Adaptive Memory-Side Last-Level GPU Caching

Xia Zhao
Ghent University, Belgium

Almutaz Adileh
Ghent University, Belgium

Zhibin Yu
Shenzhen Institutes of Advanced Technology, CAS, China

Zhiying Wang
National University of Defense Technology, China

Aamer Jaleel
Nvidia, USA

Lieven Eeckhout
Ghent University, Belgium

ABSTRACT

Emerging GPU applications exhibit increasingly high computation demands which has led GPU manufacturers to build GPUs with an increasingly large number of streaming multiprocessors (SMs). Providing data to the SMs at high bandwidth puts significant pressure on the memory hierarchy and the Network-on-Chip (NoC). Current GPUs typically partition the memory-side last-level cache (LLC) in equally-sized slices that are shared by all SMs. Although a shared LLC typically results in a lower miss rate, we find that for workloads with high degrees of data sharing across SMs, a private LLC leads to a significant performance advantage because of increased bandwidth to replicated cache lines across different LLC slices.

In this paper, we propose adaptive memory-side last-level GPU caching to boost performance for sharing-intensive workloads that need high bandwidth to read-only shared data. Adaptive caching leverages a lightweight performance model that balances increased LLC bandwidth against increased miss rate under private caching. In addition to improving performance for sharing-intensive workloads, adaptive caching also saves energy in a (co-designed) hierarchical two-stage crossbar NoC by power-gating and bypassing the second stage if the LLC is configured as a private cache. Our experimental results using 17 GPU workloads show that adaptive caching improves performance by 28.1% on average (up to 38.1%) compared to a shared LLC for sharing-intensive workloads. In addition, adaptive caching reduces NoC energy by 26.6% on average (up to 29.7%) and total system energy by 6.1% on average (up to 27.2%) when configured as a private cache. Finally, we demonstrate through a GPU NoC design space exploration that a hierarchical two-stage crossbar is both more power- and area-efficient than full and concentrated crossbars with the same bisection bandwidth, thus providing a low-cost cooperative solution to exploit workload sharing behavior in memory-side last-level caches.

CCS CONCEPTS

• **Computer systems organization** → **Single instruction, multiple data**.

ISCA '19, June 22–26, 2019, PHOENIX, AZ, USA

ACM ISBN 978-1-4503-6669-4/19/06…$15.00
https://doi.org/10.1145/3307650.3322235

ACM Reference Format:
Xia Zhao, Almutaz Adileh, Zhibin Yu, Zhiying Wang, Aamer Jaleel, and Lieven Eeckhout. 2019. Adaptive Memory-Side Last-Level GPU Caching. In *The 46th Annual International Symposium on Computer Architecture (ISCA '19), June 22–26, 2019, PHOENIX, AZ, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3307650.3322235

1 INTRODUCTION

Graphics processing units (GPUs) are widely used to accelerate a wide range of emerging throughput-oriented applications, e.g., machine learning and data analytics [1–3]. GPUs rely on streaming multiprocessors (SMs) to run a massive number of parallel threads, grouped in cooperative thread arrays (CTAs). To match the rising computational demands of GPU-compute applications, the number of SMs and the available memory bandwidth have both been steadily increasing with successive GPU generations. For example, Nvidia scaled memory bandwidth from 177 GB/s to 900 GB/s while scaling the number of SMs from 14 in Fermi [4] to 80 in Volta [5]. The large number of SMs on the one side and the high memory bandwidth on the other side increases pressure on the cache hierarchy and the Network-on-Chip (NoC).

GPUs typically feature a two-level on-chip cache hierarchy in which the first-level caches are private to each SM while the last-level cache (LLC) is a shared memory-side cache that is partitioned into equally-sized slices and accessed via the NoC. As an intermediary component between the SMs and main memory, it is essential to properly design the LLC and NoC for high-throughput GPUs. While countless research efforts have optimized the cache hierarchy for multicore CPUs [6–25], there is a fundamental difference between optimizing the cache hierarchy for a GPU versus a CPU.

Unlike latency-sensitive CPUs, GPUs desire high bandwidth access to application data. The traditional CPU solution of designing a banked large shared LLC (to accommodate large application data footprints) can create bandwidth bottlenecks for a GPU. This is especially the case when multiple SMs concurrently access shared data resident in the shared LLC. In fact, we find that GPU workloads have large read-only shared data footprints. For such *sharing-intensive* workloads, multiple SMs experience a bandwidth bottleneck when they serialize on accesses to the same shared cache line. In the CPU world, a *CPU-side cache* close to the cores (e.g., L2 cache) can provide high bandwidth access by replicating shared data cache lines. While this is a practical solution for a limited number of cores in a CPU, it does not scale to a large number of SMs due to limitations in scaling GPU die size.

This paper provides a low-cost solution to improve *memory-side last-level cache* performance for sharing-intensive workloads that

demand high bandwidth to shared data. Shared LLCs incur a performance bottleneck for workloads that frequently access data shared by multiple SMs. A shared memory-side LLC consists of multiple slices each caching a specific *memory partition*, i.e., a specific address range of the entire memory space is served by a particular memory controller. As a result, a shared cache line appears in a single LLC slice, which leads to a severe performance bottleneck if multiple SMs concurrently access the same shared data. Requests from different SMs queue up in front of the LLC slice, which leads to long queuing delays, up to the point that these queuing delays can no longer be hidden, ultimately deteriorating overall application performance. The underlying performance bottleneck is a lack of LLC bandwidth to shared data. A potential solution to the bandwidth problem may be to replicate shared data across the different LLC slices to increase the LLC bandwidth to the shared data. A private LLC achieves exactly this although it also leads to higher miss rates because of cache line replication. Because of the conflicting phenomena (higher LLC bandwidth versus increased miss rate), it is unclear whether a shared or private LLC organization is preferred for sharing-intensive workloads.

In this paper, we study private versus shared organizations for memory-side last-level caching in GPUs. We find that GPU applications with high degrees of read-only data sharing significantly benefit from a private LLC organization. Replicating cache lines across the different LLC slices in a private LLC organization leads to increased bandwidth to shared data, as the different copies of the shared cache lines can be accessed in parallel in the different LLC slices. In a shared cache on the other hand, a frequently accessed shared cache line resides in a single LLC slice providing insufficient bandwidth. GPU applications with moderate data sharing tend to benefit from a shared LLC organization because of reduced conflict misses, i.e., shared cache lines are stored only once in one of the LLC slices, improving the effective capacity of the LLC. GPU applications with no data sharing are performance-neutral to a private versus shared LLC organization. These observations suggest an opportunity to improve performance by dynamically adapting a memory-side LLC to the needs of an application's sharing behavior.

To that end, this paper proposes *adaptive memory-side caching* to dynamically choose between a shared or private memory-side LLC. Selecting a shared versus private LLC is done using a lightweight performance model. By default, the GPU assumes a shared LLC. Profiling information is periodically collected to predict LLC miss rate and bandwidth under a private LLC organization while executing under a shared LLC. If deemed beneficial, the LLC is adapted to a private cache. The LLC reverts back to a shared organization periodically and when a new kernel gets launched.

Adaptive last-level caching not only improves performance for workloads that need high bandwidth to shared data, it also provides an opportunity to save energy if the NoC is co-designed with the memory hierarchy. In particular, we propose a reconfigurable hierarchical two-stage crossbar NoC in which the first-stage crossbars, the *SM-routers*, are connected to disjoint clusters of SMs, and the second-stage crossbars, the *MC-routers*, are connected to the memory controllers via the LLC slices. NoC/LLC co-design involves that the number of MC-routers equals the number of memory controllers, and the number of SM-routers (or the number of clusters) equals the number of LLC slices per memory controller. By power-gating and bypassing the MC-routers, the SMs are connected to a subset of the LLC slices that are private to the respective SM cluster.

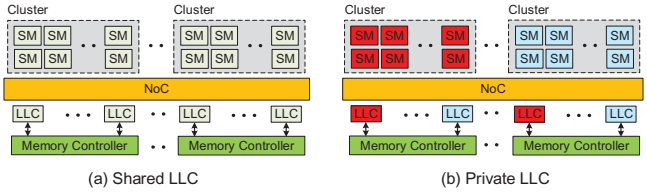


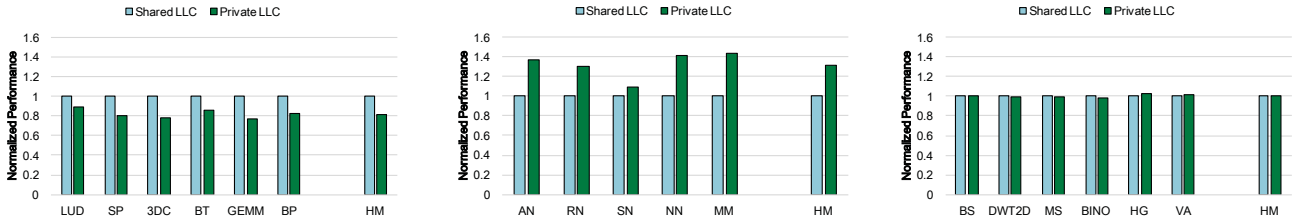
Figure 1: Shared versus private memory-side LLC organization in GPUs. *(a) In a shared LLC organization, each LLC slice is shared by all SMs. (b) In a private LLC organization, an LLC slice is private to a cluster of SMs. In either case, a slice in a memory-side LLC only caches the memory partition served by the respective memory controller.*

In summary, this paper makes the following contributions:

- We study private versus shared organizations in memory-side last-level GPU caches and make the observation that sharing-intensive GPU workloads favor a private LLC over a shared LLC because a private LLC provides higher bandwidth to read-only shared cache lines due to data replication in the different LLC slices, which increases LLC response rate and improves overall performance.
- We propose adaptive memory-side caching to dynamically reconfigure the LLC between shared versus private modes depending on the workload's execution characteristics. LLC adaptation is low-cost and is driven by a lightweight performance model that estimates private LLC miss rate and bandwidth while executing under shared caching. Adaptive caching enables saving power in a co-designed hierarchical two-stage crossbar NoC by power-gating and bypassing the second stage if the LLC is configured as a private cache.
- We perform a GPU NoC design space exploration in which we compare full, concentrated and hierarchical two-stage crossbar designs. We conclude that the hierarchical crossbar achieves the highest performance at much lower power and chip area overhead compared to a full and concentrated crossbar with the same bi-section bandwidth. Moreover, the hierarchical two-stage crossbar provides power saving opportunities under adaptive memory-side last-level caching.
- We quantitatively evaluate adaptive memory-side last-level caching and report an average performance improvement of 28.1% (and up to 38.1%) compared to a conventional shared LLC for workloads with high degrees of data sharing. Adaptive caching reduces NoC energy by 26.6% on average (and up to 29.7%) and total system energy by 6.1% on average (up to 27.2%) if the LLC is configured as a private cache.

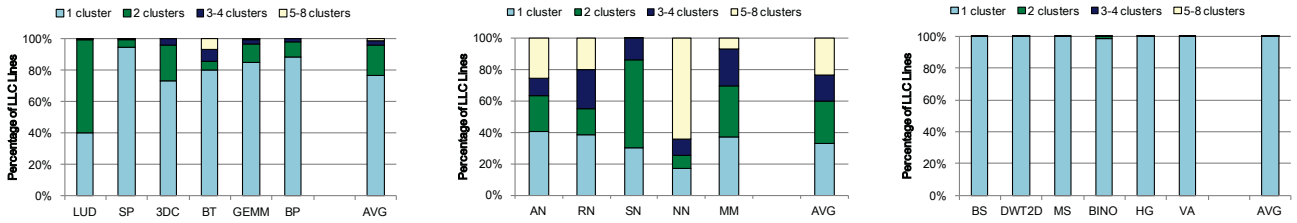
2 SHARED VERSUS PRIVATE LLC

This work is motivated by the observation that some GPU applications benefit from a shared LLC whereas other applications benefit from a private LLC. We now describe and quantitatively compare both memory-side LLC organizations.



(a) Shared cache friendly applications (b) Private cache friendly applications (c) Shared/private cache neutral applications

Figure 2: Normalized performance for a shared versus private LLC. *Shared cache friendly applications prefer shared caching because of a reduced number of conflict misses; private cache friendly applications prefer private caching because of increased LLC bandwidth to shared cache lines in different LLC slices; shared/private cache neutral applications perform equally well under both LLC organizations.*



(a) Shared cache friendly applications (b) Private cache friendly applications (c) Shared/private cache neutral applications

Figure 3: Inter-cluster locality. *Private cache friendly applications tend to exhibit high degrees of inter-cluster sharing; shared/private cache neutral applications lack inter-cluster locality; shared cache friendly applications exhibit some inter-cluster sharing.*

2.1 Memory-Side LLC Organizations

We consider a GPU with a shared and private memory-side LLC organizations as shown in Figure 1. The LLC is divided in equally-sized LLC slices. A slice in a memory-side LLC only caches data for the memory partition served by the respective memory controller.

In the *shared LLC* organization, an LLC slice is shared by all SMs, see Figure 1(a). The LLC slice for a given cache line is determined by a few address bits. Collectively, all LLC slices associated with a given memory controller cache the entire memory address space served by the memory controller.

In the *private LLC* organization, an LLC slice is private to a cluster of SMs. An LLC slice caches the entire memory partition served by the respective memory controller for only a single cluster of SMs. The LLC slice for a cache line is thus determined by the cluster ID. For example, a red LLC slice in Figure 1(b) caches the entire memory partition served by the respective memory controller for the SMs in the red cluster on the left; likewise, the blue LLC slices can only be accessed by the blue cluster on the right. This organization enables each SM to index the entire memory address space under a private LLC organization.

2.2 Workload Characterization

We now evaluate how different workloads are affected by the LLC organization. Of particular interest in this discussion is how shared cache lines are treated in both organizations. A cache line that is shared by different SMs appears only once in a single LLC slice in the shared LLC organization. In the private LLC organization on the other hand, a cache line that is shared by SMs in different clusters appears in as many LLC slices as there are clusters accessing the line. Shared cache lines in the private organization are thus replicated across the different LLC slices. Data replication leads to increased cache interference (conflict misses), but also increases the available bandwidth to the shared cache lines, i.e., the different copies of the same shared cache line in the different LLC slices can be accessed in parallel.

We now discuss how our set of GPU workloads is affected by the private versus shared LLC organizations. In this evaluation, we assume a total of 80 SMs, organized in 8 clusters of 10 SMs each, and 64 LLC slices. An important characteristic of the GPU workloads considered in this study is that the shared data footprint tends to be read-only and requires hundreds of KB to several MBs in storage space. Because of its large size, the shared data cannot reside (and be replicated) in the L1 caches. As a result, a significant fraction of the shared data tends to be serviced by the LLC. (See Section 5 for more details about our experimental setup, including a discussion of the shared data footprint size for each benchmark.)

Figure 2 reports performance for a private memory-side LLC normalized to a shared memory-side LLC. We classify the different workloads in three categories depending on their performance for the shared versus private LLC organization, namely shared cache friendly, private cache friendly, and shared/private cache neutral. To gain additional insight regarding the performance differences between the shared versus private LLC organizations, Figure 3 quantifies inter-cluster locality or the degree of sharing at the cluster level

for the shared LLC. We group the SMs per cluster as for the private LLC organization, and quantify what fraction of the last-level cache lines in the shared LLC are accessed by a single cluster, 2 clusters, 3 or 4 clusters, or 5 to 8 clusters, in a time window of 1,000 cycles.

We observe that the *private cache friendly applications* tend to exhibit high degrees of inter-cluster locality, i.e., more than 60% of the LLC cache lines are accessed by SMs in different clusters, see Figure 3(b). This leads to a bandwidth bottleneck due to serialized accesses in a shared LLC organization — different SM clusters want to access the same cache lines at the same time. Replicating shared lines across LLC slices in a private LLC organization provides more bandwidth to the shared cache lines. The increase in effective LLC bandwidth offsets the increase in miss rate due to shared cache line replication, which leads to higher performance under private caching. The *shared cache friendly applications* on the other hand exhibit much less sharing among clusters, by slightly more than 20% on average, see Figure 3(a); the only exception is LUD, however, this application suffers from a 3× increase in LLC miss rate under the private LLC organization. In general, we find that the LLC miss rate increases by 2× on average for the shared cache friendly applications under the private LLC organization. Although these workloads exhibit some inter-cluster locality, and benefit from increased LLC bandwidth to shared cache lines in a private LLC, replication of the large shared data footprint across different LLC slices increases the LLC miss rate substantially, which ultimately leads to lower performance for the private LLC. Finally, the *shared/private cache neutral applications* lack inter-cluster locality, see Figure 3(c), and hence are neutral to the LLC organization.

The overall conclusion is that whether an application prefers a shared versus private LLC organization depends on its sharing characteristics. An application with high degrees of inter-cluster sharing benefits from a private LLC. In contrast, applications with no or limited sharing intensity favor a shared LLC organization.

2.3 Adaptive LLC Opportunity

The observations from the previous section suggest an adaptive memory-side LLC in which the organization (shared versus private) is determined on a per-workload basis. The above analysis also illustrates that whether the private LLC is the preferred organization involves a delicate trade-off between increased LLC bandwidth to shared cache lines versus increased miss rate.

Note that the reconfiguration from a shared to private cache can be done in a fairly straightforward way. It suffices to dynamically change the bits that are used to index the LLC. For a shared LLC organization, the target LLC slice is determined by a few address bits of the request. In contrast, for the private LLC, the target LLC slice is determined by the cluster ID from which the request is issued.

Although adaptive memory-side caching can be implemented irrespective of the underlying NoC that connects the SMs to the LLC slices, we find an additional opportunity to save energy consumption in the NoC if the NoC is properly co-designed with the rest of the GPU. In particular, we propose a reconfigurable hierarchical two-stage crossbar NoC in which the second stage, when power-gated and bypassed, configures the adaptive LLC as a private cache.

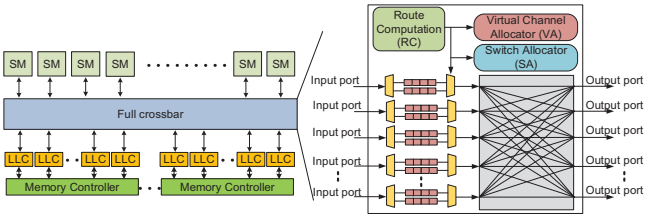


Figure 4: Full Xbar. *Full connectivity is provided between all input and output ports.*

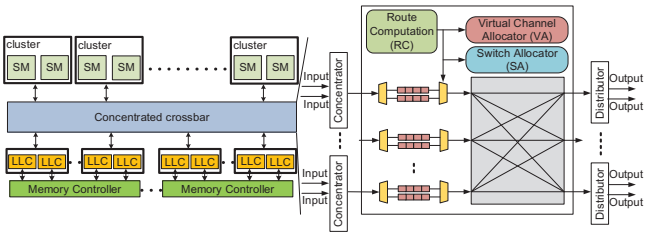


Figure 5: Concentrated Xbar. *Input and output ports are concentrated and distributed, respectively, to reduce switch complexity.*

3 GPU CROSSBAR NoC DESIGN SPACE

We now further motivate the hierarchical two-stage crossbar (H-Xbar) by contrasting it against a full crossbar and a concentrated crossbar (C-Xbar) in terms of performance, power, and chip area.

3.1 Full Crossbar

Figure 4 shows the full crossbar that connects the SMs to the LLC slices. Note that a GPU NoC in fact consists of two networks, the request network for sending requests from the SMs to the memory side and the reply network for sending replies from the memory side to the SMs. The fully connected crossbars only provide the connection between SMs and LLC slices, and vice versa. There is no communication between the SMs; likewise there is no communication among the LLC slices. The right part of Figure 4 shows the micro-architecture of a high-radix router that consists of input ports with input buffers or virtual channels (VCs), a route computation unit, a virtual channel allocator, a switch allocator and a crossbar switch that connects the input and output ports.

3.2 Concentrated Crossbar

The physical size of a full crossbar grows quadratically with port count. The area and power overhead of a full crossbar makes it impractical to build [26]. To improve cost-efficiency, concentration can be devised where SMs and LLC slices are grouped in a cluster to share one network port through a concentrator and distributor, respectively [27]. Figure 5 shows a concentrated crossbar, or *C-Xbar* for short, with a concentration of two, which means that two SMs and two LLC slices share one network port, respectively. When network contention happens, e.g., two SMs want to send packets through the concentrator at the same time, the allocator follows a round-robin policy to determine which input port can send.

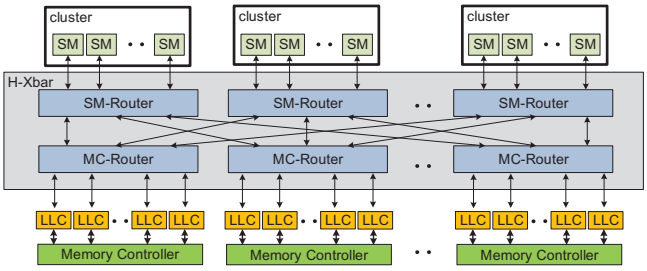


Figure 6: Hierarchical Xbar. *A two-stage crossbar network connects the SMs and LLC slices per cluster and memory controller, respectively.*

3.3 Hierarchical Crossbar

Multi-stage networks have been widely explored in the CPU world [26]. GPUs differ from CPUs though in the sense that the NoC does not need to provide connectivity between all network ports. As mentioned before, in a GPU, there is only communication between SMs and LLC slices, and vice versa, hence naively deploying CPU-optimized multi-stage networks to the GPU world would lead to many unused connections. Instead, we propose a hierarchical crossbar, or *H-Xbar* for short, which is an adaptation of a traditional 2-stage butterfly network [26] to a GPU context. Figure 6 illustrates the H-Xbar NoC with two stages of routers, the SM-router and the MC-router. The SM-routers connect the SMs to the MC-routers. The MC-routers connect the SM-routers to the LLC slices. In terms of layout, it is natural to physically place an SM-router near its respective cluster of SMs and put the MC-router near its LLC slices, and then use long links to connect the routers. The links to connect the SMs to the SM-routers and the links to connect the LLC slices to the MC-routers are relatively short.

Wormhole switching is employed in which a packet is split in fixed-length flits with a size equal to the network channel width. Credit-based flow control is used to keep track of the input buffer status of downstream ports to guarantee that the downstream router always has the available buffer space before sending a packet. Each flit needs to traverse two hops from source to destination in H-Xbar, in contrast to a single hop in a fully connected crossbar. Although this increases per-packet transfer latency, it incurs negligible impact on performance, as we will quantify.

3.4 NoC Design Space Exploration

We now compare the three crossbar designs in terms of performance, power and chip area, see Figure 7. We use GPGPU-Sim [28] for collecting performance numbers and we use DSENT [29] for evaluating chip area and power consumption, assuming a 22 nm chip technology and assuming that all crossbar designs can operate at the same clock frequency. We pair the different designs by bisection bandwidth. A full crossbar has the same bisection bandwidth (BW) as H-Xbar, both with a 32-byte channel width. A concentrated crossbar with concentration of 2 and 32-byte channel width has the same bandwidth (BW/2) as H-Xbar with 16-byte channel width, etc.

Figure 7a reports performance for the various configurations, all normalized to a full crossbar. The key take-away is that *H-Xbar achieves similar performance* as a full or concentrated crossbar

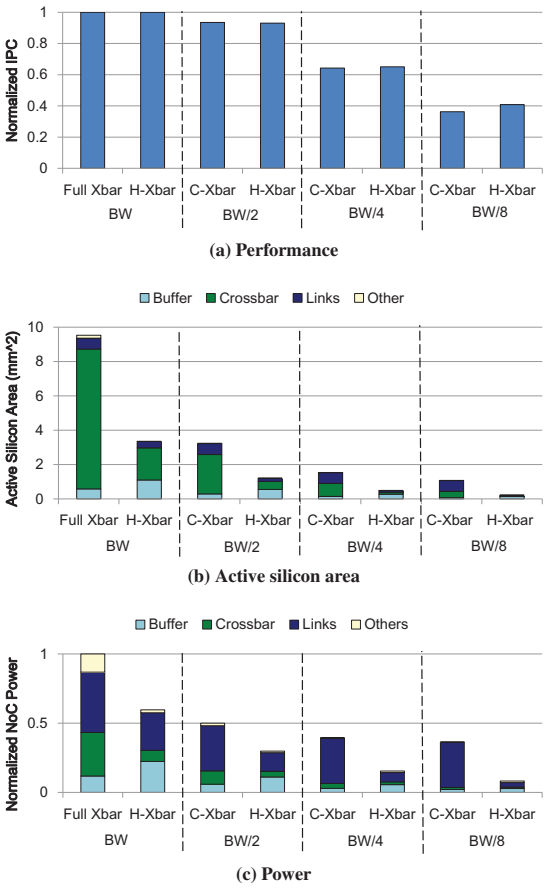


Figure 7: Performance, chip area and power for a full crossbar, C-Xbar and H-Xbar. Design points are grouped by the same bisection bandwidth. *The hierarchical crossbar achieves similar performance as a full and concentrated crossbar with the same bisection bandwidth at much higher area and power efficiency.*

with the same bisection bandwidth. The reason is that GPUs are more sensitive to bandwidth than latency, hence the extra hop count for H-Xbar compared to a concentrated (or full) crossbar does not significantly affect performance. Compared to a C-Xbar with a concentration of 8, H-Xbar achieves even better performance as it does not suffer from contention in shared network ports.

H-Xbar is more area-efficient than a full or concentrated crossbar, see Figure 7b. H-Xbar employs several low-radix crossbars that consume (much) less chip area than the high-radix crossbar used in the full and concentrated crossbars; moreover, compared to the concentrated crossbar, the channel width of the low-radix crossbars in H-Xbar is also smaller for the same bi-section bandwidth. The input buffer area however increases due to the extra input buffers of the second stage in H-Xbar. Overall, the extra buffer space is offset by the large reduction in crossbar switch area, which leads to a substantial net NoC area reduction ranging between 62% and 79%.

H-Xbar is more power-efficient than a full or concentrated crossbar, see Figure 7c. Power consumption mirrors chip area: the increase

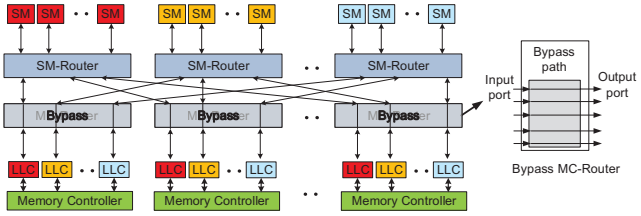


Figure 8: Private LLC in H-Xbar. An LLC slice is private to a cluster of SMs with the same color. *Bypassing the MC-Routers in H-Xbar enables private last-level caching in which an LLC slice is private to a cluster of SMs.*

in buffer power is offset by the reduction in switch power and primarily link power. The short links that connect the SMs and the LLC slices to the SM-routers and MC-routers, respectively, and the long low-bandwidth links between the SM-routers and MC-routers consume much less power than the long high-bandwidth links employed in the full and concentrated crossbars. Overall, the hierarchical crossbar consumes up to 80% less power than a concentrated design.

Because the hierarchical crossbar delivers the same performance as a full and concentrated crossbar, while drastically reducing chip area and power consumption, we consider the H-Xbar design as our baseline NoC for the remainder of this paper.

4 ADAPTIVE LAST-LEVEL CACHING

We now propose adaptive last-level caching, a novel LLC organization that dynamically reconfigures the LLC from a shared to private organization, and vice versa. Although adaptive caching can be implemented for any NoC architecture, in this paper, we specifically focus on adaptive last-level caching in the context of a hierarchical two-stage crossbar.

4.1 NoC-Enabled Adaptive LLC

By default, H-Xbar provides a shared LLC. However, bypassing the second-stage MC-routers, provides a private LLC, as illustrated in Figure 8. In particular, assuming there are as many SM-routers (or clusters) as there are LLC slices per memory controller, directly connecting the input ports to the output ports in the MC-routers leads to a private LLC organization in which an LLC slice is private to a cluster of SMs.

Note that in the private LLC organization, each LLC slice caches the entire address space served by the respective memory controller. This enables all SMs to index the entire memory space, even in the private LLC organization. In our baseline design, we assume full connectivity between the LLC slices and the memory channels per memory controller, e.g., a 3D-stacked high-bandwidth memory (HBM) system with a total of 16–32 memory channels provides full connectivity in the memory controller between the different LLC slices and the different memory channels attached to the memory controller [30]. In case of full connectivity between the LLC slices and the memory channels, providing a private LLC mode does not incur additional overhead. In a design where there is no full connectivity, the overhead of adding full connectivity between 8 LLC slices and 8 channels per memory controller is limited to 0.14% of the entire GPU chip.

NoC/LLC Co-design. Co-designing the NoC and LLC provides an interesting opportunity to save energy by power-gating and bypassing the MC-routers if the adaptive LLC is configured as a private cache. Co-design involves two requirements: (i) we need as many SM-routers as there are LLC slices per memory controller, or in other words, we need as many LLC slices per memory controller as there are clusters, and (ii) we need as many MC-routers as there are memory controllers. These requirements enable power-gating the MC-routers while enabling all SMs to still access the entire memory space under either LLC organization. Note that this also enables H-Xbar to feature the same bisection bandwidth as a full crossbar with the same channel width.

Coherence Implications. Changing the LLC from shared to private has implications for managing the cache in the context of coherence. GPUs exploit software-based coherence to circumvent the need for hardware coherence support [31–33]. This implies that the L1 cache needs to implement a write-through policy and needs to be flushed (invalidated) through cache control operations inserted by the compiler, e.g., at the execution boundary of a kernel. In conventional GPUs, because the LLC is shared by all SMs, no other support for coherence is needed. To support adaptive last-level caching, the LLC needs to support a write-through policy when configured as a private cache, and when the L1 cache is flushed, the private LLC needs to be flushed as well. Previous work adopts a similar approach to support GPU software-based coherence in a multi-chip-module GPU environment [32]. The cache flush operation typically only happens at the end of a kernel, which incurs limited overhead.

Another concern for the private LLC is how to deal with the global memory atomic operation which is handled by the raster operations (ROP) unit in the LLC [33, 34]. One way to solve this problem is to set a small size LLC near the memory controller that is (always) shared by all SMs to handle atomic operations. Alternatively, one could dynamically opt for the shared LLC organization if the workload contains global atomic operations. The workloads in this study do not contain global atomics, hence we leave this for future work.

Dynamic Reconfiguration. To guarantee correctness of operation during a transition between the two LLC organizations, we first need to stall the SMs and wait until there are no more in-flight packets in the NoC and memory system. We then need to write back the dirty cache blocks in the LLC to main memory when transitioning from shared to private caching. We need to flush (invalidate) the LLC when transitioning from private to shared caching. Finally, we power-gate or power-on the MC-routers to engage private and shared caching, respectively. In our experiments, we account for all these reconfiguration overheads. However, we find the overhead to be minimal. Writing back dirty cache lines, invalidating the cache and waiting for the in-flight packets to propagate through the network incurs a performance overhead of a couple hundreds of cycles (a couple thousand cycles at most).

Dynamic Profiling. Adaptive last-level caching is driven by online profiling. Profiling estimates LLC miss rate and bandwidth consumption of the private LLC organization while executing under a shared LLC. Each profiling phase takes 50K cycles, and we find this to be sufficient to make accurate predictions — this is especially true because CTAs from the same kernel typically exhibit similar execution characteristics [35]. We initiate a profiling phase every

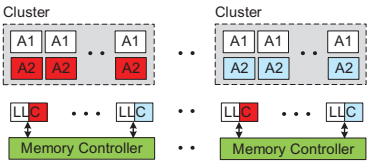


Figure 9: Multi-program support: `A1` is a shared cache friendly application that views a shared LLC, whereas `A2` is a private cache friendly applications that views a private LLC. *Different LLC organizations can be supported for co-executing applications while still benefiting from the entire LLC capacity.*

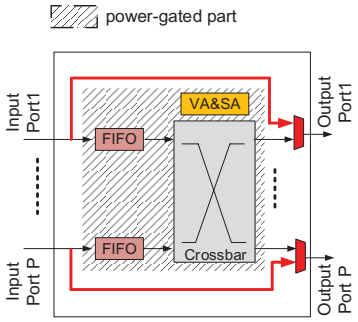


Figure 10: MC-router architecture in the reconfigurable H-Xbar. *Bypassing enables power-gating the MC-router to save power under a private LLC organization.*

1M-cycle epoch or whenever a new kernel comes in. This enables dynamically adapting to new kernels as well as to kernels that exhibit time-varying execution behavior. Our experimental evaluation shows that the runtime overhead is limited to 0.8% on average. We provide more details about dynamic profiling and LLC reconfiguration in Section 4.3.

Multi-Program Support. Adaptive last-level caching can also be used under multi-program execution in a multitasking GPU. Co-executing applications may prefer the same caching mode, which is trivial to support. If the co-executing applications prefer a different LLC mode, we can still accommodate this, i.e., different LLC modes may be employed for the different applications as they co-execute on a multi-tasking GPU. In other words, shared-friendly and private-friendly data streams co-exist in the different LLC slices without extra hardware support. We suggest to map the different programs across the different clusters as illustrated in Figure 9 to uniformly distribute the workload across the different clusters while enabling the co-executing applications to access the entire LLC capacity. Obviously, the MC-router cannot be bypassed (and power-gated) if different applications prefer a different LLC mode.

4.2 Reconfigurable H-Xbar Support

Figure 10 illustrates the reconfigurable MC-router to support adaptive last-level caching in H-Xbar. To support a shared LLC, all input ports in the MC-router can be connected to any output port, hence the MC-router needs to be powered on. For a private LLC on the other hand, an input port needs to be connected only to its corresponding output port. In other words, we can bypass the MC-router. This provides an opportunity to save (static and dynamic) power as we power-gate the MC-router. Bypassing is a commonly used technique in low-power NoCs, see for example [36–38], however, these prior proposals only power-gate routers when the network is idle. In this work on the other hand, we employ bypassing in a completely different way to alter the LLC organization while at the same time reducing NoC power. An alternative to bypassing the MC-router is to set the existing routing paths to a private LLC configuration. While this eliminates the need for additional wiring, power savings will be less. Power-gating and powering-on the MC-routers incurs a runtime overhead of a couple tens of cycles [36, 39], which we account for in our experiments.

4.3 Reconfiguration Rules

With the proposed reconfigurable NoC architecture, the challenge now is to dynamically choose between shared versus private caching. An online profiling and selection mechanism is needed to dynamically choose the most suitable LLC organization based on a workload's execution characteristics. By default, we start with a shared LLC and transition to a private LLC if deemed appropriate based on the profiling data. We transition back to a shared LLC at the beginning of a new 1M-cycle epoch or when a new kernel comes in, after which we collect profiling data for 50K cycles. The profiling information is used to decide which LLC organization to adopt according to the following transition rules:

- **Rule #1** ($S \rightarrow P$): Transition from shared to private LLC if both LLC organizations lead to similar LLC miss rates.
- **Rule #2** ($S \rightarrow P$): Transition from shared to private LLC if the benefits of increased LLC bandwidth are higher than the harm caused by increased LLC miss rate.
- **Rule #3** ($P \rightarrow S$): Transition from private to shared LLC at a new 1M-cycle epoch or when a new kernel starts.

If both LLC organizations yield the same (or similar) LLC miss rates, this indicates that the application has limited or no inter-cluster locality and is thus insensitive to private versus shared caching. We hence transition to the private LLC organization which enables power-gating the MC-routers to save power (Rule #1). Applications with high degrees of inter-cluster locality benefit from private caching because of increased bandwidth to shared cache lines. If the benefits from increased LLC bandwidth are higher than the increase in LLC miss rate, we also transition to private caching (Rule #2). If in private LLC mode, the LLC transitions back to shared mode at a new epoch or when a new kernel is launched (Rule #3).

4.4 Miss Rate and Bandwidth Model

The above transition rules lead to two challenges: we need to compare LLC miss rates for both organizations, and we need to quantify the trade-off between increased miss rate versus increased LLC bandwidth. This is non-trivial as we need to estimate the miss rate and LLC bandwidth demands under private caching while executing under shared caching. We use the following lightweight performance models to do so.

We estimate private LLC miss rate through dynamic set sampling [40] in which we profile 8 sets for a single LLC slice using

a separate hardware structure called the Auxiliary Tag Directory (ATD). Each entry in the ATD holds a tag plus one additional bit per SM-router to store which SM-router last accessed the cache line. (The hardware cost for the ATD is limited to 432 bytes.) Estimating the number of hits for the private LLC organization is done by counting the number of ATD hits for which the access originates from the same SM-router as the one stored in the ATD. Comparing the estimated LLC miss rate under private caching with the measured LLC miss rate under shared caching provides the input for Rule #1, i.e., if the miss rate under private caching is predicted to be within 2% of the miss rate under shared caching, adaptive caching transitions to private caching.

We make the trade-off for Rule #2 using a bandwidth model. We first estimate *LLC Slice Parallelism (LSP)*, or the average number of parallel LLC slice accesses:

$$LSP = \sum_{i=1}^{N} LLC_i / Max\{LLC_i\}.$$

LSP is computed as the sum of all LLC slice accesses (LLC_i) for all N slices divided by the maximum LLC slice access count. LSP equals N if the memory accesses are evenly distributed across all LLC slices; conversely, LSP equals one if all memory accesses are sent to a single LLC slice. We estimate LSP under private caching by computing the target LLC slice for each request. This is trivial, i.e., the target LLC slice is the LLC slice that corresponds with the cluster from which the request originates. Counting the number of requests to each LLC slice (LLC_i) is done using 8 16-bit counters. We do this at the SM-router of the first cluster only, to reduce hardware overhead.

LSP serves as input to the bandwidth model, which computes the overall bandwidth supplied by the memory subsystem (LLC plus main memory):

$$BW = LLC_{hit} \cdot LSP \cdot LLC_{BW} + LLC_{miss} \cdot MEM_{BW}$$

The first term computes the effective LLC bandwidth as the LLC hit rate times LSP times the raw LLC slice bandwidth. The second term computes the effective memory bandwidth as the LLC miss rate times the raw memory bandwidth. We use this bandwidth model to compute the bandwidth supplied under both private and shared caching. The LLC organization for which the bandwidth supplied is the highest is chosen following Rule #2.

5 EXPERIMENTAL SETUP

Simulated System. We use GPGPU-sim v3.2.2 [28] to evaluate performance. Table 1 lists the baseline GPU configuration. We consider the recently proposed page address entropy (PAE) mapping scheme [46] to maximize parallelism across the different LLC slices and main memory subsystem by increasing entropy in the bank and channel bits[1]. We compare adaptive memory-side last-level caching against a shared LLC organization (baseline) and a private LLC organization. We assume 128 B cache lines which is typical for modern-day GPUs. We have evaluated the effect of larger cache lines beyond 128 B and find that the number of potential sharers increases by 10% for 256 B cache lines — more sharers per cache line further exacerbates the LLC bandwidth problem which adaptive caching addresses. The H-Xbar NoC consists of 8 SM-routers, servicing a cluster of 10 SMs each, and 8 MC-routers. A router constitutes a 4-stage pipeline. All LLC slices can access the entire memory space served by the respective memory controller. The hardware overhead for making H-Xbar reconfigurable is limited to 448 bytes in total: 432 bytes (ATDs to estimate private miss rate) plus 16 bytes (private LLC slice access counters). We consider two-level round-robin CTA scheduling, which distributes CTAs across clusters and then across SMs, to balance the workload across the different clusters — we also explore sensitivity to other CTA scheduling policies.

We use DSENT v0.91 [29] to estimate NoC power assuming a 22 nm technology node. We collect activity factors through timing simulation using GPGPU-sim, which we then use as input to DSENT to compute power consumption. NoC chip area is computed as the active silicon area. We do account for repeater area for the long links;

[1]We confirm that PAE address mapping uniformly distributes memory accesses across the different LLC slices.

Table 1: Baseline GPU architecture.

Parameter	Value
Streaming Multiprocessors	80 SMs, 1400 MHz
Warp Size	32
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
Memory Controllers	8
8 LLC slices/MC	96 KB, 16-way, LRU, 128 B line
LLC	6 MB, 120 cycles access time
Interconnection Network	Crossbar, 32 B channel width 4-stage router 1 VC per port - 8 flits/VC VC/Switch allocator - Islip
DRAM Bandwidth	FR-FCFS, 16 banks/MC, 900 GB/s
GDDR5 Timing	t_{CL}=12, t_{RP}=12, t_{RC}=40, t_{RAS}=28, t_{RCD}=12, t_{RRD}=6, t_{CCD}=2, t_{WR}=12

Table 2: GPU benchmarks considered in this study.

Benchmark	Abbr.	Shared Data [MB]	Knl	LLC
LU Decomposition [41]	LUD	33.4	3	shared
Survey Propagation [42]	SP	17.0	2	shared
3D Convolution [43]	3DC	51.1	48	shared
B+TREE Search [41]	BT	13.7	1	shared
GEMM [43]	GEMM	1.8	1	shared
Backprop [41]	BP	18.8	2	shared
AlexNet [44]	AN	1.0	6	private
ResNet [44]	RN	4.2	6	private
SqueezeNet [44]	SN	0.7	1	private
NeuralNetwork [28]	NN	5.7	2	private
Matrix Multiply [45]	MM	1.9	2	private
BlackScholes [45]	BS	0.001	3	neutral
DWT2D [41]	DWT2D	0.001	1	neutral
Merge Sort[45]	MS	0.001	1	neutral
BinomialOptions[45]	BINO	0.017	1	neutral
Histogram [45]	HG	0.003	1	neutral
Vector Add [45]	VA	0.001	1	neutral

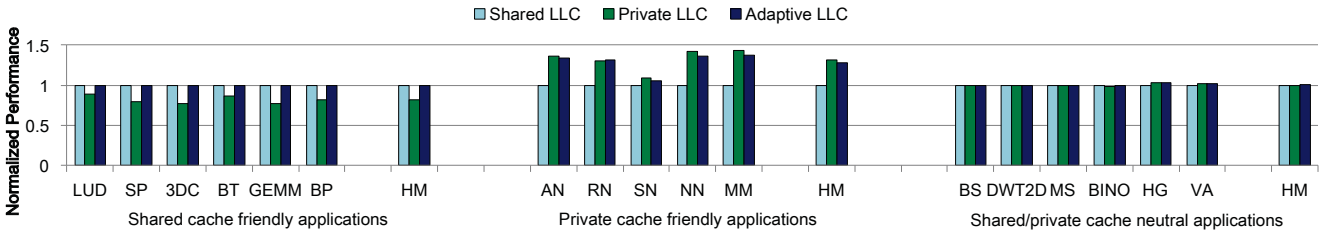


Figure 11: Normalized performance (IPC) for a shared, private and adaptive memory-side LLC. *Adaptive caching improves performance by 29.8% for the private cache friendly workloads, while being performance neutral for the other workloads.*

the links themselves are routed in the higher metal layers which attributes to the global wire area. We assume that long links measure 12.3 mm which is half the Pascal die size [47][2]. For evaluating GPU power, we use GPUWattch [49].

Workloads and Performance Metrics. We consider a representative and broad set of CUDA GPU benchmarks from a range of application domains including neural networks, machine learning, image processing, graph search, etc. These benchmarks include regular applications from Rodinia [41], CUDA SDK [45], as well as irregular applications from Lonestar [42], deep learning applications from Tango [44] and additional sources [28, 43]. Table 2 lists these benchmarks along with their classification, the number of kernels, and the shared data footprint (in MB). We quantify performance in terms of instructions per cycle (IPC). We simulate one billion instructions, which is in line with recent GPU research [50, 51].

6 EVALUATION

We evaluate adaptive caching in terms of performance and NoC energy consumption. We also provide various sensitivity analyses.

6.1 Performance

We first evaluate performance, see Figure 11. Adaptive memory-side caching improves performance by 28.1% on average and up to 38.1% for the private cache friendly workloads, while being performance-neutral for the other workloads. A private LLC on the other hand degrades performance severely for the shared cache friendly workloads, by 18.1% on average and up to 22.6%. Note that these performance results include profiling overhead.

To gain further insight into where the performance benefits are coming from for the adaptive LLC organization, we now quantify the LLC response rate for the private cache friendly applications, see Figure 12. LLC response rate is defined as the average number of responses supplied by the LLC slices per cycle. Private caching increases the LLC response rate by 35.3% on average and up to 45.9% compared to a shared LLC organization. The improvement in LLC response rate quantifies the increase in effective LLC bandwidth because of replication of shared cache lines across different LLC slices, and translates into a proportional improvement in overall application performance.

[2]This is a conservative estimate. Wire length can be reduced through optimized floorplanning [39, 48].

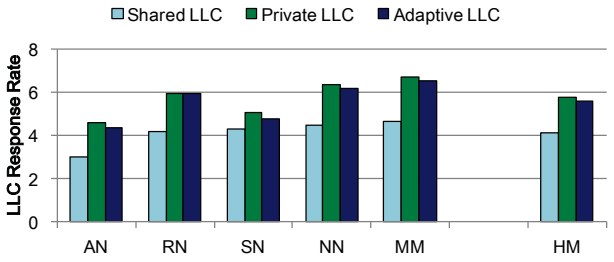


Figure 12: LLC response rate (flits/cycle) for the private cache friendly workloads for a shared, private and adaptive memory-side LLC. *Adaptive caching increases the LLC response rate by 1.5× on average.*

The shared cache friendly applications do not benefit from adaptive caching, although there is some inter-cluster locality as previously noted in Figure 3. The reason is that duplication of shared cache lines across different LLC slices in a private LLC organization, severely increases the LLC miss rate by 27.9 percent point on average and up to 52.3 percent point, as shown in Figure 13. These workloads have a large shared data footprint in the multi-MB range, see Table 2, which leads to severe interference in a private LLC. The private cache friendly applications also experience an increase in LLC miss rate but the increase is less, by 12.7 percent point on average and up to 33.3 percent point (not shown here because of space constraints).

6.2 Energy

The adaptive LLC significantly reduces NoC energy consumption by power-gating the MC-routers in the private LLC mode, see Figure 14. More specifically, NoC energy reduces by 26.6% on average, and up to 29.7%, for the private cache friendly and shared/private cache neutral applications compared to a shared LLC. Adaptive caching is energy-neutral for the shared cache friendly workloads. The private LLC organization reduces NoC energy but increases DRAM power due to an increase in DRAM traffic due to the write-through LLC policy. When quantifying total system (GPU plus DRAM) energy, we conclude though that under private LLC mode energy consumption reduces by 6.1% on average and up to 27.2% for the private cache friendly and shared/private cache neutral workloads.

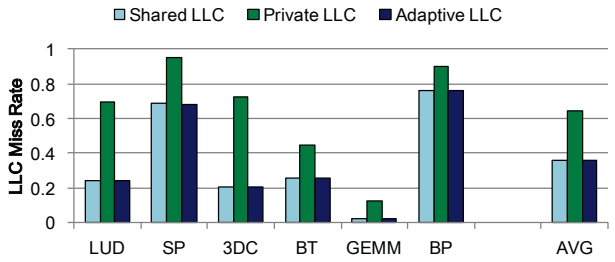


Figure 13: LLC miss rate for the shared cache friendly workloads for a shared, private and adaptive memory-side LLC. *A private cache increases the LLC miss rate by 27.9 percent point on average and up to 52.3 percent point for the shared cache friendly applications, hence adaptive caching opts for a shared LLC.*

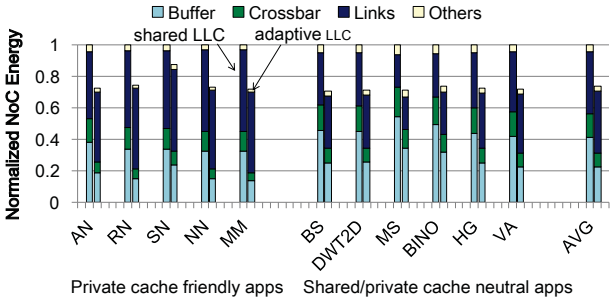


Figure 14: NoC energy consumption normalized to a shared LLC for the private cache friendly and shared/private cache neutral workloads. *Adaptive caching reduces NoC energy consumption by 26.6% on average.*

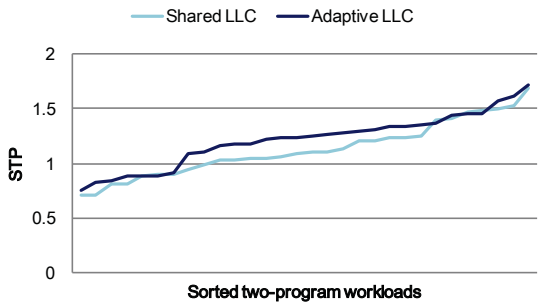


Figure 15: Multi-program performance. *Adaptive caching simultaneously provides a shared LLC for shared cache friendly applications and a private LLC for private cache friendly applications, improving overall system throughput (STP) by 8% on average compared to a shared LLC.*

6.3 Multi-Program Workloads

We now evaluate adaptive caching using multi-program workloads. We consider all possible two-program combinations of shared cache friendly and private cache friendly applications, and quantify system throughput (STP) [52], see Figure 15. The shared cache friendly application views the LLC as a shared cache, whereas the private cache friendly application views the LLC as a private cache. In these experiments, we distribute both applications across the clusters giving half the SMs per cluster to each application — this uniformly distributes the workload across the clusters and allows both applications to access the entire LLC capacity, as previously argued. STP improves by 8% on average compared to a shared LLC because of increased LLC bandwidth to shared data for the private cache friendly application.

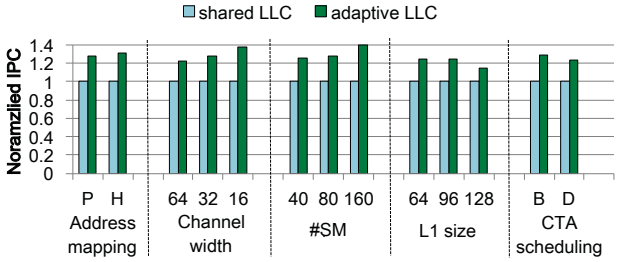


Figure 16: Sensitivity analyses. *The adaptive LLC outperforms a shared LLC across address mapping schemes (P: PAE, H: Hynix), channel widths (byte), SM count, L1 sizes (KB), and CTA scheduling policies (B: BCS, D: DCS).*

6.4 Sensitivity Analyses

We now perform a couple sensitivity analyses in which we change the address mapping scheme, channel width, number of SMs, L1 cache size and CTA scheduling policy, see also Figure 16.

Address Mapping. PAE [46], the address mapping scheme used in this paper, evenly distributes the requests. The Hynix memory address mapping scheme [53] on the other hand, incurs imbalance in the memory subsystem by unevenly distributing memory requests across channels and banks. Adaptive caching yields higher performance improvements under an imbalanced memory request stream. In particular, we note an average performance improvement by 31.1% for the Hynix address mapping scheme. This can be understood intuitively because private caching increases LLC bandwidth through data duplication across multiple LLC slices.

Channel Width. Adaptive caching is sensitive to NoC bandwidth. The performance benefit under adaptive caching is higher under constrained NoC bandwidth. We note a 22.6%, 28.1% and 38.2% average performance improvement for a channel width of 64 bytes, 32 bytes (default) and 16 bytes, respectively. Constrained channel widths renders the NoC a performance bottleneck, thereby making the LLC adaptive even more critical.

SM Count. We evaluate sensitivity with the number of SMs by changing SM count from 40 to 160 SMs; we do so while keeping the number of SMs per cluster constant, which implies that we scale the number of clusters (and thus, the number of SM-routers) and the number of LLC slices (and thus, the number of MC-routers) proportionally. We note that adaptive caching leads to an average performance improvement of 40% for 160 SMs. Adaptive caching is

more beneficial at higher SM counts, primarily because of increased NoC pressure and increased LLC contention.

L1 Cache Size. We vary the per-SM L1 cache size from 48 KB (our default) to 64, 96 and 128 KB. The performance benefit of adaptive caching remains stable around 28% on average for the 64 and 96 KB L1 caches. The performance benefit reduces to 15% on average for a 128 KB L1 cache. The increased size enables replicating shared data in the L1 cache, thereby somewhat reducing the opportunity for adaptive last-level caching.

CTA Scheduling Policy. In addition to our default CTA scheduling policy (two-level round-robin), we also consider block CTA scheduling (BCS) [54] (mapping adjacent CTAs to the same SM to improve L1 cache locality) and distributed CTA scheduling (DCS) [32] (evenly dividing CTAs across the clusters). We conclude that adaptive caching improves performance under all CTA scheduling policies, while yielding a slightly smaller benefit for distributed scheduling (23.9% average improvement) because of reduced inter-cluster locality, i.e., adjacent CTAs are mapped to the same cluster somewhat reducing the amount of inter-cluster locality.

7 RELATED WORK

The LLC and the NoC are focal points in computer architecture research. We now survey the most closely prior work.

LLC Optimization in CMPs. With the advent of CMPs, numerous efforts went into optimizing LLC sharing among cores. Huh et al. [7] propose an LLC organization that supports various sharing degrees. Dynamic NUCA techniques start either from a private or a shared LLC and seek to bring data close to the cores that use it while effectively using the cache capacity. They do so by a combination of replication [8–12], placement and migration [18, 21, 22], and cache block spilling [17, 55]

Other research efforts focus on LLC partitioning schemes in CMPs. Yeh et al. [13] and Merino et al. [14] propose approaches to split the cache into private and shared slices. Guz et al. [15] propose separate private and shared caches, laying both out in close proximity to all cores. Zhao et al. [16] propose a hybrid architecture in which each LLC slice has both a private and a share space. Jigsaw [24] uses a hardware/software approach to partition cache banks and manage the data placement in 'shares' of sub-partitions to address scalability and interference issues in shared caches. Kwon et al. [56] cluster cores and LLC banks and use a fast interconnect to provide cores with a fast inter- and intra-cluster LLC access. Abousamra et al. [57] propose a symbiotic NoC/LLC design to reduce latency and improve cache utilization.

Techniques tailored for CMPs are not readily applicable to a GPU. The main objective in CMPs is to bring data close to the relevant core(s) to reduce LLC access latency. Most CMP papers presume spatial affinity of an LLC slice to a core, which does not exist in GPUs with a memory-side LLC. Many CMP techniques rely on hardware coherence mechanisms that are not supported in GPUs. Moreover, the tradeoff is different in GPUs which are bandwidth-sensitive and less sensitive to latency than CPUs. We highlight a bottleneck in GPU workloads with significant read-only sharing, whereby numerous SMs frequently access the shared cache lines creating a bandwidth bottleneck at the corresponding LLC banks.

NoC Optimization. Hybrid and adaptive routing schemes have been explored for multicore processors. Hybrid circuit/packet switching NoCs have been proposed for coherence communication [58] and heterogeneous processors [59]. ReNoC [60] is a reconfigurable NoC architecture that dynamically changes the NoC topology in a Systems-on-Chip according to an application's communication pattern. More recently, MAERI [61] provides a reconfigurable interconnect to enable a flexible dataflow mapping in DNN accelerators. BiNoCHS [62] reconfigures network resources to traffic patterns in a heterogeneous CPU/GPU system. Jin et al. [63] propose domain-specific NoCs with large-scale caching using a novel multicast router for fast cache access, and to reduce unnecessary links in general-purpose NoCs. This technique, similar to both MAERI and BiNoCHS, targets NoCs in specialized architectures and is not readily applicable to GPUs. None of these proposals optimize the NoC and LLC cooperatively to exploit the sharing characteristics in GPU applications.

Our work uses NoC crossbars as they naturally support the communication pattern between SMs and LLC slices in a GPU. Prior techniques that optimize GPU NoCs presume mesh topologies due to their simplicity and scalability. The main goal is to reduce the hardware cost associated with providing all-to-all communication in a mesh which is not necessary in GPUs [64–68]. In this work, we propose a hierarchical two-stage crossbar that is more power and area-efficient while delivering similar performance to a full crossbar. Moreover, adaptive last-level caching enables saving NoC energy and total system energy by power-gating the second-stage routers when configured for a private LLC.

GPU Optimization. Numerous papers consider optimizing various aspects of a GPU architecture, and focus on SM-related aspects including warp scheduling [69–73], L1 cache optimization [74–77], register file optimization [78–80], and virtualized SM resources [81, 82]. Recent work explores cache hierarchy designs in multi-module GPUs within a single package [32] and across sockets [31]. They explore dedicating part of the LLC local to a GPU to hold data from remote memory modules, i.e., those attached to remote GPU modules/sockets. However, in this prior work, the LLC remains shared among the SMs within a single GPU. In this work, we demonstrate the performance benefit from a private LLC organization through adaptive last-level caching.

8 CONCLUSION

This paper studies private versus shared organizations in memory-side last-level GPU caches. We find that although a shared memory-side LLC performs well for workloads with no to moderate data sharing among SMs, sharing-intensive workloads with high degrees of read-only sharing significantly benefit from a private LLC organization because of increased bandwidth to replicated shared cache lines across different LLC slices. In response, this paper proposes adaptive last-level caching to dynamically choose between a shared versus private LLC based on the workload's execution characteristics through a lightweight performance model that predicts and trades off LLC miss rate and bandwidth under private caching while executing under shared caching. In addition to significantly improving performance, adaptive last-level caching provides an interesting

opportunity to save NoC energy and total system energy for a co-designed hierarchical two-stage crossbar in which the second stage can be bypassed and power-gated under the private LLC mode.

Our experimental results show that adaptive memory-side caching improves performance by 28.1% on average, and up to 38.1%, for sharing-intensive workloads. Furthermore, adaptive caching reduces NoC energy by 26.6%, and up to 29.7%, and system energy by 6.1% on average, and up to 27.2%, when configured as a private LLC. A NoC crossbar design space exploration demonstrates that a hierarchical two-stage crossbar is both more power- and area-efficient than a full and concentrated crossbar with the same bisection bandwidth. This leads to the overall conclusion that cooperative NoC/LLC adaptive caching is an effective and scalable solution to exploit a workload's (lack of) sharing behavior in future GPUs.

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 Grant No. 61572508, 61672526 and 61802427, NUDT Research Project No. ZK17-03-06, Science and Technology Innovation Project of Hunan Province under Grant 2018RS3083. Xia Zhao is supported through a CSC scholarship and UGent-BOF co-funding.

REFERENCES

[1] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, pp. 879–899, May 2008.

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the the international conference on Multimedia (ICMR)*, pp. 675–678, April 2014.

[3] R. Collobert, C. Farabet, K. Kavukcuoglu, and S. Chintala, "Torch." http://torch.ch/.

[4] Nvidia, "NVIDIA's Next Generation CUDA Compute Architecture:Fermi." http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf, 2009.

[5] Nvidia, "NVIDIA Tesla V100 GPU Architecture The World's Most Advanced Data Center GPU. White paper." http://www.nvidia.com/object/volta-architecture-whitepaper.html, 2017.

[6] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-uniform Cache Structure for Wire-delay Dominated On-chip Caches," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 211–222, October 2002.

[7] 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)*, pp. 31–40, June 2005.

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

[9] 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)*, pp. 357–368, June 2005.

[10] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity While Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 336–345, June 2005.

[11] 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)*, pp. 443–454, December 2006.

[12] C. Hughes, C. Kim, and Y. Chen, "Performance and Energy Implications of Many-Core Caches for Throughput Computing," *IEEE Micro*, vol. 30, pp. 25–35, November 2010.

[13] T. Y. Yeh and G. Reinman, "Fast and Fair: Data-stream Quality of Service," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pp. 237–248, September 2005.

[14] J. Merino, V. Puente, P. Prieto, and J. A. Gregorio, "SP-NUCA: A Cost Effective Dynamic Non-uniform Cache Architecture," *SIGARCH Comput. Archit. News*, vol. 36, pp. 64–71, May 2008.

[15] Z. Guz, I. Keidar, A. Kolodny, and U. C. Weiser, "Utilizing Shared Data in Chip Multiprocessors with the Nahalal Architecture," in *Proceedings of the International Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 1–10, June 2008.

[16] L. Zhao, R. Iyer, M. Upton, and D. Newell, "Towards Hybrid Last Level Caches for Chip-multiprocessors," *SIGARCH Comput. Archit. News*, vol. 36, pp. 56–63, May 2008.

[17] H. Dybdahl and P. Stenstrom, "An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 2–12, February 2007.

[18] 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)*, pp. 184–195, June 2009.

[19] S. Cho and L. Jin, "Managing Distributed, Shared L2 Caches Through OS-Level Page Allocation," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 455–468, December 2006.

[20] L. Jin, H. Lee, and S. Cho, "A Flexible Data to L2 Cache Mapping Approach for Future Multicore Processors," in *Proceedings of the Workshop on Memory System Performance and Correctness (MSPC)*, pp. 92–101, October 2006.

[21] Y. Zhang, W. Ding, M. Kandemir, J. Liu, and O. Jang, "A Data Layout Optimization Framework for NUCA-based Multicores," in *Proceedings of International Symposium on Microarchitecture (MICRO)*, pp. 489–500, December 2011.

[22] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son, "A Novel Migration-based NUCA Design for Chip Multiprocessors," in *Proceedings of the Conference on Supercomputing (SC)*, pp. 1–12, November 2008.

[23] J. Lira, C. Molina, and A. González, "The Auction: Optimizing Banks Usage in Non-Uniform Cache Architectures," in *Proceedings of the International Conference on Supercomputing (ICS)*, pp. 37–47, June 2010.

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

[25] H. Kasture and D. Sanchez, "Ubik: Efficient Cache Sharing with Strict Qos for Latency-critical Workloads," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 729–742, March 2014.

[26] N. E. Jerger, T. Krishna, and L. Peh, *On-Chip Networks: Second Edition*. Morgan & Claypool Publishers, 2017.

[27] P. Kumar, Y. Pan, J. Kim, G. Memik, and A. Choudhary, "Exploring Concentration and Channel Slicing in On-chip Network Router," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, pp. 276–285, May 2009.

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

[29] 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)*, pp. 201–210, May 2012.

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

[31] 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)*, pp. 123–135, October 2017.

[32] 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)*, pp. 320–332, June 2017.

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

[34] 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.

[35] J. Lee and H. Kim, "TAP: A TLP-Aware Cache Management Policy For a CPU-GPU Heterogeneous Architecture," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12, February 2012.

[36] L. Chen and T. M. Pinkston, "NoRD: Node-Router Decoupling for Effective Power-gating of On-Chip Routers," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 270–281, December 2012.

[37] H. Zheng and A. Louri, "EZ-Pass: An Energy & Performance-Efficient Power-Gating Router Architecture for Scalable NoCs," *IEEE Computer Architecture Letters*, vol. 17, pp. 88–91, January 2018.

[38] H. Farrokhbakht, H. M. Kamali, N. E. Jerger, and S. Hessabi, "SPONGE: A Scalable Pivot-based On/Off Gating Engine for Reducing Static Power in NoC Routers,"

in *Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED*, pp. 17:1–17:6, July 2018.
[39] 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)*, pp. 296–307, February 2014.
[40] 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)*, pp. 167–178, June 2006.
[41] 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)*, pp. 44–54, October 2009.
[42] 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)*, pp. 141–151, November 2012.
[43] 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)*, pp. 1–10, May 2012.
[44] "Tango: A Deep Neural Network Benchmark Suite for Various Accelerators." https://gitlab.com/Tango-DNNbench/Tango.
[45] "NVIDIA CUDA SDK Code Samples." https://developer.nvidia.com/cuda-downloads.
[46] 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)*, pp. 166–179, June 2018.
[47] Nvidia, "NVIDIA GP100 Pascal Architecture. White paper.." http://www.nvidia.com/object/pascal-architecture-whitepaper.html, 2016.
[48] 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)*, pp. 181–188, May 2010.
[49] 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)*, pp. 487–498, June 2013.
[50] A. Arunkumar, S. Y. Lee, V. Soundararajan, and C. J. Wu, "LATTE-CC: Latency Tolerance Aware Adaptive Cache Compression Management for Energy Efficient GPUs," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 221–234, February 2018.
[51] G. Koo, Y. Oh, W. W. Ro, and M. Annavaram, "Access Pattern-Aware Cache Management for Improving Data Utilization in GPU," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 307–319, June 2017.
[52] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," *IEEE Micro*, vol. 28, pp. 42–53, May 2008.
[53] "Hynix GDDR5 SGRAM Part H5GQ1H24AFR Revision 1.0." http://www.hynix.com/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf, 2009. Hynix.
[54] 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)*, pp. 260–271, February 2014.
[55] 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)*, pp. 45–54, February 2009.
[56] W.-C. Kwon, T. Krishna, and L.-S. Peh, "Locality-oblivious Cache Organization Leveraging Single-cycle Multi-hop NoCs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 715–728, March 2014.
[57] A. K. Abousamra, A. K. Jones, and R. G. Melhem, "NoC-aware Cache Design for Multithreaded Execution on Tiled Chip Multiprocessors," in *Proceedings of the International Conference on High Performance and Embedded Architectures and Compilers (HIPEAC)*, pp. 197–205, January 2011.
[58] N. D. E. Jerger, L. S. Peh, and M. H. Lipasti, "Circuit-Switched Coherence," in *Proceedings o the International Symposium on Networks-on-Chip (NOCS)*, pp. 193–202, April 2008.
[59] J. Yin, P. Zhou, S. S. Sapatnekar, and A. Zhai, "Energy-Efficient Time-Division Multiplexed Hybrid-Switched NoC for Heterogeneous Multicore Systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 293–303, May 2014.
[60] M. B. Stensgaard and J. Sparsø, "ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, pp. 55–64, April 2008.
[61] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 461–475, March 2018.
[62] A. Mirhosseini, M. Sadrosadati, B. Soltani, H. Sarbazi-Azad, and T. F. Wenisch, "BiNoCHS: Bimodal Network-on-Chip for CPU-GPU Heterogeneous Systems," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, pp. 7:1–7:8, October 2017.
[63] Y. Jin, E. J. Kim, and K. H. Yum, "A Domain-Specific On-Chip Network Design for Large Scale Cache Systems," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pp. 318–327, February 2007.
[64] 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)*, pp. 421–432, December 2010.
[65] 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)*, pp. 407–412, September 2012.
[66] 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)*, pp. 34:1–34:6, June 2016.
[67] 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)*, pp. 472–479, October 2016.
[68] 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)*, pp. 25:1–25:8, July 2015.
[69] 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)*, pp. 42:1–42:12, June 2016.
[70] 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)*, pp. 332–343, June 2013.
[71] 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)*, pp. 45–55, September 2016.
[72] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious Wavefront Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 72–83, December 2012.
[73] 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)*, pp. 515–527, June 2015.
[74] 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)*, pp. 89–98, June 2015.
[75] 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)*, pp. 174–185, February 2015.
[76] 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)*, pp. 76–88, February 2015.
[77] 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)*, pp. 272–283, February 2014.
[78] H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavaram, "GPU Register File Virtualization," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pp. 420–432, December 2015.
[79] 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)*, pp. 412–423, February 2013.
[80] 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)*, pp. 344–355, June 2013.
[81] 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)*, pp. 609–621, June 2016.
[82] 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)*, pp. 1–14, October 2016.