HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems

Xiaowei Ren[†‡], Daniel Lustig[‡], Evgeny Bolotin[‡], Aamer Jaleel[‡], Oreste Villa[‡], David Nellans[‡]

[†]*The University of British Columbia* [‡]*NVIDIA*

xiaowei@ece.ubc.ca {*dlustig, ebolotin, ajaleel, ovilla, dnellans*}*@nvidia.com*

***Abstract*—Prior work on GPU cache coherence has shown that simple hardware- or software-based protocols can be more than sufficient. However, in recent years, features such as multi-chip modules have added deeper hierarchy and non-uniformity into GPU memory systems. GPU programming models have chosen to expose this non-uniformity directly to the end user through scoped memory consistency models. As a result, there is room to improve upon earlier coherence protocols that were designed only for flat single-GPU hierarchies and/or simpler memory consistency models.**

In this paper, we propose HMG, a cache coherence protocol designed for forward-looking multi-GPU systems. HMG strikes a balance between simplicity and performance: it uses a readily-implementable VI-like protocol to track coherence states, but it tracks sharers using a hierarchical scheme optimized for mitigating the bandwidth limitations of inter-GPU links. HMG leverages the novel scoped, non-multi-copy-atomic properties of modern GPU memory models, and it avoids the overheads of invalidation acknowledgments and transient states that were needed to support prior GPU memory models. On a 4-GPU system, HMG improves performance over a software-controlled, bulk invalidation-based coherence mechanism by 26% and over a non-hierarchical hardware cache coherence protocol by 18%, thereby achieving 97% of the performance of an idealized caching system.

I. Introduction

As the demand for GPU compute continues to grow beyond what a single die can deliver [1–4], GPU vendors are turning to new packaging technologies such as multi-chip modules (MCMs) [5] and new networking technologies such as NVIDIA's NVLink [6] and NVSwitch [7] and AMD's xGMI [8] in order to build ever-larger GPU systems [9–11]. Consequently, as Fig. 1 depicts, modern GPU systems are becoming increasingly hierarchical. However, due to physical limitations, the large bandwidth discrepancy between existing inter-GPU links [6, 8] and on-package integration technologies [12] can contribute to Non-Uniform Memory Access (NUMA) behavior that often bottlenecks performance. Following established principles, GPUs use aggressive caching to recover some of the performance loss created by the NUMA effect [5, 13, 14], and these caches are kept coherent with lightweight coherence protocols that are implemented in software [5, 13], hardware [14, 15], or a mix of both [16].

GPU originally assumed that inter-thread synchronization would be coarse-grained and infrequent, and hence they adopted a bulk-synchronous programming model (BSP) for simplicity. This paradigm disallowed any data communication among Cooperative Thread Arrays (CTAs) of active kernels. However, in emerging applications, less-restrictive data sharing patterns and fine-grained synchronization are expected to be more frequent [17–20]. BSP is too rigid and inefficient to support these new sharing patterns.

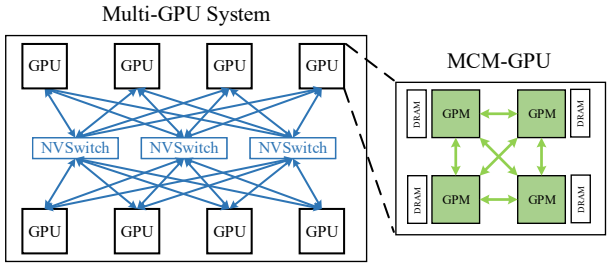


Figure 1: Forward-looking multi-GPU system. Each GPU has multiple GPU modules (GPMs).

To extend GPUs into more general-purpose domains, GPU vendors have released very precisely-defined scoped memory models [21–24]. These models allow flexible communication and synchronization among threads in the same CTA, the same GPU, or anywhere in the system, usually by requiring programmers to provide *scope* annotations for synchronization operations. Scopes allow synchronization and coherence to be maintained entirely within a narrow subset of the full-system cache hierarchy, thereby delivering improved performance over system-wide synchronization enforcement. Furthermore, unlike most CPU memory models, these GPU models are now *non-multi-copy-atomic*: they do not require that memory accesses become visible to all observers at the same logical point in time. As a result, there is room for forward-looking GPU cache coherence protocols to be made even more relaxed, and hence capable of delivering even higher throughput, than protocols proposed in prior work (as outlined in Section III).

Previously explored GPU coherence schemes [5, 13–16, 25–27] were well tuned for GPUs and much simpler than CPU protocols, but few have studied how to scale these protocols to larger multi-GPU systems with deeper cache hierarchies. To test their efficiency, we simply apply the existing software

2378-203X/20/$31.00 ©2020 IEEE
DOI 10.1109/HPCA47549.2020.00054

IEEE computer society

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

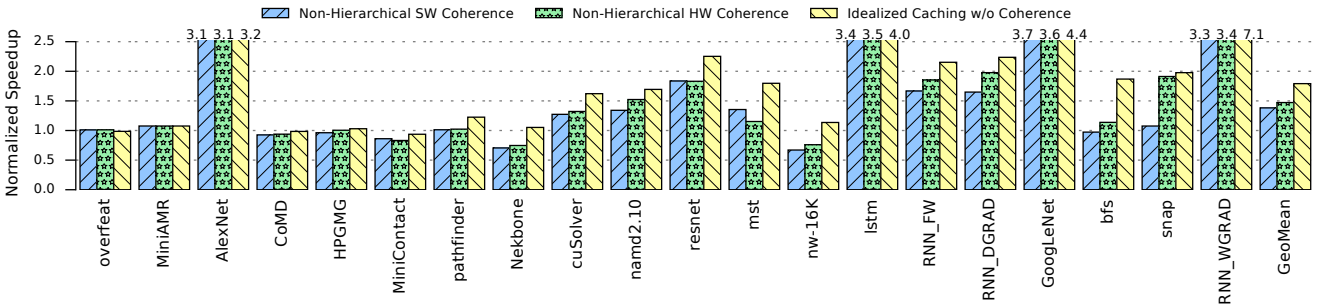


Figure 2: Benefits of caching remote GPU data under three different protocols on a 4-GPU system with 4 GPMs per GPU, all normalized to a baseline which has no such caching. The results show that existing protocols leave room for improvement when extended to multiple GPUs.

and hardware coherence protocols GPU-VI [15] on a 4-GPU system, in which each GPU consists of 4 separate GPU modules (GPMs). These protocols do not account for architectural hierarchy; we simply extend them as if the system were a flat platform of 16 GPMs. As Fig. 2 shows, in hierarchical multi-GPU systems, existing non-hierarchical software and hardware VI coherence protocols indeed leave plenty room for improvement; see Section III for details. We therefore ask the following question: how do we extend existing GPU coherence protocols across multiple GPUs while simultaneously providing high-performance support for flexible fine-grained synchronization within emerging GPU applications, and without a dramatic increase in protocol complexity?

We propose HMG, a hardware-managed cache coherence protocol designed to extend coherence guarantees across forward-looking **H**ierarchical **M**ulti-**G**PU systems with scoped memory consistency models. Unlike prior CPU and GPU protocols that enforce multi-copy-atomicity and/or track ownership within the protocol, HMG eliminates transient states and/or extra hardware structures that would otherwise be needed to cover the latency of acquiring write permissions [15, 16]. HMG also filters out unnecessary invalidation acknowledgment messages, since a write can be processed instantly if multi-copy-atomicity is not required. Similarly, unlike prior work that filters coherence traffic by tracking the read-only state of data regions [14, 16], HMG relies on precise but hierarchical tracking of sharers to mitigate the performance impact of bandwidth-limited inter-GPU links without adding unnecessary coherence traffic. As such, HMG is able to scale up to large multi-GPU systems, while nevertheless maintaining the simplicity and implementability that made prior GPU cache coherence protocols popular [15].

Overall, this paper makes the following contributions:

- We study the architectural implications of extending cache coherence protocols to multi-GPU systems. To the best of our knowledge, this is the first study of multi-MCM, multi-GPU hardware coherence under a scoped, non-multi-copy-atomic memory model.
- We show that while it is possible to extend existing software or hardware coherence protocols to multi-GPU systems, performance is very inefficient without mechanisms in place to exploit intra-GPU data locality and mitigate the inter-GPU/inter-module bandwidth constraints.
- We propose HMG, a hardware-managed cache coherence protocol for distributed L2 caches in hierarchical multi-GPU platforms. Our proposal not only mitigates the majority of the inter-GPU bandwidth related bottlenecks by improving intra-GPU data locality with hierarchical sharer tracking, but also eliminates unnecessary transient states and coherence messages found in previous proposals. HMG delivers 97% of the overall possible performance of an idealized system.

II. Background

To avoid confusion around the term "shared memory" which is used to describe scratchpad memory on NVIDIA GPUs, we use "global memory" for the virtual address space shared by all CPUs and GPUs in a system.

A. Hierarchical Multi-Module GPUs

Because the end of Moore's Law is limiting continued transistor density improvement, future multi-module GPU architectures will consist of a hierarchy in which each GPU is split into GPU modules (GPMs), as depicted in Fig. 1 [28]. Recent work has demonstrated the benefits of creating GPMs in the form of single-package multi-chip modules (MCMs) [5]. Researchers have explored the possibility of presenting a large hierarchical multi-GPU system to users as if it were a single large GPU [13], but mainstream GPU platforms today largely just expose the hierarchy directly to users so that they can manually optimize data and thread placement and migration decisions.

The constrained bandwidth of the inter-GPM/GPU links is the main performance bottleneck on hierarchical GPU architectures. To mitigate this, both MCM-GPU and Multi-GPU explorations have scheduled adjacent CTAs to the same GPM/GPU to exploit inter-CTA data locality [5, 13]. These proposals map each memory page to the first GPM/GPU that

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

touches it to increase the likelihood that caches will capture locality. They also extended a conventional GPU software coherence protocol to the L2 caches, and they showed that it worked well for traditional bulk-synchronous programs.

More recently, CARVE [14] proposed to allocate a fraction of local GPU DRAM as cache for remote GPU data, and enforced coherence using a simple protocol filtered by tracking whether data was private, read-only, or read-write shared. However, as CARVE neither tracked sharers nor used scopes, CARVE broadcast invalidation messages to all caches for read-write shared data.

B. Emerging Programs Need Fine-Grained Communication

Nowadays, many applications contain fine-grained communication between CTAs of the same kernel and/or of dependent kernels [29–35]. For example, in RNNs, abundant inter-CTA communication exists in the neuron connections between continuous timesteps [36]. In the simulation of molecule or neutron dynamics [34, 35], inter-CTA communication is necessary for the movement dependency between different particles and different simulation timesteps. Graph algorithms usually dispatch vertices among multiple CTAs or kernels that need to exchange their individual update to the graph for the next round of computing until they reach convergence [4, 30]. We provide more details on the workloads we study in this paper in Section VI. All these applications can benefit from a hierarchical GPU system for higher performance and from the scoped memory model for efficient inter-CTA synchronization enforcement.

C. GPU Memory Model

Both CUDA and OpenCL originally supported a coarse-grained bulk-synchronous programming model. Under this paradigm, data sharing between threads of the same CTA could be performed locally in the shared memory scratchpad and synchronized using CTA execution barriers; but inter-CTA synchronization was permitted only between dependent kernel calls (i.e., where data produced by one kernel is consumed by the following kernels). They could not, with guaranteed correctness, perform arbitrary communication using global memory. While many GPU applications work very well under a bulk-synchronous model with rare inter-CTA synchronizations, it quickly becomes a limiting factor for the types of emerging applications described in Section II-B.

To support data sharing more flexibly and more efficiently, both CUDA and OpenCL have shifted from bulk-synchronous models to more general-purpose scoped memory models [21–24, 37]. By incorporating the notion of *scope*, these new models allow each thread to communicate with any other threads in the same CTA (`.cta`), the same GPU (`.gpu`), and anywhere in the system (`.sys`)[1]. Scope indicates the set of threads with which a particular memory access wishes to synchronize. Synchronization of scope `.cta` is performed in the L1 cache of each GPU Streaming Multiprocessor (SM); synchronizations of scope `.gpu` and `.sys` are processed through the GPU L2 cache and via the memory hierarchy of the whole system, respectively.

D. GPU Cache Coherence

Some GPU protocols advocate for strong memory models, and hence they propose sophisticated cache coherence protocols capable of delivering good performance [27]. Most other GPU protocols enforce variants of release consistency by invalidating possibly stale values in caches when performing *acquire* operations (implicitly including the start of a kernel), and by flushing dirty data during *release* operations (implicitly including the end of a kernel). Much of the research in the area today proposes optimizations on top of these basic principles. We broadly classify this work by whether reads or writes are responsible for invalidating stale data.

Among read-initiated protocols, hLRC [26] elided unnecessary cache invalidations and flushes by lazily performing coherence actions when synchronization variables change registration. Furthermore, the recent proposals of DeNovo [16] and VIPS [38] can protect read-only or private data from invalidation. However, they incur additional overheads and/or require software changes to convey region information for read-only data, ownership tracking in word granularity, or coarse-grained (memory page level)[2] private/shared data classification.

As for write-initiated cache invalidations, previous work has observed that MESI-like coherence protocols are a poor fit for GPUs [15, 25]. QuickRelease [25] reduced the overhead of cache flush by enforcing the partial order of writes with a FIFO. However, QuickRelease needs to partition the resources required by reads and writes; it also broadcasts invalidations to all remote caches. GPU-VI [15] is a simple yet effective hardware cache coherence protocol, but it predated scoped memory models and introduced extra overheads to enforce multi-copy-atomicity, which is no longer necessary. Also, GPU-VI was proposed for use within a single GPU, and did not consider the added complexity of having various bandwidth tiers.

III. The Novel Coherence Needs of Modern Multi-GPU Systems

To scale coherence across multiple GPUs, the design of HMG not only considers the architectural hierarchy of modern GPU systems (Fig. 1), but also aggressively takes advantage of the latest scoped memory models (Section II-C). Before diving into the details of HMG, we first describe our main insights below.

[1]We use NVIDIA terminology in this paper. Equivalent scopes in HRF are work-group, device, and system [24].

[2]GPUs need large pages (e.g., 2MB) to ensure high TLB coverage. Smaller pages can cause severe TLB bottlenecks [39].

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

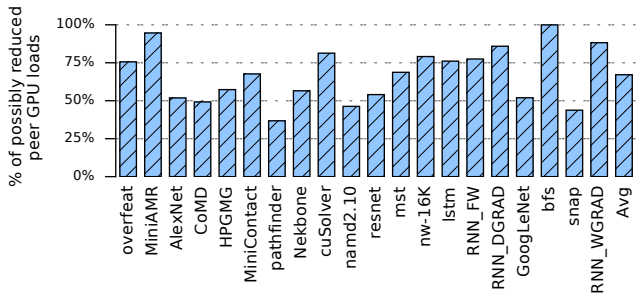


Figure 3: Percentage of inter-GPU loads destined to addresses accessed by another GPM in the same GPU.

A. Extending Coherence to Multiple GPUs

As described in Section II-D, prior GPU coherence protocols mainly focused on mechanisms that mitigate the impact of bulk cache invalidations. However, as Fig. 2 shows, even fine-grained hardware VI cannot close the gap between what non-hierarchical protocols achieve and an idealized caching scenario. In future multi-GPUs, larger shared L2 caches will only amplify the cost of coarse-grained cache invalidations and of reloading invalidated data from remote GPUs via bandwidth-limited links. Indeed, Fig. 3 shows that it is common for multiple GPMs on the same GPU to redundantly access a common range of addresses stored on remote GPUs. We therefore build HMG as a hierarchical protocol capable of being extended across multiple GPUs.

There has been much research into hierarchical cache coherence for CPUs. However, unlike GPUs, CPUs usually enforce a stronger memory model (e.g., TSO) and have much stricter latency requirements. As such, CPU coherence protocols such as MESI track ownership to exploit write data locality [40–42]. Many transient states are added to reduce the coherence stalls, resulting in prohibitive verification complexity [43]. Industrial products implemented more aggressive optimizations. For example, Sun's WildFire had special OS support for memory page replication and migration [44]. Intel's Skylake introduced IO directory cache and HitMe cache to reduce memory access latency [42]. These complexities are appropriate for latency-bound CPUs, but GPUs permit far more relaxed memory behavior, and hence HMG shows that the costs of such CPU-like protocols remain unnecessary for multi-GPUs.

B. Leveraging GPU Weak Memory Models

Besides the change of hardware architecture, scoped GPU memory models also inform the design of a good GPU coherence hierarchy. While non-scoped CPU memory models require all memory accesses to be kept coherent, GPU memory models that do explicitly expose scopes as part of the programming model require coherence to be enforced only at synchronization boundaries, and only with respect to other threads in the scope in question. The NVIDIA GPU memory model makes this relaxed nature of coherence very explicit [21]. A common pattern in multi-GPU applications will be for CTA or kernels running on a single GPU to synchronize with each other first, and with kernels on other GPUs less frequently. Such patterns rely heavily on the comparative efficiency of `.gpu` scope over `.sys` scope; while some prior work has concluded that scopes are unnecessary within a single GPU [16], the latency/bandwidth gap between the broadest and narrowest scope is an order of magnitude larger in multi-GPU environments.

Furthermore, although some prior work has proposed multi-copy-atomic memory models for GPUs [45], recent GPU scoped memory models have since formalized the lack of such a requirement [21, 24]. Loosely speaking, multi-copy-atomicity requires memory to behave as if it were a single atomic unit, with only thread-private buffering allowed between cores and memory. As GPUs share an L1 cache across an SM, GPUs today are not multi-copy-atomic. Multi-copy-atomicity also can create apparent delays for subsequent memory accesses. Most CPUs enforce multi-copy-atomicity using sophisticated coherence protocols with many transient states and by using out-of-order execution and speculation to hide the latency overheads. Some prior studies have found that single-GPU coherence protocols can also tolerate multi-copy-atomicity. For example, to reduce stalls, GPU-VI [15] added 3 and 12 transient states and 24 and 41 coherence state transitions in the L1 and L2 caches, respectively. In multi-GPU environments, however, the round trip time to remote GPUs is an order of magnitude larger and would put significantly increased pressure on the coherence protocol's ability to hide the latency. Instead, by leveraging non-multi-copy-atomicity, HMG eliminates transient states and invalidation acknowledgments altogether.

IV. Baseline Non-Hierarchical Cache Coherence

We now describe how a non-hierarchical cache coherence (NHCC) protocol can be optimized for modern weak GPU memory models. Like most scoped protocols, NHCC propagates synchronization memory accesses to different caches according to the user-provided scope annotations. As compared to GPU-VI [15], NHCC eliminates all transient states and most invalidation acknowledgments. However, it does not take the architectural hierarchy into account. As such, it will serve as our baseline during our later evaluations. In the next section, we will extend NHCC with a notion of hierarchy so that it scales better on larger multi-GPU systems like Fig. 1.

A. Architectural Overview

A high-level diagram of our baseline single-GPU architecture for NHCC is shown in Fig. 4. We assume L1 caches remain software-managed and write-through, as in GPUs today. Each GPU module (GPM) has an L2 cache that holds both local and remote-GPM DRAM accesses contending for cache capacity with a typical replacement policy such as

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

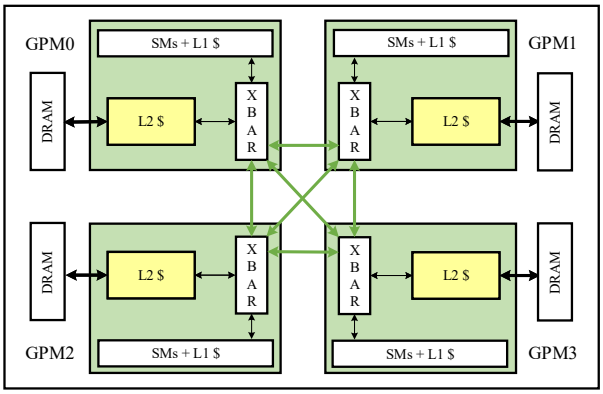


Figure 4: Future GPUs will consist of multiple GPU modules (GPMs). For example, each GPM might be a chiplet in a single package.

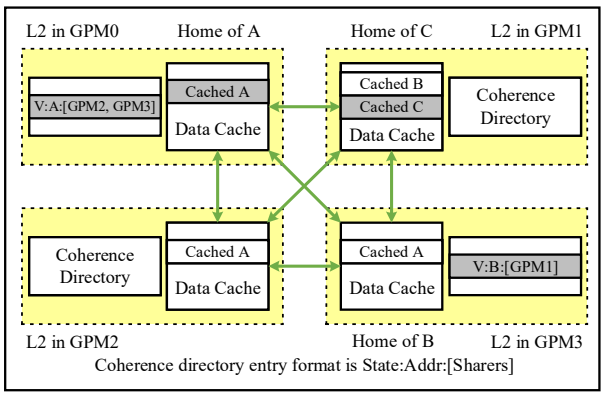


Figure 5: NHCC coherence architecture. The dotted yellow boxes are the L2 caches from Fig. 4. The shaded gray cache lines and directory entries indicate lines for which the GPM in question is the *home node*.

least-recently-used (LRU). To support hardware inter-GPM coherence, one GPM in the system is chosen by some hash function as the *home node* for any given physical address. The home node always contains the most up-to-date value at each memory location.

Like many protocols, NHCC attaches an individual directory to every L2 cache within each GPM. The coherence directory is organized as a traditional set-associative structure. Each directory entry tracks the identity of all GPM sharers, along with coherence state. Like GPU-VI [15], each line can be tracked in one of two stable states: *Valid* and *Invalid*. However, unlike GPU-VI, NHCC does not have transient states, and it requires acknowledgments only for release operations. Non-synchronizing stores (i.e., the vast majority) do not require acknowledgments in NHCC.

We assume a non-inclusive inter-GPM L2 cache architecture to enable data to be cached freely across the different GPMs. Fig. 5 shows an example in which GPM0 serves as the home node for address A. Other GPMs may cache the value at A locally, but GPM0 maintains the authoritative copy. In the same figure, address B is cached in GPM1, even though GPM3 (the home node for B) is not caching B. Similarly, data can be cached in the home node only, as with address C in our example in Fig. 5.

In NHCC, explicit coherence maintenance messages (i.e., cache invalidations) are sent only in two cases: when there is read-write sharing between CTAs on different GPMs, and when there is a directory capacity eviction. The fact that most memory accesses incur no coherence overhead ensures that the GPU does not deviate from peak throughput in the GPU common case where data is either read-only or CTA-private. We measure the impact of coherence messages in Section VII.

To explain the basics of NHCC, we track the life of a memory reference as an example. First, a memory access from the SM queries the L1 cache. Upon a L1 miss or write, the request is routed to the local GPM L2 cache. If the request misses in the L2 (or writes to L2, again assuming a write-through policy), the address is checked to determine if the local GPM is the home node for this reference. If so, the request is routed to local DRAM. Otherwise, the request is routed to the L2 cache of the home node via the inter-GPM links. The request may then hit in the home node L2 cache, or it may get routed through to that particular GPM's off-chip memory. We provide full details below.

B. Coherence Protocol Flows in Detail

Table I details the full operation of NHCC. In this table, "local" refers to operations issued by the same GPM as the L2 cache which is handling the request. "Remote" requests are those originally issued by other GPMs. We walk through the entries in the table below.

Local Loads: When a local load request reaches the local L2 cache partition, if it hits, a reply is sent back to the requester directly. If the request misses, the next destination depends on where the data is mapped by the address hash function. If the local L2 cache partition happens to be the home node for the address being accessed, the request will be sent to DRAM. Otherwise, the load request will be forwarded to the home node. Loads with `.gpu` or `.sys` scope must always miss in the L1 cache and in the non-home L2 caches to guarantee forward progress.

Local Stores: Depending on L2 design, local stores may be stored as dirty data in the L2 cache, or alternatively they may be written through and stored as clean data in the L2 cache. All stores with scope greater than `.cta` (i.e., `.gpu` and `.sys`) must be written through in order to ensure forward progress. Data which is written back or written through the L2 is sent directly to DRAM if the local L2 cache is the home node for the address in question, or it is relayed to the home node otherwise. If the local GPM is the home node and the coherence directory has recorded

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

State	Local Ld	Local St/Atom	Remote Ld	Remote St/Atom	Replace Dir Entry	Invalidation
I	-	-	add s to sharers, →V	add s to sharers, →V	N/A	-
V	-	inv all sharers, →I	add s to sharers	add s to sharers, inv other sharers	inv all sharers, →I	forward inv to all sharers (HMG only), → I

Table I: NHCC and HMG coherence directory transition table. s refers to the sender of the message.

any sharers for the address in question, then these sharers must be notified that the data has been changed. As such, a local store triggers an invalidation message being sent to each sharer. These invalidations propagate in the background, off the critical path of any subsequent reads. There are no invalidation acknowledgments.

Remote Loads: When a remote load arrives at the local home L2 cache, it either hits in the cache and returns data to the requester, or it misses and forwards the request to DRAM. The coherence directory also records the ID of the requesting node. If the line is already being tracked, the requesting ID is simply added as an additional sharer. If the line is not being tracked yet, a new entry is allocated in the directory, possibly by evicting another valid entry (discussed further below).

Remote Stores: Remote stores that arrive at a home L2 are cached and written through or written back to DRAM, depending on the configuration of the L2. Since the requesting GPM may also be caching the stored data, the requester is recorded as a sharer. Since the data has been changed, all other sharers should be invalidated.

Atomics and Reductions: Atomic operations must always be performed at the home node L2. From a coherence transition perspective, these operations are treated as stores.

Invalidations: Upon receiving an invalidation request, any local clean copy of the address in question is invalidated. No acknowledgment needs to be sent.

Directory Entry Eviction/Replacement: Because the coherence directory is implemented as a set-associative cache, there may be entry evictions due to capacity and conflict misses. To ensure correctness, invalidation messages must be sent to all sharers of the entry that is being evicted. As with invalidations triggered by stores, these invalidations propagate in the background and do not require acknowledgments to be sent in return.

Acquire: Acquire operations greater than `.cta` scope (i.e., `.gpu` and `.sys`) invalidate the entire local L1 cache, following software coherence practice. However, they do not propagate past the L1 cache, as L2 coherence in GPMs is now maintained using NHCC.

Release: Release operations trigger a writeback of all dirty data to the respective home nodes, if writeback caches are being used. Releases also ensure completion of any write-through operations and invalidation messages that are still in flight. Release operations greater than `.cta` scope are propagated through the local L2 to all remote L2s to ensure that all invalidation messages have arrived at their destinations. Once this step is complete, each remote L2 sends back an acknowledgment for the release operation itself. The local L2 then collects these acknowledgments and returns a single response to the original requester.

Cache Eviction: Two design options are possible upon cache line eviction. First, a clean cache line being evicted from an L2 cache in a non-home GPM could send a downgrade message to the home node. This allows the home node to delete the remote node as a sharer and will potentially save an invalidation message from being sent later. However, this is not required for correctness. The second option is to have valid clean cache lines get silently evicted. This eliminates the overhead of the downgrade message, but it triggers an unneeded invalidation message upon eventual eviction of the coherence directory entry. Optionally, dirty cache lines being evicted and written back can use a new message type indicating that the data must be updated but that the issuing GPM need not be tracked as a sharer going forward. Again, this optimization is not strictly required for correctness, but may be useful in implementations using writeback caches.

V. Hierarchical Multi-GPU Cache Coherence

Like most prior work, NHCC is designed for single-GPU scenarios and does not take the hierarchy between intra- and inter-GPU connections into account. This becomes a problem as we try to extend protocols like NHCC to multiple GPUs, as inter-GPU bandwidth limitations become a bottleneck.

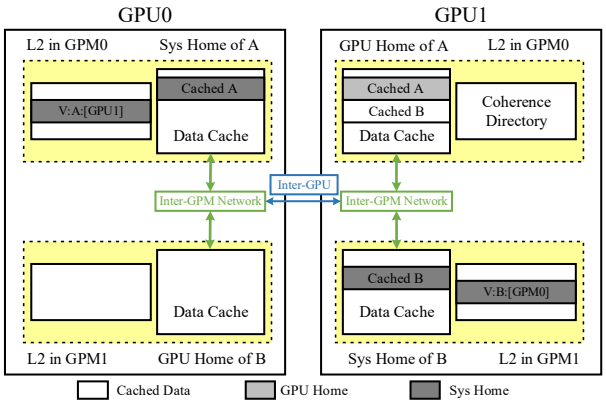
To better exploit intra-GPU data locality, we propose a *hierarchical multi-GPU* (HMG) cache coherence protocol that extends NHCC to be able to take advantage of the type of locality that Fig. 3 highlights. The HMG protocol fundamentally enables multiple cache requests from individual GPMs to be coalesced and/or cached within a single GPU *before* traversing the lower-bandwidth inter-GPU links, thereby saving bandwidth and energy.

A. Architectural Overview

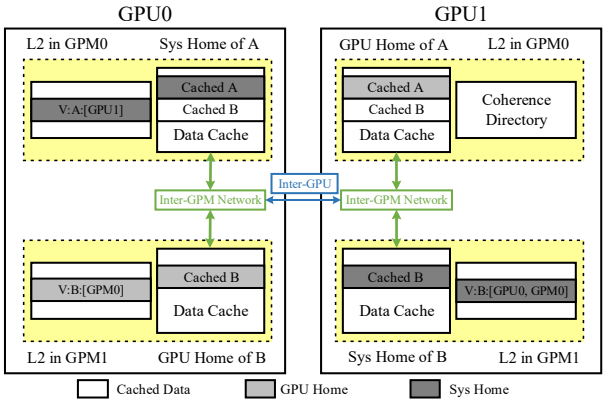
HMG is composed of two layers. The first layer is designed for intra-GPU caching, while the second layer is targeted at optimizing memory request routing in inter-GPU settings. For the intra-GPU layer, we define a *GPU home node* for each given address within each individual MCM-GPU. An MCM-GPU home node manages inter-GPM coherence using NHCC described in Section IV. Using the intra-GPU coherence layer, data that is cached within a MCM-GPU can be consumed by multiple GPMs on that GPU without consulting a remote GPU.

We define one of the GPU home nodes for each address to be the *system home node*. The choice of system home node

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.



(a) Before: GPU0:GPM0 is about to load address B



(b) After: B is cached in the L2 of the GPU0 home node for B as well as in the L2 of the original requester

Figure 6: Hierarchical coherence in multi-GPU systems. Loads are routed from the requesting GPM to the GPU home node, and then to the system home node, and responses are returned and cached accordingly.

can be made using any NUMA page allocation policy, such as first touch page placement, NVIDIA Unified Memory [37], static distribution, or any other reasonable heuristic. Among multiple GPUs, sharers are tracked by the directory using a hierarchy-aware variant of the NHCC directory design. Specifically, each GPU home node will track any sharers among other GPMs in the same GPU. Each system home node will track any sharers among other GPUs, but not individual GPMs within these other GPUs. For an M-GPM, N-GPU system, each directory entry will therefore need to track as many as $M + N - 2$ sharers.

The hierarchical caching mechanism of an example two-GPU system is shown in Fig. 6. Each GPU is shown with only two GPMs for brevity, but the protocol itself can extend to an arbitrary number of GPUs, with an arbitrary number of GPMs per GPU. In Fig. 6(a), the system home node of address A is the L2 cache residing in GPU0:GPM0. This particular L2 cache also serves as the GPU home node for the same address within GPU0. The L2 cache in GPU0:GPM1 is kept coherent with the L2 cache in GPU0:GPM0 using the intra-GPU protocol layer. The L2 cache in GPU1:GPM0 serves as the GPU1 home node for address A, and it is kept coherent with the L2 cache in GPU1:GPM1 using the intra-GPU layer. Both GPU home nodes are kept coherent using the inter-GPU protocol layer.

Furthermore, suppose that from the state shown in Fig. 6(a), GPU0:GPM0 wants to load address B, and the system home node for address B is mapped to GPU1:GPM1. GPU0:GPM1 is the GPU0 home node for B, so the load request propagates from GPU0:GPM0 to GPU0:GPM1 (the GPU home node), and then to GPU1:GPM1 (the system home node). When the response is sent back to the requester, GPU0 (but *not* GPU0:GPM0 or GPU0:GPM1) is recorded as a sharer by the directory of the system home node GPU1:GPM1, and GPU0:GPM0 is recorded as a sharer by the directory of the GPU0 home node GPU0:GPM1, as shown in Fig. 6(b).

B. Coherence Protocol Flows in Detail

HMG behaves similarly to Table I but adds the single extra transition shown in Table I. No extra coherence states are added. We highlight the important differences between NHCC and HMG as follows.

Loads: Loads progress through the cache hierarchy from the local L2 cache, to the GPU home node, to the system home node. Specifically, loads that miss in the GPM-local L2 cache are routed to the GPU home node, unless the GPM-local L2 cache is already the GPU home node. From there, loads that miss in the GPU home node are routed to the system home node, unless the GPU home node is also the system home node. Loads that miss in the system home node are routed to DRAM.

Non-synchronizing loads (i.e., the vast majority) and loads with `.cta` scope can hit in all caches. However, loads with `.gpu` scope must miss in all caches prior to the GPU home node. Loads with `.sys` scope must also miss in the GPU home node; they may only hit in the system home node.

Loads propagating from the GPU home node to the system home node do not carry information about the GPM that originally requested the data. Because this information is already stored by the GPU home node, it would be redundant to store it again in the directory of the system home node. Instead, invalidations are propagated to sharers hierarchically as described below.

Stores: Stores are routed through a similar hierarchy as they write-through and/or write-back. Specifically, stores propagating past the GPM-local L2 cache are routed to the GPU home node (unless the GPM-local L2 is already the GPU home node), and stores propagating past the GPU home node are routed to the system home node (unless the GPU home node is already the system home node). Stores propagating past the system home node are written to DRAM. Similar to loads, stores or write-back/write-through operations

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

propagating from the GPU home node to the system home node carry only the GPU identifier, not the identifier of the GPM within that GPU.

Stores must be written through at least to the home node for the scope in question: the L1 cache for non-synchronizing and `.cta`-scoped stores, the GPU home node for `.gpu`-scoped stores, and the system home node for `.sys`-scoped stores. This ensures that synchronization operations will make forward progress.

Atomics and Reductions: Atomics are always performed in the home node for the scope in question and they continue to be treated as stores for the purposes of coherence protocol transitions, just as in NHCC. Once performed at the home node, responses are propagated back to the requester just as load responses are handled and the result is stored as a dirty line or written through to subsequent levels of the cache hierarchy, just as a store would be. For example, the result of a `.gpu`-scoped atomic read-modify-write operation performed in the GPU will be written through to the system home node, in systems which configure the GPU home node to be write-through for stores.

Invalidations: Because sharers are tracked hierarchically, invalidations sent due to stores and directory evictions must also propagate hierarchically. Invalidations sent from the system or GPU home node to other GPMs in the same GPU are processed and dropped without acknowledgment, just as in NHCC. However, in HMG any invalidations received by a GPU home node from the system home node must also be propagated to any and all GPM sharers within the same GPU. This is the special transition shown in Table I for HMG.

Acquire: As before, `.cta`-scoped acquire operations invalidate the local L1 cache, but nothing more, as all levels of L2 cache are being kept hardware-coherent.

Release: Release operations trigger writeback of all dirty data, at least to the home node for the scope being released. They also still ensure completion of any write-through operations and invalidation messages still in flight to the home node for the scope in question. A `.gpu`-scoped release operation, however, need not flush all write-back operations across the inter-GPU network before returning a completion acknowledgment to the original requester.

VI. Evaluation Methodology

To evaluate HMG, we use a proprietary industrial simulator to model a multi-GPU system described in Table II. The simulator is driven by program traces that record instructions, registers, memory addresses, and CUDA events. All micro-architectural scheduling, and thus time for execution, is dynamic within the simulator and respects functional dependencies such as work scheduling, barrier synchronization, memory access latencies. However, it cannot accurately model spin-lock synchronizations in memory. While this type of communication is legal on current NVIDIA hardware, it is not yet widely adopted due to performance overheads and

Structure	Configuration
Number of GPUs	4
Number of SMs	128 per GPU, 512 in total
Number of GPMs	4 per GPU
GPU frequency	1.3GHz
Max number of warps	64 per SM
OS Page Size	2MB
L1 data cache	128KB per SM, 128B lines
L2 data cache	12MB per GPU 128B lines, 16 ways
L2 coherence directory	12K entries per GPU module each entry covers 4 cache lines
Inter-GPM bandwidth	2TB/s per GPU, bi-directional
Inter-GPU bandwidth	200GB/s per link, bi-directional
Total DRAM bandwidth	1TB/s per GPU
Total DRAM capacity	32GB per GPU

Table II: Configuration of simulated architecture.

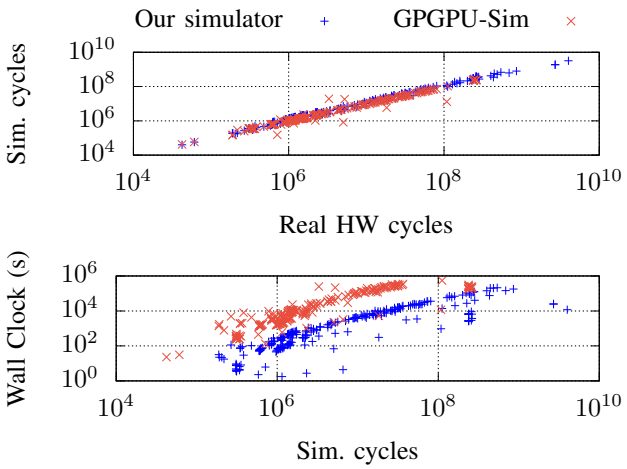


Figure 7: Simulator correlation vs. a NVIDIA Quadro GV100 and simulation runtime for our simulator and GPGPU-Sim [46–49].

not present in our suite of workloads. Simulating the system-level effects of fine-grained synchronization, in reasonable time, without sacrificing fidelity [20, 50] remains an open problem for GPU researchers.

Fig. 7 shows our simulator correlation versus a NVIDIA Quadro-GV100 GPU across a range of targeted microbenchmarks, public, and proprietary workloads. Fig. 7 also shows the corresponding data for GPGPU-Sim, a widely-used academic GPU architecture simulator [51], with simulations capped at running for about one week. Our simulator has a correlation coefficient of 0.99 and average absolute error of 0.13. This compares favorably to GPGPU-Sim (at 0.99 and 0.045, respectively), as well as other recently reported simulator results [52] while being significantly faster, which allows us to run forward-looking GPU configurations more easily. Our simulator inherits the contiguous CTA scheduling and first-touch page placement polices from prior work [5, 13] to maximize data locality in memory.

To perform our evaluation, we choose a public subset of workloads (shown in Table III) [29–35] that have sufficient parallelism to fill a 4-GPU system. These benchmarks utilize

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

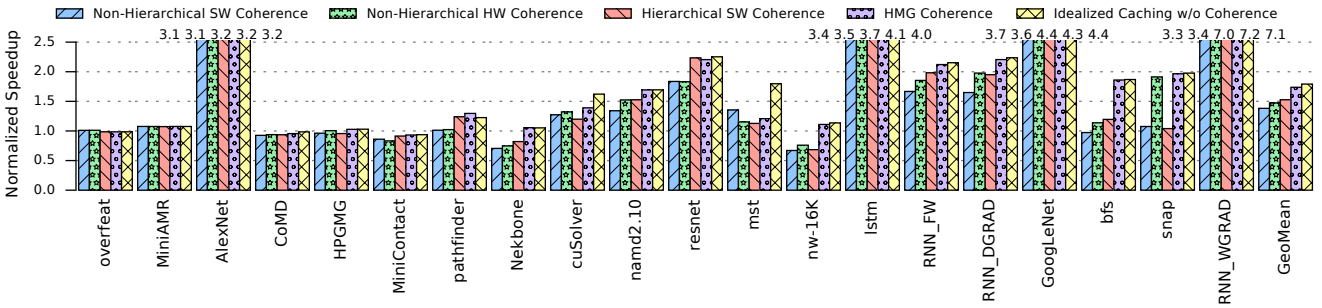


Figure 8: Performance of a 4-GPU system, where each GPU is composed of 4 GPMs. Performance is normalized to a 4-GPU system that disallows caching of remote GPU data. Five configurations are evaluated: software protocols with non-hierarchical and hierarchical implementations, NHCC, HMG, and ideal caching without coherence overhead.

Benchmark	Abbrev.	Footprint
cuSolver	cuSolver	1.60 GB
HPC_CoMD-xyz49	CoMD	313 MB
HPC_HPGMG	HPGMG	1.32 GB
HPC_MiniAMR-test2	MiniAMR	1.80 GB
HPC_MiniContact	MiniContact	246 MB
HPC_namd2.10	namd2.10	72 MB
HPC_Nekbone-10	Nekbone	178 MB
HPC_snap	snap	3.44 GB
Lonestar_bfs-road-fla	bfs	26 MB
Lonestar_mst-road-fla	mst	83 MB
ML_AlexNet_conv2	AlexNet	812 MB
ML_GoogLeNet_conv2	GoogLeNet	1.15 GB
ML_lstm_layer2	lstm	710 MB
ML_overfeat_layer1	overfeat	618 MB
ML_resnet	resnet	3.20 GB
ML_RNN_layer4_DGRAD	RNN_DGRAD	29 MB
ML_RNN_layer4_FW	RNN_FW	40 MB
ML_RNN_layer4_WGRAD	RNN_WGRAD	38 MB
Rodinia_nw-16K-10	nw-16K	2.00 GB
Rodinia_pathfinder	pathfinder	1.49 GB

Table III: Benchmarks used for evaluation.

scoped and/or inter-kernel synchronization patterns. This ensures that performance does not regress on traditional workloads even as we accelerate workloads with more fine-grained sharing. Specifically, `cuSolver`, `namd2.10`, and `mst` use `.gpu`-scoped synchronization explicitly, others utilize inter-kernel communication by launching frequent dependent kernels, and a few are traditional bulk-synchronous providing a historical comparative baseline.

Coherence Protocol Implementations: This work implements and compares 4 coherence possibilities: a non-hierarchical software protocol (conventional software coherence with scopes and bulk-invalidation of caches), a non-hierarchical hardware protocol (NHCC), a hierarchical software protocol (conventional software coherence with hierarchical extension to leverage scopes), and our proposed hierarchical hardware protocol (HMG). We also compare them to idealized caching that does not enforce coherence; this serves as a loose upper bound for performance that can be achieved via hardware caching. For non-hierarchical protocols, multi-GPU systems like Fig. 1 behaves as a single flat GPU with more GPMs.

NHCC and HMG behave according to Section IV and V respectively. Load-acquire operations in our software coherence protocols trigger bulk cache invalidations in any caches between the issuing SM and the home node for the scope in question. For example, `.gpu`-scoped loads will invalidate both the L1 cache of the issuing SM and the GPM-local L2 cache. In the hierarchical protocol, `.sys`-scoped loads invalidate the L1 cache of the issuing SM and all L2 caches of the issuing GPU. However, in the non-hierarchical protocol, `.sys`-scoped loads need not to invalidate L2 caches in other GPMs of the same GPU, as subsequent loads will not fetch stale data from those caches. Store-release operations stall subsequent operations until the home node for the scope in question clears all pending writes.

In our evaluation, all caches are write-through. We do not implement the optional sharer downgrade messages. We model one directory optimization: each entry tracks the state of four cache lines together. This enables $12K \times 4 \times 128B = 6MB$ of data assigned to each GPM to be actively shared by other GPMs and/or GPUs. Section VII-B later shows performance sensitivity to the choices of these parameters.

VII. Results and Discussion

We first compare the performance of HMG to NHCC, software coherence protocols, and idealized caching without any coherence overhead. Then we conduct sensitivity analysis to explore the design space of HMG.

A. Performance Analysis

Single-GPU System: Like prior work, we observe that for most benchmarks, both software and hardware coherence generally perform similarly and close to an idealized non-coherent caching scheme. The relatively small L2 caches and relatively large inter-GPM bandwidths can minimize the performance penalty of cache invalidations in single-GPU systems, and hence we do not elaborate on them further here.

Multi-GPU System: While software coherence may be sufficient within individual GPUs, even for benchmarks with fine-grained thread-to-thread communication, Fig. 8 shows that the benefits of HMG are much more pronounced in

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

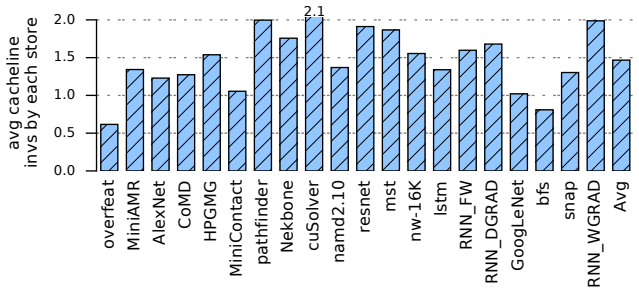


Figure 9: Average number of cache lines invalidated by each store request on shared data.

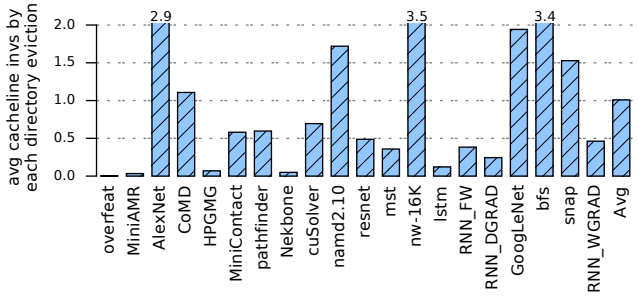


Figure 10: Average number of cache lines invalidated by each coherence directory eviction.

deeply hierarchical multi-GPU systems, especially for the applications which have more fine-grained data sharing (i.e, the right half side). In a 4-GPU system, HMG generally outperforms both software coherence protocols and NHCC. Both software and hardware hierarchical protocols significantly benefit from the additional intra-GPU data locality. Meanwhile, the non-hierarchical protocols suffer from larger inter-GPU latency and bandwidth penalties.

Fig. 9 and 10 show that cache line invalidations due to store instructions or coherence directory evictions do not have a significant impact on performance of HMG. This is because stores only trigger invalidations if there is a sharer for the same address and typically only a small percentage of the memory footprint of each workload contains read-write shared data. Even among stores or directory evictions that do trigger sharer invalidations, there are generally no more than two sharers in our workloads. These observations highlight the benefit of tracking sharers dynamically, rather than e.g., classifying data sharing type alone [14].

Graph workloads' fine-grained, often conflicting access patterns can lead to false sharing. Store operations in software coherence protocols will simply write this data through, but HMG might trigger frequent invalidations (in these experiments, at the granularity of four cache lines per directory entry), depending on the input sets. In such cases, the hardware protocol HMG will have higher overhead. This explains the performance of `mst`, for example. For most other applications, the benefits of HMG outweigh the costs.

We also profile the bandwidth overhead of invalidation

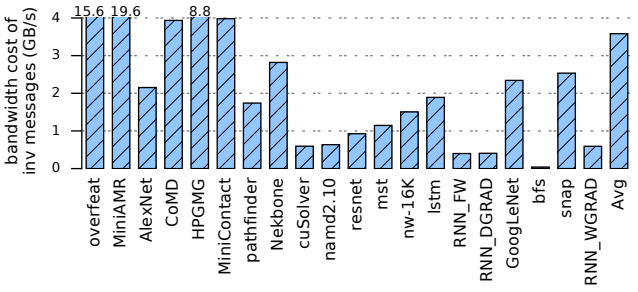


Figure 11: Total bandwidth cost of invalidation messages.

messages. Fig. 11 shows that the total bandwidth cost of invalidation messages is generally as low as just a few gigabytes per second. This is consistent with prior data since there is little read-write sharing and a low number of sharers when invalidations must be sent out. The size of each invalidation message is also relatively small compared to a GPU cache line. Combined with the fact that GPU workloads are generally latency tolerant, it becomes clear that HMG for hierarchical multi-GPUs can deliver high performance, at high efficiency, with relatively simple hardware implementation. Overall, our results confirm prior suggestions that complicated CPU-like coherence protocols are unnecessary, even in hierarchical multi-GPU contexts. By providing a lightweight coherence enforcement mechanism specifically tuned to the scoped memory model, HMG is able to deliver 97% of the ideal speedup that inter-GPU caching can possibly enable.

B. Sensitivity Analysis

To understand the relationship between our architectural parameters and the performance of HMG, we performed sensitivity studies across a range of design space parameters.

- Bandwidth-limited inter-GPU links are the main cause of NUMA effects that often bottleneck multi-GPU performance. Fig. 12 shows that when sweeping across inter-GPU bandwidths, HMG is always the best performing coherence option, even when absolute performance begins to saturate due to sufficient inter-GPU bandwidth.
- The impact of L2 cache size on performance is shown in Fig. 13. Because of the overhead of cache invalidation, the benefits of increased L2 capacity are restricted by software coherence protocols. Conversely, the performance of HMG increases as capacity grows, indicating the advantage of HMG will only become more favorable in systems with larger caches.
- Coherence directory sizing presents a trade-off between power/area and coverage/performance. As Fig. 14 shows, the performance of our proposed HMG is somewhat sensitive to directory size. The benefit of hardware-managed coherence over software coherence shrinks if the directory is not able to track enough sharers and is forced to perform additional cache invalidations across GPUs. However, our modestly-sized directories are large

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

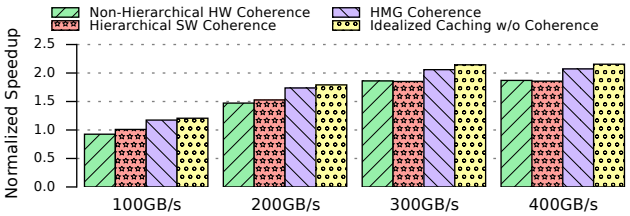


Figure 12: Performance sensitivity to inter-GPU bandwidth (baseline is no caching with configurations of Table II).

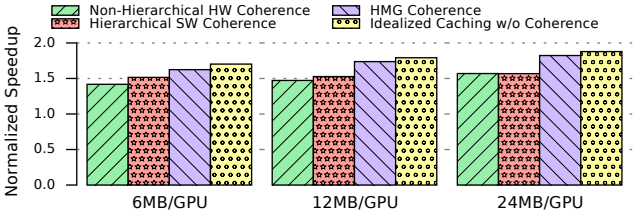


Figure 13: Performance sensitivity to L2 cache size (baseline is no caching with configurations of Table II).

enough to successfully capture the locality needed to deliver near-ideal caching performance.

- Coarse-grained directory entry tracking granularity (e.g., where each entry tracks four cache lines at a time) allows directories to be made smaller, but it also introduces a risk of false sharing. In order to quantify this impact, we varied the granularity tracked by each directory entry while simultaneously adjusting the total number of entries in order to keep the total coverage constant. The results (not pictured) showed minimal sensitivity, and we therefore conclude that coarse-grained directory tracking is a useful optimization for HMG.

C. Hardware Costs

In our HMG implementation, each directory entry needs to track as many as six sharers: three GPMs in the same GPU and three other GPUs. Therefore, a 6 bit vector is required for the sharer list. Because our protocol uses just two states, Valid and Invalid, only one bit is needed to track directory entry state. We assume 48 bits for tag addresses, so each entry in the coherence directory requires 55 bits of storage. Every GPM has 12K directory entries, so the total storage cost of the coherence directories is 84KB, which is only 2.7% of each GPM's L2 cache data capacity, a small price to pay for large performance improvements in future multi-GPUs.

D. Discussion

On-package integration [5, 12, 53] along with off-package integration technologies [6–8] enable more and more GPU modules to be integrated in a single systems. However, NUMA effects are exacerbated as the number of GPMs, GPUs, and non-uniform topologies increase within the system. In these situations, HMG's coherence directory would need

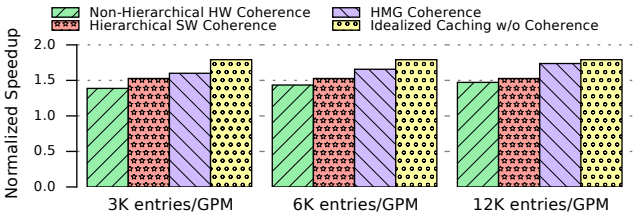


Figure 14: Performance sensitivity to the coherence directory size (baseline is no caching with configurations of Table II).

to record more sharers and cover a larger footprint of shared data, but the system performance will likely be more sensitive to the link speeds and the actual network topology. As shown in Fig. 14, HMG can perform very well even after we reduce the coherence directory size by 50%, showing that there is still room to scale HMG to larger systems. We envision our proposed coherence protocol being applicable for systems that can be comprised by a single NVSwitch-based network within a single operating system node. Systems significantly larger than this (e.g., 1024-GPU systems) may be decomposed into a hierarchy consisting of hardware-coherent GPU-clusters which are in turn share data using software mechanisms such as MPI or SHMEM [54, 55].

The rise of MCM-GPU-like architectures might seem to motivate adding scopes in between `.cta` and `.gpu`, to minimize the negative effects of coherence. However, our single-GPU performance results indicate that our workloads are minimally sensitive to the inter-GPM coherence mechanism due to high inter-GPM bandwidth. As a result, the performance benefits of introducing a new `.gpm` scope may not outweigh the added programmer burden of using numerous scopes. We expect further exploration of other software-hardware coherence interactions to remain an active area of research as GPU systems continue to grow in size.

VIII. Related Work

Memory Consistency: Some previous work enforced sequential consistency (SC) in GPUs [27, 56, 57]. They found that TSO and relaxed memory models could not significantly outperform SC. However, modern GPU products still enforce relaxed memory models, as weak models allow for more microarchitectural flexibility and arguably better performance/power/area tradeoffs.

Cache Coherence: We have discussed most of the existing GPU coherence protocols in Section II-D. Others have also proposed GPU coherence based on physical or logical timestamps [15, 27]. Without explicit invalidation messages, cache lines are self-invalidated after timestamps are expired, so they can reduce traffic load and improve performance efficiently. However, they considered neither architecture hierarchy nor scoped memory model. In this paper, we explore the coherence support for future deeply hierarchical GPU systems with scoped memory model enforcement.

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

Coherence hierarchy has been commonly employed in CPUs [58, 59]. Most hierarchical CPU designs [40–42, 44, 60] have adopted MESI-like coherence, which has been proven to be a poor fit for GPUs [15, 25]. HMG shows that the complexity of extra states is also unnecessary for hierarchical GPUs. Both DASH [41] and WildFire [44] increased the complexity even more by employing the mixed coherence policy: intra-cluster snoopy coherence and inter-cluster directory-based coherence. To implement consistency model efficiently, Alpha GS320 [60] separated the commit events to allow time-critical replies to bypass inbound requests without violating memory order constraints. HMG can achieve almost optimal performance without such overheads.

Shared data synchronization in the unified memory space of heterogeneous systems also requires efficient coherence protocols [61–63]. We expect that HMG would be integrated nicely with such schemes due to its simple states and clear coherence hierarchy.

GPU Scaling: As Section II-A described, previous work built larger GPU systems with distributed architectures [5, 13, 14]. They alleviated the NUMA effect with aggressive caching, which correctness is enforced with either conventional software or hardware coherence without recording exact sharers. However, none of them targeted deeply hierarchical systems and scoped memory models.

IX. Conclusion

In this paper, we introduce HMG, a novel cache coherence protocol specifically tailored to scale well to hierarchical multi-GPU systems. HMG provides efficient support for fine-grained synchronization now permitted under recently-formalized scoped GPU memory models. We find that, without much complexity, simple hierarchical extensions and optimizations to existing coherence protocols can take advantage of relaxations now permitted in scoped memory models to achieve 97% performance of an ideal caching scheme that has no coherence overhead. Thanks to its cheap hardware implementation and high performance, HMG demonstrates the most practical solution available for extending cache coherence to future hierarchical multi-GPU systems, and thereby for enabling continued performance scaling of applications onto larger and larger GPU-based systems.

X. Acknowledgment

We are grateful to Mieszko Lis, and the anonymous reviewers for their insightful feedback.

References

[1] Inside HPC, "TOP500 Shows Growing Momentum for Accelerators," https://insidehpc.com/2015/11/top500-shows-growing-momentum-for-accelerators/, 2015, accessed on 2019-08-04.

[2] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," *CoRR*, vol. abs/1410.0759, October 2014. [Online]. Available: http://arxiv.org/abs/1410.0759

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, September 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[4] P. Harish and P. Narayanan, "Accelerating Large Graph Algorithms on the GPU Using CUDA," in *International conference on high-performance computing (HiPC)*. Springer, 2007, pp. 197–208.

[5] 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 44th International Symposium on Computer Architecture (ISCA)*. ACM, 2017, pp. 320–332.

[6] NVIDIA, "NVIDIA NVLink: High Speed GPU Interconnect," https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/, accessed on 2019-08-04.

[7] NVIDIA, "NVIDIA NVSwitch: The World's Highest-Bandwidth On-Node Switch," https://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf, accessed on 2019-08-04.

[8] "AMD's answer to Nvidia's NVLink is xGMI, and it's coming to the new 7nm Vega GPU," https://www.pcgamesn.com/amd-xgmi-vega-20-gpu-nvidia-nvlink, accessed on 2019-08-04.

[9] NVIDIA, "NVIDIA DGX-1: Essential Instrument for AI Research," https://www.nvidia.com/en-us/data-center/dgx-1/, 2017, accessed on 2019-08-04.

[10] ——, "NVIDIA DGX-2: The world's most powerful AI system for the most complex AI challenges," https://www.nvidia.com/en-us/data-center/dgx-2/, 2018, accessed on 2019-08-04.

[11] ——, "NVIDIA HGX-2: Powered by NVIDIA Tesla V100 GPUs and NVSwitch," https://www.nvidia.com/en-us/data-center/hgx/, 2018, accessed on 2019-08-04.

[12] J. W. Poulton, W. J. Dally, X. Chen, J. G. Eyles, T. H. Greer, S. G. Tell, J. M. Wilson, and C. T. Gray, "A 0.54 pJ/b 20 Gb/s Ground-Referenced Single-Ended Short-Reach Serial Link in 28 nm CMOS for Advanced Packaging Applications," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 48, no. 12, pp. 3206–3218, 2013.

[13] 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 50th International Symposium on Microarchitecture (MICRO)*. ACM, 2017, pp. 123–135.

[14] V. Young, A. Jaleel, E. Bolotin, E. Ebrahimi, D. Nellans, and O. Villa, "Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems," in *Proceedings of the 51th International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 339–351.

[15] I. Singh, A. Shriraman, W. W. Fung, M. O'Connor, and T. M. Aamodt, "Cache Coherence for GPU Architectures," in *Proceedings of the 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 578–590.

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

[16] M. D. Sinclair, J. Alsop, and S. V. Adve, "Efficient GPU Synchronization Without Scopes: Saying No to Complex Consistency Models," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*. ACM, 2015, pp. 647–659.

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

[18] J. Kim and C. Batten, "Accelerating Irregular Algorithms on GPGPUs Using Fine-Grain Hardware Worklists," in *Proceedings of the 47th International Symposium on Microarchitecture (MICRO)*. IEEE, 2014, pp. 75–87.

[19] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding Irregular GPGPU Graph Applications," in *International Symposium on Workload Characterization (IISWC)*. IEEE, 2013, pp. 185–195.

[20] M. D. Sinclair, J. Alsop, and S. V. Adve, "HeteroSync: A Benchmark Suite for Fine-Grained Synchronization on Tightly Coupled GPUs," in *International Symposium on Workload Characterization (IISWC)*. IEEE, 2017, pp. 239–249.

[21] D. Lustig, S. Sahasrabuddhe, and O. Giroux, "A Formal Analysis of the NVIDIA PTX Memory Consistency Model," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2019, pp. 257–270.

[22] "HSA Platform System Architecture Specification Version 1.2," http://www.hsafoundation.com/?ddownload=5702, accessed on 2019-07-07.

[23] "The OpenCL Specification Version 2.2," https://www.khronos.org/registry/OpenCL/specs/2.2/pdf/OpenCL_API.pdf, accessed on 2019-07-07.

[24] D. R. Hower, B. A. Hechtman, B. M. Beckmann, B. R. Gaster, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous-Race-Free Memory Models," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2014, pp. 427–440.

[25] B. A. Hechtman, S. Che, D. R. Hower, Y. Tian, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "QuickRelease: A Throughput-oriented Approach to Release Consistency on GPUs," in *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 189–200.

[26] J. Alsop, M. S. Orr, B. M. Beckmann, and D. A. Wood, "Lazy Release Consistency for GPUs," in *Proceedings of the 49th International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, p. 26.

[27] X. Ren and M. Lis, "Efficient Sequential Consistency in GPUs via Relativistic Cache Coherence," in *Proceedings of the 23rd International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 625–636.

[28] W. J. Dally, C. T. Gray, J. Poulton, B. Khailany, J. Wilson, and L. Dennison, "Hardware-Enabled Artificial Intelligence," in *Symposium on VLSI Circuits*. IEEE, 2018, pp. 3–6.

[29] L. Chien, "How to Avoid Global Synchronization by Domino Scheme," *NVIDIA GPU Technology Conference (GTC)*, 2014.

[30] M. Kulkarni, M. Burtscher, C. Casçaval, and K. Pingali, "LoneStar: A Suite of Parallel Irregular Programs," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2009, pp. 65–76.

[31] J. Gong, S. Markidis, E. Laure, M. Otten, P. Fischer, and M. Min, "Nekbone Performance on GPUs with OpenACC and CUDA Fortran Implementations," *The Journal of Supercomputing*, vol. 72, no. 11, pp. 4160–4180, 2016.

[32] D. Li and M. Becchi, "Multiple Pairwise Sequence Alignments with the Needleman-Wunsch Algorithm on GPU," in *SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*. IEEE, 2012, pp. 1471–1472.

[33] 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 *International Symposium on Workload Characterization (IISWC)*. IEEE, 2009, pp. 44–54.

[34] R. J. Zerr and R. S. Baker, "SNAP: SN (discrete ordinates) Application Proxy Description," *Los Alamos National Laboratories, Tech. Rep. LAUR-13-21070*, 2013.

[35] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable Molecular Dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[36] G. Diamos, S. Sengupta, B. Catanzaro, M. Chrzanowski, A. Coates, E. Elsen, J. Engel, A. Hannun, and S. Satheesh, "Persistent RNNs: Stashing Recurrent Weights On-Chip," in *International Conference on Machine Learning (ICML)*, 2016, pp. 2024–2033.

[37] NVIDIA, "Unified Memory in CUDA 6," https://devblogs.nvidia.com/unified-memory-in-cuda-6/, Nov 2013, accessed on 2019-08-04.

[38] K. Koukos, A. Ros, E. Hagersten, and S. Kaxiras, "Building Heterogeneous Unified Virtual Memories (UVMs) without the Overhead," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 1, p. 1, 2016.

[39] R. Ausavarungnirun, J. Landgraf, V. Miller, S. Ghose, J. Gandhi, C. J. Rossbach, and O. Mutlu, "Mosaic: A GPU Memory Manager with Application-Transparent Support for Multiple Page Sizes," in *Proceedings of the 50th International Symposium on Microarchitecture (MICRO)*. ACM, 2017, pp. 136–150.

[40] S.-L. Guo, H.-X. Wang, Y.-B. Xue, C.-M. Li, and D.-S. Wang, "Hierarchical Cache Directory for CMP," *Journal of Computer Science and Technology*, vol. 25, no. 2, pp. 246–256, 2010.

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.

[41] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor," in *Proceedings of the 17th International Symposium on Computer Architecture (ISCA)*. IEEE, 1990, pp. 148–159.

[42] D. Mulnix, "Intel Xeon Processor Scalable Family Technical Overview," https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview, 2017, accessed on 2019-08-04.

[43] D. J. Sorin, M. D. Hill, and D. A. Wood, "A Primer on Memory Consistency and Cache Coherence," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 3, pp. 1–212, 2011.

[44] E. Hagersten and M. Koster, "WildFire: A Scalable Path for SMPs," in *Proceedings of the 5th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1999, pp. 172–181.

[45] J. Alglave, M. Batty, A. F. Donaldson, G. Gopalakrishnan, J. Ketema, D. Poetzl, T. Sorensen, and J. Wickerson, "GPU Concurrency: Weak Behaviours and Programming Assumptions," in *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2015, pp. 577–591.

[46] A. Jain, M. Khairy, and T. G. Rogers, "A Quantitative Evaluation of Contemporary GPU Simulation Methodology," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)*, p. 35, 2018.

[47] M. Khairy, A. Jain, T. M. Aamodt, and T. G. Rogers, "Exploring Modern GPU Memory System Design Challenges through Accurate Modeling," *CoRR*, vol. abs/1810.07269, October 2018. [Online]. Available: http://arxiv.org/abs/1810.07269

[48] M. A. Raihan, N. Goli, and T. M. Aamodt, "Modeling Deep Learning Accelerator Enabled GPUs," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 79–92.

[49] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers *et al.*, "Analyzing Machine Learning Workloads Using a Detailed GPU Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 151–152.

[50] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao, H. Barclay, A. K. Ziabari, Z. Chen, R. Ubal, J. L. Abellán, J. Kim, A. Joshi, and D. Kaeli, "MGPUSim: Enabling Multi-GPU Performance Modeling and Optimization," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*. ACM, 2019, pp. 197–209.

[51] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2009, pp. 163–174.

[52] A. Gutierrez, B. Beckmann, A. Dutu, J. Gross, J. Kalamatianos, O. Kayiran, M. Lebeane, M. Poremba, B. Potter, S. Puthoor, M. D. Sinclair, M. Wyse, J. Yin, X. Zhang, A. Jain, and T. G. Rogers, "Lost in Abstraction: Pitfalls of Analyzing GPUs at the Intermediate Language Level," in *Proceedings of the 24th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 141–155.

[53] AMD, "Multi-Chip Module Architecture: The Right Approach for Evolving Workloads," http://developer.amd.com/wordpress/media/2017/11/LE-62006-SB-Latency-170824-Final-1.pdf, August 2017.

[54] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard, Version 3.1," https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf, June 2015.

[55] OpenSHMEM Project, "OpenSHMEM Application Programming Interface," http://www.openshmem.org/site/sites/default/site_files/OpenSHMEM-1.4.pdf, December 2017.

[56] B. A. Hechtman and D. J. Sorin, "Exploring Memory Consistency for Massively-Threaded Throughput-Oriented Processors," in *Proceedings of the 40th International Symposium on Computer Architecture (ISCA)*. ACM, 2013, pp. 201–212.

[57] A. Singh, S. Aga, and S. Narayanasamy, "Efficiently Enforcing Strong Memory Ordering in GPUs," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*. ACM, 2015, pp. 699–712.

[58] A. W. Wilson Jr, "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors," in *Proceedings of the 14th International Symposium on Computer Architecture (ISCA)*. ACM, 1987, pp. 244–252.

[59] D. A. Wallach, "PHD: A Hierarchical Cache Coherent Protocol," Ph.D. dissertation, Massachusetts Institute of Technology, 1992.

[60] K. Gharachorloo, M. Sharma, S. Steely, and S. Van Doren, "Architecture and Design of AlphaServer GS320," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2000, pp. 13–24.

[61] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous System Coherence for Integrated CPU-GPU Systems," in *Proceedings of the 46th International Symposium on Microarchitecture (MICRO)*. ACM, 2013, pp. 457–467.

[62] L. E. Olson, M. D. Hill, and D. A. Wood, "Crossing Guard: Mediating Host-Accelerator Coherence Interactions," in *Proceedings of the 22th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2017, pp. 163–176.

[63] J. Alsop, M. D. Sinclair, and S. V. Adve, "Spandex: A Flexible Interface for Efficient Heterogeneous Coherence," in *Proceedings of the 45th International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 261–274.

Authorized licensed use limited to: Auckland University of Technology. Downloaded on June 05,2020 at 12:37:41 UTC from IEEE Xplore. Restrictions apply.