesis Lectures on Computer Architecture*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:38177873

[13] M. S. Gaur, V. Laxmi, M. Zwolinski, M. Kumar, N. Gupta, and Ashish, "Network-on-chip: Current issues and challenges," in *2015 19th International Symposium on VLSI Design and Test*, 2015, pp. 1–3.

[14] M. Jalili and M. Erez, "Reducing load latency with cache level prediction," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 648–661.

[15] A. Bakshi, J.-L. Gaudiot, W.-Y. Lin, M. Makhija, V. K. Prasanna, W. W. Ro, and C. Shin, "Memory latency : to tolerate or to reduce ?" 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:634879

[16] X. Chen, L.-W. Chang, C. I. Rodrigues, J. Lv, Z. Wang, and W.-M. Hwu, "Adaptive cache management for energy-efficient gpu computing," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 343–355.

[17] X. Zhu, R. Wernsman, and J. Zambreno, "Improving first level cache efficiency for gpus using dynamic line protection," ser. ICPP '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3225058.3225104

[18] B. Li, J. Wei, J. Sun, M. Annavaram, and N. S. Kim, "An efficient gpu cache architecture for applications with irregular memory access patterns," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 3, jun 2019. [Online]. Available: https://doi.org/10.1145/3322127

[19] Y. Oh, G. Koo, M. Annavaram, and W. W. Ro, "Linebacker: Preserving victim cache lines in idle register files of gpus," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 183–196.

[20] Y. Tian, S. Puthoor, J. L. Greathouse, B. M. Beckmann, and D. A. Jiménez, "Adaptive gpu cache bypassing," ser. GPGPU-8. New York, NY, USA: Association for Computing Machinery, 2015, p. 25–35. [Online]. Available: https://doi.org/10.1145/2716282.2716283

[21] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access pattern-aware cache management for improving data utilization in gpu," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 307–319.

[22] N. Agarwal, D. Nellans, M. O'Connor, S. W. Keckler, and T. F. Wenisch, "Unlocking bandwidth for gpus in cc-numa systems," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 354–365.

[23] S. Dublish, V. Nagarajan, and N. Topham, "Cooperative caching for gpus," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, dec 2016. [Online]. Available: https://doi.org/10.1145/3001589

[24] M. A. Ibrahim, H. Liu, O. Kayiran, and A. Jog, "Analyzing and leveraging remote-core bandwidth for enhanced performance in gpus," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019, pp. 258–271.

[25] M. A. Ibrahim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and leveraging shared l1 caches in gpus," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 161–173. [Online]. Available: https://doi.org/10.1145/3410463.3414623

[26] M. A. Ibrahim, O. Kayiran, Y. Eckert, G. H. Loh, and A. Jog, "Analyzing and leveraging decoupled l1 caches in gpus," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 467–478.

[27] X. Xu, L. Wang, L. Xiao, L. Liu, Y. Lv, X. Xie, M. Han, and H. Liu, "Ata-cache: Contention mitigation for gpu shared l1 cache with aggregated tag array," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2023.

[28] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163–174.

[29] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[30] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," in *2012 Innovative Parallel Computing (InPar)*, 2012, pp. 1–10.

[31] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a mapreduce framework on graphics processors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 260–269. [Online]. Available: https://doi.org/10.1145/1454115.1454152

[32] A. Karki, C. Palangotu Keshava, S. Mysore Shivakumar, J. Skow, G. Madhukeshwar Hegde, and H. Jeon, "Tango: A deep neural network benchmark suite for various accelerators," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 137–138.

[33] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU-3. New York, NY, USA: Association for Computing Machinery, 2010, p. 63–74. [Online]. Available: https://doi.org/10.1145/1735688.1735702

[34] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 738–753. [Online]. Available: https://doi.org/10.1145/3466752.3480063

[35] B.-W. Cheng, E.-M. Huang, C.-H. Chao, W.-F. Sun, T.-T. Yeh, and C.-Y. Lee, "Colab: Collaborative and efficient processing of replicated cache requests in gpu," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 314–319. [Online]. Available: https://doi.org/10.1145/3566097.3567838

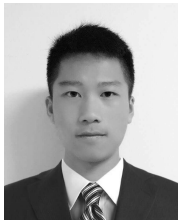
[36] J. Wang, L. Jiang, J. Ke, X. Liang, and N. Jing, "A sharing-aware l1.5d cache for data reuse in gpgpus," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 388–393. [Online]. Available: https://doi.org/10.1145/3287624.3287633

[37] K. Choo, W. Panlener, and B. Jang, "Understanding and optimizing gpu cache memory performance for compute workloads," in *2014 IEEE 13th International Symposium on Parallel and Distributed Computing*, 2014, pp. 189–196.

Guosheng Wang is currently a master student in School of Computer Science and Technology, Wuhan University of Technology. His current research interests include cache design and optimization of GPU.

Yajuan Du received joint PhD degrees of City University of Hong Kong and Huazhong University of Science and Technology in Feb. 2018 and Dec. 2017, respectively. She is now an associate professor in School of Computer Science and Technology, Wuhan University of Technology.

Her research interest focuses on optimizing GPU performance and the performance of nonvolatile memories.



Weiming Huang received the M.S. degree from the Wuhan University of Technology, Wuhan, China. His current research interests include cache and TLB design of GPU.