Analyzing and Leveraging Decoupled L1 Caches in GPUs

Mohamed Assem Ibrahim*†, Onur Kayiran†, Yasuko Eckert†, Gabriel H. Loh†, Adwait Jog*
*William & Mary †Advanced Micro Devices, Inc.
maibrahim@email.wm.edu, onur.kayiran@amd.com, yasuko.eckert@amd.com, gabriel.loh@amd.com, ajog@wm.edu

***Abstract*—Graphics Processing Units (GPUs) use caches to provide on-chip bandwidth as a way to address the memory wall. However, they are not always efficiently utilized for optimal GPU performance. We find that the main source of this inefficiency stems from the tightly-coupled design of cores with L1 caches. First, such a design assumes a per-core private local L1 cache in which each core independently caches the required data. This allows the same cache line to get replicated across cores, which wastes precious cache capacity. Second, due to the many-to-few traffic pattern, the tightly-coupled design leads to low per-core L1 bandwidth utilization while L2/memory is heavily utilized.**

To address these inefficiencies, we renovate the conventional GPU cache hierarchy by proposing a new DC-L1 (DeCoupled-L1) cache – an L1 cache separated from the GPU core. We show how decoupling the L1 cache from the GPU core provides opportunities to reduce data replication across the L1s and increase their bandwidth utilization. Specifically, we investigate how to aggregate the DC-L1s; how to manage data placement across the aggregated DC-L1s; and how to efficiently connect the DC-L1s to the GPU cores and the L2/memory partitions. Our evaluation shows that our new cache design boosts the useful L1 cache bandwidth and achieves significant improvement in performance and energy efficiency across a wide set of GPGPU applications while reducing the overall NoC area footprint.

***Index Terms*—Bandwidth, GPUs, Locality, Network-on-Chip**

I. Introduction

Graphics Processing Unit (GPU) architectures are a critical component in most high-performance computing systems as they provide faster and more energy efficient execution for many general purpose applications. GPUs employ a conventional two-level cache hierarchy where each core incorporates a private L1 cache and all the GPU cores are connected via a Network-on-Chip (NoC) to a shared and banked L2 cache. The L1 and L2 caches are used to boost the on-chip bandwidth as a means to address the well-known memory wall problem [1]. An increase in the on-chip bandwidth translates into performance improvements for memory-sensitive applications [2]–[5]. Therefore, prior research efforts developed hardware and software schemes to improve cache performance [3], [6]–[11]. However, we find that the conventional cache hierarchy leads to inefficient utilization of the valuable on-chip caches. Specifically, the tight coupling between the GPU cores and the L1 caches results in the following two inefficiencies.

The first inefficiency stems from the many-to-few communication between the L1s and the L2 banks. This puts more pressure on the few L2s and less pressure on the many per-core L1s, which results in a low bandwidth utilization for the per-core L1s [12]. The second inefficiency is due to the private nature of the L1 caches. This may lead to high cache line (data) replication across the L1 caches [3], [5], [13], [14] as each GPU core may independently cache the same cache line. Such replication effectively wastes the overall L1 cache capacity, leading to lower L1 hit rates and hence reduces its useful bandwidth. If cache line replication is reduced, then the L1 caches can effectively provide more capacity to cache more data, leading to higher hit rates, more delivered on-chip bandwidth, and reduced pressure on the L2 and memory.

In this paper, we address these two inefficiencies by breaking the tight coupling between the GPU cores and the L1 caches. To achieve that, we renovate the GPU two-level cache hierarchy and propose **De**Coupled**-L1** (**DC-L1**) caches, where we separate the L1 caches from the GPU cores. The decoupled nature of the DC-L1 caches enables aggregating the DC-L1 caches into bigger caches (while maintaining the total L1 cache capacity), in which each DC-L1 cache is accessed by multiple GPU cores. Aggregating DC-L1 caches improves their individual bandwidth utilizations and reduces data replication across the DC-L1s as more cores are accessing a given DC-L1. Although extreme aggregation of DC-L1s (all cores accessing one DC-L1) eliminates replication and improves DC-L1 bandwidth utilization, it can drastically reduce the overall peak L1 bandwidth and hence performance. In this paper, we use the aggregation granularity as a knob to reduce replication and improve cache bandwidth utilization while managing the overall peak L1 bandwidth.

Once we achieve a suitable aggregation granularity, we propose managing data placement across the DC-L1s to further reduce replication. Specifically, we evaluate a shared DC-L1 cache design to eliminate replication across the DC-L1 caches. With a shared DC-L1 cache design, each DC-L1 *exclusively* caches a unique slice of the address range. This ensures only one copy of data exists across DC-L1s, thereby eliminating replication and making better use of the finite cache capacity. However, we show that the shared DC-L1 cache design requires all-to-all communication between the GPU cores and the DC-L1s, which imposes significant NoC area/power overheads and NoC scalability/clocking challenges. Therefore, we propose to vary the sharing granularity using a **Clustered DC-L1** cache design to balance the trade-off between the replication waste and the NoC overheads. With such a design, we group the DC-L1 caches into clusters and enable the shared cache organization only within each cluster instead of enabling a fully shared cache across all DC-L1s. Therefore, we eliminate replication within the DC-L1 cluster and reduce replication across all the DC-L1s in a controlled fashion. This improves overall GPU throughput while reducing the overall GPU area and energy requirements.

To enable these DC-L1-based cache designs, a revamped NoC design is also required to connect the DC-L1 caches to the GPU cores and the memory partitions. The updated NoC design depends on the granularity of DC-L1 aggregation and the granularity of sharing under the clustered DC-L1 cache design. Also, given the shared nature of the L2 slices and the unique address range assigned to each DC-L1 within a cluster, each DC-L1 will communicate only with a few L2 slices. This further reduces the overall area and energy requirements.

Contributions: This paper contributes the following:

- We propose DC-L1 caches where we dissociate the L1 caches from the GPU cores and aggregate them.
- We propose co-designing the DC-L1 caches and the NoC to build a shared DC-L1 cache organization that *eliminates* cache line replication across the DC-L1 caches. We show that our holistic approach significantly improves the collective L1 hit rates and reduces the bandwidth pressure to the lower levels of the memory hierarchy for the applications that are sensitive to high replication volume.
- To address the drawbacks of the shared DC-L1 organization (NoC area/power overheads, scalability, and clocking challenges), we propose a clustered shared DC-L1 cache design that *limits* data replication. This cache design enables a cluster of GPU cores to access a cluster of shared DC-L1 caches, thus eliminating data replication within the cluster and reducing it across the GPU.
- We evaluate our clustered DC-L1 design across 28 GPGPU applications. On average, our proposal boosts performance by 75% (up to 8×) for the applications that are sensitive to high data replication without degrading performance of the applications that are insensitive. Additionally, our proposal reduces the total NoC area by 50%.

II. Motivation and Analysis

In this section, we discuss the inefficiencies of tightly coupled L1 caches in GPUs and make a case for separating and aggregating the L1 caches to address those drawbacks.

A. Inefficiency#1: Cache Line Replication across L1 Caches

With the baseline private L1 cache design, each GPU core satisfies its L1 requests from the local L1 cache. On a miss, each core *independently* fetches the required data from the L2 cache. This may lead to replication across L1s if the cores request the same cache line, leading to wasted cache capacity.

Wasted L1 Cache Capacity. The volume of replication of the evaluated applications is shown in Figure 1 in terms of *Replication Ratio*, sorted in ascending order. Replication ratio is defined as the ratio of L1 misses that can be found in other L1 caches to total L1 misses. We observe that the replication ratio varies across the evaluated applications. Specifically, some applications have no replication (e.g., `C-BLK`) or low replication (e.g., `C-RAY`), while others have high replication (e.g., `C-BFS`). We also observe that `T-AlexNet`, `T-ResNet`, and `T-SqueezeNet` from Tango benchmark suite [15] have significantly high replication. For example, `T-AlexNet` has a replication ratio of 95%.

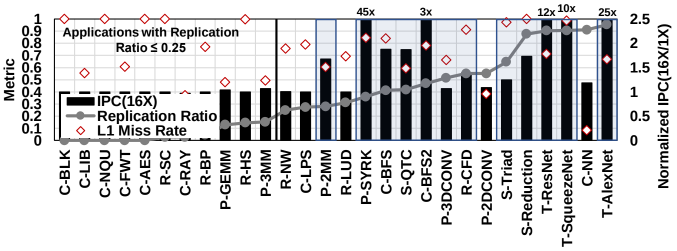


Fig. 1: Performance of the evaluated applications in terms of IPC improvement under 16× the L1 cache (normalized to baseline), L1 miss rate, and cache line replication ratio. The left-hand y-axis represents replication ratio and raw L1 miss rate. The experimental methodology is detailed in Section VII.

Identifying Replication-sensitive Applications. The waste due to data replication may not affect all applications. Only the applications that are sensitive to larger cache space are *expected* to benefit if the wasted cache space is reduced/eliminated. Therefore, we study performance of the evaluated applications under a 16× larger L1 cache in Figure 1. We observe that 15 applications are both capacity-sensitive and possess high data replication. To identify the subset of the capacity-sensitive applications that are *replication-sensitive*, we study their L1 miss rates. Applications with low L1 miss rates (e.g., `C-NN`) may not suffer under private L1 caches because the majority of their requests can be satisfied locally. In general, we consider an application to be replication-sensitive if it 1) has a replication ratio of >25%, 2) has an L1 miss rate of >50%, and 3) observes a speedup of >5% with 16× capacity. Based on these criteria, we observe that 12 applications are replication-sensitive (marked by the blue boxes in Figure 1).

Effect of Eliminating Replication. To estimate the potential performance benefits of eliminating data replication for the replication-sensitive applications, we evaluate a hypothetical design where all GPU cores access a *single* L1 cache (while maintaining the total L1 cache capacity and bandwidth) to ensure no replication. We observe that the L1 miss rate is reduced significantly by an average of 89.5% under such design. This is because removing replication allows for more data to be cached in L1s, thus improving L1 hit rates. For `T-AlexNet`, `T-ResNet`, and `T-SqueezeNet`, we observe an exceptional 99% reduction in the L1 miss rates as they have high replication volume (Figure 1). Overall, for the replication-sensitive applications, the significant reduction in L1 miss rates leads to more delivered on-chip bandwidth from the L1s, which translates to an IPC improvement of 2.9× on average.

B. Inefficiency#2: Low L1 Cache Utilization

The tight coupling of the L1 caches and GPU cores along with the many-to-few communication pattern (between the L1s and the L2 banks) puts more pressure on the few L2 banks and less pressure on the many L1 caches. This leads to low bandwidth utilization of the per-core L1 caches. We define the per-core L1 bandwidth utilization as the ratio of a core's L1 accesses (requests) over the total cycle count.

Figure 2 shows the maximum bandwidth utilization of the L1 cache data port, across all L1s, under all the evaluated applications sorted in ascending order. We observe that the highest bandwidth utilization of the L1 data ports is 18%. The low bandwidth utilization of the L1 caches is also shown by recent work [12], [16]. For a comprehensive view, we also study the utilization of NoC links that carry the data replies from the L2 to the GPU cores. Figure 2 shows the maximum NoC link utilization, across all links connected to GPU cores, for all evaluated applications in ascending order. Similar to the data port, the maximum link utilization is low (30%), which further shows the underutilization of the per-core L1s.

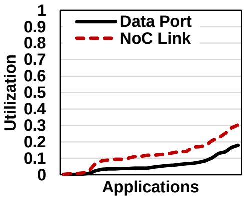


Fig. 2: L1 cache data port and NoC link utilization.

C. Solution: Decouple and Aggregate L1 Caches

We propose a **DeCoupled-L1** cache (**DC-L1**) – an L1 cache separated from the GPU core (Section III). This breaks the tight coupling between these entities and enables optimizations to reduce replication across the L1s and boost their bandwidth utilizations. These optimizations include aggregating the DC-L1 caches (Section IV) and managing data placement across the aggregated caches (Section V and Section VI).

III. Decoupled-L1 (DC-L1) Design

In this section, we describe Decoupled-L1 (DC-L1) caches and demonstrate how the DC-L1s, GPU cores, and L2/memory are connected. Also, we discuss the request/reply flow under the DC-L1-based design.

DC-L1 Node and NoC Design. Figure 3 🅐 shows our DC-L1 node design. A DC-L1 node simply contains the DC-L1 cache (DC-L1$), two queues to handle the traffic from/to the GPU core, and two queues to handle the traffic to/from the L2 and memory partitions. A GPU core in our design is a *Lite Core*. A lite GPU core is similar to the baseline GPU core but without the L1 data cache and the associated Miss Status Holding Registers (MSHR). Because the L1 caches are now separated from the GPU cores, we breakdown the NoC into two parts. The first NoC 🅑 (NoC#1) connects the GPU cores and the DC-L1 nodes. The second NoC 🅒 (NoC#2) connects the DC-L1 nodes and the L2/memory. The design of both NoCs is determined by the number of DC-L1 nodes and the cache organization.

Handling Read Requests. With a DC-L1-based design, an L1 read request is injected into NoC#1 to the target DC-L1 node as the GPU core does not have an L1 cache (and associated MSHR) anymore. The target DC-L1 node queues the received request into *Q1* ① to be served by the DC-L1$ in FIFO manner. The request at the head of the queue accesses the DC-L1$. If the request hits in the DC-L1$, then the DC-L1 node queues the read reply into *Q2* ② for injection into NoC#1 back to the GPU core. If the request misses in the DC-L1$, then the DC-L1 node queues the request into *Q3* ③ to be forwarded to the L2 cache through NoC#2. Once a read reply is received from the L2 via NoC#2, the DC-L1 node queues the reply into *Q4* ④ to be installed in its DC-L1$. Concurrently while caching the reply, the DC-L1 queues the reply to be injected to the requester GPU core through NoC#1 ⑤. This read reply only needs to provide the data requested by the requester GPU core. This is because, in our design, a GPU core does not have an L1 cache to install the data in. Hence, if the whole cache line is not required by the core, then sending the read reply at a full cache line granularity from the DC-L1 will waste NoC#1 bandwidth [17].

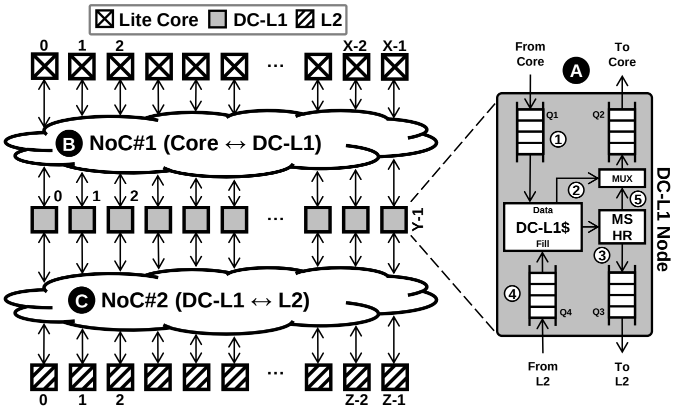


Fig. 3: Decoupled-L1 (DC-L1) node and NoC design.

Handling Write Requests. A write request follows the same flow as a read request. However, because we use a write-evict policy for the DC-L1 caches (Section VII), on a write hit, a given write request evicts the cache line from the DC-L1$. The evicted cache line is forwarded to the L2 cache through NoC#2. On a write miss, no cache line is allocated at the DC-L1$ and the updated data is delivered to the L2 cache as we use a no-write-allocate policy. Once a write ACK is received from the L2, the DC-L1 node forwards the write ACK to the requester GPU core via NoC#1.

Handling Non-L1 Requests. Because the DC-L1 node is on the path to L2, all the instruction, texture, and constant cache misses from the GPU core must go through the DC-L1 node to be forwarded to the L2 via NoC#2. These non-L1 requests do not access the DC-L1$. A given non-L1 request is simply moved from *Q1* to *Q3* bypassing the DC-L1$. Similarly, a non-L1 reply is moved from *Q4* to *Q2*. For clarity, the bypassing of DC-L1$ is not shown in Figure 3.

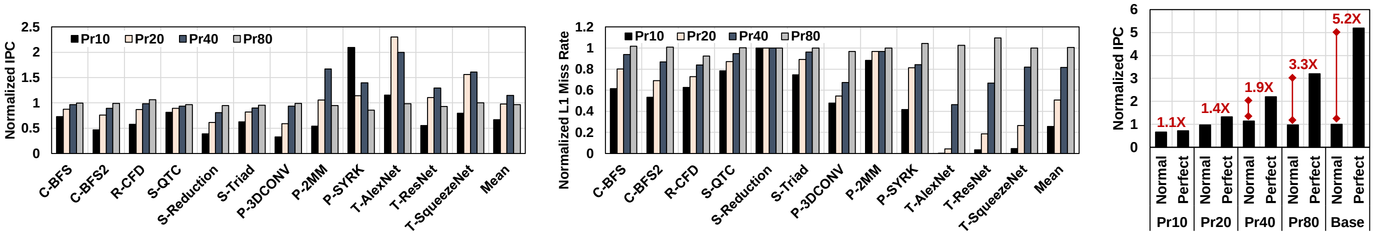
Handling Atomic Operations. In the baseline, atomic operations skip the L1 cache and are handled at L2/MC (memory controller) [18]. Similarly, in our design, atomic operations skip the DC-L1 cache and are handled at the unaltered L2/MC.

IV. Private DC-L1 Caches

In this section, we address the inefficiencies discussed in Section II. Specifically, to reduce data replication across the DC-L1s and improve their individual bandwidth utilizations, we investigate aggregating the DC-L1s into larger DC-L1s.

A. Designing Private DC-L1 Caches

Given a system with X GPU cores and X DC-L1 nodes, where each DC-L1 node hosts a single DC-L1$ with size



(a) IPC (b) L1 miss rate (c) IPC scope

Fig. 4: Performance under private DC-L1 design. Results are normalized to the private L1 baseline.

C, we aggregate these X DC-L1$ to form Y bigger DC-L1$ ($X > Y$). Each of the Y larger DC-L1$ has a size of $(X \times C)/Y$ and is hosted in a DC-L1 node. Under this design, each DC-L1 node is accessed privately by a group of $N = X/Y$ cores via an $N \times 1$ crossbar in NoC#1. We refer to this private aggregated DC-L1 design as **Pr**Y. In that sense, we can vary Y to control the granularity of aggregation (X/Y). Table I shows the different NoC configurations of a private DC-L1 design using different Y values under our 80-core baseline (Section VII). For example, Figure 5 shows the design of Pr40, where 80 DC-L1s are aggregated into 40 DC-L1s each with double the cache capacity. With Pr40, each DC-L1 node is privately accessed by two cores via a 2×1 crossbar in NoC#1. Such a private cache organization allows each DC-L1 to cache any line. For example, given the different address ranges represented by different patterns in Figure 5, a private DC-L1 cache can store any line from all address ranges.

TABLE I: NoC size and peak L1 bandwidth reduction under different private DC-L1 configurations.

Config.	NoC#1 Crossbars	NoC#2 Crossbars	Peak L1 BW	Peak L1 BW Drop
Baseline	NA	(×1) 80 × 32 XBar	$\frac{128\ Bytes}{1\ Cycle} \times 80$	-
Pr80	(×80) Direct Links	(×1) 80 × 32 XBar	$\frac{128\ Bytes}{4\ Cycles} \times 80$	4×
Pr40	(×40) 2 × 1 XBar	(×1) 40 × 32 XBar	$\frac{128\ Bytes}{4\ Cycles} \times 40$	8×
Pr20	(×20) 4 × 1 XBar	(×1) 20 × 32 XBar	$\frac{128\ Bytes}{4\ Cycles} \times 20$	16×
Pr10	(×10) 8 × 1 XBar	(×1) 10 × 32 XBar	$\frac{128\ Bytes}{4\ Cycles} \times 10$	32×

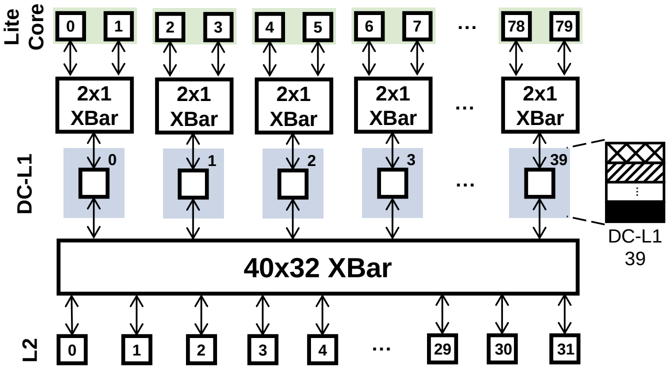


Fig. 5: Pr40 design.

B. Evaluating Private DC-L1 Caches

Performance. We evaluate performance of a private DC-L1 cache design on the replication-sensitive applications in terms of IPC and DC-L1 miss rate, normalized to the private L1 baseline in Figure 4. We start with Pr80 where we decouple the L1 caches without any aggregation. As shown in Table I, Pr80 connects the GPU cores to the corresponding DC-L1 nodes using *32-Byte* direct links in NoC#1, while connecting the DC-L1 nodes to L2/memory using a 80×32 crossbar in NoC#2. Because of the 32B links, the 128B cache line fetched from a given DC-L1 will be decomposed into four 32B chunks (assuming no control metadata) to be delivered sequentially to a requester core. Therefore, the peak theoretical DC-L1 cache bandwidth is 4× less than baseline (Table I). Nonetheless, as shown in Figure 4a, performance of Pr80 in terms of IPC is similar to baseline (3% drop on average). This is attributed to the latency tolerance property of GPGPU applications. This also shows that the peak L1 bandwidth is sufficiently abundant (as shown in [12], [16]). However, Pr80 does not reduce DC-L1 miss rate as shown in Figure 4b. This indicates no reduction in data replication across the DC-L1s. This is expected as, under Pr80, we do not aggregate the DC-L1s.

Under aggregated DC-L1s, a group of cores access a common caching resource (a single DC-L1$). For example, under Pr40, two cores access a single DC-L1$. As there is no cache line replication within a single cache, DC-L1 aggregation should reduce replication and enhance the collective hit rate of the DC-L1s. This is shown in Figure 4b where the DC-L1 miss rate drops by 19%, 49%, and 74% under Pr40, Pr20, and Pr10, respectively. In terms of throughput, Pr40 improves the IPC of the replication-sensitive applications by 15%, on average, compared to baseline. This is because of the reduction in data replication, hence higher DC-L1 hit rates, which leads to an increase in the on-chip bandwidth and overall performance. On the other hand, Pr20 and Pr10 reduce average performance by 3% and 34%, respectively. This is due to the significant drop in their peak L1 bandwidth (Table I) and the lower NoC bisection bandwidth due to using smaller crossbars in NoC#2.

To understand the scope of the private DC-L1 design under different DC-L1 node counts, we assume a perfect DC-L1$ with 100% hit rate. Figure 4c shows the average IPC improvement for the replication-sensitive applications under both normal and perfect DC-L1$ normalized to the private L1 baseline. Three observations are in order. First, Pr10 under perfect DC-L1$ still leads to a drop in performance by 28%

due to the reduced DC-L1 cache and NoC bandwidth. Second, Pr20 and Pr40 improve performance under perfect DC-L1$ by 40% and 90%, respectively, compared to their normal DC-L1$ counterparts. However, Pr40 has a higher IPC boost of 2.2× compared to the baseline with normal L1 cache. Finally, Pr80 under perfect DC-L1$ boosts performance by 3.3× compared to Pr80 with normal DC-L1$. However, it does not match the 5.2× improvement of having a perfect L1 cache in the baseline private case (denoted as *Base*). This is due to the 4× drop in the peak L1 bandwidth under Pr80.

Area & Power. We study NoC area and static power breakdown under different private DC-L1 configurations, normalized to the private baseline in Figure 6. We use DSENT [19] to model the crossbars in both NoC#1 and NoC#2, assuming a 22nm technology and assuming that all the evaluated crossbars can operate at the same clock frequency. We observe the following. First, Pr80 adds insignificant area and static power overhead compared to the baseline. This is because Pr80 only adds links to connect a given GPU core to its corresponding DC-L1 node. Second, Pr40, Pr20, and Pr10 reduce the NoC area by 28%, 54%, and 67%, respectively. This is due to breaking down the baseline 80×32 crossbar into smaller crossbars, thus reducing the NoC area [10], [20]. Third, the static power reduction under Pr40 is just 4%. This is because the small crossbars of Pr40 reduces the per-router static power from the *Crossbar* and the *Switch Allocator* components; however, it increases the static power from *Buffer* components (due to more routers). Finally, the static power reduction under Pr20 or Pr10 is more than Pr40. This is because Pr40 uses more small crossbars in NoC#1 and a bigger crossbar in NoC#2.

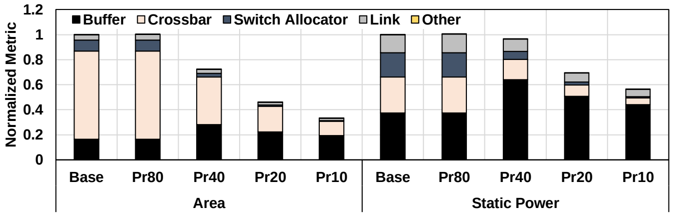


Fig. 6: NoC area and static power under private DC-L1 design. Results are normalized to the private L1 baseline.

Verdict. Because Pr40 improves throughput while reducing the NoC area and maintaining the power consumption (compared to baseline), we choose *40* DC-L1 nodes for the rest of this paper. However, to bridge the Pr40 performance gap between normal and perfect DC-L1, we need to investigate other innovative ways to reduce data replication, thus further improving the DC-L1s collective hit rates.

V. Shared DC-L1 Caches

To eliminate data replication across the DC-L1 caches, we investigate enabling a shared DC-L1 cache organization. Under a shared organization, the entire address range is interleaved across all the DC-L1s and such mapping is fixed. In other words, each DC-L1 exclusively caches data from a non-overlapping address range. A DC-L1 that can cache a given cache line is the *home DC-L1* of that line. For example, Figure 7 shows that the black address range can be cached by only DC-L1 39. In other words, DC-L1 39 is the home of the black address range. Because an exclusive slice of the address range maps to a single DC-L1, the shared organization ensures no cache line replication across DC-L1 caches.

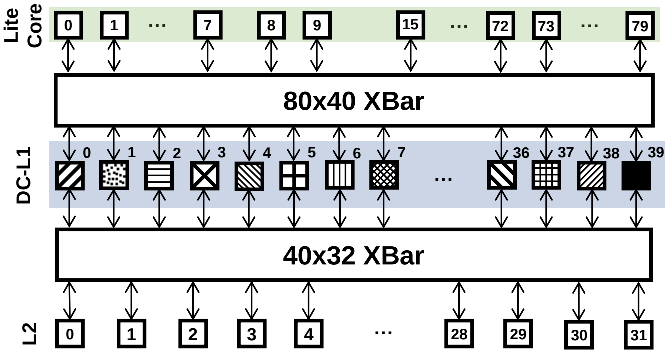


Fig. 7: Sh40 design.

A. Designing Shared DC-L1 Caches

To enable a shared DC-L1 organization, any core needs access to any DC-L1 node. Figure 7 shows one possible design to achieve that under our setup (Section VII). In this design, 80 GPU cores are connected to 40 DC-L1 nodes via an 80×40 crossbar in NoC#1. We refer to this design as **Sh40** (or **Sh**Y in general, where Y is the total number of DC-L1 nodes).

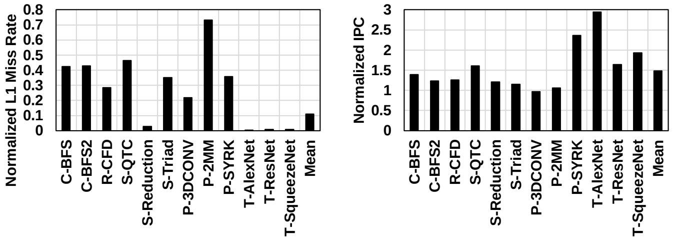
Selecting the Home DC-L1. To select the home DC-L1 for a given cache line, we use the *home bits*. These home bits are selected from the physical address of the request. The process of selecting these bits is analogous to selecting the appropriate L2 bank based on the physical address. In general, ShY design requires $\lceil log_2(Y) \rceil$ home bits to identify the home DC-L1.

Handling Requests. An L1 or a non-L1 request/reply (read or write) follows the same flow in Section III. The only difference is that the request/reply is forwarded to the home DC-L1.

B. Evaluating Shared DC-L1 Caches

Performance. We evaluate the performance of Sh40 on the replication-sensitive applications in terms of DC-L1 miss rate and IPC, normalized to the private L1 baseline in Figure 8. Under Sh40, the DC-L1 miss rate drops significantly by an average of 89% (minimum = 27%, maximum = 99%). The significant drop in the DC-L1 miss rate is expected as these applications have high data replication across the DC-L1s (Section II), which is eliminated under shared DC-L1 design. This effectively provides L1 cache capacity to store more cache lines, thus improving the L1 hit rate and the on-chip bandwidth. The boosted on-chip bandwidth from the DC-L1s is translated into a throughput boost of 48% on average (up to 2.9× for `T-AlexNet`) as shown in Figure 8b.

However, two replication-sensitive applications do not benefit from Sh40. Specifically, `P-2MM` achieves only 6% speedup because Sh40 can lead to a *partition camping* problem. Partition camping [21] is caused by cache accesses that are skewed toward a subset of the DC-L1 nodes. This leads to



(a) L1 miss rate

(b) IPC

Fig. 8: Performance under Sh40. Results are normalized to the private L1 baseline.

load imbalance between the DC-L1s and limits the benefits of the shared cache design. As for P-3DCONV, it suffers a 3% performance loss with Sh40 due to its sensitivity to available L1 cache bandwidth. Specifically, the traffic in NoC#1 is high due to the absence of the L1 caches from the GPU cores and the high DC-L1 hit rate with Sh40. Thus, the reduced peak cache bandwidth with 40 DC-L1s (Table I) limits the performance benefits of P-3DCONV under Sh40.

Area & Power. Although Sh40 improves performance of the replication-sensitive applications, it uses an 80×40 crossbar in NoC#1 to route the traffic from/to any GPU core to/from any DC-L1 node. This crossbar, in addition to a 40×32 crossbar in NoC#2, incurs a NoC area overhead of 69% and a NoC static power overhead of 57% compared to the private baseline.

Replication-insensitive Applications. We evaluate performance of Sh40 on the applications that are classified as *replication-insensitive* in Figure 9. We observe the following. First, most of these applications perform as well as the baseline (e.g., R-LUD and C-BLK). These applications have a high tolerance to the latency overhead induced by the DC-L1 design. Second, R-SC performs better than the baseline. This is because under the baseline, R-SC suffers from work distribution imbalance as some cores are assigned more cooperative thread arrays (CTAs). This leads to imbalance in L1 cache accesses across the cores. However, given the shared nature of the DC-L1 under Sh40, such imbalance in DC-L1 cache accesses is mitigated, which reduces the bottlenecks and improves performance. Second, five applications suffer a drop in performance with Sh40 (minimum = 40%, maximum = 85%). We refer to these applications as *poor-performing applications* in Figure 9. These applications either have high L1 hit rates and low latency tolerance (C-NN), suffer from partition camping (C-RAY, P-3MM, and P-GEMM), or are sensitive to the reduced peak L1 cache bandwidth (P-2DCONV).

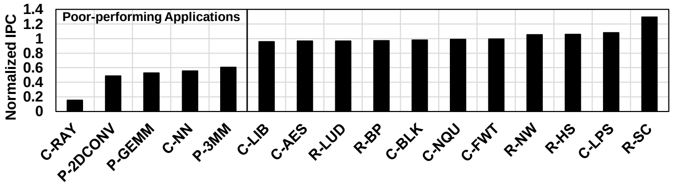


Fig. 9: Performance of replication-insensitive applications under Sh40. Results are normalized to the private L1 baseline.

Verdict. Although Sh40 significantly improves performance of replication-sensitive applications, it incurs a considerable NoC area and static power overhead. Also, some replication-insensitive applications suffer significant performance loss with Sh40. Therefore, to make a strong case for DC-L1-based designs, we need to address these two issues.

VI. Clustered Shared DC-L1 Caches

We need a design that provides performance boost for the replication-sensitive applications while reducing area and energy requirements. Also, it should *not* negatively affect the replication-insensitive applications. Therefore, in this section, we investigate limiting replication instead of eliminating it. Also, we utilize the fact that smaller crossbars in NoC#1 can be operated at a higher frequency to boost performance of both replication-sensitive and replication-insensitive applications.

A. Designing Clustered Shared DC-L1 Caches

The main reason behind the NoC area and power overhead of Sh40 is the 80×40 crossbar used in NoC#1. This crossbar is essential to enable the fully-shared cache organization. On the other hand, with Pr40, replication is still high, but the overall NoC area and static power is reduced. This presents a trade-off between the reduction in replication and the NoC area/power requirements. Therefore, we propose a cluster-based shared DC-L1 design where we enable the shared cache model across a cluster of DC-L1 caches instead of all of them. This eliminates replication across the DC-L1s of the same cluster, as shown in Figure 10. However, it still allows replication across the DC-L1s in different clusters. Using the number of clusters as a design parameter, we can limit replication while controlling the NoC area and power requirements.

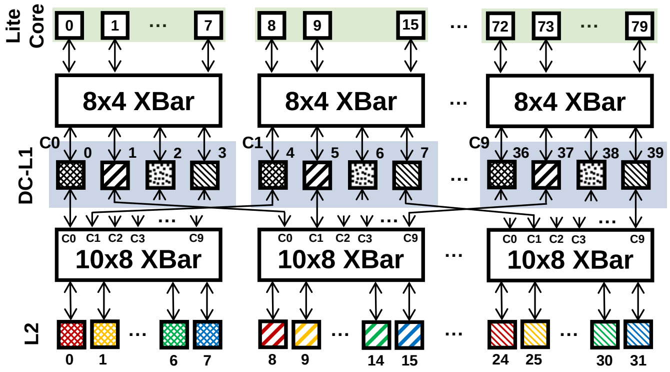


Fig. 10: Sh40+C10 design.

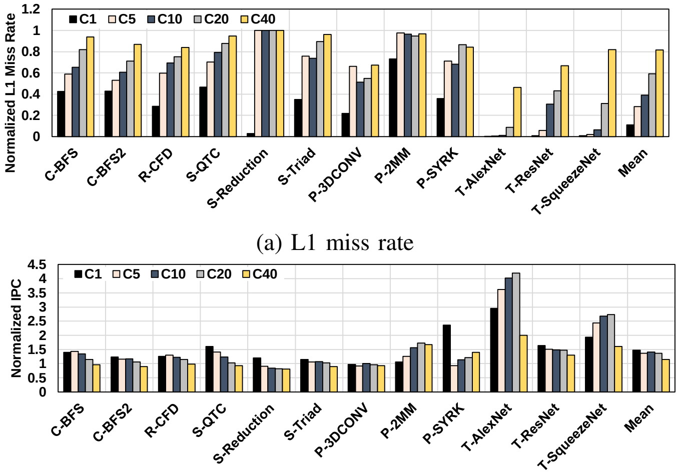
With this design, a cluster of M DC-L1 nodes is accessed by N cores via an $N \times M$ crossbar in NoC#1. We refer to this design as **Sh**Y**+C**Z, where Y is the total number of DC-L1 nodes and Z is the number of clusters ($Z = Y/M$). Additionally, because of the shared nature of the L2 slices and that each DC-L1 within a cluster is assigned a unique address range, a given DC-L1 will communicate only with a few L2 slices. Therefore, instead of using a full $Y \times L$ crossbar in NoC#2 to connect the Y DC-L1 nodes to the L

L2 slices ($L \geq M$, $L \bmod M = 0$), a given DC-L1 will communicate only with $O = L/M$ L2 slices via an $Z \times O$ crossbar in NoC#2. For example, Figure 10 shows the design of Sh40+C10 with 40 DC-L1s and 10 clusters. Each cluster consists of 8 cores accessing 4 shared DC-L1s via an 8×4 crossbar in NoC#1. The 40 DC-L1s are connected to the 32 L2 slices via four 10×8 crossbars in NoC#2. Specifically, all the 10 DC-L1s across the clusters that are assigned the same address range access the 8 L2 slices that serve such address range via a 10×8 crossbar in NoC#2. To illustrate, in Figure 10, the DC-L1s that serve the cross-hatched address range are connected to the L2 slices 0 to 7 (shown as different colors) that jointly serve such address range.

Selecting the Home DC-L1. As discussed in Section V-A, the selection of the home DC-L1 is based on the home bits. The only difference is the number of bits used from the physical address of a request. Specifically, ShY+CZ design requires $\lceil log_2(Y/Z) \rceil$ home bits.

B. Evaluating Clustered Shared DC-L1 Caches

Performance. In Figure 11, we evaluate performance of the clustered shared DC-L1 cache design on the replication-sensitive applications under different cluster counts. The results are normalized to the private L1 baseline. In this figure, $C1$ and $C40$ are equivalent to Sh40 and Pr40 designs, respectively. We make several observations. First, the L1 miss rate is higher when cluster count is more than one ($C > 1$). This is due to increased replication compared to the $C1$ case that keeps only a single copy of a given cache line across the DC-L1s. Specifically, up to 5, 10, 20, 40, and 80 copies of a cache line can exist across the DC-L1s with $C5$, $C10$, $C20$, $C40$, and baseline, respectively. This leads to an average L1 miss rate reduction (compared to baseline) of 72%, 61%, and 41% with $C5$, $C10$, and $C20$, respectively.



(a) L1 miss rate

(b) IPC

Fig. 11: Performance of Sh40 under different cluster counts. Results are normalized to the private L1 baseline.

Second, the performance improvement is still significant under $C5$, $C10$, and $C20$ even with the smaller reduction in the L1 miss rate (compared to $C1$). For example, $C10$ improves performance by 41%, on average, over the private L1 baseline. This represents a 5% drop in performance compared to $C1$. Third, the majority of the replication-sensitive applications perform better with $C1$ because of their sensitivity to the additional effective cache capacity achieved by eliminating the replication. On the other hand, some applications (e.g., `T-AlexNet`) perform better with clustering. This is because the controlled replication using clustering balances the useful L1 bandwidth from the additional cache capacity and from having multiple copies (hence sources) of a given cache line. This shows that controlled data replication may not negatively affect, and can even help improve, overall performance.

Finally, `P-3DCONV` does not obtain speedup with the clustered design and `S-Reduction` loses performance (15% drop). As discussed in Section V-B, the low performance of `P-3DCONV` is due to its sensitivity to the reduced peak cache bandwidth with the DC-L1-based designs. As for `S-Reduction`, its performance improves only with the fully-shared $C1$ design due to its replication pattern. This is evident by the 97% drop in L1 miss rate with $C1$. With other clustering options, L1 miss rate does not drop, which means no reduction in replication. Thus, neither the on-chip bandwidth nor performance is boosted. On the contrary, due to the decoupled nature of the DC-L1 design and the latency intolerance of the application, we observe performance degradation.

Area & Power. We evaluate the NoC area and static power of different cluster counts, normalized to the private L1 baseline, in Figure 12. Similar to our observation in Section IV-B, breaking the 80×40 crossbar in NoC#1 with $C1$ (Sh40) and using smaller crossbars to form the clusters leads to savings in the NoC area and static power. Also, using smaller crossbars in NoC#2 instead of the 40×32 crossbar further improves such savings. Specifically, compared to baseline, we observe a NoC area savings of 45%, 50%, and 45% with $C5$, $C10$, and $C20$, respectively. As for NoC static power, we observe a reduction of 15%, 16%, and 14% with $C5$, $C10$, and $C20$, respectively. Given the performance improvement and area/power savings of $C10$, we choose this design for the rest of this paper. We refer to it as **Sh40+C10**.

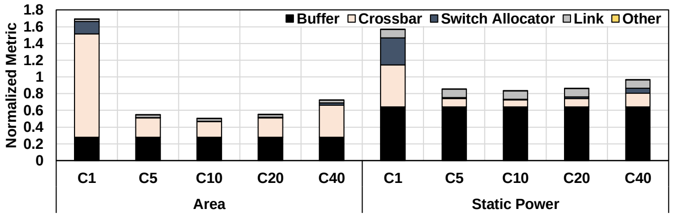
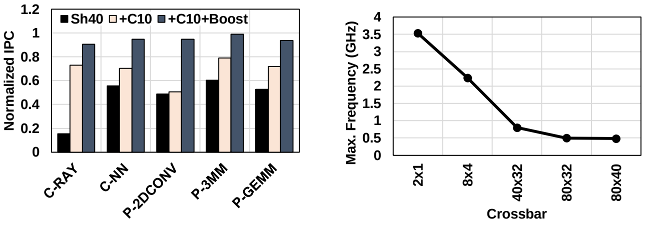


Fig. 12: NoC area and static power under different cluster counts. Results are normalized to the private L1 baseline.

Replication-insensitive Applications. Figure 13a shows performance of the poor-performing applications that significantly suffered with Sh40. We observe that Sh40+C10 drastically improves performance of three of these applications compared to Sh40. Specifically, `C-RAY`, `P-3MM`, and `P-GEMM` benefit



(a) Performance (b) Maximum frequency

Fig. 13: Performance of poor-performing replication-insensitive applications under Sh40+C10 (normalized to baseline) and the maximum frequency of various crossbars.

as the effect of partition camping is lower with the clustered design. In other words, the DC-L1 contention from partition camping is relieved by having multiple home DC-L1s (ten under Sh40+C10). However, even with this improvement, performance losses in these five applications are still significant compared to the private L1 baseline with a maximum drop of 49% in `P-2DCONV`. Therefore, we need to further improve the performance of these applications.

C. *Frequency-boosted Clustered Shared DC-L1 Design*

To further boost performance of both replication-sensitive and replication-insensitive (especially poor-performing) applications, we improve the performance of NoC#1. This NoC between the GPU cores and the DC-L1 nodes is busy with request and reply traffic due to the absence of L1 caches in the GPU cores and the high hit rate of the DC-L1s under the clustered shared design. To improve performance, we utilize the fact that our Sh40+C10 uses smaller crossbars in NoC#1. This enables us to boost the frequency of these small crossbars with minimal effect on the overall NoC dynamic power (evaluated in Section VIII). Using DSENT, we estimate the maximum operating frequency of different crossbars used in our designs in Figure 13b. We observe the low maximum operating frequency of the 80×32 crossbar used in the baseline and the 80×40 crossbar used in Sh40. On the other hand, the small crossbars used in Pr40 (2×1) and Sh40+C10 (8×4) can operate at significantly higher frequencies. Therefore, in Sh40+C10, we double the baseline frequency of the 8×4 crossbars in NoC#1 while keeping the baseline frequency of the 10×8 crossbars in NoC#2 the same.[1] We refer to this design as **Sh40+C10+Boost**. From Figure 13a, we observe that the frequency-boosted design improves performance of the poor-performing replication-insensitive applications significantly. The performance impact is particularly evident in `P-2DCONV` as it is sensitive to the available peak cache bandwidth (discussed in Section V-B). By doubling the frequency of NoC#1, Sh40+C10+Boost partially compensates for the drop in the peak cache bandwidth due to using 40 DC-L1s (Table I). Specifically, instead of enduring 8× peak cache bandwidth reduction with Sh40+C10 (compared to baseline), Sh40+C10+Boost has a 4× reduction.

Verdict. Sh40+C10+Boost is a balanced design that limits replication to at most 10 replicas. It achieves significant performance improvements for the replication-sensitive applications while reducing the NoC area and static power. Additionally, the boosted frequency in NoC#1 recovers most of the lost performance of the poor-performing applications.

VII. Experimental Setup

Simulated System. Our *private L1 baseline* assumes a generic GPU, consisting of multiple cores (also called Compute Units, or CUs) that have private L1 caches. These caches are connected to multiple address-sliced L2 cache banks via a NoC. We use two separate networks (request and reply) to avoid protocol deadlocks [18]. Our baseline and proposed designs assume a separate scratchpad memory and L1 data cache. The software-managed scratchpad memory is local per-core and its performance characteristics (latency and bandwidth) are unchanged across all designs. We model our baseline and proposed designs using GPGPU-Sim v.3 cycle-level simulator [18]. Table II provides a detailed platform configuration.

TABLE II: Configuration parameters of the simulated GPU.

Core Features	1400MHz core clock, 80 cores (CUs), SIMD width = 32 (16 × 2)
Resources / Core	48KB scratchpad, 32KB register file, Max. 1536 workitems (48 wavefronts, 32 workitems/wavefront)
L1 Caches / Core	16KB 4-way Write-evict L1 data cache - Latency = 28 cycles [16] 12KB 24-way texture cache, 8KB 2-way constant cache, 2KB 4-way I-cache, 128B cache block size
L2 Cache	8-way 128 KB/memory channel (4MB in total) 128B cache block size - Latency = 120 cycles
Memory Model	16 GDDR5 Memory Controllers (MCs) FR-FCFS scheduling, 16 DRAM-banks, 4 bank-groups/MC, 924 MHz memory clock, Global linear address space is interleaved among partitions in chunks of 256 bytes Hynix GDDR5 Timing [22]
Interconnect	80 × 32 crossbar topology, 700MHz interconnect clock, 32B flit size, 4 VCs per port, 4 flits/VC, iSLIP VC and switch allocators

Evaluated Applications. We evaluate 28 applications from five representative and diverse benchmarks suites (CUDA-SDK (C) [23], Rodinia (R) [24], SHOC (S) [25], PolyBench (P) [26], and Tango (T) [15]).

VIII. Experimental Results

In this section, we evaluate and compare the following against the private L1 baseline:

- **Pr40:** The proposed private DC-L1 cache design (Section IV) in which we reduce the number of the DC-L1 nodes to 40 while maintaining the total DC-L1 cache capacity.
- **Sh40:** The proposed fully-shared DC-L1 cache design (Section V) in which we enable a shared DC-L1 cache organization to eliminate data replication across the DC-L1s.
- **Sh40+C10:** The proposed cluster-based DC-L1 cache design (Section VI-A) in which we apply the shared cache design across a cluster of DC-L1s to eliminate data replication within the cluster and limit replication in the GPU.
- **Sh40+C10+Boost:** The proposed frequency-boosted Sh40+C10 (Section VI-C) that doubles the frequency of the *small* 8×4 crossbars in NoC#1 under Sh40+C10 design.

Effect on Performance. Figure 14 shows performance of our proposed designs in terms of IPC normalized to the private L1

[1] We do not boost NoC#2 frequency as it has less traffic due to the high hit rate of the DC-L1s.

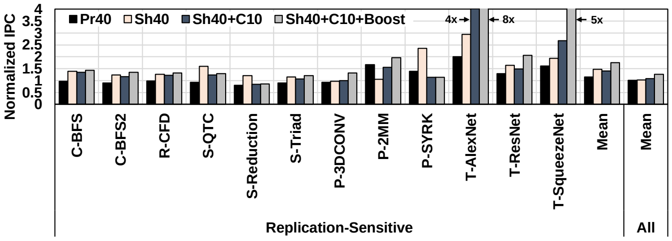


Fig. 14: The effect of the proposed designs on IPC. Results are normalized to the private L1 baseline.

baseline. We observe the following. First, all the proposed designs improve performance of the replication-sensitive applications by varying degrees. Specifically, an improvement of 15%, 48%, 41%, and 75% is achieved under Pr40, Sh40, Sh40+C10, and Sh40+C10+Boost, respectively. Second, performance of P-3DCONV only improves under Sh40+C10+Boost (31%). This is because the cache bandwidth sensitivity of P-3DCONV (Section V-B) is addressed by the frequency boost in NoC#1, which partially compensates the lost peak cache bandwidth under the DC-L1-based designs. Third, performance of P-2MM improves with Sh40+C10(+Boost) as the DC-L1 contention from partition camping is alleviated by having 10 home DC-L1s. Fourth, S-Reduction still suffers a drop in performance (14%) under Sh40+C10+Boost due to its replication pattern that can only be eliminated/reduced under the fully shared Sh40, as discussed in Section VI-B. For the same reason, P-SYRK achieves a lower IPC improvement of 13% with Sh40+C10+Boost compared to 2.4× with Sh40.

For the replication-insensitive applications (not shown due to lack of space), a 7%, 22%, and 11% drop in performance is incurred under Pr40, Sh40, and Sh40+C10, respectively. However, Sh40+C10+Boost maintains performance of these applications with an average IPC drop of only <1%. This is because of the frequency boost in NoC#1, which in return pushed performance of the poor-performing replication-insensitive applications (e.g., C-NN and P-2DCONV). Overall, Sh40+C10+Boost improves performance of all evaluated applications by 27%, as shown in Figure 14. To demonstrate that, we show the speedup of all evaluated applications sorted in ascending order under the proposed designs in Figure 15. This shows that Sh40+C10+Boost can provide performance benefits for the replication-sensitive applications. Also, Sh40+C10+Boost pushes the tail of the S-curve towards the private L1 baseline, thus maintaining performance of the replication-insensitive applications.

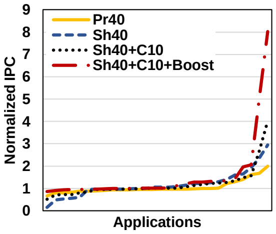


Fig. 15: The effect of the proposed designs on IPC (normalized to baseline) as S-curve.

Effect on L1 Miss Rate. Figure 16 shows the effectiveness of our designs in reducing the DC-L1 miss rate. The results are normalized to the private L1 baseline. The reduction in the DC-L1 miss rate under Sh40+C10(+Boost) is higher compared to Pr40 and lower compared to Sh40. Such a reduction is directly proportional to the reduction in data replication. Specifically, for the replication-sensitive applications, only a single copy of the data (i.e., zero replicas) is maintained under Sh40. However, under the private L1 baseline, each L1 can store any cache line, which may lead to more replicas across the L1s (7.7 replicas on average). Pr40 can reduce the replica count (5.7 replicas on average) compared to baseline. Sh40+C10+Boost strikes a middle ground between Pr40 and Sh40 (2.8 replicas on average).

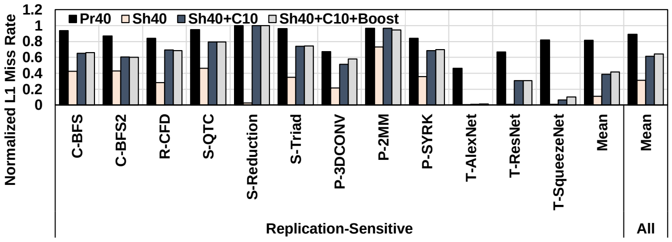


Fig. 16: The effect of the proposed designs on L1 miss rate. Results are normalized to the private L1 baseline.

Effect on L1 Utilization. Figure 17 illustrates the bandwidth utilization of the L1/DC-L1 cache data port (maximum per L1/DC-L1 accesses over the execution time) with our designs for all evaluated applications sorted in ascending order. We observe that all the proposed designs show higher DC-L1 data port utilization compared to baseline L1 data port utilization. This is from reducing the number of DC-L1 nodes (by aggregating the DC-L1 caches), which leads to more requests served by each DC-L1 compared to baseline. We also studied the utilization of NoC links that carry the data replies from the L2 to the DC-L1s and observed similar trends.

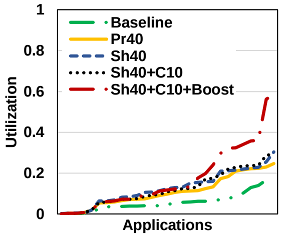
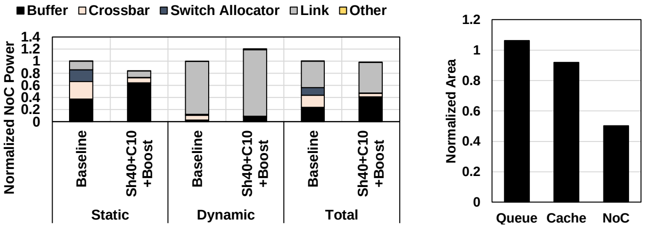


Fig. 17: L1 cache data port utilization.

Energy Analysis. With Sh40+C10+Boost, an 8×4 crossbar (in NoC#1) connects a nearby cluster of 8 GPU cores and 4 DC-L1 nodes via short 3.3mm links. The DC-L1 nodes and the L2 slices are connected to the 10×8 crossbars (in NoC#2) via long 12.3mm links. These conservative estimations are similar to prior work [10], [20]. We use DSENT to estimate the power consumption of the crossbars in both NoC#1 and NoC#2 assuming a 22nm technology. We use GPGPU-Sim to collect the flit count and NoC link activity to estimate the injection rates from the GPU cores, DC-L1 nodes, and L2 banks. We feed these estimates into DSENT to compute NoC dynamic power. Figure 18a shows the static, dynamic, and total NoC power breakdown for Sh40+C10+Boost normalized to the private L1 baseline. We observe the following. First, Sh40+C10+Boost reduces the NoC static power by 16% compared to baseline. Second, compared to baseline, the dynamic power of Sh40+C10+Boost is on average 20% higher because of the high traffic volume in NoC#1. Finally, even with the high dynamic power toll, the overall NoC power



(a) NoC power (b) Area

Fig. 18: Power and area under Sh40+C10+Boost. Results are normalized to the private L1 baseline.

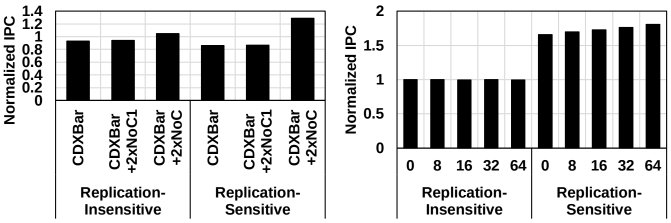
under Sh40+C10+Boost 2% lower than baseline. Given the improvement in the overall throughput and execution time, the average energy savings under Sh40+C10+Boost is 35% compared to baseline. Therefore, Sh40+C10+Boost improves performance-per-watt and energy efficiency (performance-per-energy), on average, by 29.5% and 95%, respectively.

Area Analysis. Figure 18b shows the area overhead/savings of Sh40+C10+Boost in terms of the queues within the DC-L1 nodes, the DC-L1 caches, and the NoC. We discussed the NoC area breakdown in Section VI-B and showed that Sh40+C10 reduces NoC area requirements by 50%. The Boost optimization affects the NoC dynamic power and minimally affects the NoC area. As for the queues within the DC-L1 nodes, we use four queues (Figure 3) in each DC-L1 node. Each queue holds up to four 128B entries. All the queues impose an overhead of 6.25% compared to the total baseline L1 cache. This overhead is compensated by the 8% cache area savings from aggregating the DC-L1 caches in a fewer number of DC-L1 nodes. The cache area is estimated using CACTI 6.5 [27]. Even though the cache budget is maintained under Sh40+C10+Boost, we save area as we use fewer cache ports. Specifically, Sh40+C10+Boost has 50% fewer DC-L1 cache banks and hence fewer cache ports.

Latency Analysis. The decoupled nature of the DC-L1s imposes additional latency for the communication between the GPU cores and the DC-L1s. We estimated such latency under the evaluated applications with Sh40+C10+Boost, and observed an overhead of 54 cycles, on average. Another source of latency overhead is the aggregation of the DC-L1s. Specifically, with Sh40+C10+Boost, each DC-L1 cache is double the size of the baseline L1 cache, which adds a 7% increase in the DC-L1 access latency. Specifically, the DC-L1s with Sh40+C10+Boost have an access latency of 30 cycles, compared to 28 cycles L1 access latency in the baseline (Table II). Such latency overheads do not negatively affect the evaluated applications because of the latency tolerance of the GPGPU applications. In fact, given the additional provided on-chip bandwidth from the DC-L1s with Sh40+C10+Boost, we observe a 53% reduction in the overall round trip time to fetch the required data, compared to the private L1 baseline.

A. *Sensitivity Studies*

Hierarchical Crossbar. Zhao *et al.* [10], [20] proposed a hierarchical two-stage crossbar design to improve the NoC scalability, area, and power. In Figure 19a, we evaluate a hierarchical crossbar design similar to [20] (denoted as *CDXBar*) normalized to the private L1 baseline. We observe that both the replication-insensitive and the replication-sensitive applications incur performance degradation of 7% and 14% with CDXBar, respectively. This is because the main design goal of CDXBar is not performance. For a fair comparison with Sh40+C10+Boost, we study doubling the NoC frequency of the small crossbars in the first stage of CDXBar (denoted as *CDXBar+2xNoC1*) and observed a minor performance improvement (<1%) compared to CDXBar. This is because CDXBar (and CDXBar+2xNoC1) does not reduce data replication across the L1 caches, which puts pressure on the crossbars of the second stage of CDXBar. Hence, once we double the frequency of both stages of CDXBar (denoted as *CDXBar+2xNoC*), we observe performance improvement of 29% for the replication-sensitive applications. Such improvement is 26% lower compared to the 75% improvement under Sh40+C10+Boost. As for the replication-insensitive applications, CDXBar+2xNoC improves their performances by 5% compared to a slight <1% loss under Sh40+C10+Boost. However, CDXBar+2xNoC incurs higher dynamic NoC power overhead due to doubling the frequency of all the crossbars. In summary, Sh40+C10+Boost improves performance significantly compared to CDXBar-based designs, while achieving similar NoC area and power savings.



(a) Hierarchical crossbar (b) L1 access latency

Fig. 19: Sensitivity studies.

L1 Access Latency. In our baseline, we assume 28 cycles access latency for the L1 caches (Table II). Figure 19b shows performance of Sh40+C10+Boost under different L1 (and DC-L1) access latency, ranging from zero to 64 cycles, normalized to its respective private L1 baseline. We observe that Sh40+C10+Boost achieves a significant 66% performance improvement for the replication-sensitive applications even under zero access latency while maintaining the performance of the replication-insensitive applications (<1% drop).

CTA Scheduling. We evaluate the effect of the state-of-the-art distributed CTA scheduler [28] compared to the default round-robin CTA scheduler under Sh40+C10+Boost. Even with such scheduler, we observe 46% performance improvement for the replication-sensitive applications. The reduction in performance benefits is attributed to mapping the nearby CTAs to the same core which may reduce replication.

System Size. We study the scalability of our frequency-boosted clustered shared design (Section VI) by evaluating Sh60+C10+Boost under a 120-core system with 60 DC-L1s, 48 L2s, and 24 memory channels. We observe that performance follows a similar trend to the evaluated 80-core system. Specifically, we gain significant IPC improvement of 67% for the replication-sensitive applications, and maintain the private performance for the replication-insensitive applications.

Boosted Baseline. We investigate various *boosted* baselines with 2× the per-core L1 cache capacity, 2× the NoC frequency, and 5× the flit size, respectively. For the replication-sensitive applications, we observe that these boosted baselines achieve performance improvement of 33%-36% normalized to the private L1 baseline. Such improvement is 22% lower compared to the 75% improvement under Sh40+C10+Boost. As for the replication-insensitive applications, the boosted baselines can improve their performance by 2%-6% compared to a slight <1% loss under Sh40+C10+Boost. However, these boosted baselines incur significant overheads. Specifically, using DSENT and CACTI, the cache-boosted baseline incurs a cache area overhead of 84%, and the flit-boosted baseline incurs a NoC area and static power overhead of 18.5× and 4.2×, respectively. As for the frequency-boosted baseline, the 80×32 crossbar cannot be operated using 2× the baseline frequency. Finally, our proposed designs are expected to improve performance with larger DC-L1s or boosted NoC resources.

IX. Related Work

In this section, we briefly discuss works that are most relevant to this study.

Intra-core Locality in GPUs. Prior works focused on exploiting the locality that exists within a private L1 cache [6], [7], [9], [29], [30]. In this work, we focus on the locality that exists across L1 caches. Other works proposed CTA schedulers [28], [31], [32] using different heuristics to exploit the locality across CTAs and improve cache performance. However, these schedulers are not ideal, and the problem of *uncontrolled* replication across L1 caches persists. Our proposed designs restrict replication to a preset limit (e.g., at most 10 copies with Sh40+C10+Boost) and do not require any software support. In general, prior L1 cache capacity management techniques [17], [32]–[34] do not control replication across L1s. However, these works can improve performance of each individual DC-L1, while our designs facilitate coordination across DC-L1s for their better utilization.

Inter-core Locality in GPUs. Prior works focused on improving the private L1 bandwidth utilization by exploiting inter-core locality and enabling inter-core communication. This was achieved by using a ring to connect the GPU cores [14] or coherence-like mechanisms [35]. Ibrahim *et al.* [3] optimized inter-core communication via data sharing prediction and parallel probing schemes. These works do not reduce replication across L1s. However, our designs reduce replication and eliminate the need for inter-core communication. Prior work [36], [37] proposed sharing an L1 data cache across a group of cores. This cache design is similar to the private DC-L1 cache design (Section IV) which suffers from high data replication compared to our design (Section VI). Zhao *et al.* [10] utilized inter-core locality to address bandwidth bottlenecks at L2 by replicating cache lines across different L2 slices. This work is complementary to our work as it targets the L2 bandwidth, while ours improves the L1 capacity and its bandwidth utilization.

Replication Control in CPUs. Prior CPU works investigated the trade-offs between shared and private cache design for the last-level caches [38]–[47]. These works focused on latency as it is often the first-order challenge in CPU workloads. However, to our knowledge, our work is the first to propose replication control and clustered shared decoupled L1 cache design in GPUs in order to boost on-chip bandwidth.

X. Conclusions

In this work, we showed that rethinking the cache hierarchy and interconnect design in GPUs can be rewarding in terms of performance, area, and energy. Specifically, we introduced the DC-L1 cache, an L1 cache decoupled from the GPU core to address the low bandwidth utilization of the L1s and the wasted L1 cache capacity due to cache line replication across the L1 caches. We used the DC-L1s and proposed a clustered-based DC-L1 cache organization, where a cluster of GPU cores access a cluster of shared DC-L1s. With a clustered shared cache organization, we eliminated data replication within each cluster and limited the overall replication in the GPU. Our designs improve the effective L1 cache capacity, which significantly boosts on-chip bandwidth and overall performance.

Acknowledgment

The authors thank the anonymous reviewers, Brad Benton, and members of the Insight Computer Architecture Lab at William & Mary for their feedback. This material is based upon work supported by the National Science Foundation (NSF) CAREER award (#1750667). This work was performed in part using computing facilities at William & Mary. ©2021 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

References

[1] W. A. Wulf and S. A. McKee, “Hitting the Memory Wall: Implications of the Obvious,” *ACM SIGARCH Computer Architecture News*, 1995.

[2] H. Wang, F. Luo, M. Ibrahim, O. Kayiran, and A. Jog, “Efficient and Fair Multi-programming in GPUs via Effective Bandwidth Management,” in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[3] M. A. Ibrahim, H. Liu, O. Kayiran, and A. Jog, “Analyzing and Leveraging Remote-core Bandwidth for Enhanced Performance in GPUs,” in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2019.

[4] H. Wang and A. Jog, “Exploiting Latency and Error Tolerance of GPGPU Applications for an Energy-Efficient DRAM,” in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2019.

[5] 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 International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2020.

[6] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-Conscious Wavefront Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2012.

[7] A. Jog, O. Kayiran, N. C. Nachiappan, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[8] A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Orchestrated Scheduling and Prefetching for GPGPUs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2013.

[9] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2013.

[10] X. Zhao, A. Adileh, Z. Yu, Z. Wang, A. Jaleel, and L. Eeckhout, "Adaptive Memory-Side Last-Level GPU Caching," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2019.

[11] H. Wang, M. Ibrahim, S. Mittal, and A. Jog, "Address-Stride Assisted Approximate Load Value Prediction in GPUs," in *Proceedings of the International Conference on Supercomputing (ICS)*, 2019.

[12] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking," *arXiv*, April 2018.

[13] A. Li, S. L. Song, W. Liu, X. Liu, A. Kumar, and H. Corporaal, "Locality-Aware CTA Clustering for Modern GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[14] S. Dublish, V. Nagarajan, and N. Topham, "Cooperative Caching for GPUs," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2016.

[15] A. Karki, C. P. Keshava, S. M. Shivakumar, J. Skow, G. M. Hegde, and H. Jeon, "Detailed Characterization of Deep Neural Networks on GPUs and FPGAs," in *Proceedings of the Workshop on General Purpose Processing Using GPU (GPGPU)*, 2019.

[16] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2020.

[17] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A Locality-aware Memory Hierarchy for Energy-efficient GPU Architectures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2013.

[18] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.

[19] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2012.

[20] X. Zhao, S. Ma, Z. Wang, N. E. Jerger, and L. Eeckhout, "CD-Xbar: A Converge-Diverge Crossbar Network for High-Performance GPUs," *IEEE Transactions on Computers (TC)*, 2019.

[21] A. M. Aji, M. Daga, and W.-c. Feng, "Bounding the Effect of Partition Camping in GPU Kernels," in *Proceedings of the International Conference on Computing Frontiers (CF)*, 2011.

[22] Hynix, "Hynix GDDR5 SGRAM Part H5GQ1H24AFR Revision 1.0," 2009. [Online]. Available: http://www.hynix.com/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf

[23] NVIDIA, "CUDA C/C++ SDK Code Samples," 2011. [Online]. Available: http://developer.nvidia.com/cuda-cc-sdk-code-samples

[24] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2009.

[25] 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 Workshop on General Purpose Processing Using GPU (GPGPU)*, 2010.

[26] L.-N. Pouchet, "Polybench: The Polyhedral Benchmark Suite," 2012. [Online]. Available: http://web.cs.ucla.edu/~pouchet/software/polybench/

[27] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2007.

[28] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[29] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Divergence-Aware Warp Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2013.

[30] D. Li, M. Rhu, D. R. Johnson, O. Mike, M. Erez, D. Burger, D. S. Fussell, and S. W. Redder, "Priority-Based Cache Allocation in Throughput Processors," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2015.

[31] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Improving GPGPU Resource Utilization Through Alternative Thread Block Scheduling," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2014.

[32] A. Tabbakh, M. Annavaram, and X. Qian, "Power Efficient Sharing-Aware GPU Data Management," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2017.

[33] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access Pattern-Aware Cache Management for Improving Data Utilization in GPU," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[34] A. Arunkumar, S. Lee, V. Soundararajan, and C. Wu, "LATTE-CC: Latency Tolerance Aware Adaptive Cache Compression Management for Energy Efficient GPUs," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[35] D. Tarjan and K. Skadron, "The Sharing Tracker: Using Ideas from Cache Coherence Hardware to Reduce Off-Chip Memory Traffic with Non-Coherent Caches," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2010.

[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 Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019.

[37] K. Choo, W. Panlener, and B. Jang, "Understanding and Optimizing GPU Cache Memory Performance for Compute Workloads," in *Proceedings of the International Symposium on Parallel and Distributed Computing (ISPDC)*, 2014.

[38] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2004.

[39] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2005.

[40] Z. Chishti, M. D. Powell, and T. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2005.

[41] B. M. Beckmann, M. R. Marty, and D. A. Wood, "ASR: Adaptive Selective Replication for CMP Caches," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2006.

[42] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006.

[43] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2007.

[44] M. K. Qureshi, "Adaptive Spill-Receive for Robust High-Performance Caching in CMPs," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2009.

[45] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-optimal Block Placement and Replication in Distributed Caches," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2009.

[46] G. Kurian, S. Devadas, and O. Khan, "Locality-Aware Data Replication in the Last-Level Cache," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2014.

[47] P.-A. Tsai, N. Beckmann, and D. Sanchez, "Nexus: A New Approach to Replication in Distributed Shared Caches," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2017.