SAC: Sharing-Aware Caching in Multi-Chip GPUs

Shiqing Zhang
Ghent University
Ghent, Belgium
shiqing.zhang@ugent.be

Mahmood Naderan-Tahan
Ghent University
Ghent, Belgium
mahmood.naderan@ugent.be

Magnus Jahre
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
magnus.jahre@ntnu.no

Lieven Eeckhout
Ghent University
Ghent, Belgium
lieven.eeckhout@ugent.be

ABSTRACT

Bandwidth non-uniformity in multi-chip GPUs poses a major design challenge for its last-level cache (LLC) architecture. Whereas a memory-side LLC caches data from the local memory partition while being accessible by all chips, an SM-side LLC is private to a chip while caching data from all memory partitions. We find that some workloads prefer a memory-side LLC while others prefer an SM-side LLC, and this preference solely depends on which organization maximizes the effective LLC bandwidth. In contrast to prior work which optimizes bandwidth beyond the LLC, we make the observation that the effective bandwidth ahead of the LLC is critical to end-to-end application performance. We propose Sharing-Aware Caching (SAC) to adopt either a memory-side or SM-side LLC organization by dynamically reconfiguring the routing policies in the intra-chip interconnection network and LLC controllers. SAC is driven by a simple and lightweight analytical model that predicts the impact of data sharing across chips on the effective LLC bandwidth. SAC improves average performance by 76% and 12% (and up to 157% and 49%) compared to a memory-side and SM-side LLC, respectively. We demonstrate significant performance improvements across the design space and across workloads.

CCS CONCEPTS

• **Computer systems organization** → **Single instruction, multiple data**; • **Networks** → *Network on chip*.

KEYWORDS

Multi-socket GPU, Multi-Chip-Module (MCM) GPU, Network-on-Chip (NoC), data sharing, cache organization.

ACM Reference Format:
Shiqing Zhang, Mahmood Naderan-Tahan, Magnus Jahre, and Lieven Eeckhout. 2023. SAC: Sharing-Aware Caching in Multi-Chip GPUs. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589078

ISCA '23, June 17–21, 2023, Orlando, FL, USA.

ACM ISBN 979-8-4007-0095-8/23/06…$15.00
https://doi.org/10.1145/3579371.3589078

1 INTRODUCTION

Graphics Processing Units (GPUs) have become the defacto standard accelerator for compute and memory-intensive applications due to their massively parallel architecture and programming model [8, 32, 41]. Scaling GPU performance is traditionally achieved by increasing the number of Streaming Multiprocessors (SMs)[1] as well as memory bandwidth. GPU manufacturers have increased die size which comes at the cost of lowering manufacturing yield while approaching the reticle limit. Continuing to scale GPU performance hence requires moving beyond a single chip [1, 7, 39].

Multi-chip GPU architectures can be broadly classified as (i) multi-socket GPUs [28, 30], where the system contains multiple GPU and memory chips that are connected with each other through the Printed Circuit Board (PCB), and (ii) Multi-Chip-Module (MCM) GPUs [2, 9], where multiple GPU and memory dies are interconnected using silicon interposers or organic substrates within a single package [39]. In both architectures, multiple GPU chips are connected to each other via inter-chip links, and each GPU chip is connected to a local memory partition. The key difference is the bandwidth available between GPU chips. MCM-GPUs offer the highest inter-chip bandwidth, but also incur the highest cost while providing limited memory capacity. In contrast, multi-socket GPUs incur lower cost and provide higher memory capacity, but offer lower inter-chip bandwidth. A key challenge in multi-chip GPUs, both multi-socket GPUs and MCM-GPUs, is how to best overcome the bandwidth non-uniformity in inter-chip versus intra-chip bandwidth.

As reported by prior work [2, 25, 36], the last-level cache (LLC) in multi-chip GPUs needs to be carefully designed to maximize performance. Broadly speaking, the LLC can be *memory-side*, caching data in the local memory partition on behalf of the SMs in any chip, or *SM-side*, caching data from any memory partition on behalf of the SMs within the chip. A memory-side LLC maximizes the effective cache capacity since each cache line is cached in at most one LLC. On the flip side, accesses to a remote memory partition need to traverse across lower-bandwidth inter-chip links. An SM-side organization on the other hand caches remote data locally, making it accessible at higher bandwidth. However, SM-side caches replicate shared cache lines across chips, thereby reducing the effective cache capacity. In addition, SM-side caches need to be kept coherent.

[1] Industrial terminology is Streaming Multiprocessor (SM) for Nvidia products and Compute Engine (CU) for AMD devices. In this paper, we adopt Nvidia's terminology.

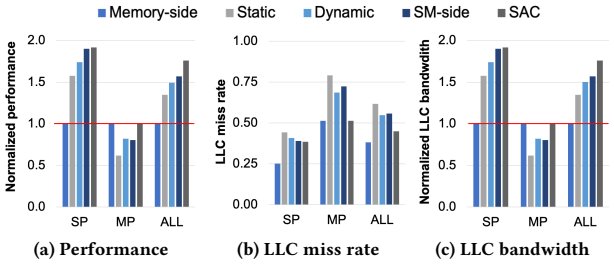


Figure 1: Performance, LLC miss rate and effective LLC bandwidth for different LLC organizations in a 4-chip GPU for SM-side preferred (SP) and memory-side preferred (MP) benchmarks. *Sharing-Aware Caching (SAC) yields the highest performance (and effective LLC bandwidth) across all workload categories.*

Our key observation is that some workloads prefer a memory-side LLC while other workloads prefer an SM-side LLC, as shown in Figure 1a. SM-side preferred (SP) benchmarks notice 91% higher average performance under an SM-side cache compared to a memory-side cache. In contrast, memory-side (MP) preferred benchmarks witness 32% higher average performance under a memory-side cache compared to an SM-side cache. (Details about our experimental setup follow in Section 4.) The SM-side organization uniformly leads to a higher miss rate as the LLC caches both local and remote data, which in case of true data sharing (i.e., multiple chips accessing the same cache lines) leads to data replication and thus increased cache pressure, see Figure 1b.

Counter-intuitively perhaps, the higher LLC miss rate under an SM-side LLC may lead to higher performance. This occurs when the higher LLC miss rate is outweighted by an increase in effective LLC bandwidth due to higher intra-chip bandwidth for accessing remote data in local caches instead of traversing lower-bandwidth inter-chip links to access remote caches — this is what happens for the SM-side preferred benchmarks, see Figure 1c. For the memory-side preferred benchmarks however, the increase in LLC miss rate is not outweighted by an increase in effective LLC bandwidth, which, in fact, decreases. While it is well-known that GPU performance critically depends on raw memory bandwidth, this analysis points out the importance of maximizing the effective LLC bandwidth. Indeed, whether a workload prefers a memory-side versus SM-side organization solely depends on which LLC organization maximizes the effective LLC bandwidth, which ultimately depends on the degree and nature of inter-chip data sharing in the workload. Clearly, the optimum LLC organization varies across workloads and their inputs. In particular, an application or kernel may prefer an SM-side organization for a small input set — because there is sufficient LLC capacity to replicate the shared data set — while preferring a memory-side LLC for a large input set — as the shared working set exceeds the LLC capacity.

Previously proposed solutions are suboptimal as they (explicitly or implicitly) optimize the effective bandwidth *beyond* the LLC rather than the bandwidth *ahead* of the LLC. In particular, recent work by Milic et al. [25] proposed a *Dynamic LLC* organization in which cache ways are dynamically (at run time) partitioned to cache local versus remote data to optimize and balance the outgoing local memory bandwidth versus the incoming inter-chip bandwidth. Prior to this work, Arunkumar et al. [2] proposed the L1.5 cache (for caching remote data locally) as a complement to a memory-side LLC (for caching local data) — this *Static LLC* organization, as we call it in this work, in effect statically (at design time) partitions the available LLC capacity for local versus remote data. Unfortunately, both the Static and Dynamic LLCs do not maximize the effective LLC bandwidth, see also Figure 1.

We propose *Sharing-Aware Caching (SAC)* which maximizes the effective bandwidth ahead of the LLC. SAC is enabled by our insight that the LLC can be configured to adopt memory-side or SM-side organizations by changing the NoC routing policy and adding simple bypass and control logic to the LLC slices — thereby retaining practically the same NoC area and power overhead as the memory-side architecture and reducing NoC power and area by 21% and 18%, respectively, compared to the SM-side architecture. To select the most favorable LLC organization, SAC employs a simple and lightweight analytical model that predicts the impact of data sharing across chips on the effective LLC bandwidth. We find that a short profiling window at the beginning of each kernel, which collects inputs for the model using hardware performance counters (620 bytes per chip), is sufficient for SAC to decide to adopt a memory-side or SM-side organization for the remainder of the kernel's execution. Our evaluation shows that SAC effectively selects the best-performing LLC organization across a broad range of GPU-compute benchmarks and input sets. More specifically, and as shown in Figure 1a, SAC improves performance by 76% and 12% on average (and up to 157% and 49%) compared to a memory-side and SM-side configuration, respectively. SAC improves performance by 31% and 18% on average (and up to 92% and 27%) compared to the L1.5 static LLC [2] and dynamic LLC partitioning [25], respectively.

In summary, we make the following contributions:

- We observe that the best performing LLC organization (memory-side versus SM-side) in a multi-chip GPU system is the one that maximizes the effective LLC bandwidth. In spite of the fact that an SM-side organization increases the LLC miss rate, caching remote data locally and replicating shared data across chips leads to a net performance improvement if this significantly increases the effective LLC bandwidth and outweighs the coherence overhead. If not, a memory-side LLC organization is preferred.
- We demonstrate that previously proposed solutions are suboptimal by (explicitly or implicitly) optimizing the effective bandwidth beyond the LLC. Instead, performance on multi-chip GPU systems is optimized by maximizing the effective bandwidth ahead of the LLC.
- We propose Sharing-Aware Caching (SAC) which dynamically chooses a memory-side or SM-side LLC organization. We observe that minorly adjusting the NoC of a memory-side LLC organization enables supporting both a memory-side and SM-side organization by changing the routing policy and by adding LLC bypass paths.
- SAC leverages a novel analytical model to steer LLC reconfiguration by predicting which LLC organization is likely to maximize the effective LLC bandwidth.
- We evaluate SAC across the design space while considering a broad range of workloads, inputs, system configurations

with varying inter-chip bandwidth, memory bandwidth, etc. SAC provides a significant performance benefit across the design space, especially for system configurations where the difference in intra-chip versus inter-chip bandwidth is relatively large.

2 MOTIVATION

Figure 2 shows our baseline multi-chip GPU which consists of four GPU chips, similar to prior work [2, 25]. Each chip contains 64 SMs with 128 KB private L1 caches, 4 MB of Last-Level Cache (LLC) capacity, and 8 memory controllers. We model conventional caches in our baseline to be consistent with prior work [2, 25], but SAC also supports sectored caches [18]. The Network-on-Chip (NoC) connects 32 SM clusters (two SMs share one network port) to 16 LLC slices with a total bisection bandwidth of 4 TB/s. The inter-chip links are connected to a chip's NoC and modeled after Nvidia's NVLink [34] in which each bidirectional link provides 64 GB/s bandwidth; we further assume an inter-chip ring network with 6 links per chip and hence 3 links between each pair of chips. The NoC is hence a 38×22 crossbar, i.e., 32 SM clusters plus 6 inter-chip links on the input side of the crossbar versus 16 LLC slices plus 6 inter-chip links on the output side; commercial GPUs often use (concentrated) crossbar NoCs [29]. We model a concentrated hierarchical crossbar in our baseline because it provides similar performance to a concentrated crossbar while being substantially more power and area efficient [48]. We refer to the SMs and caches within a chip as local and the SMs and caches in other chips as remote. Similarly, the memory partition attached to a GPU chip is local whereas the memory partitions of other GPU chips are remote. Our baseline employs software coherence which is common in GPUs [2, 25].

2.1 Memory-Side versus SM-Side LLC

Figure 3 illustrates the memory-side, SM-side and SAC LLC organizations in a multi-chip GPU for a simplified (for illustration purposes) GPU chip with four SMs (with private L1 caches), two LLC cache slices, two memory channels, and two inter-chip links. Figure 3a illustrates the memory-side LLC organization in which the LLC slices cache data from the local memory partition for both local and remote SMs [2]. Each LLC slice is responsible for a subset of the address space as determined by the memory page allocation policy. The memory-side LLC organization requires a NoC between the SMs (and their private L1s) and the LLC slices [20]. There are direct connections between the LLC slices and their corresponding memory controllers. The NoC between the SMs and LLC slices

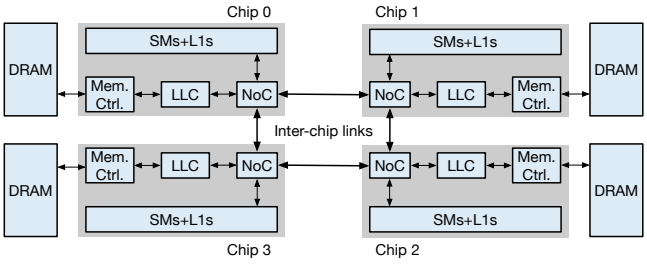


Figure 2: Schematic of our baseline multi-chip GPU.

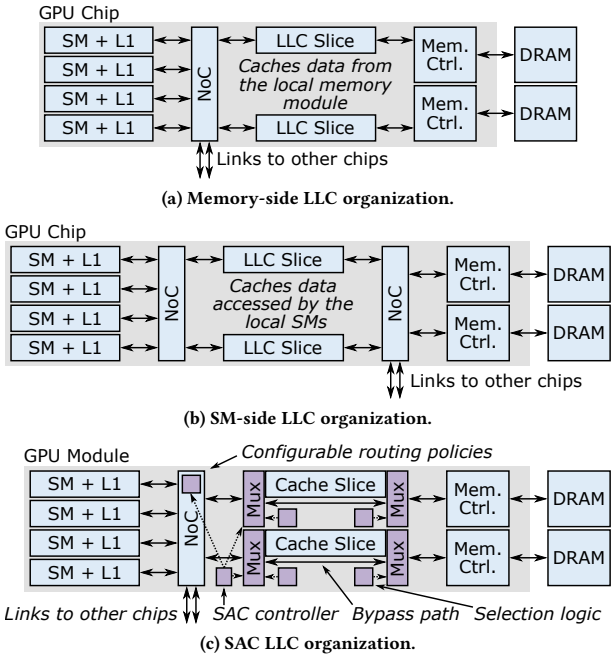


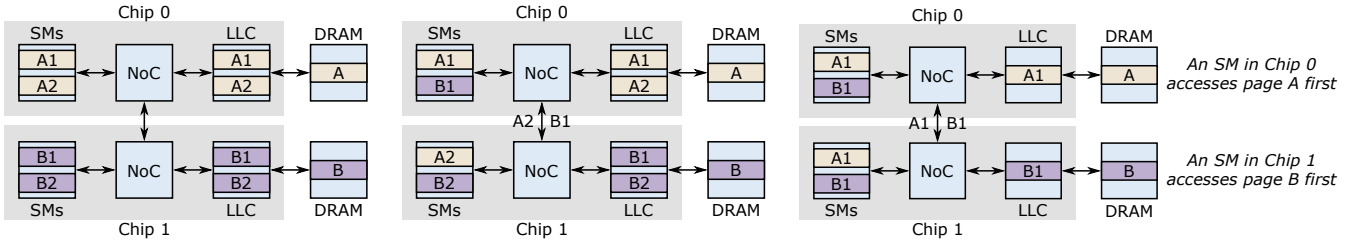
Figure 3: The memory-side, SM-side and SAC LLC organizations in multi-chip GPUs. *The key observation underpinning SAC is that minorly modifying the memory-side architecture enables supporting both memory-side and SM-side LLC configurations.*

provides links to the other chips, i.e., local requests are sent to the LLC slices and remote requests are sent across the inter-chip links.

The LLCs in the SM-side organization cache data from the entire address space, which includes both the local memory partition *and* the remote memory partitions [25, 36], as illustrated in Figure 3b. The LLC slices in the SM-side organization are on the SM-side of the NoC and hence require an additional NoC to connect the LLC slices to the memory controllers and inter-chip links. This provides an inherent latency and bandwidth advantage over the memory-side LLC because the inter-chip traffic does not compete with the intra-chip traffic between the SMs and LLC slices. On the flip side, the two NoCs for the SM-side LLC lead to 21% and 18% higher NoC power and area, respectively, compared to the memory-side LLC NoC according to DSENT [43] (assuming a 22 nm technology node). SM-side LLCs also require support for coherence. Under software-based coherence [2], flushing and invalidation of the L1 cache upon a cache control operation inserted by software or at kernel boundaries needs to be extended to the LLC, i.e., both the L1 and LLC need to be flushed and invalidated. Under hardware-based coherence [36], a directory needs to keep track of sharers across all SM-side LLCs; all sharers need to be invalidated upon a write.

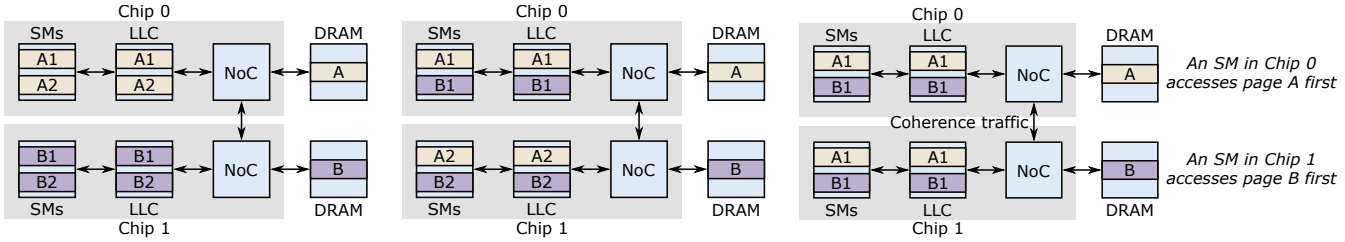
2.2 SAC: Supporting Both Memory-Side and SM-Side LLC

The key observation that underpins SAC is that minor modifications to the memory-side LLC baseline architecture enables supporting *both* memory-side and SM-side LLC policies (see Figure 3c). More specifically, we need to (1) enable LLC bypassing (i.e., add bypass



(a) No inter-chip sharing. (b) False sharing. (c) True sharing.

Figure 4: Data sharing in a memory-side LLC. *The memory-side LLC configuration is beneficial for workloads with limited data sharing because first-touch page placement then mostly maps data to the chip's local memory.*



(a) No inter-chip sharing. (b) False sharing. (c) True sharing.

Figure 5: Data sharing in an SM-side LLC. *SM-side LLC is beneficial for applications with falsely-shared data and/or a small truly shared data set: local accesses within a chip provide high effective bandwidth.*

paths, selection logic and multiplexers/demultiplexers), (2) modify the NoC routing algorithm to route remote requests to remote LLC slices under a memory-side LLC organization and local LLC slices under an SM-side organization, (3) change the LLC controller to send remote LLC misses to remote memories under an SM-side organization (all misses access the local memory in a memory-side organization), and (4) support the coherence operations required for SM-side operation (i.e., flushing the LLC on kernel boundaries under software coherence and invalidating sharers upon a write under hardware coherence). Adopting a configurable LLC architecture on top of a memory-side organization does not incur significant overhead (see Section 3.1 for a detailed discussion).

A configurable LLC organization is preferable because whether a workload prefers a memory-side or SM-side LLC organization depends on its shared data set and access pattern. Akin to the well-known true sharing versus false sharing phenomenon observed at the cache line granularity in CPU cache coherence protocols [13], *true inter-chip sharing* occurs when SMs in different GPU chips access the same cache line. In contrast, *false inter-chip sharing* occurs when SMs in different chips access different cache lines from the same memory page. In other words, a cache line is truly shared if it is accessed by multiple chips, and a cache line is falsely shared if it is accessed by a single chip only, but at least one cache line within the same memory page is accessed by another chip. A cache line is not shared if it is accessed by a single chip only, and no other cache lines from the same memory page are accessed by another chip.

2.3 Which LLC Organization is Preferred?

In the examples that follow, we denote memory accesses in the form *Xi* where *X* identifies the memory page and *i* the cache line within the page, e.g., A1 refers to cache line 1 within page A. To simplify the example, we focus on a dual-chip GPU system with software coherence in which each chip has a single LLC slice and is connected to a single memory partition. We further assume a *first-touch page allocation policy* [2] which installs a memory page in the memory partition of the chip that first accesses it, i.e., Chip 0 (C0) is assumed to access page A first whereas Chip 1 (C1) is assumed to access page B first; as a result, A is mapped to C0's memory partition whereas B is mapped to C1's memory partition.

Non-shared cache lines. Data that is not shared across chips leads to the exact same placement of cache lines under either LLC organization. Assume that C0 accesses A1 and A2 while C1 accesses B1 and B2, i.e., there is no sharing across chips. The memory-side LLC in Figure 4a caches A1 and A2 in C0 because they are stored in the local memory partition. The SM-side LLC in Figure 5a also caches A1 and A2 in C0, but for a different reason, namely because they are accessed by the local SMs. Similarly, the SMs in C1 access page B which leads to B1 and B2 being cached in C1's LLC under both LLC organizations. As a result, because there is no difference between either LLC organizations, they are equally preferred (apart from the overhead introduced by the coherence protocol).

Falsely shared cache lines. For falsely shared cache lines, see Figures 4b and 5b, there is an important difference between both LLC organizations. Assume that C0 accesses cache blocks A1 and B1, whereas C1 accesses A2 and B2, i.e., these are all falsely shared cache lines. The memory-side LLC organization stores the cache

lines of page A in C0 and the cache lines of page B in C1, resulting in A2 and B1 traversing the inter-chip links on repeated accesses. In contrast, the SM-side LLC stores A1 and B1 in C0 and A2 and B2 in C1 and hence avoids consuming inter-chip bandwidth on repeated accesses. (The cold-start traversal of the inter-chip links for A2 and B1 is not shown in Figure 5b.) For falsely shared cache lines, the SM-side LLC is hence the preferred LLC organization because it exposes higher effective memory bandwidth by caching the frequently accessed data in the on-chip LLC slices of the chip that accesses it.

Truly shared cache lines. The situation is more involved for truly shared cache lines, and which LLC organization is preferred depends on the workload. Assume that both C0 and C1 access A1 and B1, i.e., both cache lines are truly shared. In the memory-side LLC (Figure 4c), C0's requests to B1 need to go to C1's memory partition and C1's requests to A1 need to access C0's memory partition, hence increasing the traffic on the inter-chip links. In contrast, the SM-side LLC will replicate A1 and B1 across the LLCs in both chips (see Figure 5c), thereby increasing the effective bandwidth to these shared cache lines. However, cache line replication across the LLCs in different chips reduces the effective LLC cache capacity and incurs coherence overhead. In other words, the SM-side LLC is beneficial when (i) the bandwidth benefit of replicating truly shared cache lines outweighs the bandwidth cost of increased LLC miss rates, and (ii) there is sufficient true data sharing to outweigh the cost of maintaining coherence among the LLCs of the different chips. A memory-side LLC avoids this coherence issue and yields higher LLC utilization because it does not replicate data, but, on the flip side, it does not expose as high effective bandwidth to shared cache lines.

Interaction with L1 caches. The SMs cache the same cache lines in their respective L1 caches under the memory-side and SM-side organizations, as illustrated in Figures 4 and 5. The LLC organization indirectly impacts the L1 caches: data replication under the SM-side organization may lead to thrashing the L1 caches under an inclusive cache hierarchy in case the truly shared data set exceeds the LLC capacity.

3 SHARING-AWARE CACHING

The key take-away from the above analysis is that there is a case to be made for an LLC organization that, depending on the workload's inter-chip data sharing characteristics, reconfigures itself to being either memory-side or SM-side. *Sharing-Aware Caching (SAC)* achieves exactly this.

3.1 The SAC Reconfigurable LLC Architecture

The SAC reconfigurable LLC architecture, shown in Figure 3c, requires selection logic to decide whether or not to bypass the cache slice, a physical bypass path (wires) to connect to the memory controller, and a multiplexer/demultiplexer on both the SM side and the memory side of the LLC slice. On the SM side, requests need to be demultiplexed to either the LLC slice or bypass path, while responses are multiplexed into the NoC port. Similarly, requests are multiplexed and responses demultiplexed on the memory side. Local misses (from the local LLC slice) and remote misses (bypassing the local LLC slice) share the request queue in front of the memory

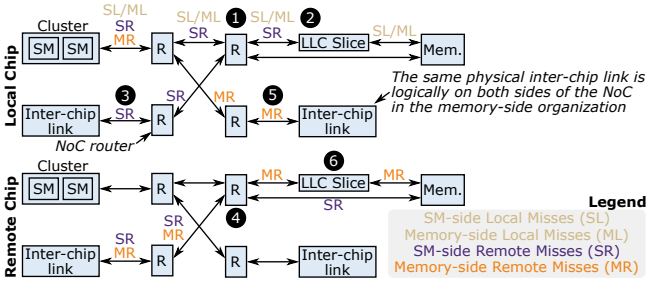


Figure 6: Miss routing in the SM-side and memory-side LLC.

controller in our setup. If there is no available queue space, our selection logic forces the request to wait in the queue ahead of the local LLC slice. We account for these queueing delays in our simulation experiments.

Figure 6 illustrates how LLC misses traverse the NoC in this architecture. For LLC misses that access the local memory partition, i.e, SM-side Local (SL) and Memory-side Local (ML) misses, there is no difference between the two configurations (see ❶). SM-side Remote (SR) misses first access the local LLC slice at ❷ before being routed to the appropriate inter-chip link at ❸. They later appear in the remote chip where they must bypass the LLC slice (see ❹) because it caches data for the remote chip's SMs. The response then follows the same path to insert the cache block into the local chip's LLC slice before delivering the requested data to the SM. (We model separate request and response networks in our evaluation.) Memory-side Remote (MR) misses are in contrast routed directly to the local chip's inter-chip links (see ❺) before accessing the remote LLC slice at ❻.

Enabling SAC's configurable LLC on top of a memory-side architecture does not incur significant overhead because we do not need to modify the crossbar NoC, i.e., we need a 38×22 crossbar for both organizations to connect 32 SM clusters and 6 inter-chip links to 16 LLC slices and 6 inter-chip links. Enabling a configurable LLC does however require changing the NoC routing policy which slightly increases design complexity but does not affect bandwidth. When configured as an SM-side LLC, the selection logic (1) forwards SR-misses from local SMs to the NoC, and (2) bypasses the LLC slice for SR-misses from remote SMs. The selection logic however only marginally increases complexity, area and power compared to the memory-side architecture (see Section 3.6.)

3.2 Runtime Support

SAC reconfigures the LLC on a per-kernel basis. During a short profiling window at the beginning of each kernel, a memory-side configuration is assumed and profiling information is collected using custom hardware performance counters. The profiling data then serves as input to the EAB analytical model to predict whether a memory-side or SM-side LLC is likely to yield the highest performance. If the model predicts the SM-side configuration to yield the highest performance, the LLC is reconfigured as an SM-side cache for the remainder of the kernel. If not, the LLC remains configured as a memory-side cache. In our setup, we assume a profiling window of 2K clock cycles. We explored the impact of the profiling window and concluded that 2K cycles at the beginning of each kernel invocation is adequate for our workloads. We also explored

Symbol	Memory-side configuration		SM-side configuration	
	Local Requests	*Remote Requests*	*Local Requests*	*Remote Requests*
$B_{SM\_LLC,local\,|\,remote}$	B_{intra}	B_{inter}	$B_{intra} \times R_{local}$	$B_{intra} \times R_{remote}$
$B_{LLC\_hit,local\,|\,remote}$	$B_{LLC\_hit} \times R_{local}$	$B_{LLC\_hit} \times R_{remote}$	$B_{LLC\_hit} \times R_{local}$	$B_{LLC\_hit} \times R_{remote}$
$B_{LLC\_miss,local\,|\,remote}$	$B_{LLC\_miss} \times R_{local}$	$B_{LLC\_miss} \times R_{remote}$	$B_{LLC\_miss} \times R_{local}$	$B_{LLC\_miss} \times R_{remote}$
$B_{LLC\_mem,local\,|\,remote}$	–	–	–	B_{inter}
$B_{mem,local\,|\,remote}$	$B_{mem} \times R_{local}$	$B_{mem} \times R_{remote}$	$B_{mem} \times R_{local}$	$B_{mem} \times R_{remote}$

Table 1: Computing the EAB bandwidth numbers for the different LLC configurations and local vs. remote requests.

Symbol	Definition
EAB_{local}	The EAB provided for local requests
EAB_{remote}	The EAB provided for remote requests
R_{local}	Fraction local requests
R_{remote}	Fraction remote requests
B_{SM_LLC}	Bandwidth between SMs and LLC slices
B_{LLC_mem}	Bandwidth between the LLC and memory
B_{intra}	Bandwidth of intra-module links
B_{inter}	Bandwidth of inter-module links
B_{LLC}	Raw LLC bandwidth
B_{LLC_hit}	Bandwidth for LLC hits
B_{LLC_miss}	Bandwidth for LLC misses
LLC_{hit}	LLC hit rate
LSU	LLC slice uniformity
N	Number of LLC slices
R_i	Number of requests to LLC slice i
B_{mem}	Raw memory bandwidth

Table 2: The EAB model inputs.

periodically re-profiling the workload during kernel execution (e.g., every 100K or 1M cycles) and found this to be unnecessary for our workloads. In any case, profiling overhead is less than 1% (analysis omitted due to space constraints).

3.3 The EAB Model

We now describe the *Effective Available Bandwidth (EAB)* model, which computes the EAB under the memory-side and SM-side LLC organizations. By comparing both EABs, the runtime system decides which organization to adopt.

The effective available bandwidth is defined as the bandwidth that the system can provide given the workload's access pattern. We compute the overall EAB as follows:

$$EAB_{\text{total}} = EAB_{\text{local}} + EAB_{\text{remote}},$$

where EAB_{local} and EAB_{remote} are the effective available bandwidth provided by the system for local and remote requests, respectively. The local and remote requests are bottlenecked by different bandwidth limitations depending on the specific LLC configuration. We compute the local/remote EAB as follows:

$$EAB_{\text{local}\,|\,\text{remote}} = \min(B_{SM\_LLC}, B_{LLC\_hit} + \min(B_{LLC\_miss}, B_{LLC\_mem}, B_{mem})).$$

EAB is computed as the minimum of the bandwidth between the SMs and the LLC slices (B_{SM_LLC}) and the bandwidth for LLC hits (B_{LLC_hit}) plus the available bandwidth for LLC misses. The latter is computed as the minimum of the bandwidth provided by LLC misses (B_{LLC_miss}), the bandwidth between the LLC and main memory (B_{LLC_mem}), and the raw bandwidth provided by memory (B_{mem}). How to compute these bandwidth numbers depends on (1) whether they concern local or remote requests, and (2) the specific LLC organization. We now describe how we compute them, see also Tables 1 and 2.

B_{SM_LLC}: For the memory-side configuration, the available bandwidth between the SMs and the LLC is bounded by the intra and inter-chip network bandwidth for local and remote requests, respectively. Under an SM-side organization, the intra-GPU network bandwidth is shared by local and remote requests, we hence compute the effective available bandwidth for local and remote requests as the intra-GPU network bandwidth times the respective fraction of requests.

B_{LLC_hit}: The LLC hit bandwidth is computed as the raw LLC bandwidth (B_{LLC}) times the available *LLC Slice Uniformity (LSU)* and the effective LLC hit rate (LLC_{hit}):

$$B_{LLC\_hit} = B_{LLC} \times LSU \times LLC_{hit}$$

LSU quantifies the degree of distribution across the N LLC slices, and is computed as follows:

$$LSU = \frac{1}{N} \sum_{i=1}^{N} \frac{R_i}{\max(R_1, ..., R_N)}$$

In this equation, R_i equals the number of requests to LLC slice i. LSU thus equals one if the requests are uniformly distributed across all slices, and equals $1/N$ if all requests go to a single slice. LSU and LLC hit rate are a function of the specific LLC organization and they are both affected by the degree of data replication under a specific LLC organization. A cache line that is shared by multiple SMs in a memory-side LLC leads to a non-uniform access distribution, and thus low LSU, as accesses are concentrated to a single slice. In contrast, in an SM-side LLC, a shared cache line is replicated across multiple slices in different chips, thereby raising LSU but also decreasing the LLC hit rate due to increased pressure on cache capacity.

B_{LLC_miss}: A miss in the LLC leads to a memory access, in either a local or remote memory partition depending on the LLC organization. In any case, the sum of the LLC hit and miss bandwidth cannot exceed the raw LLC bandwidth. The LLC miss bandwidth is computed as follows:

$$B_{LLC\_miss} = B_{LLC} \times LSU \times LLC_{miss}.$$

The LLC miss rate reflects the pressure data replication imposes on cache capacity under an SM-side configuration.

B_{LLC_mem}: We assume that the LLC misses that access a local memory partition are not bandwidth-limited. This is a reasonable assumption assuming point-to-point links between an LLC slice and its corresponding memory controller. In an SM-side configuration, LLC misses to remote data need to go to a remote memory partition

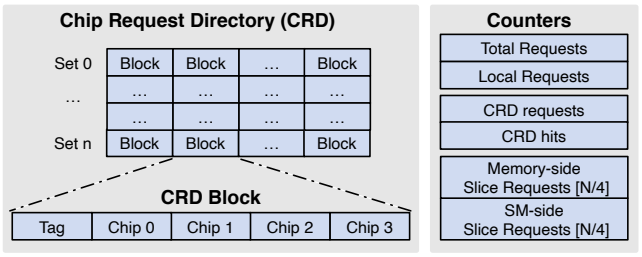


Figure 7: The Chip Request Directory (CRD) and counters to collect the EAB model inputs.

over the inter-chip network. Those requests are hence limited by the inter-chip link bandwidth.

***B_{mem}*:** The available memory bandwidth is assumed to be equal to the designed memory bandwidth. This is reasonable since we consider the PAE address mapping policy [22] which evenly distributes memory accesses across memory channels. We verified that this is indeed the case for our setup.

3.4 Collecting the Model Inputs

Some EAB model parameters are a function of the architecture only (i.e., intra and inter-GPU interconnect bandwidth B_{intra} and B_{inter} as well as raw LLC and memory bandwidth B_{LLC} and B_{mem}). Other parameters are a function of the workload only, namely the fraction of local and remote memory accesses (i.e., R_{local} and R_{remote}). The final set of parameters is a function of the interaction between the workload and the specific LLC configuration, namely the LLC hit/miss rate (LLC_{hit} and LLC_{miss}), and the LLC slice uniformity (LSU).

We rely on hardware performance counters for computing the application-dependent parameters R_{local}, LSU and LLC_{hit}. (Note that LLC_{miss} is computed as $1 - LLC_{hit}$; similarly, $R_{remote} = 1 - R_{local}$.) R_{local} depends on the application alone, whereas LSU and LLC_{hit} depend on the LLC configuration as well. We thus need to compute the latter two for both LLC configurations. We rely on existing hardware performance counters to compute LLC_{hit} under the memory-side configuration. For computing the other model inputs, we propose a custom hardware performance counter architecture, as illustrated in Figure 7. To compute LSU, we count the number of 'slice requests' R_i to each slice for both LLC configurations. We have $N/4$ slices per chip, hence we effectively have an array of $N/4$ 'slice requests' counters, one for the memory-side configuration and one for the SM-side configuration. For computing R_{local}, we keep track of the number of 'total requests' (i.e., all L1 misses) and 'local requests' (i.e., all L1 misses going to the local LLC).

Computing LLC_{hit} for the SM-side LLC is somewhat more involved, for which we develop a lightweight structure called the *Chip Request Directory (CRD)*, inspired by the Auxiliary Tag Directory (ATD)[35] and the Replication Degree Directory (RDD) [49]. We modified the mechanism so that CRD can predict the LLC hit rate of the SM-side configuration when running the memory-side configuration. As illustrated in Figure 7, CRD samples n sets of the local cache for which it provides an entry (i.e., CRD block) to record whether this block has been accessed and replicated in any of the chips under the SM-side configuration. Each CRD block contains a tag and four bits called 'Chip i' to record whether this block has been accessed by chip i or not. These bits are initially set to 0. Upon the first access with the matching tag to chip i, the corresponding bit is set. Then, when chip i accesses the cache line again (i.e., 'Chip i' equals one), we know that it will be a cache hit in the SM-side configuration. If the architecture uses sectored caches [18, 21, 38], we need to increase the size of the 'Chip i' fields to include one bit for each sector. The number of LLC hits is kept track of in the 'CRD hits' counter; we also keep track of the total number of 'CRD requests'; dividing these two counters provides an estimate for the hit rate under the SM-side LLC. Recall that profiling is done while the LLC is configured as a memory-side cache. Doing so guarantees that the CRD sees all requests whose data addresses are mapped to this memory partition.

3.5 Putting It Together

At the end of the profiling phase, the EAB model is provided with the inputs collected with the hardware performance counter architecture. The EAB model then computes and compares the EABs for the memory-side and SM-side configurations. Because the SM-side configuration incurs additional LLC coherence overhead compared to the memory-side configuration — which the EAB model does not account for (to not overly complicate the model) — we consider a threshold θ when comparing the EABs. If the EAB of the SM-side configuration exceeds the EAB of the memory-side configuration by more than θ, the runtime system will reconfigure the LLC to an SM-side organization. If not, the LLC remains configured as a memory-side cache. We consider a balanced threshold $\theta = 5\%$ (sensitivity analysis omitted due to space constraints).

3.6 Overhead

The hardware overhead of SAC is small. In our setup, the CRD consists of 8 sets with 16 ways per set. Each CRD block contains a 30-bit tag and 4 bits to record the requesting chip for conventional caches. Supporting sectored caches requires 4 bits per chip as we model 4 sectors per cache line. The total hardware overhead for the CRD amounts to 544 and 736 bytes per chip for conventional and sectored caches, respectively. To compute LSU, we assume one 16-bit counter per LLC slice for both the memory-side and SM-side configurations. The LSU counters amount to 64 bytes per chip. Finally, we need four more counters, which we assume to be 24-bit each, for a total of 12 bytes. Put together, the total hardware overhead with conventional and sectored caches amounts to 620 and 812 bytes per chip, respectively. We combined Synopsys DesignWare Library [44], DSENT [43] and CACTI [26] to estimate the overhead of the LLC bypassing logic including all components (i.e., selection logic, multiplexers, demultiplexers and wires); the LLC slice width is estimated to be 0.69 mm for a 256 KB LLC slice according to CACTI (assuming a 22 nm technology node). We conclude that the total area and power overhead of enabling LLC bypassing equals 1.9% and 1.6% of the total NoC area and power, respectively, over a memory-side LLC.

The runtime overhead of the EAB model is negligible (less than 1%). The model inputs are collected while the application is running during the profiling phase. Computing the model formulas involves

Parameter	Value
Number of chips	4
Number of SMs	64 per chip, 256 in total
GPU frequency	1 GHz
SM compute capability	CUDA 8.0; 64 warps per SM
Warp scheduler	Greedy-Then-Oldest (GTO) [37]
Inter-chip bandwidth	768 GB/s ring, 12 bidirectional links in total 6 links per chip, 64 GB/s bidirectional per link
LLC bandwidth	64 slices, 16 TB/s in total
DRAM bandwidth	32 channels, 1.75 TB/s in total
L1 data cache size	128 KB per SM
LLC capacity	128 B per cacheline, 4 MB per chip, 16 MB in total
Page size and allocation	4 KB, first-touch [2]
CTA allocation	Distributed CTA scheduling [2]
Coherency protocol	Software-managed

Table 3: Simulated baseline configuration.

a couple dozen computations and a final comparison. Reconfiguring the LLC to an SM-side organization at the end of the profiling window (if needed) requires that we (1) wait for the completion of in-flight requests, (2) write-back and invalidate the dirty cache lines in the LLC, and (3) switch the routing algorithm in the NoCs to SM-side. When a kernel terminates, we revert back to a memory-side configuration (if needed). This involves (1) waiting for the completion of in-flight requests and (2) switching the routing algorithm to memory-side. We model these runtime overheads carefully.

4 EXPERIMENTAL SETUP

Simulated System. We faithfully extended GPGPU-Sim [3] to model a multi-chip GPU system with four chips. As shown in Table 3, our baseline configuration consists of 64 SMs and a 4 MB LLC per chip, or 256 SMs and 16 MB of LLC in total, which is in line with prior work and commercial GPUs [2, 25, 29]. As aforementioned, all NoCs in this work are concentrated hierarchical crossbars. We consider a ring topology to connect the four chips with a total inter-chip bandwidth of 768 GB/s. We assume four memory partitions with eight memory channels each, for a total of 32 memory channels and 1.75 TB/s memory bandwidth. Our simulated baseline is configured similarly as the second-generation NVLink [34] in terms of link bandwidth, and after GDDR6 [24] in terms of memory bandwidth. We perform various sensitivity analyses considering higher and lower inter-chip and memory bandwidth configurations. We model a multi-level cache hierarchy with a private write-through L1 cache per SM and a write-back last-level (L2) cache. We further assume software-controlled cache coherence for the L1s, which we extend to L2 when configured as SM-side.

We adopt a first-touch page allocation policy [2] which maps a page to the memory partition of the GPU chip that first accesses a cache block within the page. We use the state-of-the-art PAE randomized address mapping scheme [22] to uniformly distribute memory accesses across LLC slices, memory channels and banks. We further consider distributed CTA scheduling [2] which divides the set of CTAs across the four chips by assigning a contiguous set of CTAs per chip to maximize inter-CTA data locality within a chip.

Workloads. We consider a diverse set of 16 benchmarks from five benchmark suites, including Rodinia [6], Polybench [12], Tango [16], Nvidia SDK [33], and Parboil [42], see Table 4. We use large default input sets for each of the benchmarks.

Benchmark	Number of CTAs	Footprint (MB)	True-Shared Data (MB)	False-Shared Data (MB)
RN [16]	512	21	11	4
AN [16]	1,024	20	9	3
SN [16]	512	18	2	13
CFD [6]	4,031	97	9	33
BFS [6]	1,954	37	10	14
3DC [12]	2,048	98	17	38
BS [33]	480	76	0	56
BT [6]	48,096	31	4	19
SRAD [6]	65,536	753	30	3
GEMM [12]	2,048	174	14	21
LUD [6]	131,068	317	38	51
STEN [42]	1,024	205	18	17
3MM [12]	4,096	109	12	7
BP [6]	65,536	76	4	0
DWT [6]	91,373	207	3	10
NN [16]	60,000	1,388	154	0

Table 4: Simulated workloads: SM-side (top half) versus memory-side (bottom half) LLC preferred benchmarks.

5 EVALUATION

We compare the following configurations:

- **Memory-side LLC:** Our baseline, a commonly used LLC organization in commercial GPUs [29, 31].
- **SM-side LLC:** This is the two-NoC implementation of the SM-side LLC configuration (see Section 3.1).
- **Static LLC:** This is the L1.5 cache [2] which provides half the LLC capacity to cache remote data and half the LLC capacity to cache local data.
- **Dynamic LLC**: Starting from a half-local, half-remote LLC configuration, this LLC organization dynamically reserves capacity for local versus remote data, following the design proposed in [25].
- **Sharing-Aware Caching (SAC):** Our scheme as described in Section 3 which dynamically chooses between a memory-side and an SM-side configuration.

5.1 Performance

Figure 8 reports the speedup for different LLC organizations relative to the baseline memory-side LLC. The benchmarks are grouped by their LLC organization preference. The benchmarks on the left-hand side of the figure prefer the SM-side LLC organization while the benchmarks on the right-hand side prefer the memory-side LLC organization; we report the average (harmonic mean) speedup for the two benchmark categories and the overall average on the far right. SAC achieves the best of both worlds. SAC performs (almost) equally well as the SM-side LLC for the SM-side preferred benchmarks, while performing equally well as the memory-side LLC for the memory-side preferred benchmarks. Overall, on average across all benchmarks, SAC outperforms the memory-side LLC and SM-side LLC organizations by 76% and 12%, respectively. SAC outperforms the previously proposed static and dynamic LLC organizations by 31% and 18% on average, respectively. The small

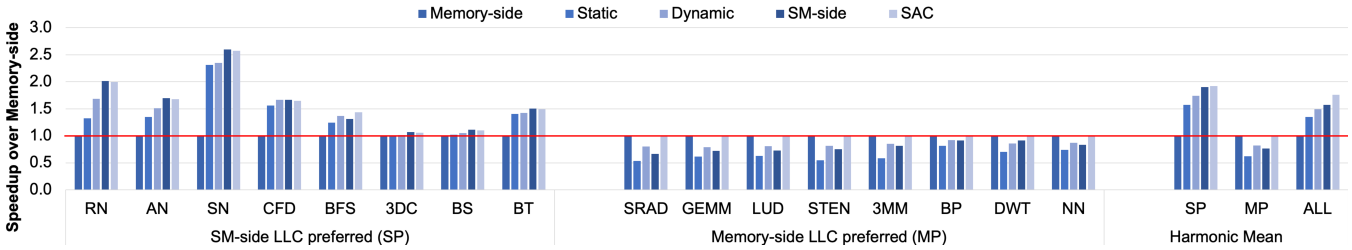


Figure 8: Speedup for the different LLC organizations relative to the memory-side LLC. *The sharing-aware LLC outperforms the alternative LLC organizations across a broad set of benchmarks.*

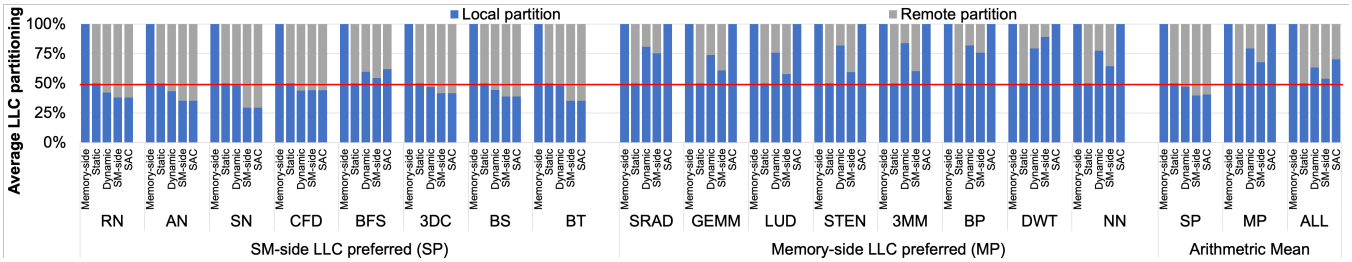


Figure 9: Quantifying how the different LLC organizations cache local versus remote data. *SAC allocates a large fraction of remote data for the SM-side preferred benchmarks, while only allocating local data for the memory-side preferred benchmarks.*

performance gap between SAC and the SM-side LLC for the SM-side preferred benchmarks is primarily a result of the SAC profiling and reconfiguration overhead, and not the inherent bandwidth and latency benefit of the two-NoC SM-side LLC because of the order-of-magnitude difference in intra-chip LLC bandwidth compared to the inter-chip bandwidth.

To understand why SAC outperforms the alternative LLC organizations, Figure 9 reports the fraction of the LLC that caches local data (from the local chip's memory partition) versus remote data (from a remote chip's memory partition). A memory-side LLC, by definition, only caches local data. The Static LLC caches 50% local data and 50% remote data. The other three LLC organizations exhibit different behaviors for the SM-side versus memory-side preferred benchmarks. For the former (except BFS, see later), the Dynamic LLC caches more remote data than the Static LLC, and the SM-side LLC and SAC allocate even more remote data. This leads to an increase in effective memory bandwidth. For the memory-side preferred benchmarks, the Dynamic and SM-side LLC cache more local data than the Static LLC. SAC only caches local data, and no remote data is cached.

The key conclusion is that a workload fundamentally prefers *either* a memory-side organization *or* an SM-side organization. Alternative LLC organizations that cache partly local and partly remote data are hence suboptimal. The memory-side LLC is unable to cache remote data locally, while the SM-side LLC caches some remote data, which is detrimental for the SM-side and memory-side preferred benchmarks, respectively. The Static LLC is too rigid: it is either unable to allocate enough remote data (for the SM-side preferred benchmarks) or it allocates too much remote data (for the memory-side preferred benchmarks). The Dynamic LLC performs slightly better than the Static LLC, however, its heuristic leads to a local optimum in which the LLC does not allocate enough local data. SAC fundamentally solves this problem by choosing one of the two extremes, a memory-side versus an SM-side LLC configuration.

5.2 Effective LLC Bandwidth

The performance speedup obtained through SAC correlates strongly with the effective LLC bandwidth, as reported in Figure 10.[2] Improving the effective LLC bandwidth leads to a corresponding improvement in overall performance. Figure 10 further breaks down the improvement in effective LLC bandwidth by classifying the LLC responses from where they originate, namely the local LLC, a remote LLC, the local memory partition, versus a remote memory partition. For the SM-side preferred benchmarks, SAC trades remote LLC accesses for local LLC accesses. This leads to a significant improvement in effective LLC bandwidth because of the higher intra-chip versus inter-chip network bandwidth. For the memory-side preferred benchmarks, SAC maintains high effective bandwidth to the local LLC or memory partition as opposed to the alternative LLC organizations.

5.3 Sharing Behavior

We now dive deeper to understand the root cause of the improvement in effective memory bandwidth. Figure 11 reports the working set size (in MB and capped at 32 MB) under the SM-side LLC organization for different time windows of 1,000 to 100,000 cycles. The red line denotes the system's total LLC capacity (16 MB). The working set is categorized in three groups: (1) true-sharing, (2) false-sharing, and (3) non-sharing, as previously described in Section 2.1.

[2]The effective memory access latency also improves substantially under SAC, and while there is a strong correlation with the overall performance improvement, the correlation is not as strong as for the effective LLC bandwidth, which is why we focus on the latter. The correlation is weaker because latency gets exposed only when there is insufficient bandwidth.

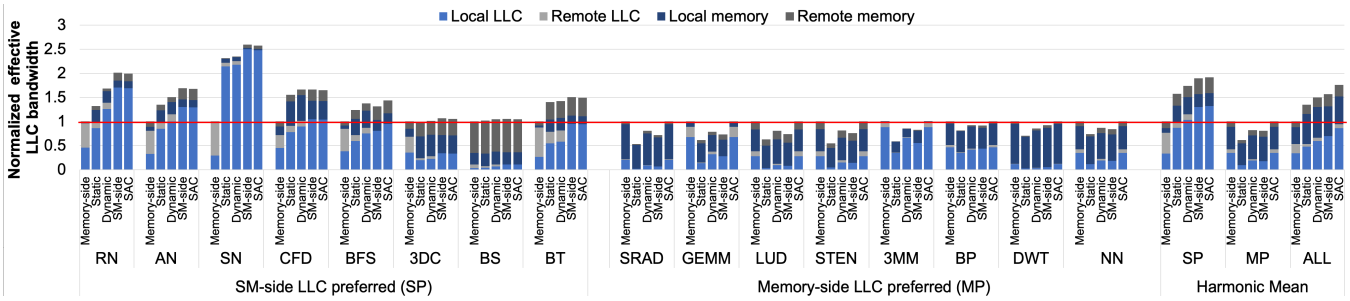


Figure 10: Normalized effective LLC bandwidth breakdown showing the number of LLC responses per cycle including their origin. *SAC improves the effective LLC bandwidth which explains the performance speedup reported in Figure 8.*

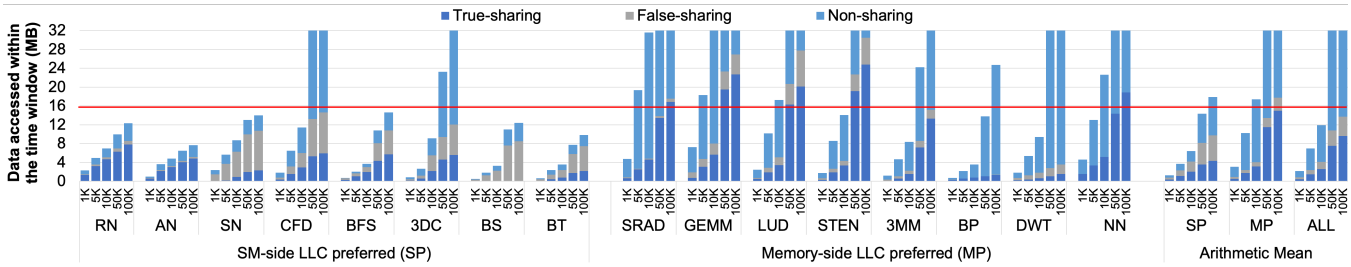


Figure 11: Working set size (in MB, capped at 32 MB) for different time windows under the SM-side LLC organization. The red line denotes the system total LLC capacity of 16 MB. *Replicating the shared data set does not exceed LLC capacity for the SM-side preferred benchmarks in contrast to the memory-side preferred benchmarks.*

There are several interesting observations to make here. First, the truly shared working set is relatively small for the SM-side preferred benchmarks while it is large for the memory-side preferred benchmarks. Note that the truly shared working set gets replicated under an SM-side LLC organization as different chips are accessing the same cache lines. Replicating a relatively small truly shared working set across chips does not pressure cache capacity while significantly improving the effective bandwidth to these cache lines. This explains the performance improvement under SAC for several of the SM-side preferred benchmarks, e.g., RN, AN, CFD and BFS. In contrast, the memory-side preferred benchmarks have a fairly large truly shared working set. Replicating this truly shared working set across chips exceeds the total LLC capacity over large time windows, which leads to cache trashing and explains why these benchmarks prefer a memory-side LLC organization.

Second, most of the SM-side preferred benchmarks have a fairly large false-sharing working set. A chip caches the false-sharing working set locally under an SM-side organization which leads to higher effective bandwidth compared to a memory-side organization which needs to access the false-sharing working set across the inter-chip network.

Third, some benchmarks are atypical in the sense that the performance difference between a memory-side and SM-side LLC is minor, see in particular 3DC, BS, BP and DWT. Recall that the EAB model opts for the SM-side LLC organization in case the predicted bandwidth improvement is larger than for the memory-side LLC (by more than $\theta = 5\%$).

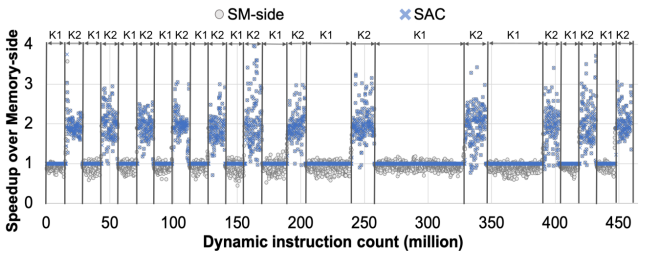


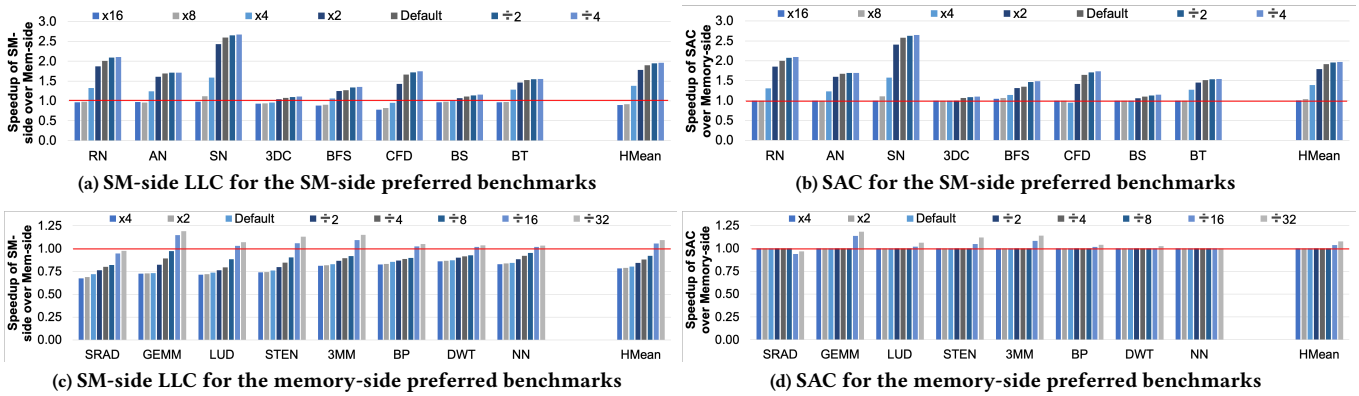
Figure 12: Time-varying execution behavior for BFS: performance for SM-side and SAC relative to the memory-side LLC. *SAC selects the optimum LLC organization on a per-kernel basis, i.e., memory-side LLC for K1 and SM-side LLC for K2.*

5.4 Time-Varying Behavior

As noted in Figure 8, SAC slightly under-performs compared to the SM-side LLC for the SM-side preferred benchmarks, the reason being the profiling and runtime overhead (less than 1%) to dynamically reconfigure the LLC from a memory-side to an SM-side organization. The one exception is BFS, for which SAC outperforms the SM-side LLC. Figure 12 illustrates that BFS' execution alternates between a memory-side preferred kernel K1 and an SM-side preferred kernel K2. SAC chooses the optimum organization on a per-kernel basis which leads to the overall improvement over the SM-side configuration.

5.5 Input Set

Given that the primary reason why SAC outperforms the alternative LLC organizations originates from a workload's shared working set relative to its total working set, it is important to evaluate SAC's



(a) SM-side LLC for the SM-side preferred benchmarks

(b) SAC for the SM-side preferred benchmarks

(c) SM-side LLC for the memory-side preferred benchmarks

(d) SAC for the memory-side preferred benchmarks

Figure 13: Analyzing input set sensitivity for the SM-side preferred benchmarks: (a) SM-side LLC vs. (b) SAC; and for the memory-side preferred benchmarks: (c) SM-side LLC vs. (d) SAC. *SAC selects the optimum LLC organization across inputs.*

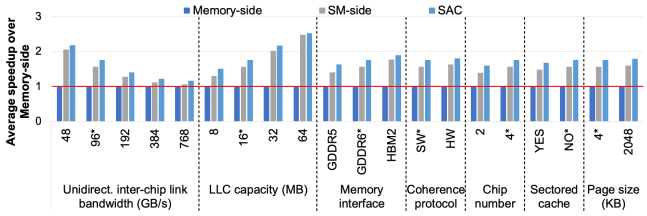


Figure 14: SAC sensitivity relative to the memory-side LLC. The configurations with an asterisk denote our baseline. *SAC performs well across the broad design space.*

sensitivity to input size. Figure 13 reports speedup relative to the memory-side LLC for the SM-side and SAC LLC configurations across a wide range of input set sizes. For the SM-side preferred benchmarks, we change the input set size from ×8 to ÷4 the default input. For the memory-side preferred benchmarks, we change the input from ×4 to ÷32 the default input. For the benchmarks for which it is impossible to alter the input (i.e., RN, AN, SN and BT), we take an alternative approach and scale the LLC capacity. The key take-away from Figure 13 is that SAC chooses the optimum LLC organization across inputs. For the SM-side preferred benchmarks (Figure 13a and 13b), SAC selects the SM-side LLC organization when beneficial while reverting to a memory-side LLC for the largest input sizes, i.e., replicating the increasingly large shared working set leads to cache thrashing effects under an SM-side configuration. For the memory-side preferred benchmarks (Figure 13c and 13d), SAC selects the SM-side LLC organization for the smallest inputs when it is beneficial to do so, i.e., the shared working set is small enough so that replication does not thrash the cache.

5.6 Sensitivity Analyses

We now explore SAC across the design space, see Figure 14.

Inter-chip bandwidth. SAC outperforms the memory-side and SM-side LLC across a broad range of inter-module link bandwidth configurations. The smaller 48 GB/s unidirectional link bandwidth configuration corresponds to PCIe [40]. The second and third generation of NVLink [34] offer 150 GB/s and 300 GB/s bidirectional link bandwidth, respectively, in a 4-GPU setup organized in a ring topology. Our default configuration (96 GB/s unidirectional link bandwidth) is situated in-between these two NVLink systems. The higher 384 GB/s and 768 GB/s unidirectional link configurations are in the range of interposer-based multi-chip-module (MCM) GPUs [2]. Overall, the performance improvement by SAC relative to the memory-side LLC decreases with increasing inter-chip bandwidth: the higher the inter-chip bandwidth, the less critical it is to cache remote data locally and to replicate shared data.

LLC capacity. Larger LLC capacity increases the performance improvement by SAC (and the SM-side LLC) compared to the memory-side LLC. The reason is that a larger LLC enables caching more data locally and replicating a larger shared working set across chips.

Memory interface. SAC's performance improvement over the memory-side LLC increases with increasing memory bandwidth. Increased memory bandwidth from GDDR5 [23] to GDDR6 [24] and HBM2 [15] shifts the system bottleneck to the inter-chip network. Replicating the shared data set is hence more critical for higher memory bandwidth systems, which leads to higher performance benefits through SAC.

Coherence protocol. Hardware coherence keeps cache lines consistent across the system by tracking sharers and invalidating remote copies upon a write, whereas software coherence relies on flush operations to write back dirty cache lines upon software-inserted synchronization primitives. Software coherence is currently the prevalent approach in commercial products, while hardware coherence has been evaluated in research on multi-GPU systems [36]. We evaluate SAC under hardware coherence which updates the local copy while invalidating all other copies in the system upon a write; this requires a dirty bit to be added in the directory[3]. SAC provides a slightly higher performance benefit under hardware coherence compared to software coherence, the reason being lower overhead for reconfiguring the LLC.

GPU count. To evaluate SAC's sensitivity to GPU count, we set up an experiment in which we reduce the number of GPUs from four

[3]This is different from HMG [36] which assumes that the local copy as well as the copy in the home GPU are updated and all other copies are invalidated. The reason for choosing a different implementation is to avoid the unnecessary write traffic and wastage of LLC capacity caused by false sharing — a problem also recognized in [36].

to two, while keeping the total inter-chip bandwidth unchanged. This implies that the per-link bandwidth in the two-GPU setup is effectively twice the per-link bandwidth in the four-GPU setup — this is similar to how inter-GPU bandwidth scales for Nvidia's NVLink [34]. We find that SAC's effectiveness increases with GPU count. The reason is that as the number of GPUs increases, the amount of per-GPU bandwidth decreases which increases the opportunity for SAC to increase the effective bandwidth.

Sectored cache. Sectored caches [18, 21, 38] reduce tag overhead by having cache lines share tag information per sector. Because the effective cache line size is smaller in a sectored cache, there is less opportunity for true sharing. Nevertheless, SAC still provides a significant performance benefit.

Page size. SAC is largely insensitive to page size as it only affects false sharing, not true sharing which is key for SAC.

6 RELATED WORK

SAC focuses on maximizing the effective LLC bandwidth in multi-chip GPU systems. The most closely related works to SAC are hence Dynamic LLC partitioning [25] and the L1.5 cache [2] (i.e., Static LLC), which we have shown are outperformed by SAC. Page migration [4, 10] improves beyond-LLC bandwidth by migrating pages to the memory module of the local chip. Memory management techniques such as CARVE [47] and GPS [27] store fine-grained remote data in local memory and hence replicate shared data across memory modules. All these techniques aim at improving memory bandwidth (i.e., beyond-LLC) whereas SAC increases the effective bandwidth ahead of the LLC.

A number of works have focused on improving how GPUs handle shared data (which is common in GPUs [45]). In particular, the Adaptive LLC [48] and SelRep LLC [49] replicate shared read-only cache blocks across LLC slices within a single GPU to avoid congestion when many SMs access the same cache block around the same time. Note that the Adaptive and SelRep LLC proposals only replicate cache lines in the LLC of the home GPU, because these schemes assume a memory-side LLC. By consequence, other GPUs still need to access those shared cache lines over the inter-chip links, which leads to bandwidth congestion issues. SAC overcomes exactly this inter-chip bandwidth bottleneck by dynamically reconfiguring the LLC between memory-side versus SM-side with the latter configuration enabling replication and caching of truly and falsely shared cache lines across GPUs.

Locality-Aware Data Management (LADM) [17] coordinates the placement of data and CTAs to reduce data sharing between GPU chips. LADM builds upon the Dynamic LLC [25] and proposes a compiler-assisted 'cache-remote-once' cache insertion policy to avoid wasting cache capacity for falsely shared cache blocks in the Dynamic LLC's remote partition. By doing so, LADM is in effect similar to SM-side caching. SAC makes the observation that an additional performance improvement is to be obtained by dynamically reconfiguring the LLC to be a memory-side or SM-side cache.

Ibrahim et al. [14] propose an L1 cache organization that is shared by all SMs. A shared L1 is orthogonal to SAC by maximizing the effective L1 bandwidth within a GPU chip, whereas SAC aims at maximizing the effective LLC bandwidth across GPU chips in a multi-GPU system. While a workload with data sharing should prefer a shared L1 organization (to eliminate data replication) [14], the conclusion is different at the multi-GPU level: if the shared data set is small enough relative to the LLC size, it is beneficial to replicate across chips under an SM-side configuration — higher LLC miss rate but higher effective bandwidth — however, if the shared data is too big to fit in the cache, it is better not to replicate under a memory-side configuration.

A large body of prior work has investigated various LLC organizations in the context of multi-core CPUs [5, 11, 19, 46]. These approaches are fundamentally different from GPU-focused LLC organizations as they focus on placing data close to the requesting core(s) to minimize latency whereas maximizing bandwidth is the critical concern in GPUs.

7 CONCLUSION

We presented Sharing-Aware Caching (SAC) which adopts either a memory-side or an SM-side LLC organization depending on which of the two configurations it predicts will provide higher bandwidth. The key components of SAC are (1) the simple EAB analytical model that takes into account the degree and nature of data sharing in the workload, and (2) a low-overhead routing-based LLC reconfiguration mechanism. SAC improves performance by 76%, 12%, 31%, and 18% on average compared to a memory-side LLC baseline, an SM-side LLC configuration, the L1.5 static LLC organization [2], and dynamic LLC partitioning [25], respectively.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. Shiqing Zhang is supported by a CSC scholarship (Grant No. 201903-170128), and Magnus Jahre is supported by the Research Council of Norway (Grant No. 286596). Lieven Eeckhout is supported in part by the UGent-BOF-GOA grant No. 01G01421, and the European Research Council (ERC) Advanced Grant agreement No. 741097.

REFERENCES

[1] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-Layer CNN Accelerators. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[2] Akhil Arunkumar, Evgeny Bolotin, Benjamin Cho, Ugljesa Milic, Eiman Ebrahimi, Oreste Villa, Aamer Jaleel, Carole-Jean Wu, and David Nellans. 2017. MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 320–332.

[3] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. 2009. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 163–174.

[4] Trinayan Baruah, Yifan Sun, Ali Tolga Dinçer, Saiful A Mojumder, José L Abellán, Yash Ukidave, Ajay Joshi, Norman Rubin, John Kim, and David Kaeli. 2020. Griffin: Hardware-Software Support for Efficient Page Migration in Multi-GPU Systems. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 596–609.

[5] Bradford M Beckmann, Michael R Marty, and David A Wood. 2006. ASR: Adaptive Selective Replication for CMP Caches. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 443–454.

[6] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*. IEEE, 44–54.

[7] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. *ACM SIGARCH Computer Architecture News* 42, 1 (2014), 269–284.

[8] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[9] William J Dally, C Thomas Gray, John Poulton, Brucek Khailany, John Wilson, and Larry Dennison. 2018. Hardware-Enabled Artificial Intelligence. In *Proceedings of the International Symposium on VLSI Technology and Circuits (VLSI)*. IEEE, 3–6.

[10] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. 2013. Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems. *ACM SIGPLAN Notices* 48, 4 (2013), 381–394.

[11] Babak Falsafi and David A Wood. 1997. Reactive NUMA: A Design for Unifying S-COMA and CC-NUMA. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 229–240.

[12] John Cavazos Scott Grauer-Gray. 2015. PolyBench/GPU 1.0. https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/. [Online; accessed 2022-04-16].

[13] John L Hennessy and David A Patterson. 2017. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers.

[14] Mohamed Assem Ibrahim, Onur Kayiran, Yasuko Eckert, Gabriel H Loh, and Adwait Jog. 2020. Analyzing and Leveraging Shared L1 Caches in GPUs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 161–173.

[15] JEDEC. 2021. High Bandwidth Memory (HBM) DRAM. https://www.jedec.org/standards-documents/docs/jesd235a. [Online; accessed 2022-04-16].

[16] Aajna Karki, Chethan Palangotu Keshava, Spoorthi Mysore Shivakumar, Joshua Skow, Goutam Madhukeshwar Hegde, and Hyeran Jeon. 2019. Detailed Characterization of Deep Neural Networks on GPUs and FPGAs. In *Proceedings of the Workshop on General Purpose Processing Using GPUs (GPGPU)*. 12–21.

[17] Mahmoud Khairy, Vadim Nikiforov, David Nellans, and Timothy G Rogers. 2020. Locality-Centric Data and Threadblock Management for Massive GPUs. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 1022–1036.

[18] Mahmoud Khairy, Zhesheng Shen, Tor M Aamodt, and Timothy G Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 473–486.

[19] George Kurian, Srinivas Devadas, and Omer Khan. 2014. Locality-Aware Data Replication in the Last-Level Cache. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1–12.

[20] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. 2008. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro* 28, 2 (2008), 39–55.

[21] J. S. Liptay. 1968. Structural Aspects of the System/360 Model 85, II: The Cache. *IBM Systems Journal* 7, 1 (1968), 15–21.

[22] Yuxi Liu, Xia Zhao, Magnus Jahre, Zhenlin Wang, Xiaolin Wang, Yingwei Luo, and Lieven Eeckhout. 2018. Get Out of the Valley: Power-Efficient Address Mapping for GPUs. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 166–179.

[23] Micron. 2014. High Performance, High Bandwidth: GDDR5 for Networking. https://media-www.micron.com/-/media/client/global/documents/products/product-flyer/flyer_gddr5_networking.pdf. [Online; accessed 2022-04-16].

[24] Micron. 2021. Technical Note: GDDR6 Design Guide. https://media-www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn-ed-04_gddr6_design_guide.pdf. [Online; accessed 2022-04-16].

[25] Ugljesa Milic, Oreste Villa, Evgeny Bolotin, Akhil Arunkumar, Eiman Ebrahimi, Aamer Jaleel, Alex Ramirez, and David Nellans. 2017. Beyond the Socket: NUMA-Aware GPUs. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 123–135.

[26] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. 2007. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 3–14.

[27] Harini Muthukrishnan, Daniel Lustig, David Nellans, and Thomas Wenisch. 2021. GPS: A Global Publish-Subscribe Model for Multi-GPU Memory Management. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 46–58.

[28] Nvidia. 2016. NVIDIA DGX-1: Essential Instrument of AI Research. https://www.nvidia.com/en-gb/data-center/dgx-systems/dgx-1/. [Online; accessed 2022-04-16].

[29] Nvidia. 2016. NVIDIA Tesla P100: The Most Advanced Datacenter Accelerator Ever Built. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf. [Online; accessed 2022-04-16].

[30] Nvidia. 2018. NVIDIA DGX-2: Break Through the Barriers to AI Speed and Scale. https://www.nvidia.com/en-us/data-center/dgx-2/. [Online; accessed 2022-04-16].

[31] Nvidia. 2020. NVIDIA A100 Tensor Core GPU Architecture: The World's Most Advanced Data Center GPU. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf. [Online; accessed 2022-04-16].

[32] Nvidia. 2020. NVIDIA cuDNN. https://developer.nvidia.com/cudnn. [Online; accessed 2022-04-16].

[33] Nvidia. 2022. NVIDIA CUDA SDK Code Samples. https://docs.nvidia.com/cuda/cuda-samples/index.html. [Online; accessed 2022-04-16].

[34] Nvidia. 2022. NVIDIA NVLINK: High-speed GPU Interconnect. https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/. [Online; accessed 2022-04-16].

[35] Moinuddin K Qureshi, Daniel N Lynch, Onur Mutlu, and Yale N Patt. 2006. A Case for MLP-Aware Cache Replacement. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 167–178.

[36] Xiaowei Ren, Daniel Lustig, Evgeny Bolotin, Aamer Jaleel, Oreste Villa, and David Nellans. 2020. HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 582–595.

[37] Timothy G Rogers, Mike O'Connor, and Tor M Aamodt. 2012. Cache-Conscious Wavefront Scheduling. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 72–83.

[38] A. Seznec. 1994. Decoupled Sectored Caches: Conciliating Low Tag Implementation Cost and Low Miss Ratio. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. 384–393.

[39] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 14–27.

[40] Debendra Das Sharma. 2020. PCI Express® 6.0 Specification: A Low-Latency, High-Bandwidth, High-Reliability, and Cost-Effective Interconnect With 64.0 GT/s PAM-4 Signaling. In *Proceedings of the Symposium on High-Performance Interconnects (HOTI)*. IEEE, 1–8.

[41] Addison Snell and Laura Segervall. 2017. *HPC Application Support for GPU Computing*. Technical Report. Intersect360 Research.

[42] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. 2012. *Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing*. Technical Report. IMPACT-12-01, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign. 29 pages.

[43] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. 2012. 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)*. IEEE, 201–210.

[44] Synopsys. 2022. DesignWare Library - Datapath and Building Block IP. https://www.synopsys.com/dw/buildingblock.php. [Online; accessed 2023-02-15].

[45] Abdulaziz Tabbakh, Murali Annavaram, and Xuehai Qian. 2017. Power Efficient Sharing-Aware GPU Data Management. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 698–707.

[46] Po-An Tsai, Nathan Beckmann, and Daniel Sanchez. 2017. Nexus: A New Approach to Replication in Distributed Shared Caches. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 166–179.

[47] Vinson Young, Aamer Jaleel, Evgeny Bolotin, Eiman Ebrahimi, David Nellans, and Oreste Villa. 2018. Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 339–351.

[48] Xia Zhao, Almutaz Adileh, Zhibin Yu, Zhiying Wang, Aamer Jaleel, and Lieven Eeckhout. 2019. Adaptive Memory-Side Last-Level GPU Caching. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 411–423.

[49] Xia Zhao, Magnus Jahre, and Lieven Eeckhout. 2020. Selective Replication in Memory-Side GPU Caches. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 967–980.