# Selective Replication in Memory-Side GPU Caches

Xia Zhao
Academy of Military Science
China

Magnus Jahre
Norwegian University of Science and Technology
Norway

Lieven Eeckhout
Ghent University
Belgium

***Abstract*—Data-intensive applications put immense strain on the memory systems of Graphics Processing Units (GPUs). To cater to this need, GPU memory systems distribute requests across independent units to provide high bandwidth by servicing requests (mostly) in parallel. We find that this strategy breaks down for shared data structures because the shared Last-Level Cache (LLC) organization used by contemporary GPUs stores shared data in a single LLC slice. Shared data requests are hence serialized — resulting in data-intensive applications not being provided with the bandwidth they require. A private LLC organization can provide high bandwidth, but it is often undesirable since it significantly reduces the effective LLC capacity.**

**In this work, we propose the Selective Replication (SelRep) LLC which selectively replicates shared read-only data across LLC slices to improve bandwidth supply while ensuring that the LLC retains sufficient capacity to keep shared data cached. The compile-time component of SelRep LLC uses dataflow analysis to identify read-only shared data structures and uses a special-purpose load instruction for these accesses. The runtime component of SelRep LLC then monitors the caching behavior of these loads. Leveraging an analytical model, SelRep LLC chooses a replication degree that carefully balances the effective LLC bandwidth benefits of replication against its capacity cost. SelRep LLC consistently provides high performance to replication-sensitive applications across different data set sizes. More specifically, SelRep LLC improves performance by 19.7% and 11.1% on average (and up to 61.6% and 31.0%) compared to the shared LLC baseline and the state-of-the-art Adaptive LLC, respectively.**

## I. Introduction

Data-intensive applications such as machine learning are becoming critically important workloads across the computing spectrum. In data centers, Graphics Processing Units (GPUs) are commonly used to accelerate these and other workloads [5], [21], [46]. To improve performance, modern GPUs contain an ever-increasing number of Streaming Multiprocessors (SMs) supported by a highly-parallel, often die-stacked, high-bandwidth memory system. Unfortunately, the benefit of a highly parallel memory system is void if architectural policies cause serialization.

Data-intensive applications rely on shared data to globally coordinate the thousands of threads that are in flight at any given time. These global data structures are typically accessed frequently. Hence, they have high locality and tend to be cached in the GPU's highly-distributed Last Level Cache (LLC). Unfortunately, the shared LLC organization employed in contemporary GPUs creates a serialization bottleneck. More specifically, the requests for shared global data all go to the same LLC slice — serializing memory requests and creating significant congestion. Although LLC management has been studied extensively in the context of multi-core processors (e.g., [12], [13], [16], [19], [23], [25], [26], [37], [44], [63]), these approaches are not directly applicable to GPUs since they focus on reducing latency rather than increasing bandwidth.

One way of providing more bandwidth to shared data is to adopt a private LLC organization in which the requests from different (clusters of) SMs are routed to different LLC slices. Unfortunately, this significantly reduces the effective LLC capacity as copies of shared data are now stored in multiple LLC slices. The state-of-the-art Adaptive LLC [75] dynamically selects either a shared or private organization based on application behavior. Unfortunately, this all-or-nothing approach only addresses the LLC serialization problem when the shared data set is small enough to fit in the LLC with maximum replication. We find that a more fine-grained approach to shared data replication in GPUs is needed. The key challenge is to tune the replication degree such that bandwidth is maximized — by routing requests for shared data to different LLC slices — while protecting locality — to ensure that the shared data set remains cached.

Our Selective Replication (SelRep) LLC approach answers this call. SelRep LLC focuses on replicating shared read-only data since (i) read-only data is common in GPU-compute workloads (i.e., 97% of shared cache lines are read-only), and (ii) read-only data can be replicated without creating coherence issues. At compile time, SelRep LLC uses dataflow analysis to identify read-only shared data structures and replaces accesses to these structures with a special-purpose read-only load instruction. At run time, we leverage a novel light-weight hardware mechanism we call the Replication Degree Directory (RDD) to analyze the behavior of these loads. More specifically, the RDD concurrently predicts the LLC hit rates for all replication degrees. These hit rates are fed to an analytical model to predict and weigh the overall impact of the positive effect (i.e., improved bandwidth) versus the negative effect (i.e., more LLC misses) of increased replication. SelRep LLC's dynamic mechanism is extremely light-weight and requires only 692 bytes of storage.

Our evaluation shows that SelRep LLC effectively identifies appropriate replication degrees for our replication-sensitive applications across a broad range of data set sizes. In particular, SelRep LLC is within 2.3% (on average) of the performance that can be achieved by exhaustively searching through all static replication degrees and then selecting the top performer. Unlike the shared and adaptive LLC organizations, SelRep LLC consistently provides high performance across a broad range

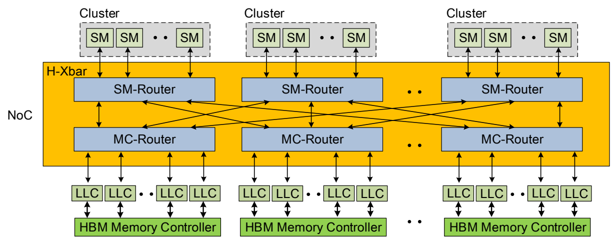


**Fig. 1: GPU architecture with a hierarchical crossbar (H-Xbar) NoC.** *To reduce the hardware overhead of the NoC, the SMs (LLC slices) are clustered, and the SMs (MCs) of a cluster share an SM-router (MC-router). All SM-routers have direct links to all MC-routers.*

of data sets for our replication-sensitive applications. More specifically, SelRep LLC improves performance by 19.7% and 11.1% on average (and up to 61.6% and 31.0%) compared to the baseline shared LLC organization and state-of-the-art Adaptive LLC [75], respectively.

In summary, we make the following major contributions:

- We show, for the first time, that selective replication enables making fine-grained trade-offs between bandwidth to shared read-only data and LLC capacity.
- We propose a light-weight compiler pass that uses dataflow analysis to identify shared read-only data structures and communicate this information to hardware through a special-purpose load instruction.
- We devise a light-weight dynamic measurement mechanism we call the Replication Degree Directory (RDD) which we combine with an analytical bandwidth model to identify the most appropriate replication degree while only requiring 692 bytes of storage.
- SelRep LLC enables robust selective replication of shared read-only data across a broad range of input sets for our replication-sensitive applications. More specifically, SelRep LLC improves performance by 19.7% and 11.1% on average (and up to 61.6% and 31.0%) compared to the shared LLC baseline and state-of-the-art Adaptive LLC [75], respectively.

## II. Background and Motivation

In this section, we provide the necessary background for understanding our SelRep LLC scheme. We first provide background on the Network-on-Chip (NoC) and Last-Level Cache (LLC) subsystems of high-performance GPUs and then describe how these systems can be modified to enable adaptively switching between shared and private LLC organizations. This discussion lays the foundation for describing our SelRep LLC approach in Section III.

**GPU Architecture.** We first provide some background on the high-level architecture of the memory systems of contemporary GPUs. Since the GPU programming model is highly latency-tolerant, the main objective is to maximize bandwidth. This is achieved by creating a highly parallel memory system in which the SMs are connected to a large number of LLC slices through a NoC (see Figure 1). To increase bandwidth, the LLC is divided into a number of equally-sized slices that each share a memory controller.

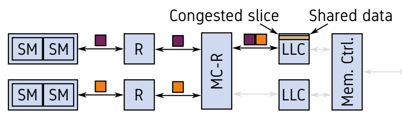
Figure 1 also shows the internal structure of the NoC. Since contemporary GPUs feature a large number of SMs, a fully connected crossbar is not attractive due to its poor scalability. More specifically, the hardware cost of the full crossbar increases quadratically with the number of ports. Prior work [74], [75] observed that a hierarchical crossbar (H-Xbar) scales better than a full crossbar in terms of hardware complexity and power efficiency while providing the same bisection bandwidth. The H-Xbar achieves this through a two-level router structure, i.e., the SM-router and the MC-router in Figure 1, in which each router is shared by clusters of SMs and LLC slices, respectively. Each SM-router has a direct link to all MC-routers. Thus, each request coming from an SM has to traverse two router hops in H-Xbar compared to a single router hop in a fully connected crossbar.

Each Memory Controller (MC) has a number of associated LLC slices to cache the data from the memory space provided by that MC. The memory accesses of the SMs are routed to the different LLC slices and MCs based on the memory address. In this work, we use the recently proposed PAE address mapping which evenly distributes memory requests to different addresses across LLC slices and memory controllers to maximize parallelism in the memory subsystem [42].

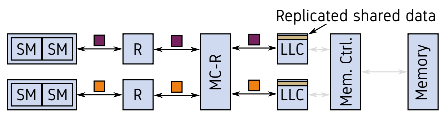
**Shared versus Private LLC Organization.** Figure 2 illustrates the different ways in which the LLC slices can be organized. The most common organization is that LLC capacity is shared among SMs since this maximizes LLC utilization, i.e., each cache block is stored in at most one LLC slice. Unfortunately, the shared LLC organization may be suboptimal for read-only shared data (see Figure 2a). The problem is that locating frequently accessed shared data in one LLC slice concentrates memory traffic into this LLC slice, thus creating a bandwidth bottleneck.

Figure 2b illustrates that a private cache organization can improve bandwidth to read-only shared data by replicating this data across LLC slices. In the shared LLC, the MC-router determines the destination of memory requests based on a subset of the memory address bits. To implement a private LLC organization, this policy is simply changed to selecting the LLC slice based on the SM-cluster from which the memory request originates. Figure 3 illustrates this organization and shows that the requests of each SM cluster is routed to a separate LLC slice. In H-Xbar, a private organization is straightforwardly implemented by selecting LLC slices based on the SM-cluster the request was sent from [75].

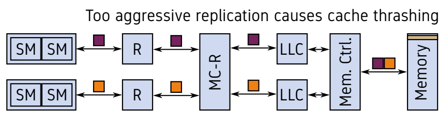
A private LLC organization increases the effective LLC bandwidth at the cost of decreasing the effective cache capacity. Figure 2c illustrates the case where replication decreases the effective cache capacity to the extent that the shared data set no longer fits in the LLC. Thus, performance is significantly reduced compared to a shared organization since memory bandwidth, which is much lower than LLC bandwidth, becomes the limiting factor.



(a) Shared LLC.



(b) Private LLC with small data set.



(c) Private LLC with large data set.

**Fig. 2: Example illustrating the need for intelligent replication of shared data.** *In a shared LLC, SMs tend to access shared read-only data in the same LLC slice around the same time which causes congestion. A private LLC organization can improve performance by distributing requests for shared read-only data across LLC slices, but replicating too aggressively destroys locality and incurs a significant performance penalty.*

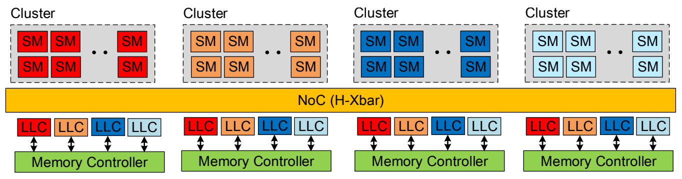


**Fig. 3: Private LLC configuration.** *SMs are divided into clusters; cluster count equals the number of LLC slices per memory controller. The colors illustrate LLC-slice to SM-cluster allocation (e.g., red SMs access red LLC slices).*

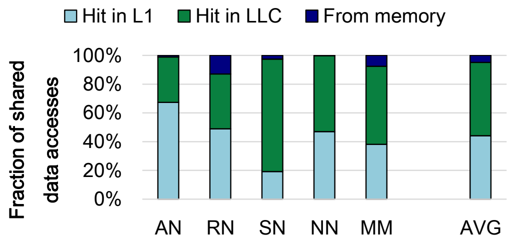


**Fig. 4: Shared data set memory access breakdown.** *The shared data set typically exceeds the L1 cache capacity and needs to be fetched from the LLC.*

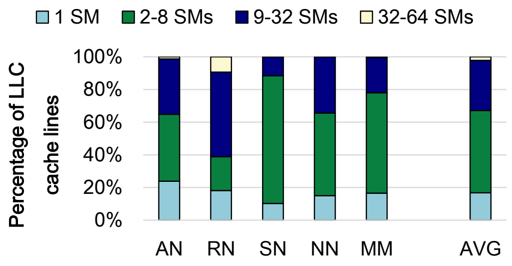


**Fig. 5: Accesses to read-only cache lines from different SMs within a 1000-cycle time window.** *On average, over 80% of the LLC cache lines are accessed by at least 2 SMs within the 1000-cycle time window.*

**Adaptive LLC.** The recently proposed Adaptive LLC [75] dynamically chooses *either* a shared or a private LLC organization using a coarse-grained approach, i.e., the shared data set is either fully replicated across all slices (a private LLC organization) or there is no replication at all (a shared LLC organization). For a small shared data set size, it is beneficial to replicate the shared data set across all LLC slices to maximize the effective LLC bandwidth — as achieved under a private LLC organization. At the other end of the spectrum, i.e., for a large shared data set, it is best to not replicate to avoid cache thrashing — as achieved under a shared LLC organization. The Adaptive LLC dynamically chooses the best performing LLC organization for the given workload at hand. By doing so, it achieves the best of both worlds: the Adaptive LLC reverts to a private LLC when the shared data set is small and reverts to a shared LLC when the shared data set is large. This strategy enables the Adaptive LLC to dynamically adopt the best performing LLC organization.

**Towards Selective Replication.** Unfortunately, the Adaptive LLC leaves significant performance on the table. The fundamental reason is that the Adaptive LLC is too coarse-grained, i.e., it is an all-or-nothing approach. Ideally, one would want to selectively replicate data to carefully balance the effective LLC capacity and bandwidth as a function of the shared data set size. It is expected that selective replication performs equally good as a private LLC for small shared data sets. On the other hand, it is expected that selective replication performance is equally good as a shared LLC for large data sets. However, for the important range of medium-sized shared data sets, selective replication should significantly outperform either approach. In other words, *selective replication should be robust across data set sizes*.

**Shared Data Set Characteristics.** We now characterize the shared data set to further support selective replication in the LLC. We further find that the shared data set is too large to fit in the L1 caches. More specifically, more than half the shared data set is accessed in the LLC, by 50.9% on average and up to 78.2% for SN (see Figure 4). (See Section IV for details regarding the experimental setup.) Moreover, we find that multiple SMs access the same cache lines in the LLC *around the same time*. In particular, we find that 83.3% of the read-only cache lines in the LLC are accessed by more than 2 SMs within a 1000-cycle time window, and 32.8% of the LLC cache lines are accessed by more than 9 SMs (see Figure 5). The fact that the majority of the shared data set is located in the LLC, combined with the fact that multiple SMs access the same shared data elements around the same time, leads to intermittent request camping in LLC slices. Selectively replicating shared read-only cache lines across LLC slices alleviates the camping problem and improves overall performance by increasing the

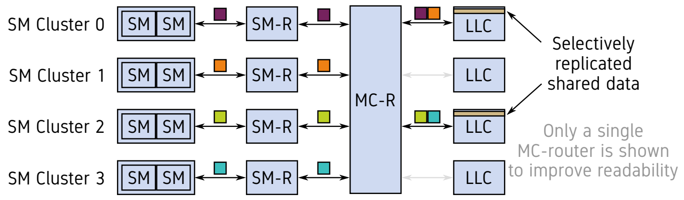

SM Cluster 0
SM Cluster 1
SM Cluster 2
SM Cluster 3
SM
SM-R
MC-R
LLC
Selectively replicated shared data
Only a single MC-router is shown to improve readability

**Fig. 6: Selective replication example.** *SelRep LLC improves performance compared to a shared and private LLC by selectively replicating shared data across LLC slices to improve bandwidth while simultaneously ensuring that shared data remains cached.*

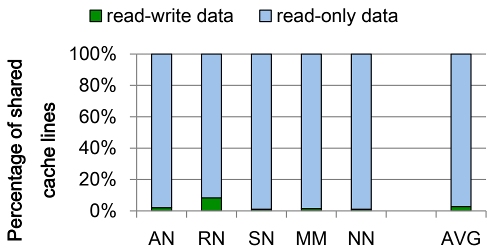


**Fig. 7: Fraction of read-only versus read-write shared cache lines.** *The key take-away is that more than 97% of the shared cache lines are read-only on average.*

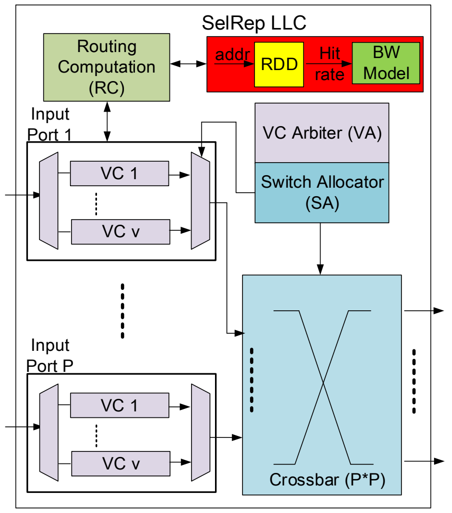


**Fig. 8: A block diagram of an MC-router that supports SelRep LLC.** *SelRep LLC minorly modifies the router by adding the RDD and bandwidth model logic (both structures are off the critical path).*

effective LLC bandwidth.

## III. Enabling Selective Replication

We now describe our SelRep LLC approach which addresses the shortcomings of the current state-of-the-art LLC management schemes by selectively replicating read-only shared data across LLC slices to maximize bandwidth while avoiding LLC thrashing. Figure 6 illustrates selective replication in which a cache line that contains shared read-only data elements is replicated across 2 of the 4 LLC slices. SM clusters 0 and 1 access the shared data structure in the first LLC slice, whereas SM clusters 2 and 3 access the third LLC slice. Selective replication increases the effective LLC bandwidth while balancing LLC miss rate.

We first provide a general overview of SelRep LLC in Section III-A before we describe in detail how SelRep LLC predicts LLC misses and performance across different LLC organizations in Section III-B and III-C, respectively. We then discuss the potential overheads of SelRep LLC in Section III-D before we illustrate how we can leverage its compiler support to simplify the coherence protocol and support for atomic operations compared to prior adaptive and private LLC organizations in Section III-E.

### *A. SelRep LLC Overview*

**Compile-time analysis.** SelRep LLC's compiler analysis marks data structures as read-only within a kernel boundary. We accomplish this by using data flow analysis at the level of the PTX intermediate representation [50] to identify all accesses to each data structure and then determine if they are reads or writes. A data structure is read-only within a particular kernel if its elements are never written within its boundaries. Otherwise, we mark the data structure as read-write. Note that a data structure that is read-only in one kernel may be read-write in a different kernel (e.g., one kernel writes the input data for a subsequent kernel to read).

After identifying read-only data structures, the compiler replaces its loads of global data (i.e., `ld.global` instructions) with a special load read-only instruction (i.e., `ld.global.ro`). To exploit this information at runtime, we add a read-only bit to the metadata of memory requests. Thus, SelRep LLC identifies the memory requests that are eligible for replication by inspecting this metadata bit. This extra bit does not add overhead because a read request only needs to transmit the address to be fetched while a write request needs to transmit both the address and the data. In other words, a read request only uses a few of the wires in the request links (i.e., 8 bytes out of 32 bytes in our setup).

The style in which GPU-compute applications are programmed work in favor of SelRep LLC. First, GPU kernels often have regular memory access behavior which makes dataflow analysis straightforward. Second, shared read-only data structures are typically used for kernel inputs and are hence common. Figure 7 supports this observation for the replication-sensitive benchmarks in this study, i.e., on average more than 97% of the shared cache lines are read-only. Third, aliasing is not common [41]. In fact, our dataflow analysis correctly identifies all read-only data structures within the benchmarks we consider in this work.

**Run-time support.** Figure 8 shows that SelRep LLC can be implemented with minor modifications to the MC-router. The Replication Degree Directory (RDD) predicts if a read-only memory request will hit in the LLC with different degrees of replication as well as a shared and private organization. The memory address is available within the Routing Computation (RC) unit, and we forward the address of the request to the

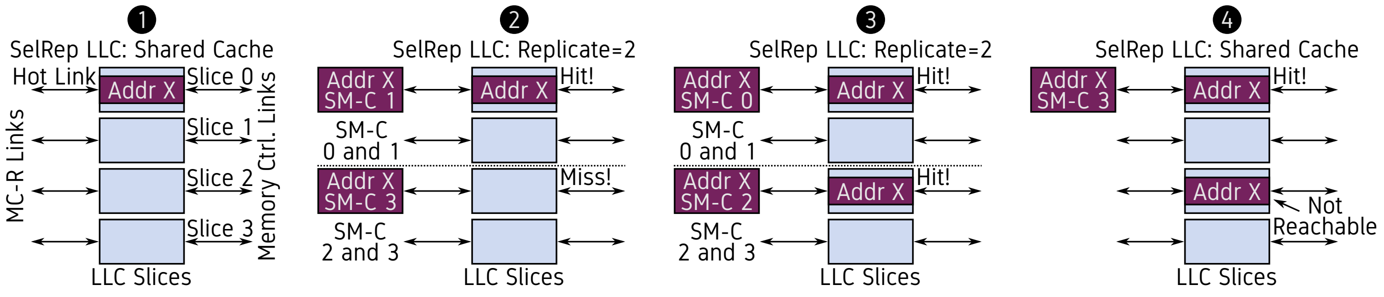


**Fig. 9: Example illustrating SelRep LLC's run-time operation.** *SelRep LLC seamlessly adapts to new LLC organizations by lazily and selectively replicating cache blocks.*

RDD if the read-only bit is set.

SelRep LLC divides time into fixed-length epochs (e.g., 20 K clock cycles) and re-evaluates the policy's selection at epoch boundaries. At the end of an epoch, hit rates for all replication degrees are retrieved from the RDD and provided to the bandwidth model (Section III-B). SelRep LLC uses an analytical bandwidth model to predict which LLC organization maximizes bandwidth (Section III-C). This configuration is then selected for the next epoch. Since memory-intensive GPU applications are typically sensitive to bandwidth, improving bandwidth tends to increase performance.

**Run-time example.** Figure 9 drills down into the LLC behavior when SelRep LLC dynamically adjusts the replication degree. Similarly to Figure 6, we assume 4 SM clusters and 4 LLC slices. To limit the complexity of the example, we focus on the LLC slices of a single memory controller.

Initially (see ❶), SelRep LLC has chosen a shared cache organization, and cache line X is stored in LLC slice 0. SelRep LLC then chooses an LLC organization with replication degree 2 (see ❷). This means that requests from SM-cluster 0 and 1 (2 and 3) are routed to LLC slice 0 and 1 (2 and 3). At this point, SM-cluster 1 and SM-cluster 3 both issue memory requests for cache line X. The request from SM-cluster 1 is routed to LLC slice 0 which results in a cache hit. SM-cluster 3's request on the other hand is now routed to LLC slice 2. This LLC slice does not currently contain cache line X which results in a cache miss. The LLC services this miss as it would with any other miss and thereby installs a replica of cache line X in LLC slice 2 (see ❸).

SM-clusters 0 and 2 now access cache line X (see ❸), and both requests hit in LLC slice 0 and 2, respectively. This alleviates congestion in LLC slice 0 as requests for line X are now distributed across slice 0 and 2. Finally, SelRep LLC decides to revert to the shared LLC organization (see ❹). All requests for cache line X are now routed to LLC slice 0 (see the cache hit request from SM-cluster 3). The replica in LLC slice 2 is no longer reachable and will eventually be evicted. By relying on the regular insertion and replacement policies of the LLC, SelRep LLC quickly adjusts to new cache organizations while limiting complexity and overheads.

## B. Predicting LLC Misses

To predict the LLC misses for all possible replication degrees we devise a light-weight mechanism we call the Replication Degree Directory (RDD). The RDD is inspired by the Auxiliary Tag Directory (ATD) [55] which is an independent tag directory commonly used to predict per-application cache misses as a function of allocated ways in shared LLCs (see e.g., [27], [76]). Unlike an ATD, the RRD is (i) located within an MC-router rather than an LLC slice, and (ii) predicts misses across replication degrees and not miss curves. The RDD contains a collection of bit vectors which we call ClusterID along with each tag to predict the hit rates for different replication degrees (see Figure 10). There is up to one bit for each SM-cluster in the ClusterID structure. When the tag is first brought into the RDD, the bits that represent the SM-cluster that accessed the tag is set and all others are zero. For each subsequent access, the bits representing the SM-cluster that accessed the cache block is set to 1.

For each RDD access, we determine if it would hit or miss in the LLC for a particular cache organization. For the shared LLC, the request is a hit if the tag matches. The other extreme is the private cache in which the tag needs to match and the memory access must come from the same SM-cluster. Hence, the ClusterID vector needs as many bits as there are SM-clusters. Selective replication works similarly to the private cache except that each SM-cluster can access more than one LLC slice (i.e., there are fewer bits in the ClusterID vectors). The RDD counts the number of accesses and keeps a separate hit counter for each policy and can thus predict the LLC hit rates of all policies concurrently.

We use Figure 10 to describe the operation of the RDD. The example RDD samples two sets and has the same associativity as the LLC slices (16 in the example). We further assume that there are 8 SM clusters and 8 LLC-slices per MC-router. Thus, the ClusterID vector for the private LLC configuration contains 8 bits. A request from SM-cluster 3 with a matching tag will be a hit in the private LLC organization in the example since ClusterID bit 3 is set to 1. A request from SM-cluster 2 with a matching tag will be a miss in the private LLC organization, but a hit with replication degree 4 (2) since the bits for clusters 2 and 3 (0 to 3) are set to 1.

Adaptive LLC [75] also uses a hardware mechanism to select a private or shared LLC organization. Unlike SelRep LLC, it is not able to predict LLC hit rates for selective replication. Further, it can only predict LLC hit rates when the shared LLC organization is used. The reason is that Adaptive

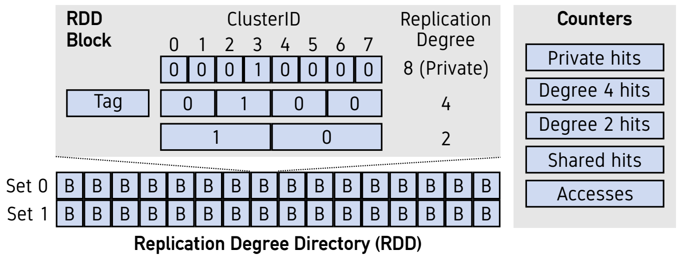


**Fig. 10: The Replication Degree Directory (RDD).**

LLC locates its predictor in an LLC slice which results in it only seeing the accesses of a single SM-cluster in the private configuration. Thus, Adaptive LLC needs to return to the shared LLC organization periodically to account for phase behavior. SelRep LLC avoids this problem by placing the RDD in an MC-router which enables it to monitor all accesses regardless of LLC organization.

### C. Predicting Performance

We now describe the analytical bandwidth model that SelRep LLC uses to dynamically select the replication degree. We first compute the LLC Slice Parallelism (LSP):

$$\mathrm{LSP} = \sum_{i=0}^{S-1} \frac{R_i}{\max(R_0, R_1, \ldots, R_{S-1})}. \quad (1)$$

LSP is the sum of the requests $R_i$ to slice $i$ divided by the maximum number of requests to any cache slice; there are $S$ LLC slices in total. LSP expresses how balanced the read-only shared LLC requests are. More specifically, LSP equals $S$ if the requests are evenly distributed, and LSP equals 1 if all requests go to a single LLC slice.

The total effective bandwidth $B$ is a function of the LSP and the bandwidth each LLC slice can provide. In turn, the bandwidth offered by an LLC slice depends on its hit rate $H$:

$$B = \mathrm{LSP} \times (H \times B_{\mathrm{LLC}} + Min\{(1-H) \times B_{\mathrm{LLC}}, B_{\mathrm{mem}}\}). \quad (2)$$

More specifically, the effective bandwidth is the sum of the bandwidth provided to LLC hits (i.e., $H \times B_{\mathrm{LLC}}$ where $B_{\mathrm{LLC}}$ is the maximum bandwidth of each LLC slice) and LLC misses. If the bandwidth demand of the LLC misses is less than the memory bandwidth, the memory system will be able to hide the LLC miss impact. Thus, the LLC misses are effectively receiving the same bandwidth as the LLC hits. Otherwise, the miss bandwidth is determined by the LLC slice's proportional share of the total DRAM bandwidth $B_{\mathrm{mem}}$.

Equation 2 precisely captures the bandwidth versus LLC capacity trade-off that is central to effectively leveraging selective replication. More specifically, increasing the replication degree increases LSP and $R_{\mathrm{miss}}$, and Equation 2 enables SelRep LLC to intelligently balance these conflicting effects.

At runtime, SelRep LLC computes Equation 1 and 2 for all sharing degrees at the end of each epoch. To accomplish this, SelRep LLC maintains counters for the number of requests to each LLC slice (i.e., $R_i$ in Equation 1). In addition, the RDD supplies the predicted number of hits (i.e., $R_{\mathrm{hit}}$) for all possible replication degrees. $B_{\mathrm{LLC}}$ and $B_{\mathrm{mem}}$ are architectural constants and $R_{\mathrm{miss}}$ can be computed from the predicted hits and accesses. Finally, SelRep LLC finds the replication degree with the highest predicted bandwidth $B$ and instructs the MC-routers to use this replication degree in the next epoch.

The replication selection procedure is off the critical path. Thus, application execution continues with the old replication degree while SelRep LLC determines the (new) replication degree. Regardless, the latency of computing the bandwidth model is low because the number of LLC slices per MC-router and replication degrees are limited. More specifically, the GPU we model in this work has 16 LLC slices per MC-router (i.e., $S$ is 16 in Equation 1), and we consider 5 replication degrees (i.e., Equation 2 is computed 5 times).

### D. Overheads

**Hardware Overhead.** SelRep LLC is lean. To limit the overheads, we only add the RDD to a single MC-router and apply set sampling aggressively. More specifically, the RDD contains only 2 sets with 16 ways each, which results in an overall storage cost of 520 bytes. In addition, SelRep LLC needs to count the number of requests to each LLC slice as well as the hits and accesses for the different LLC organizations. We find that 16-bit counters are sufficient, which adds a storage overhead of 172 bytes. The total storage overhead of the RDD and the analytical model is hence 692 bytes.

Enabling replication also requires increasing the size of the LLC tags since address bits that are constant in the shared LLC case (as they are used for slice selection) can change in the private case. Assuming a 30-bit physical address [42], a 4-bit larger LLC tag increases the storage overhead of our default LLC by less than 0.4% because the tags are small compared to the data. Since this overhead is due to enabling replication, it affects Adaptive LLC [75] and SelRep LLC equally.

**Routing Latency.** SelRep LLC potentially re-routes memory requests to different LLC slices. Thus, it needs to consider the SM-cluster the requests was sent from, the current replication degree, and the memory address during the routing computation. In contrast, the baseline shared LLC organization only considers the memory address. SelRep LLC may therefore increase the latency of the routing computation. However, the virtual channel or switch allocation is usually the critical path of the router [35]. Since the logic required to perform SelRep LLC's routing computation is simple, we do not expect that it increases the latency enough for it to become the critical path. Regardless, we show that SelRep LLC performs well across a range of router latencies in Section V-C.

RDD access and bandwidth model computation is off the critical path. More specifically, the bandwidth model reads the RDD hit and access counters once every epoch (e.g., 20 K cycles). Then, it computes the most appropriate replication degree for the next epoch which is finally communicated to the MC-routers. The new LLC organization takes effect when this information has reached all routers, typically after a few tens of clock cycles. As SelRep LLC replicates cache blocks lazily, it can easily tolerate this latency.

**DRAM Organization.** All SMs must be able to access the whole memory address space regardless of the LLC organization. To enable this, each LLC slice needs to be given access to all channels managed by the respective memory controller. For the recent commercial stacked High-Bandwidth Memory (HBM) considered in our work [70], we therefore need two crossbars (one for either direction) per memory controller that connects all LLC slices to all respective memory channels. If the memory controller does not contain such a crossbar, the area overhead of adding it is limited. We use DSENT [61] to estimate the active silicon area of the crossbar in a 22 nm technology node. We account for the area of repeaters for long links which we assume are 12.3 mm long (half of the Pascal die length [48]). The links themselves are routed in the higher metal layers and hence consume global wire area. We find that the overall cost of the crossbars and all long links are 1.01 $\text{mm}^2$ and 0.42 $\text{mm}^2$, respectively, which is approximately 0.2% of the GPU die area.

### *E. Cache Coherence Implications*

While many GPUs only cache read-only data in the L1, recent GPUs such as the Volta cache all data and adopt a write-through policy [49]. In this case, cache coherence is typically ensured by requiring that software flushes the L1 cache when necessary [8], [45]. A conventional LLC does not need to be flushed as it is shared between all SMs. This approach does not work for prior private or adaptive LLC schemes (e.g., Adaptive LLC [75]) as dirty copies of the same cache block may exist in different LLC slices. Previous work addresses this issue by making the LLC write-through as well as flushing the LLC on synchronization boundaries (e.g., sync instructions or kernel boundaries). This increases system complexity and results in unnecessary writes to memory.

SelRep LLC's compiler support guarantees that we only replicate read-only data structures. Hence, dirty replicates do not exist, and we can simplify cache coherence support compared to prior work. In fact, SelRep LLC only needs to flush the LLC on kernel boundaries as a data structure which is read-only in one kernel might be read-write in a different kernel (we account for this overhead in our evaluation).

Atomic instructions also need special consideration in prior private and adaptive LLC schemes. These instructions are typically implemented within the raster operations unit [2], [20] and rely on all instructions accessing a common memory location. To enable this, prior schemes contain an additional (small) shared LLC for atomic instructions. In SelRep LLC, the shared cache organization is used for all data that is not explicitly marked as shared and read-only. Thus, atomic operations work seamlessly within SelRep LLC.

## IV. Experimental Setup

**Simulated System.** We evaluate SelRep LLC through cycle-level simulation and integrate GPGPU-sim [11] with Ramulator [36] to model a high-end GPU system attached to an HBM memory subsystem. The baseline GPU configuration is listed in Table I. We consider a GPU with 64 SMs and a 4 MB LLC, which is similar to previous work and commercial GPUs [6], [7], [8], [45], [48]. We assume 4 HBM stacks with eight memory channels each, for a total of 32 memory channels and an aggregate memory bandwidth of 600 GB/s. We further assume two LLC slices per channel, and a total number of 64 LLC slices or 16 LLC slices per HBM stack. We use the state-of-the-art PAE randomized address mapping scheme to uniformly distribute memory accesses across LLC slices, memory channels, and banks [42]. We further assume a typical cache line size of 128 B. We find that a larger cache line size exacerbates the LLC contention problem as the number of sharers increases. The H-Xbar NoC consists of 16 SM-routers, connecting to a cluster of 4 SMs each, and 4 MC-routers. The SM- and MC-routers have a 4-stage pipeline. We adopt two-level round-robin (2L-RR) CTA scheduling to balance the workload across the different SM-routers, i.e., 2L-RR first distributes CTAs across SM-routers and then across SMs within an SM-router [74], [75]. Other CTA scheduling policies are explored in the sensitivity analysis.

**TABLE I: Baseline GPU architecture.**

| Parameter | Value |
|---|---|
| Streaming Multiprocessors | 64 SMs, 1400 MHz |
| Schedulers/Core | 2 (GTO) |
| Number of Threads/Core | 2,048 |
| Registers/Core | 65,536 |
| Shared Memory/Core | 64 KB |
| L1 Data Cache/Core | 48 KB, 6-way, LRU, 128 B line |
| LLC | 4 MB in total (64 slices, 16-way), 120 clock cycles latency |
| Interconnection Network | H-Xbar, 32 B channel width<br>16 SM-routers, 4 MC-routers<br>4 VC per port – 8 flits/VC<br>VC/Switch allocator – Islip |
| HBM Memory | 4 memory stacks, 8 channels/stack, open-page policy, FR-FCFS, 16 banks/channel, 600 GB/s |
| HBM Timing | tRC=24, tRCD=7, tRP=7, tCL=7, tWL=2, tRAS=17, tRRDl=5, tRRDs=4, tFAW=20, tRTP=7, tCCDl=1, tCCDs=1 |

Compiler support for SelRep LLC analyzes PTX, CUDA's intermediate code representation, which is compiled by `nvcc` v4.0 [47]. The compiler pass replaces `ld.global` instructions that operate on read-only data structures — read-only data structures are identified through dataflow analysis — with `ld.global.ro` instructions, our instruction set extension. We use GPUWattch [40] to evaluate the GPU's power consumption assuming a 22 nm technology node; we model SelRep LLC's hardware extensions and account for their impact on GPU power. We use DSENT [61] to estimate NoC power, and assume long links to connect SM-routers and MC-routers, with a long link length measuring 12.3 mm (half the Pascal die size [48]).[1] Activity factors from Ramulator are used as input to GPUWattch to estimate overall system (GPU plus memory) power.

[1]The long link length is a conservative estimate which can be reduced through optimized floorplanning [14], [18], [34].

**TABLE II: Replication-sensitive GPU-compute benchmarks with their configurations from small to large.**

| Benchmark | Configurations (from small to large) | | | | |
|---|---|---|---|---|---|
| **AN** | *an_0* | *an_1* | *an_2* | *an_3* | *an_4* |
| *Shared Data [MB]* | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| *LLC Size [MB]* | 4 | 1 | 0.5 | 0.25 | 0.125 |
| **RN** | *rn_0* | *rn_1* | *rn_2* | *rn_3* | *rn_4* |
| *Shared Data [MB]* | 4.2 | 4.2 | 4.2 | 4.2 | 4.2 |
| *LLC size [MB]* | 4 | 1 | 0.5 | 0.25 | 0.125 |
| **SN** | *sn_0* | *sn_1* | *sn_2* | *sn_3* | *sn_4* |
| *Shared Data [MB]* | 0.7 | 0.7 | 0.7 | 0.7 | 0.7 |
| *LLC size [MB]* | 8 | 6 | 4 | 2 | 1 |
| **NN** | *nn_0* | *nn_1* | *nn_2* | *nn_3* | *nn_4* |
| *Shared Data [MB]* | 0.6 | 2.8 | 5.7 | 11.4 | 22.8 |
| *LLC size [MB]* | 4 | 4 | 4 | 4 | 4 |
| **MM** | *mm_0* | *mm_1* | *mm_2* | *mm_3* | *mm_4* |
| *Shared Data [MB]* | 0.04 | 0.1 | 0.6 | 1.9 | 3.8 |
| *LLC size [MB]* | 4 | 4 | 4 | 4 | 4 |

**TABLE III: Non-replication-sensitive GPU benchmarks.**

| Benchmark | Abbr. | Shared Data [MB] | Preferred LLC |
|---|---|---|---|
| LU Decomposition [17] | LUD | 33.4 | shared |
| Survey Propagation [15] | SP | 17.0 | shared |
| 3D Convolution [22] | 3DC | 51.1 | shared |
| B+TREE Search [17] | BT | 13.7 | shared |
| GEMM [22] | GEMM | 1.8 | shared |
| Backprop [17] | BP | 18.8 | shared |
| BlackScholes [51] | BS | 0.001 | neutral |
| DWT2D [17] | DWT2D | 0.001 | neutral |
| Merge Sort [51] | MS | 0.001 | neutral |
| BinomialOptions [51] | BINO | 0.017 | neutral |
| Histogram [51] | HG | 0.003 | neutral |
| Vector Add [51] | VA | 0.001 | neutral |

**Workloads.** We consider all benchmarks used in recent prior work [75], including both replication-sensitive workloads as well as non-replication-intensive workloads. Table II lists the five replication-sensitive benchmarks: AlexNet (AN) [1], ResNet (RN) [1], SqueezeNet (SN) [1], Neural Network (NN) [11] and Matrix Multiply (MM) [51]. We consider five *configurations* in which we vary the size of the shared data set relative to the LLC size from *small* to *large* — this is to show that SelRep LLC is robust across data set sizes. We vary the input data set for the latter two benchmarks, NN and MM, while keeping the LLC size constant at 4 MB. The shared data set size varies between 600 KB and 22.8 MB for NN, and between 40 KB and 3.8 MB for MM. It is impossible to vary the input data set size for the other three benchmarks, AN, RN and SN — the input data set is hard-coded in the benchmark — nevertheless, these workloads will be executed with different input data set sizes in production. We therefore take a different approach for these benchmarks: instead of varying the input data set size, we take a dual approach and vary the LLC size. In particular, the shared data set size is fixed for these benchmarks (1 MB for AN, 4.2 MB for RN, and 0.7 MB for SN), and we vary the LLC size from 4 MB to 128 KB for AN and RN, and from 8 MB to 1 MB for SN.

The non-replication-intensive benchmarks are listed in Table III. These benchmarks are insensitive to (selective) data replication, yet we include these workloads in the analysis to demonstrate that SelRep LLC is performance-neutral for such workloads. We identify two categories: workloads with a large shared data set (but low sharing degree) — these workloads prefer a shared LLC organization because data replication dramatically increases the LLC miss rate while providing limited bandwidth benefits — versus workloads with a very small shared data set — these workloads are neutral to the LLC organization, i.e., a shared and private LLC perform equally well for these workloads.

## V. Evaluation

We now evaluate SelRep LLC quantitatively and consider the following LLC organizations:

- **Shared LLC.** This is the standard shared LLC organization, i.e., each LLC slice can be accessed by all SM clusters. Shared data appears in at most one LLC slice.
- **Private LLC.** This is a private LLC organization in which an LLC slice is private to an SM cluster. Shared data appears in multiple LLC slices if accessed by multiple SM clusters.
- **Adaptive LLC.** This is the state-of-the-art adaptive LLC organization [75] which dynamically chooses between a shared versus private LLC. Adaptive LLC is an all-or-nothing approach, i.e., the data is either replicated across all LLC slices or it is not replicated at all.
- **SelRep LLC.** This is the LLC organization proposed in this work which determines the replication degree by balancing LLC capacity and bandwidth.
- **Best Static Replication (BSR).** This is a best-static approach in which we determine the best replication degree through offline profiling. We evaluate all power-of-two replication degrees (0, 2, 4, 8 and 16) and select the best one on a per-kernel, per-configuration basis. Assuming the same configuration for profiling and evaluation is unrealistic, hence BSR only serves as a point of comparison.

### *A. Performance*

We first evaluate SelRep LLC's impact on performance. Figure 11 compares the Shared, Private, Adaptive and SelRep LLC organizations against the Best Static Replication (BSR) approach, for each of the five replication-sensitive benchmarks across all five configurations; the bottom-right figure reports average performance. The Private LLC organization achieves the best performance for the small configuration: because the shared data set is small compared to the LLC size, replicating it across LLC slices substantially increases the achievable LLC bandwidth without overly penalizing the LLC miss rate. In contrast, the Shared LLC organization achieves the best performance for the large configurations: replicating a large shared data set increases pressure on LLC capacity leading to increased LLC miss rates, which offsets the increased LLC bandwidth potential. Adaptive LLC achieves the best of both

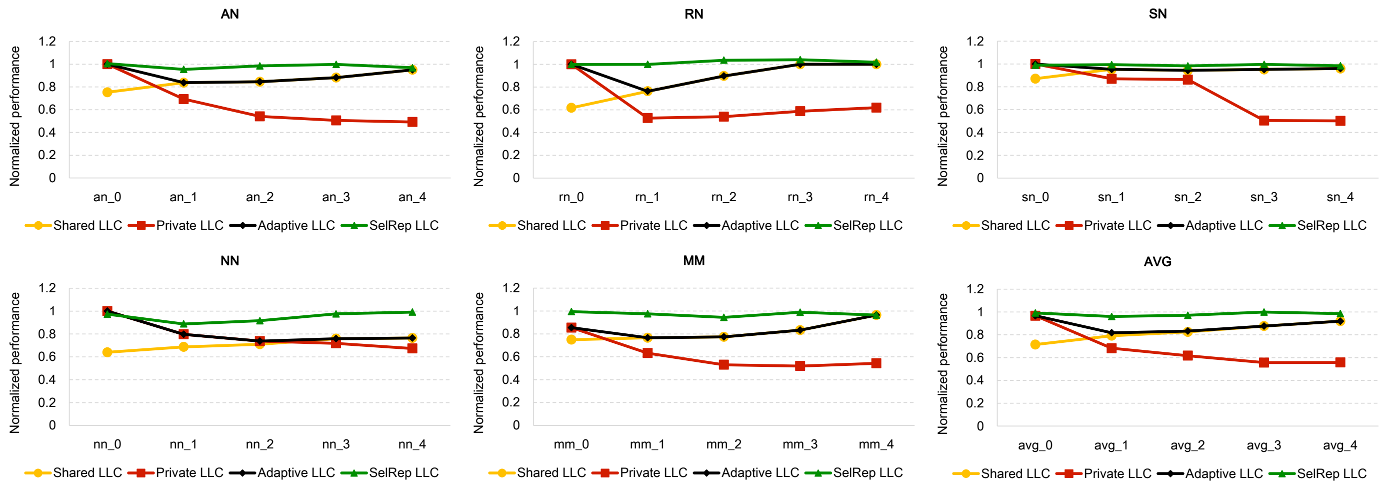


**Fig. 11: Performance normalized to Best Static Replication (BSR) for the Shared LLC, Private LLC, Adaptive LLC and SelRep LLC organizations.** *SelRep LLC approaches BSR across all configurations.*

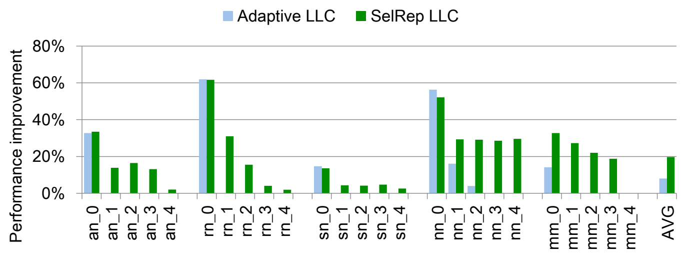


**Fig. 12: Performance improvement for the Adaptive and SelRep LLC organizations relative to a Shared LLC.** *SelRep LLC improves performance by 19.7% on average compared to 7.9% for the Adaptive LLC.*

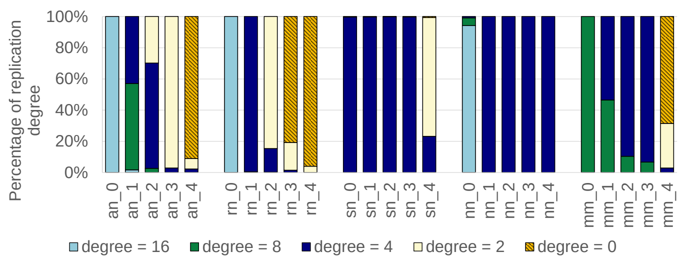


**Fig. 13: Replication degree distribution.** *SelRep LLC dynamically selects the replication degree for the given workload and configuration at hand.*

worlds and dynamically selects the Private or Shared LLC organization that achieves the highest performance for the given configuration.

Unfortunately, the Adaptive LLC leaves significant performance on the table compared to BSR. Full replication (as in Private LLC) versus no replication (as in Shared LLC) is a too coarse-grained decision. Determining the optimum replication degree for a given configuration, as obtained through BSR on a per-kernel basis, is shown to provide a significant performance benefit. Unfortunately, and as mentioned before, BSR assumes offline profiling for the same configuration as for the evaluation, which is unrealistic.

SelRep LLC closely approaches BSR for all benchmarks. By selectively replicating the shared data set across LLC slices, SelRep LLC is able to balance the LLC miss rate and effective LLC bandwidth to maximize performance. We find that SelRep LLC is within 2.3% compared to BSR on average (and by at most 7.3% for NN). It is interesting to note that SelRep LLC surpasses BSR for RN by 1.9%. The reason is that the optimum replication degree varies dynamically during the execution.

**Performance Improvement over Prior Work.** Figure 12 reports the performance improvement through SelRep LLC relative to a Shared LLC organization and compared against the state-of-the-art Adaptive LLC. SelRep LLC improves performance by 19.7% on average (and up to 61.6%) relative to the Shared LLC and by 11.1% on average compared to the Adaptive LLC. Whereas the Adaptive LLC only improves performance for the smallest configurations, SelRep LLC improves performance across the broad range of configurations. This demonstrates SelRep LLC's key feature: *SelRep LLC robustly improves performance over a Shared and Adaptive LLC organization, irrespective of the size of the shared data set relative to the LLC*. SelRep LLC significantly improves performance upon both the Shared LLC and the Adaptive LLC organizations for the medium-sized configurations ('⋆_1' through '⋆_3'), i.e., by up to 31.0% (RN), 29.6% (NN), 27.3% (MM) and 16.5% (AN).

**Replication Degree.** Figure 13 reports the replication degree for the different benchmarks and configurations under SelRep LLC. These results are computed by recording the replication degree for each epoch under SelRep LLC, out of which we then compute a distribution of the replication degree. SelRep LLC increases replication for smaller shared data sets. Also, different applications prefer different replication degrees. In particular, for AN and RN, we note the full swing from full replication to no replication as we range from small to large configurations. For SN, NN and MM on the other hand, the

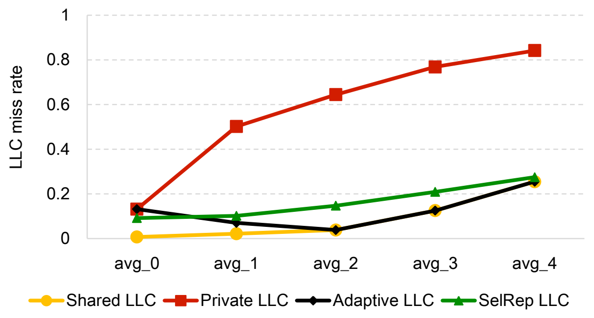


**Fig. 14: LLC miss rate across LLC organizations.** *SelRep LLC slightly increases LLC miss rates due to data replication compared to the Shared LLC across all configurations.*

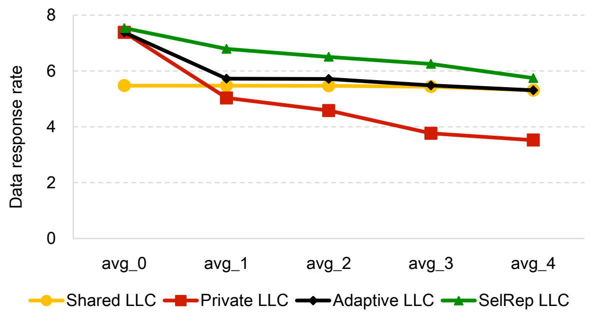


**Fig. 15: Effective LLC bandwidth measured as LLC response rate (replies/cycle).** *SelRep LLC achieves the highest effective LLC bandwidth which explains why it outperforms the other LLC organizations.*

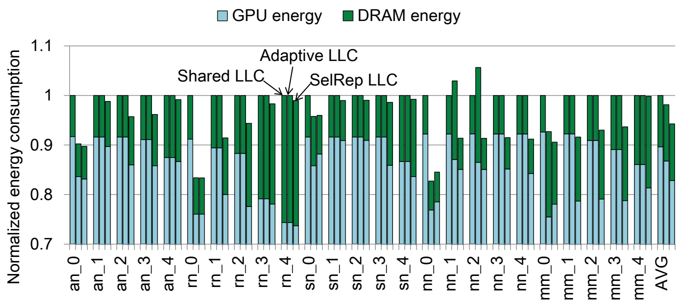


**Fig. 16: Normalized energy consumption.** *SelRep LLC reduces energy consumption by 5.7% on average compared relative to the shared cache configuration.*

range is limited to 2–4, 4–16, and 0–8, respectively.

**Discussion and Analysis.** We now dive deeper and analyze where the performance improvements are coming from. We report the LLC miss rate and effective LLC bandwidth in Figures 14 and 15, respectively. The effective LLC bandwidth is measured as the LLC response rate, or the number of LLC data replies per cycle. (We report average curves because of space constraints — curves for the individual benchmarks show similar trends.) The LLC miss rate increases sharply from small to large configurations under the Private LLC organization. Shared-data replication increases pressure on LLC capacity which leads to a higher miss rate. The miss rate is substantially lower under the Shared LLC organization because only a single copy of the shared data set is maintained. For all but the smallest configuration (i.e., `avg_1` through `avg_4`), the effective LLC bandwidth is higher for the Shared LLC compared to the Private LLC organization, which explains why the Shared LLC outperforms the Private LLC organization. The situation is different for the smallest configuration (`avg_0`). Data replication leads to a higher effective LLC bandwidth for the Private LLC while having a limited impact on the LLC miss rate. This explains why the Private LLC outperforms the Shared LLC organization by a significant margin for the smallest configuration.

The Adaptive LLC organization combines the best of both worlds by reverting to a Private LLC organization for shared data sets that are small compared to the LLC size. In other words, replicating a small shared data set across all LLC slices has limited impact on the LLC miss rate, while at the same time substantially increasing the effective LLC bandwidth. The increase in LLC bandwidth outweighs the increase in LLC miss rate, which leads to a substantial performance improvement for the smallest configuration (`avg_0`).

SelRep LLC achieves the highest effective LLC bandwidth, which explains why it outperforms the other LLC organizations across all configurations. However, SelRep LLC increases the LLC miss rate across all configurations compared to the Shared LLC. This increase in LLC miss rate is offset by the increase in effective LLC bandwidth. Note that SelRep LLC's performance improvement over the Shared LLC is higher for the smaller configurations, which is explained by the larger gap in effective LLC bandwidth. We observe a similarly large gap in effective LLC bandwidth between SelRep LLC and Adaptive LLC for the large configurations, which likewise explains SelRep LLC's performance improvements over the Adaptive LLC for those large configurations.

In summary, the key insight behind SelRep LLC is that replicating shared data across a *select* number of LLC slices improves overall application performance by carefully balancing the increase in effective LLC bandwidth versus the increase in LLC miss rate. In contrast to the state-of-the-art Adaptive LLC, which dynamically selects between full replication across all LLC slices or no replication, the SelRep LLC organization dynamically controls the degree of replication to balance LLC bandwidth and miss rate in a fine-grained manner to optimize performance.

## *B. Energy Consumption*

We now evaluate how the LLC organization affects energy consumption. On the one hand, data replication increases the LLC miss rate and therefore memory traffic, which in turn leads to higher power consumption in the memory subsystem. On the other hand, improved performance through increased effective LLC bandwidth reduces execution time, which reduces energy consumption. Figure 16 breaks down overall system (GPU and memory) energy consumption (lower-is-better) for the Adaptive

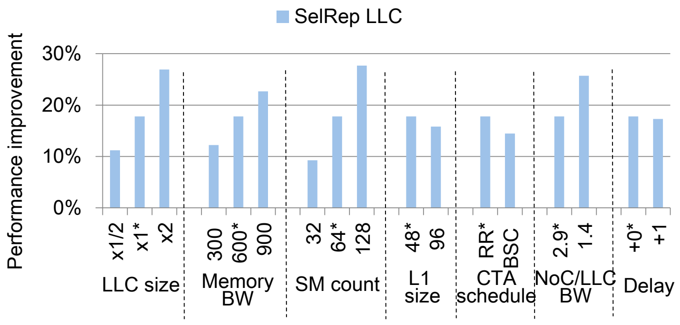


**Fig. 17: Sensitivity analyses. (An asterisk denotes the baseline configuration.)** *SelRep LLC yields higher performance benefits with larger LLC sizes, higher memory bandwidth (GB/s), larger SM count and smaller NoC/LLC bandwidth (TB/s). The performance benefit dampens with a larger L1 cache size (KB) and the BCS CTA scheduling policy. SelRep LLC is insensitive to increased MC-Router delay.*

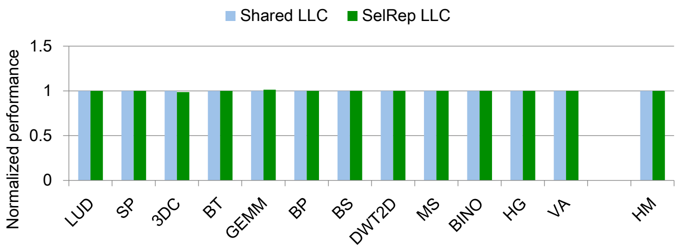


**Fig. 18: Evaluating SelRep LLC for the non-replication-sensitive benchmarks.** *SelRep LLC does not adversely impact non-replication-sensitive workloads compared to the baseline Shared LLC.*

LLC and SelRep LLC organizations relative to a Shared LLC. Overall, SelRep LLC reduces energy consumption by 4.6% and 5.7% on average compared to Adaptive LLC and the shared cache, respectively. The reason is that SelRep LLC reduces execution time significantly while only minorly increasing power consumption.

## *C. Sensitivity Analyses*

We now perform a number of sensitivity analyses to demonstrate SelRep LLC's effectiveness across the broader design space, see also Figure 17. The design points with an asterisk denote the baseline configuration.

**LLC size.** Selective replication is sensitive to LLC size. A smaller LLC size decreases its hit rate and decreases the opportunity from shared-data replication. A larger LLC size on the other hand increases the opportunity: replicating shared data across LLC slices increases the effective LLC bandwidth without being offset by the increase in LLC miss rate. We report a 26.9% average performance improvement over a Shared LLC organization for an LLC size that is twice as big as the default LLC size.

**Memory bandwidth.** SelRep LLC is also sensitive to memory bandwidth. Reduced memory bandwidth dampens the performance improvement through SelRep LLC because of a reduction in the effective LLC bandwidth, which implies limited data replication. In contrast, increasing memory bandwidth increases the opportunity and improves performance by 22.7% on average for a 900 GB/s memory system.

**SM count.** We now vary SM count while proportionally scaling memory bandwidth and LLC slice count to maintain an overall balanced system. We note that performance improves under the SelRep LLC organization while scaling the system for higher parallelism. The reason is that the concurrent access rate to shared data increases with an increased number of SMs, i.e., a larger number of concurrently executing CTAs access the same shared data set which increases pressure on the LLC. Selective replication increases the effective LLC bandwidth, thereby improving overall performance. We report an average performance improvement of 27.7% for a GPU with 128 SMs. We thus conclude that SelRep LLC is a scalable solution for future large-scale GPU systems.

**L1 cache size.** Increasing the L1 data cache size improves its hit rate and decreases the number of LLC accesses. This in turn decreases the performance opportunity for SelRep. Yet, SelRep LLC still improves performance by 15.8% on average for an increased L1 cache size of 96 KB.

**CTA scheduling.** In addition to the baseline two-level round-robin CTA scheduling policy (2L-RR), we also consider block CTA scheduling (BCS) which assigns consecutive CTAs to the same SM to improve L1 cache locality [38]. SelRep LLC improves performance by 14.5% on average, which is slightly less than for the round-robin policy because of improved L1 cache performance.

**NoC/LLC bandwidth.** We decreased the data width of the crossbar and LLC from 32 bytes to 16 bytes, thereby decreasing NoC/LLC bandwidth from 2.9 TB/s to 1.4 TB/s. With less NoC/LLC bandwidth, SelRep LLC improves performance by 25.7% on average compared to the Shared LLC configuration. Lower NoC/LLC bandwidth creates more contention, hence increasing the relative importance of SelRep LLC.

**SelRep LLC delay.** Supporting SelRep LLC may incur an additional delay in the MC-routers. Adding (an) extra cycle(s) however does not affect SelRep LLC's achieved performance improvement. The reason is that GPUs are mostly sensitive to bandwidth and not latency, i.e., the extra cycle(s) do not affect NoC bandwidth. Moreover, it affects memory request latency and not the reply latency — and prior work has shown that GPU performance is more sensitive to reply network performance than request network performance [29], [78].

## *D. Other GPU workloads*

Not all workloads are sensitive to data sharing and replication. It is therefore important to demonstrate that the proposed SelRep LLC organization does not adversely impact the performance for such workloads. Figure 18 reports performance for SelRep LLC normalized to Shared LLC for the non-replication-sensitive benchmarks and shows that these workloads are performance-neutral to SelRep LLC.

## VI. Related Work

**GPU LLC organization.** The most closely related work to ours is the Adaptive LLC [75] which dynamically selects either a shared or private LLC organization based on application behavior. We show in Section V that the Adaptive LLC leaves significant performance on the table since it, in contrast to SelRep LLC, does not support selective replication.

**CPU LLC organization.** Private versus shared LLC organizations have been studied extensively in the context of multi-core processors with Non-Uniform Cache Access (NUCA) LLCs. For instance, Beckmann et al. [12] replicate data blocks based on probabilistic filtering, and Chang et al. [16] propose cooperative caching. Chishti et al. [19] propose controlled replication to intelligently place replicas to balance latency and capacity, and Hossain et al. [25] leverage the cache coherence protocol to provide larger LLC capacity with localized data and metadata access for shared and private data. Kurian et al. [37] build a locality classifier on top of Reactive-NUCA [23] which only replicates the cache lines with high reuse, while ESP-NUCA [44] uses replicas and victims to take advantage of both private and shared cache organizations. A number of other works also explore latency-capacity trade-offs in multi-core LLCs (e.g., [13], [23], [26], [63]). These approaches are fundamentally different from GPU-focused LLC management because their key benefit comes from placing data close to the cores that need it and thereby reduce latency. In contrast, GPUs are latency-tolerant but bandwidth-sensitive, which means that replication schemes like SelRep LLC need to increase bandwidth rather than reduce latency.

A different direction of related work propose schemes that dynamically change cache organization in multi-core processors. Qureshi [54] proposes dynamic spill-receive to efficiently share cache capacity between applications in private caches, and Jaleel et al. [28] propose a thread-aware dynamic insertion policy. PIPP [69] combines dynamic insertion and promotion policies in shared caches, while Vantage [58] enables fine-grained cache partitioning. Herrero et al. [24] use distributed cache partitioning to optimize cache use, and MorphCache [60] dynamically alters the cache topology to enable sharing multiple cache slices between cores. GDP [27] allocates LLC capacity to processes based on slowdown predictions, while Rolan et al. [57] propose adaptive set-granular cooperative caching. These works are not directly applicable to GPUs as they exploit that different threads (processes) in multi-threaded (multi-programmed) workloads have different memory requirements. In contrast, the SMs of the GPU typically execute CTAs from the same kernel which results in limited diversity in memory requirements across SMs.

**GPU architecture.** Other related work optimizes various aspects of the GPU architecture, e.g., warp scheduling [33], [39], [43], [56], [66], L1 cache management [31], [59], [65], [68], register file design [3], [30], [32], NoC optimization [10], [35], [73], [77], [78], and SM resource virtualization [64], [72]. Recent work also provides approaches for efficient multitasking in GPUs [4], [52], [53], [62], [67], [71], [76], virtual memory management [9], and design considerations for multi-module GPUs [8], [45]. These approaches are all orthogonal to SelRep LLC as they do not consider selective replication of shared read-only data.

## VII. Conclusion

We have now presented SelRep LLC which selectively replicates shared read-only data across LLC slices to improve bandwidth while preserving shared-data locality. SelRep LLC achieves this by first identifying read-only shared data structures at compile-time and accessing these with a special-purpose load instruction. At runtime, SelRep LLC monitors these accesses to determine the appropriate balance point where replication improves bandwidth to shared data while ensuring that the data set remains cached. We show that SelRep LLC, unlike prior work, consistently provides high performance across a broad range of shared data set sizes for our replication-sensitive applications. More specifically, SelRep LLC improves performance by 19.7% and 11.1% on average (and up to 61.6% and 31.0%) compared to the shared LLC baseline and Adaptive LLC [75], respectively.

## Acknowledgements

We thank the anonymous reviewers for their valuable comments. This work was supported in part by European Research Council (ERC) Advanced Grant agreement no. 741097, FWO projects G.0434.16N and G.0144.17N, NSFC under Grants no. 61572508, 61672526 and 61802427. Magnus Jahre is supported by the Research Council of Norway (Grant no. 286596).

## References

[1] Tango: A Deep Neural Network Benchmark Suite for Various Accelerators. https://gitlab.com/Tango-DNNbench/Tango.

[2] T. M. Aamodt, W. W. L. Fung, and T. G. Rogers, *General-Purpose Graphics Processor Architectures*. Morgan & Claypool Publishers, 2018.

[3] M. Abdel-Majeed and M. Annavaram, "Warped Register File: A Power Efficient Register File for GPGPUs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, February 2013, pp. 412–423.

[4] J. T. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte, "The Case for GPGPU Spatial Multitasking," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2012, pp. 1–12.

[5] Amazon. Amazon web services. https://aws.amazon.com/cn/ec2/.

[6] AMD. AMD Radeon Pro Vega II. https://www.techpowerup.com/gpu-specs/radeon-pro-vega-ii.c3426.

[7] AMD. AMD Radeon RX Vega 64. https://www.techpowerup.com/gpu-specs/radeon-rx-vega-64.c2871.

[8] 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)*, June 2017, pp. 320–332.

[9] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. J. Rossbach, and O. Mutlu, "MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018, pp. 503–518.

[10] A. Bakhoda, J. Kim, and T. M. Aamodt, "Throughput-Effective On-Chip Networks for Manycore Accelerators," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2010, pp. 421–432.

[11] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proceeding of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2009, pp. 163–174.

[12] 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)*, December 2006, pp. 443–454.

[13] N. Beckmann and D. Sanchez, "Jigsaw: Scalable software-defined caches," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2013, pp. 213–224.

[14] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler, "Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2018, pp. 43–55.

[15] M. Burtscher, R. Nasre, and K. Pingali, "A Quantitative Study of Irregular Programs on GPUs," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, November 2012, pp. 141–151.

[16] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2006, pp. 264–276.

[17] 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 *Proceedings of the International Symposium on Workload Characterization (IISWC)*, October 2009, pp. 44–54.

[18] L. Chen, L. Zhao, R. Wang, and T. M. Pinkston, "MP3: Minimizing performance penalty for power-gating of Clos network-on-chip," in *Proceedings of the Symposium on High Performance Computer Architecture (HPCA)*, February 2014, pp. 296–307.

[19] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2005, pp. 357–368.

[20] D. B. Glasco, P. B. Holmqvist, G. R. Lynch, P. R. Marchand, K. Mehra, and J. Roberts, "Cache-based Control of Atomic Operations in Conjunction With an External ALU Block." Google Patents, March 2012.

[21] Google. Graphics Processing Unit (GPU) — Google Cloud. https://cloud.google.com/gpu/.

[22] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a High-Level Language Targeted to GPU Codes," in *Innovative Parallel Computing (InPar)*, May 2012, pp. 1–10.

[23] 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)*, June 2009, pp. 184–195.

[24] E. Herrero, J. González, and R. Canal, "Elastic Cooperative Caching: An Autonomous Dynamically Adaptive Memory Hierarchy for Chip Multiprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2010, pp. 419–428.

[25] H. Hossain, S. Dwarkadas, and M. C. Huang, "POPS: Coherence Protocol Optimization for Both Private and Shared Data," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, October 2011, pp. 45–55.

[26] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," in *Proceedings of the International Conference on Supercomputing (ICS)*, June 2005, pp. 31–40.

[27] M. Jahre and L. Eeckhout, "GDP: Using Dataflow Properties to Accurately Estimate Interference-Free Performance at Runtime," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 296–309.

[28] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, and J. Emer, "Adaptive Insertion Policies for Managing Shared Caches," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, October 2008, pp. 208–219.

[29] H. Jang, J. Kim, P. Gratz, K. H. Yum, and E. J. Kim, "Bandwidth-efficient On-chip Interconnect Designs for GPGPUs," in *Proceedings of the Design Automation Conference (DAC)*, June 2015, pp. 9:1–9:6.

[30] H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavaram, "GPU Register File Virtualization," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2015, pp. 420–432.

[31] W. Jia, K. A. Shaw, and M. Martonosi, "MRPB: Memory Request Prioritization for Massively Parallel Processors," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, February 2014, pp. 272–283.

[32] N. Jing, Y. Shen, Y. Lu, S. Ganapathy, Z. Mao, M. Guo, R. Canal, and X. Liang, "An Energy-efficient and Scalable eDRAM-based Register File Architecture for GPGPU," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2013, pp. 344–355.

[33] 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)*, June 2013, pp. 332–343.

[34] Y. H. Kao, N. Alfaraj, M. Yang, and H. J. Chao, "Design of High-Radix Clos Network-on-Chip," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, May 2010, pp. 181–188.

[35] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Providing Cost-effective On-Chip Network Bandwidth in GPGPUs," in *Proceedings of the International Conference on Computer Design (ICCD)*, September 2012, pp. 407–412.

[36] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, January 2016.

[37] 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)*, February 2014, pp. 1–12.

[38] 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)*, February 2014, pp. 260–271.

[39] S.-Y. Lee, A. Arunkumar, and C.-J. Wu, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration of GPGPU Workloads," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 515–527.

[40] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2013, pp. 487–498.

[41] G. Li and G. Gopalakrishnan, "Scalable SMT-Based Verification of GPU Kernel Functions," in *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, November 2010, pp. 187–196.

[42] Y. Liu, X. Zhao, M. Jahre, Z. Wang, X. Wang, Y. Luo, and L. Eeckhout, "Get Out of the Valley: Power-Efficient Address Mapping for GPUs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 166–179.

[43] Y. Liu, Z. Yu, L. Eeckhout, V. J. Reddi, Y. Luo, X. Wang, Z. Wang, and C. Xu, "Barrier-Aware Warp Scheduling for Throughput Processors," in *Proceedings of the International Conference on Supercomputing (ICS)*, June 2016, pp. 42:1–42:12.

[44] J. Merino, V. Puente, and J. A. Gregorio, "ESP-NUCA: A Low-cost Adaptive Non-Uniform Cache Architecture," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, January 2010, pp. 1–10.

[45] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the Socket: NUMA-aware GPUs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, October 2017, pp. 123–135.

[46] Nvidia. Accelerating Data Center Workloads With GPUs. https://www.nvidia.com/en-us/data-center/.

[47] Nvidia. CUDA COMPILER DRIVER NVCC. https://docs.nvidia.com/pdf/CUDA_Compiler_Driver_NVCC.pdf.

[48] Nvidia. (2016) NVIDIA Tesla P100. White paper. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[49] Nvidia. (2018) VOLTA Architecture and performance optimization. http://on-demand.gputechconf.com/gtc/2018/presentation/s81006-volta-architecture-and-performance-optimization.pdf.

[50] Nvidia. (2019) Parallel Thread Execution ISA Version 6.5. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html.

[51] NVIDIA CUDA SDK Code Samples. https://developer.nvidia.com/cuda-downloads. NVIDIA Corporation.

[52] J. J. K. Park, Y. Park, and S. Mahlke, "Chimera: Collaborative Preemption for Multitasking on a Shared GPU," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2015, pp. 593–606.

[53] J. J. K. Park, Y. Park, and S. Mahlke, "Dynamic Resource Management for Efficient Utilization of Multitasking GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2017, pp. 593–606.

[54] 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)*, February 2009, pp. 45–54.

[55] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A Case for MLP-Aware Cache Replacement," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2006, pp. 167–178.

[56] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious Wavefront Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2012, pp. 72–83.

[57] D. Rolan, B. B. Fraguela, and R. Doallo, "Adaptive Set-Granular Cooperative Caching," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2012, pp. 1–12.

[58] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and Efficient Fine-Grain Cache Partitioning," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2011, p. 5768.

[59] A. Sethia, D. A. Jamshidi, and S. Mahlke, "Mascar: Speeding up GPU Warps by Reducing Memory Pitstops," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, February 2015, pp. 174–185.

[60] S. Srikantaiah, E. Kultursay, T. Zhang, M. Kandemir, M. J. Irwin, and Y. Xie, "MorphCache: A Reconfigurable Adaptive Multi-level Cache Hierarchy," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, February 2011, pp. 231–242.

[61] 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)*, May 2012, pp. 201–210.

[62] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero, "Enabling Preemptive Multiprogramming on GPUs," in *Proceeding of the International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 193–204.

[63] P. Tsai, N. Beckmann, and D. Sanchez, "Nexus: A New Approach to Replication in Distributed Shared Caches," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2017, pp. 166–179.

[64] N. Vijaykumar, K. Hsieh, G. Pekhimenko, S. Khan, A. Shrestha, S. Ghose, A. Jog, P. B. Gibbons, and O. Mutlu, "Zorua: A Holistic Approach to Resource Virtualization in GPUs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, October 2016, pp. 1–14.

[65] B. Wang, W. Yu, X.-H. Sun, and X. Wang, "DaCache: Memory Divergence-Aware GPU Cache Management," in *Proceedings of the International Conference on Supercomputing (ICS)*, June 2015, pp. 89–98.

[66] B. Wang, Y. Zhu, and W. Yu, "OAWS: Memory Occlusion Aware Warp Scheduling," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*, September 2016, pp. 45–55.

[67] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous Multikernel GPU: Multi-tasking Throughput Processors via Fine-Grained Sharing," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 358–369.

[68] X. Xie, Y. Liang, Y. Wang, G. Sun, and T. Wang, "Coordinated Static and Dynamic Cache Bypassing for GPUs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, February 2015, pp. 76–88.

[69] Y. Xie and G. H. Loh, "PIPP: Promotion/Insertion Pseudo-Partitioning of Multi-Core Shared Caches," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2009, pp. 174–183.

[70] XILINX. (2018) AXI High Bandwidth Memory Controller v1.0. https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf.

[71] Q. Xu, H. Jeon, K. Kim, W. W. Ro, and M. Annavaram, "Warped-Slicer: Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 230–242.

[72] M. K. Yoon, K. Kim, S. Lee, W. W. Ro, and M. Annavaram, "Virtual Thread: Maximizing Thread-Level Parallelism beyond GPU Scheduling Limit," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 609–621.

[73] X. Zhao, S. Ma, C. Li, L. Eeckhout, and Z. Wang, "A Heterogeneous Low-cost and Low-latency Ring-Chain Network for GPGPUs," in *Proceedings of the International Conference on Computer Design (ICCD)*, October 2016, pp. 472–479.

[74] 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*, vol. 68, no. 9, pp. 1283–1296, September 2019.

[75] 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)*, July 2019, pp. 411–423.

[76] X. Zhao, M. Jahre, and L. Eeckhout, "HSM: A Hybrid Slowdown Model for Multitasking GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[77] X. Zhao, S. Ma, Y. Liu, L. Eeckhout, and Z. Wang, "A Low-Cost Conflict-Free NoC for GPGPUs," in *Proceedings of the Design Automation Conference (DAC)*, June 2016, pp. 34:1–34:6.

[78] A. K. Ziabari, J. L. Abellán, Y. Ma, A. Joshi, and D. Kaeli, "Asymmetric NoC Architectures for GPU Systems," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, July 2015, pp. 25:1–25:8.