RESEARCH-ARTICLE

Locality-Aware Optimizations for Improving Remote Memory Latency in Multi-GPU Systems

LEUL BELAYNEH, University of Michigan, Ann Arbor, Ann Arbor, MI, United States

HAOJIE YE, University of Michigan, Ann Arbor, Ann Arbor, MI, United States

KUAN-YU CHEN, University of Michigan, Ann Arbor, Ann Arbor, MI, United States

DAVID BLAAUW, University of Michigan, Ann Arbor, Ann Arbor, MI, United States

TREVOR MUDGE, University of Michigan, Ann Arbor, Ann Arbor, MI, United States

RONALD DRESLINSKI, University of Michigan, Ann Arbor, Ann Arbor, MI, United States

View all

Published: 27 January 2023

Citation in BibTeX format

PACT '22: International Conference on Parallel Architectures and Compilation Techniques
October 8 - 12, 2022
Illinois, Chicago

Conference Sponsors:
SIGARCH

Locality-aware Optimizations for Improving Remote Memory Latency in Multi-GPU Systems

Leul Belayneh, Haojie Ye, Kuan-Yu Chen, David Blaauw, Trevor Mudge, Ronald Dreslinski, Nishil Talati

Computer Science and Engineering, University of Michigan
Ann Arbor, Michigan, USA
Email: leulb@umich.edu

ABSTRACT

With generational gains from transistor scaling, GPUs have been able to accelerate traditional computation-intensive workloads. But with the obsolescence of Moore's Law, single GPU systems are no longer able to satisfy the computational and memory requirements of emerging workloads. To remedy this, prior works have proposed tightly-coupled multi-GPU systems. However, multi-GPU systems are hampered from efficiently utilizing their compute resources due to the Non-Uniform Memory Access (NUMA) bottleneck. In this paper, we propose **DualOpt**, a lightweight *hardware-only* solution that reduces the remote memory access latency by delivering optimizations catered to a workload's locality profile. DualOpt uses the *spatio-temporal locality* of remote memory accesses as a metric to classify workloads as **cache insensitive** and **cache-friendly**. Cache insensitive workloads exhibit low spatio-temporal locality, while cache-friendly workloads have ample locality that is not exploited well by the conventional cache subsystem of the GPU. For cache insensitive workloads, DualOpt proposes a fine-granularity transfer of remote data instead of the conventional cache line transfer. These remote data are then coalesced so as to efficiently utilize inter-GPU bandwidth. For cache-friendly workloads, DualOpt adds a *remote-only* cache that can exploit locality in remote accesses. Finally, a **decision engine** automatically identifies the class of workload and delivers the corresponding optimization, which improves overall performance by 2.5× on a 4-GPU system, with a small hardware overhead of 0.032%.

CCS CONCEPTS

• **Computer systems organization** → **Multicore architectures**; **Interconnection architectures**.

KEYWORDS

GPGPU, multi-GPU, data movement, GPU cache management

ACM Reference Format:
Leul Belayneh, Haojie Ye, Kuan-Yu Chen, David Blaauw, Trevor Mudge, Ronald Dreslinski, Nishil Talati. 2022. Locality-aware Optimizations for Improving Remote Memory Latency in Multi-GPU Systems . In *International Conference on Parallel Architectures and Compilation Techniques (PACT '22), October 10–12, 2022, Chicago, IL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3559009.3569649

PACT '22, October 10–12, 2022, Chicago, IL, USA

ACM ISBN 978-1-4503-9868-8/22/10…$15.00
https://doi.org/10.1145/3559009.3569649

1 INTRODUCTION

GPUs, with their high computational throughput, have been at the forefront of accelerating data-parallel tasks. To date, GPUs have satisfied the increasing computational demand from emerging workloads via transistor scaling. However, the slowdown of Moore's Law [8] has hindered the scaling of single-GPU performance. In addition, there has been increasing computational and storage demand from emerging workloads. As an alternative, major GPU vendors have recently started offering tightly-coupled multi-GPU systems [2, 30]. However, despite the availability of ample compute resources, utilization of the full potential of multi-GPU system still remains a challenge.

One of the key challenges of utilizing multi-GPU systems is the Non-Uniform Memory Access (NUMA) bottleneck, which arises from the large bandwidth gap between local and remote memory accesses. While local memory accesses enjoy the high bandwidth memory (HBM), remote memory accesses are served via pin-limited slow inter-GPU links. This results in up to a 12× discrepancy between local and remote memory bandwidth [25, 42]. This discrepancy still holds in modern interconnect technologies such as NVLink and NVSwitch [15, 29].

To address the NUMA bandwidth bottleneck, prior works have followed two approaches: (1) page migration schemes [5], which rely on runtime page partitioning, and (2) replication-based schemes [26, 42], wherein remote data is duplicated in local memory of the GPUs. While the former scheme achieves a balanced page distribution, there still remains inter-GPU communication from shared pages. On the other hand, the replication overhead in the latter optimization schemes can incur prohibitive storage costs. Therefore, achieving high scalability in multi-GPU systems for workloads with large working set sizes still remains a challenge.

To overcome the limitations of existing works, we propose **DualOpt**, a *low-cost hardware-only* solution that reduces remote memory access latency via locality-dependent optimizations. The main objectives of our design are to: (1) automatically identify locality traits of a workload, and (2) deliver an optimization tailored to the workload's locality characteristics. We propose to classify different workloads based on their *spatio-temporal locality* property into **cache insensitive** and **cache-friendly**. Cache insensitive workloads have low spatio-temporal locality and benefit little from caching, while cache-friendly ones exhibit better locality. Using

this characterization, DualOpt introduces two novel optimizations: *remote access coalescing* and *remote data caching*.

Remote access coalescing. We first make the key observation that cache insensitive workloads have low cache line utilization. Only a small fraction of remotely transferred cache lines are utilized by the GPU Compute Units (CUs). Based on this observation, DualOpt proposes to apply a *finer granularity* remote data transfers to conserve the inter-GPU bandwidth. This technique is akin to the sector cache employed in the IBM 360 in the 1960's [24]. In addition, DualOpt *coalesces* as much of these fine-grained accesses as possible into a single interconnect packet.

Remote data caching. Second, we observe an under-utilized opportunity to exploit data locality in remote accesses of cache-friendly workloads. This under-utilization is due to: (1) high contention of L1 caches in GPUs, shared by thousands of threads, (2) sharing of the L1 cache by local and remote data, and (3) private L1 cache design that leads to remote data replication for the same cache lines accessed by different CUs. As a result, DualOpt augments a *shared remote cache* for storing remotely accessed data near Remote Data Memory Access (RDMA).

DualOpt employs a **decision engine** to automatically deliver locality-aware optimizations. A decision engine monitors, characterizes, and identifies the class of a workload based on its *spatio-temporal locality*. Once the category of a workload is identified, the decision engine will declare an optimization and initiate the required hardware components.

Contributions. We make the following key contributions:

- We perform a detailed characterization of multi-GPU workloads to divide them into **cache insensitive** and **cache-friendly** workloads.
- We propose two novel optimizations: *remote access coalescing* of fine-grained accesses and *remote data caching* for cache insensitive and cache-friendly workloads, respectively.
- We design a *decision engine* to monitor, accurately identify, and declare a workload-specific optimization.
- Finally, we evaluate the end-to-end performance of DualOpt, showing a 4.4× reduction of inter-GPU traffic that translates to an average memory access latency reduction of 2.4× and overall performance improvement of 2.5×.

2 BACKGROUND

2.1 Multi-GPU Programming

GPU programming frameworks such as OpenCL and CUDA provide programmers an interface to launch thousands of work items on a GPU in a SPMD (single program, multiple data) fashion. A group of work items form a wavefront that is executed in a lock-step fashion. In turn, a group of wavefronts make up a workgroup, that are launched to the same GPU. The recent advance of big data applications has led to the exploration of multi-GPU platforms. These frameworks are adding support for multi-GPU programming with a discrete multi-GPU model. In OpenCL, a command queue is associated with the available GPUs and all the kernel launches (e.g., memory copy, kernel launch) run on the associated GPU. On the other hand, CUDA has the *cudaSetDevice* API that gives the programmer the option of which GPU to launch kernels on. There has been an increasing interest to explore a unified multi-GPU model by hiding multiple GPUs behind a single GPU interface, thus, removing the programmer's burden of modifying legacy codebase and managing multiple GPUs. In this paper, we use a unified multi-GPU model, which has been widely explored in related prior works [4, 20, 25, 26, 36, 42, 46]. A GPU program is dispatched in work groups (using OpenCL terminology) across GPUs transparently without specifying the GPU ID, and all GPUs share a unified address space with implicit communication [36].

2.2 Multi-GPU Architecture

With the antiquation of Moore's law, single-GPU systems can no longer satisfy the increasing compute and storage demand of emerging workloads. One possible solution is a multi-GPU system, where multiple GPUs orchestrate the execution of a workload in tandem. Fig. 1 shows the interconnection of 2 AMD GPUs. These GPUs are interconnected via high speed inter-GPU fabric such as PCIe. Recent works have also shown package-level integration of multiple GPUs [4]. However, the offered bandwidth of state-of-the-art interconnects still falls far behind the available local memory bandwidth. In practice, local memory accesses enjoy up to 12× more bandwidth compared to remote memory accesses. Therefore, optimizing the remote access bottleneck can have significant positive impact on the overall performance of multi-GPU systems [25, 42].

Each GPU shown in Fig. 1 is hierarchically divided into multiple Shader Engines (SE), which have their own L1 scalar caches. Numerous compute units (CU) reside within an SE co-located with an L1 vector cache, (2) in the figure. A scalar cache is used to read a single piece of data for a group of threads and is used for caching read-only data such as constants. A vector cache stores individual thread data. Accesses missed at the L1 cache have two options: 1) if the data is located at local memory, then a multi-banked L2 cache of the local GPU will serve requests, 2) if it is a remote access, then either a specific cache line is retrieved from a remote memory, or a page migration request is initiated to migrate a page to local memory. Explanation of these communication mechanisms with their trade-offs is detailed in the next section.

2.3 Multi-GPU Communication

Data partitioned across GPUs can be communicated in two ways: Page Migration, and Direct Cache Access (DCA). Page migration refers to the transfer of a page and ownership from one GPU to another. Fig. 1 shows the sequence of steps taken to transfer a page from GPU 1 to GPU 0. Initially, GPU 0 checks its translation look-aside buffers (TLB) to translate its address and upon a TLB miss it translates via the Input Output Memory Management Unit (IOMMU)(1). The IOMMU is a central directory that tracks page locations for all GPUs. After translation, if the IOMMU locates the page in another GPU (GPU1 in Fig. 1), it will initiate a page fault. Upon detecting a page fault, a driver will flush the CU pipeline, cache, TLB, and in-flight memory requests of the owner GPU (GPU1) to avoid leaving stale translations and data (2). Then, a Page Migration Controller (PMC) will transfer the page from GPU 1 to GPU 0 (3). Finally, future accesses to the same page can be retrieved from the memory of GPU 0. Page migration transfers data at coarse granularity and facilitates bandwidth optimized access. On the other hand, page migration incurs performance penalty due

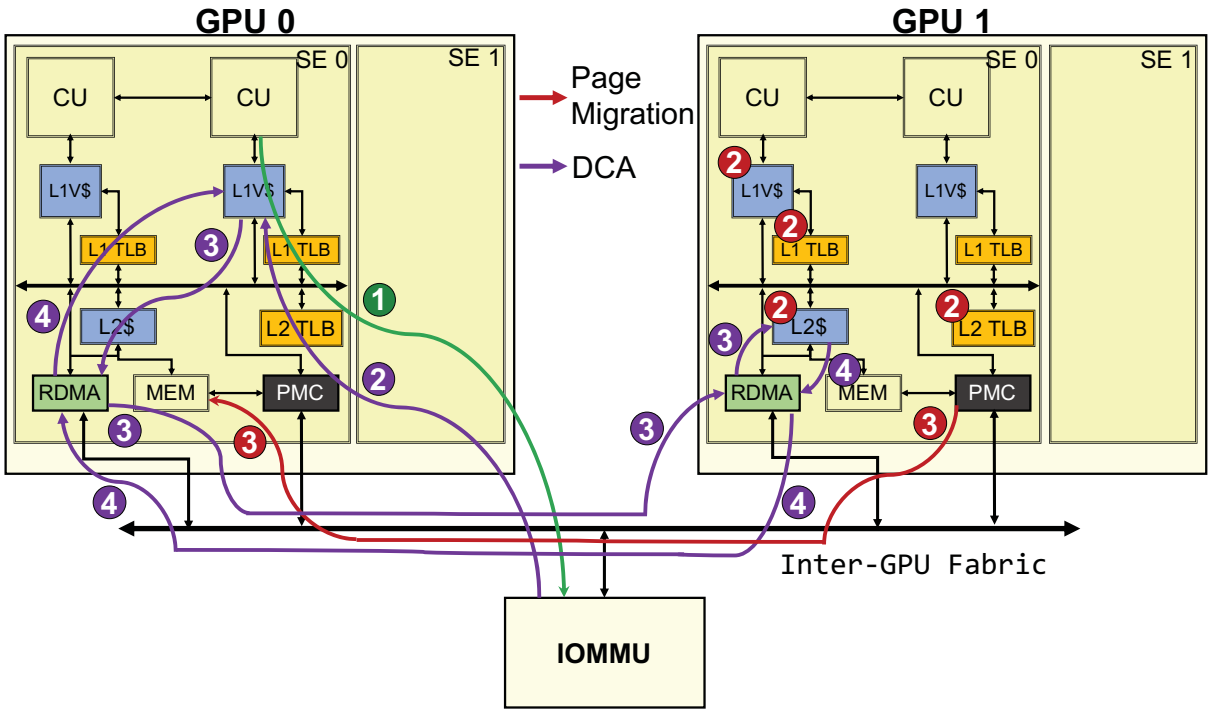

Figure 1: Simplified multi-GPU system with 2 GPUs. Sequences of transactions for page migration and DCA are shown in red and purple, respectively. SE 1 has the same components as SE 0, but omitted due to space limitations.

to flushing of outstanding memory accesses and contents in cache, TLB, and CU pipeline. This is especially costly in GPUs, where thousands of threads have in-flight requests and translations.

Page migration hurts performance when there is frequent sharing of a page between GPUs resulting in a ping-pong effect. We can avoid this penalty by relying on DCA. DCA allows accessing data from remote memory at cache line granularity. Unlike page migration, it evades the costly flushing of CU pipeline, cache, and TLB. Note that DCA still requires significant latency at the slow inter-GPU interconnect for each remote accesses. This gets exacerbated on a GPU, where a stall in a single thread accessing remote data can delay the progress of an entire wavefront [9, 35]. Fig. 1 shows the main steps in DCA. A requesting GPU (GPU 0) translates the address with the help of the IOMMU (①, ②). Then it directly sends its request via the RDMA unit if it misses at the L1 cache (③). Finally, a response arrives via RDMA and is cached at the L1 until eviction (④). DCA has been used in numerous prior works [5, 25, 42]. Thus, this work uses a unified GPU based DCA [36] as a baseline and focuses on how to further optimize it.

3 UNDERSTANDING THE INTER-GPU COMMUNICATION BOTTLENECK ON MULTI-GPU SYSTEMS

In this section, we analyze the inter-GPU communication behavior and its impact on the overall performance of multi-GPU systems. We also identify the source of bottlenecks and pinpoint mechanisms to address the bottlenecks. We simulate the multi-GPU system on MGPUSim [36] based on the configuration in Table 1. For most of this paper, we model a multi-GPU system with 4 GPUs. To best understand the inter-GPU communication bottleneck, we ask the following key questions:

> **(Q1)** What is the maximum attainable benefit of addressing the inter-GPU communication bottleneck?
> **(Q2)** What is the distribution of accesses into local and remote memory address space?
> **(Q3)** Where do the remote memory accesses spend most of their trip time?
> **(Q4)** What is the cache line utilization of remote accesses?
> **(Q5)** What is the coalescing opportunity in the remote memory accesses?

We present a detailed performance analysis of multi-GPU workloads to answer these questions, as shown below.

3.1 Characterization of Workloads

Q1. The inter-GPU link is the main source of contention in multi-GPU systems. To understand the maximum attainable benefit of addressing the inter-GPU bottleneck, we implement a multi-GPU system with different hypothetical interconnect frequencies. Fig. 2 shows the effect of increasing inter-GPU frequency on the performance of a wide range of workloads (see Table 2). It shows the opportunity for speedup achievable by addressing inter-GPU congestion. Though impractical, increasing the baseline frequency by a

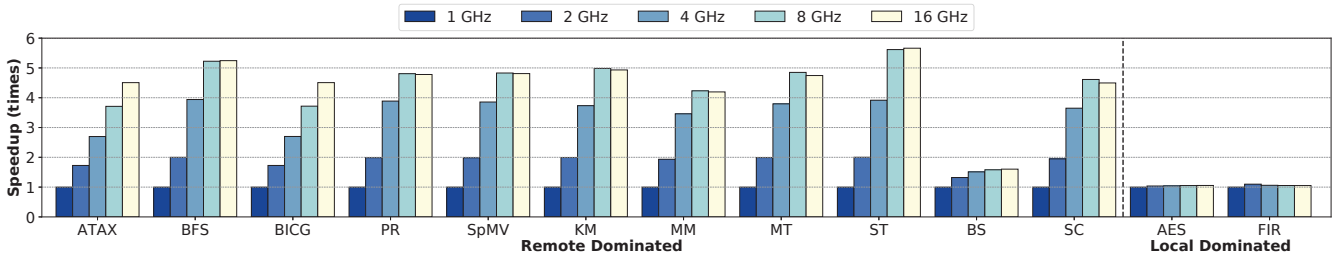

Figure 2: Sensitivity of performance to inter-GPU frequency. Increasing the inter-GPU frequency improves remote-dominated workloads while local-dominated works remain frequency-agnostic.

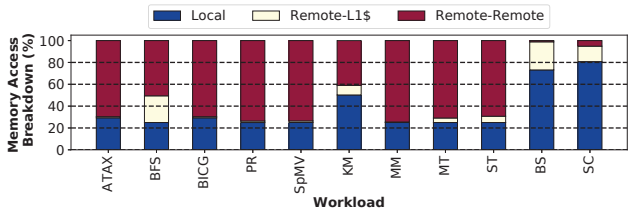

Figure 3: Memory accesses to local and remote memory address space. Remote accesses can be served from either local L1 cache or remote GPU.

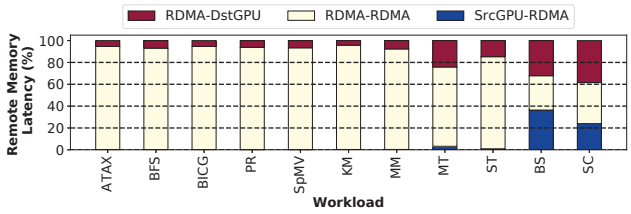

Figure 4: Latency distribution of remote memory accesses. The latency has three phases: pre inter-GPU link (within source GPU), within inter-GPU link, and post inter-GPU link (at remote GPU).

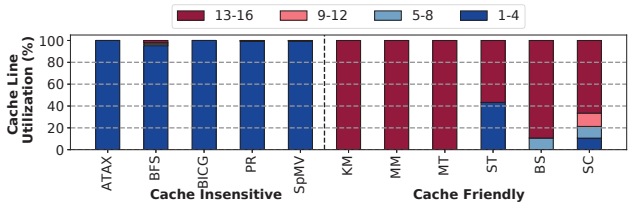

Figure 5: Cache line utilization in remote memory accesses. A 64 byte cache line is divided into 16 4-byte pieces. A CU can consume upto 16 cache line pieces.

factor of 2, 4, 8, and 16 offers an average performance improvement of 1.8×, 2.4×, 3.1×, 3.2× respectively. Note that the speedup gain for frequencies above 8 GHz saturates for all workloads.

Two categories of workloads clearly emerge based on our findings in Fig. 2. The first category of workloads are **remote-dominated**, where increasing the inter-GPU bandwidth boosts the overall performance. The second category are **local-dominated** workloads such as aes and fir. Due to their data layout and algorithmic property, data partitioned across GPUs are solely used by their host GPU. Hence, these workloads are hardly impacted by inter-GPU bandwidth. In contrast, accesses from remote-dominated workloads are not confined to local address space, which incurs inter-GPU traffic. Therefore, our analysis and optimizations afterwards focus on remote-dominated workloads.

Q2. Fig. 3 shows a breakdown of memory accesses into local and remote memory address spaces. Accesses to local memory address space are always serviced by local memory units, while remote memory accesses can either be retrieved from local L1 cache ($Remote - L1\$$) or remote memory ($Remote - Remote$). Remote-dominated workloads have around 40% of memory requests served from remote memory. These workloads heavily rely on the scarce inter-GPU link bandwidth to request and transfer data from remote GPU. We also notice that certain workloads such as bs and sc are able to serve significant remote memory accesses from the L1 cache incurring less inter-GPU transactions.

Q3. Here, we analyze the cycles spent within the inter-GPU link relative to the end-to-end latency of remote memory accesses. We break down the path taken by remote memory accesses into three phases: pre inter-GPU link (within source GPU), within inter-GPU link, and post inter-GPU link (at remote GPU). Fig. 4 plots a breakdown of a remote memory access latency. Overall, remote-dominated workloads spend 80% of remote memory access latency traversing the inter-GPU links. Exceptions to this trend are bs and sc, where inter-GPU latency is low due to their relatively low inter-GPU congestion. Note that, the high L1 cache hit rate of both workloads (see Fig. 6) results in lower the inter-GPU traffic.

3.2 Sources of Inefficiency

Q4. As discussed in §2, DCA relies on the cache line transfer of remote data. Here, we further dissect the effectiveness of the cache line transfer of remote data employed in DCA techniques. Conventionally, a 64 byte cache line is transferred per memory transaction. We use cache line utilization to quantify how much of a cache line is consumed by a CU (on average). Fig. 5 shows the effective utilization of remote cache line data transferred via inter-GPU links. The first class of workloads with low spatial locality

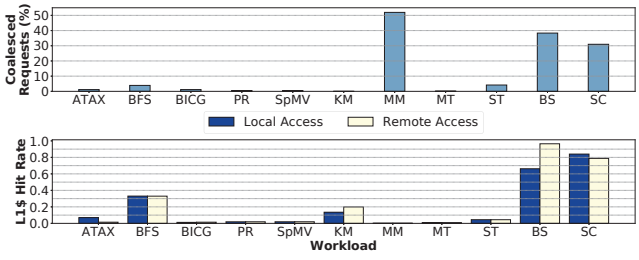

Figure 6: Coalescing potential of remote memory request at the RDMA and L1 cache hit rate of local and remote memory accesses.

are **cache insensitive**. These workloads exhibit very low cache line utilization; on average, 99% of accesses consume less than four piece (16 byte) of the cache line. Hence, cache insensitive workloads suffer from (1) inefficient data transfer where a sparsely utilized cache line is transferred wasting inter-GPU bandwidth, and (2) eviction of useful cache entries by these poorly utilized cache lines.

Q5. The second class of workloads are **cache-friendly**. As shown in Fig. 5, these workloads exhibit a good cache line utilization. However, their temporal locality is still low with an average L1 cache hit rate of 22% as shown in Fig. 6. This is because L1 caches are (1) small but shared by thousands of threads leading to frequent cache evictions, and (2) private to CUs where inter-CU locality is left unexploited. For instance, to study the potential inter-CU locality, we monitor outgoing memory requests leaving the GPU at the RDMA. We augment an ideal (infinite-sized) coalescing buffer at the RDMA to merge remote accesses to the same memory location. Fig. 6 shows percentage of accesses that can ideally be coalesced at the RDMA. One can observe that some workloads (*i.e.,* mm, bs, and sc) exhibit ample locality among remote memory requests.

4 DUALOPT DESIGN

In this section, we present DualOpt, a low-cost hardware-based solution that introduces locality-aware optimizations to tackle the inter-GPU link congestion. DualOpt, based on the spatio-temporal characteristics of a workload (detailed in §3), proposes two independent optimizations.

Remote Access Coalescing. Conventionally, remote data is transferred and cached at a cache line granularity. However, as discussed in §3.2, cache insensitive workloads struggle to enjoy the benefit of cache line transfer due to their very low spatio-temporal locality (see Fig. 5 and 6). Thus, we replace the conventional cache line transfer of remote data with a fine-grained one. By transferring only useful portion of cache lines at fine granularity, one can save on the scarce inter-GPU bandwidth. This entails appending additional metadata, during address generation, to identify specific locations of fetched remote data. Due to the mismatch between the fine-grained transfers and cache block sizes, DualOpt opts to bypass caches for remote accesses. Note that caching is already ineffective for cache insensitive workloads, as described in §3.2. On the positive side, these fine-grained remote transactions offer coalescing opportunity to further reduce inter-GPU network traffic.

Remote Data Caching. In §3, we observe that cache-friendly workloads exhibit good cache line utilization, but are not effectively served by current GPU caches. Hence, to exploit locality in these workloads, we propose a local cache at the RDMA. The **RDMA Cache** stores only remote data and acts as a victim cache for remote accesses that miss the L1 caches. Conventionally, remote accesses that miss L1 caches are served via the slow inter-GPU links. However, by serving remote memory accesses at the RDMA Cache, we will avoid these long latency remote accesses.

Fig. 7 shows a high-level diagram of additional units introduced by DualOpt. DualOpt has four basic units:

Bitmask Generation Unit (BGU): handles the generation of addresses. The generated addresses permit fine-grained accesses by including a bitmask to identify fetched entries in a cache line.

Cache Extensions: identifies remote accesses in cache insensitive workloads and bypasses them from using the L1 cache. Note that these remote accesses are fine-grained. A slight extensions to miss status handling registers (MSHR) is also required to track these fine-grained remote memory accesses.

RDMA CoDec Unit: has two functions: 1) coalescing fine-grained outgoing remote responses until it fits a packet size, and 2) decoalescing back an incoming packet (remote response) into individual responses so as to be processed by the receiving GPU.

RDMA Cache: a local in-RDMA cache that stores remotely accessed data for cache-friendly workloads.

The following sections explain these units in detail.

4.1 Remote Access Coalescing

4.1.1 ***Bitmask Generation Unit (BGU).*** The main purpose of BGU is to generate fine-grained addresses. Conventionally, a memory address generated from a CU (coalescer) points to a cache line. A coalescer is used to merge memory accesses from one or more threads in a wavefront that access the same cache line. Fig. 7(b) shows how a coalescer unit merges memory accesses from a wavefront that has 4 threads. The coalescer merges accesses from thread 2 (`0x100`) and 3 (`0x104`), since both threads access the same cache line. The other two can not be coalesced as they refer to different cache lines. In total, the coalescer generates 3 memory accesses.

In DualOpt, we augment BGU to the coalescer. Whereas the coalescer generates cache line addresses, the BGU appends an extra bitmask entry. Fig. 7(b) shows how the bitmask is generated at the coalescing unit. The bitmask has 16 bits, where each bit refers to a 4-byte entry within a cache line; a cache line has a total of 16 4-byte entries. Hence, the BGU uses 4 bits of an address ([5:2]) to calculate the bitmask in parallel with the coalescing unit. And, when accesses from two or more threads are coalesced (*e.g.*`0x100` and `0x104` in Fig. 7(b)), the BGU bitwise-OR's their respective bitmasks to generate the final bitmask. DualOpt appends the bitmask to the cache line address and propagates it to the memory subsystem, thereby allowing flexible fetching of data at a small granularity. A bitmask has no traffic overhead since it is carried by a read request that can repurpose its unused reserved bits.

4.1.2 ***Cache Extensions.*** DualOpt extends the cache subsystem to take care of requests generated at the BGU. We extend two key components of the L1 cache, namely cache controller and MSHR.

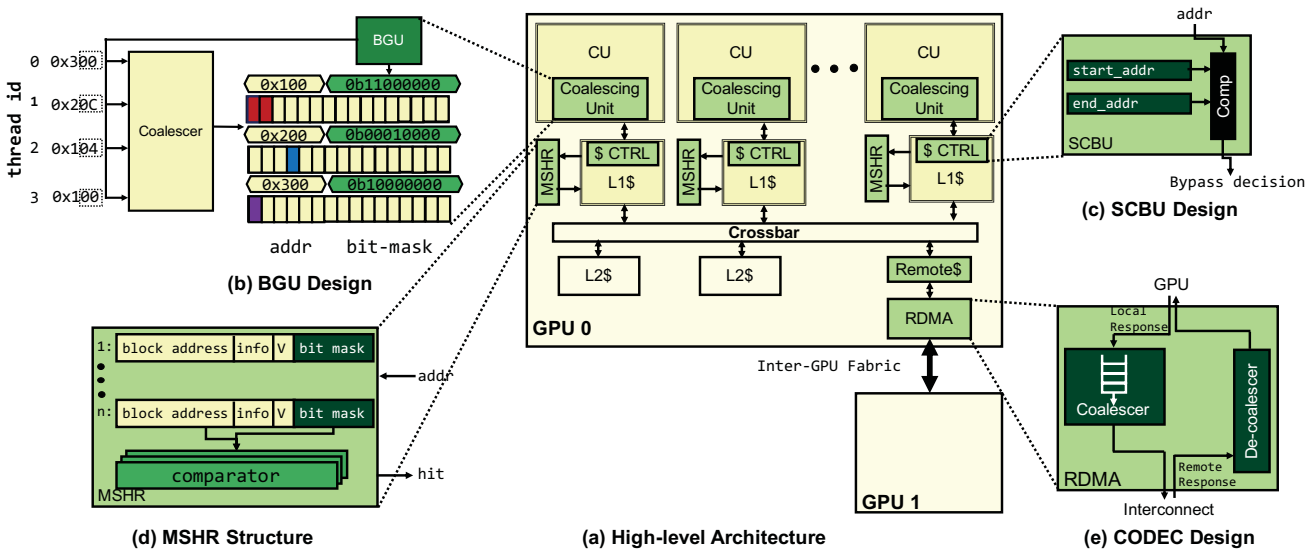

Figure 7: High-level architecture of a multi-GPU system with two GPUs. DualOpt extends the coalescing unit, cache subsystem, and RDMA (shown in green).

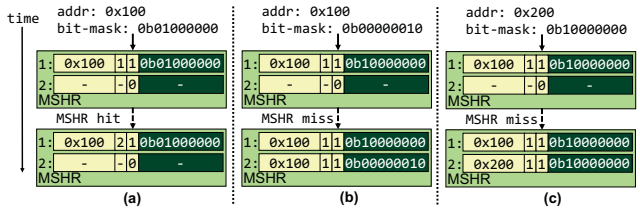

Figure 8: Simplified example of MSHR operations in a 2-entry MSHR. Incoming request contains an 8-bit bitmask.

Selective Cache Bypassing Unit (SCBU). Since remote memory requests operate at finer granularity, DualOpt bypasses the cache for remote requests. To this end, DualOpt uses an SCBU to identify and bypass remote memory accesses at the cache controller. Fig. 7 (c) shows an SCBU that contains an address registers holding the start and end physical addresses of local GPU memory. After a memory request arrives at an L1 cache, the SCBU consult the address registers to decide bypassing. Note that SCBU operates in the physical address space. If the requested address falls outside the start and end address registers, then it is a remote memory request and will bypass the L1 cache. Otherwise, it will follow conventional cache access procedure.

MSHR Extensions. MSHRs are used for holding in-flight memory requests. Conventionally, upon arrival of new memory request, an MSHR is checked. If there exist entry to the same cache line address, it means there is prior outgoing memory request to the same address and MSHR hit occurs. Hence, the current memory request is recorded and will use the memory response of the prior request. On MSHR miss, a new MSHR entry is allocated and a memory request is sent to the lower memory hierarchy.

In DualOpt, however, the flexibility in memory access granularity prohibits the use of conventional MSHR structure. Hence, DualOpt extends the MSHR to carefully handle the fine-grained remote requests. Fig. 7 (c) shows the additional bitmask entry added to the MSHR. The bitmask tracks specific cache line entries fetched by outgoing memory requests. Hence, MSHR in DualOpt should consider bitmasks in its operation. Fig. 8 shows a case-by-case execution flow of MSHR queries in DualOpt.

(A) **Same address, same mask**: For an incoming memory request, if there exist an MSHR entry that has the same address with a bitmask containing[1] the incoming bitmasks, then it will be an MSHR hit. Fig. 8(a) shows an incoming request that has similar address (`0x100`) as the MSHR entry. In addition, the bitmask of the incoming request (`0b01000000`) is the subset of the corresponding bitmask of the MSHR entry (`0b01000000`). In other words, the fine-grained data being fetched by the in-flight request can serve the incoming request. Hence, it will be an MSHR hit.

(B) **Same address, different mask**: If an MSHR entry has the same address but does not contain all bitmasks of an incoming memory request, then it will be an MSHR miss. Fig. 8(b) shows an incoming request with a bitmask of `0b00000010` (7th entry of a cache line). Even though there exists an MSHR entry with the same address, it is fetching the first entry of a cache line (`0b10000000`). Therefore, it will be an MSHR miss. Additional MSHR entry is allocated and a request is sent to lower memory hierarchy.

(C) **Different address**: If there exists no MSHR entry with the same address as the incoming request, then it will be an MSHR miss. Fig. 8(c) shows an incoming memory request that has no matching entry in the MSHR. Hence, the request is allocated its own MSHR entry and will be forwarded to lower memory hierarchy.

[1]bits set in a bitmask of an incoming memory request are all set in a bit mask of an MSHR entry, then the MSHR entry contains the bit mask of the incoming memory request

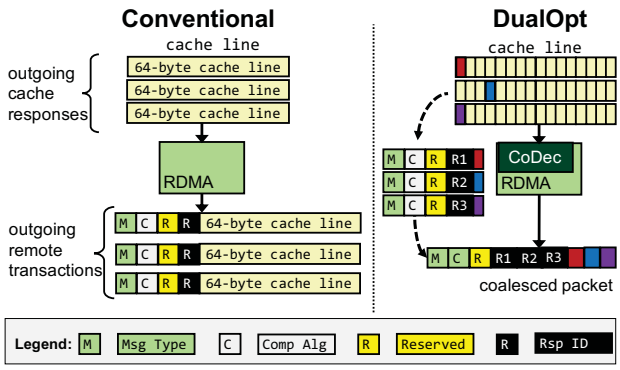

Figure 9: Comparison of RDMA operations with and without CoDec. A packet header includes the message type, compression algorithm, response ID, and few reserved bits. Packet sent to the same destination GPU carry similar header except the response ID. DualOpt carries the response ID of each coalesced access for identification at the receiver.

4.1.3 ***RDMA CoDec Unit.*** The RDMA unit acts as an interface to the inter-GPU link through which incoming and outgoing memory transactions are transferred. In DualOpt, fine-grained remote memory transactions are coalesced into a single packet to efficiently use interconnect bandwidth. And on the receiving side, the RDMA should be capable of de-coalescing the incoming packet. DualOpt has a dedicated CoDec unit in RDMA that handles the coalescing and de-coalescing of remote memory transactions. Note that we do not apply fine-grained memory accesses at L2 cache and DRAM. Our analysis shows that because of the high memory bandwidth at the crossbars connecting L1 cache with L2 cache, there is no performance headroom from fine-grained L2 cache accesses. Up to 2% loss in performance is incurred after applying fine-grained caching at L2 cache.

Fig. 7(e) shows the hardware structure of the CoDec unit. A CoDec has two basic components, a coalescer and de-coalescer.

Coalescer coalesces fine-grained outgoing memory responses. Since DualOpt transfers fine-grained data, it is possible to merge them into one packet. A buffer is used for staging outgoing responses to the coalescer. Since a coalescer merges responses to the same destination, DualOpt separately buffers responses to different destination. Therefore, *n-1* buffers should be allocated in RDMA, where *n* is the number of GPUs. Fig. 7(e) is a two-GPU system, hence outgoing responses are queued in a single buffer. This simplifies the coalescer logic as it can merge and send packets from each buffer in a round-robin fashion. To avoid deadlock, we set a buffer timeout to flush the buffer during inactivity.

Fig. 9 shows how coalescer merges multiple outgoing responses in comparison with conventional data transfer. Conventionally, an individual response packet carries a 64-byte cache line from each response in addition to packet header. In DualOpt, a response packet can carry up to 10 fine-grained transactions per packet, where each transaction is 6 bytes (4-byte data and 2-byte response ID). However, the packet might not be fully utilized to avoid performance penalty from long waiting times at the CoDEC buffers.

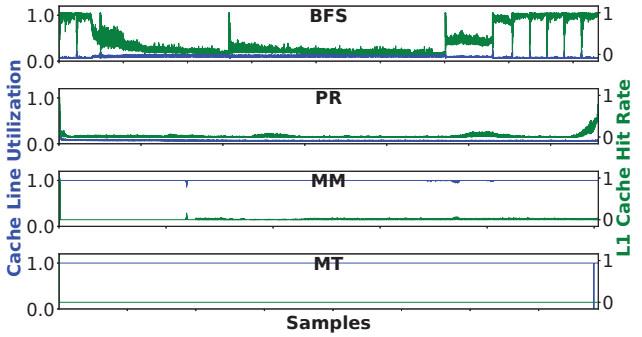

Figure 10: Cache line utilization and L1 cache hit rate during the execution of the representative workloads.

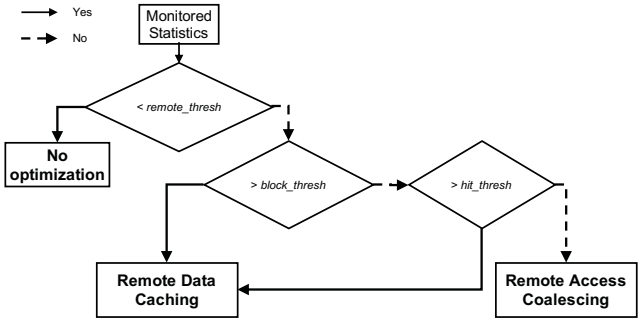

Figure 11: Flow chart of a decision engine.

De-coalescer performs the reverse of the coalescer; de-coalescing incoming packets at the RDMA. The de-coalescer receives an incoming packet and breaks it up into multiple individual fine-grained responses. It uses the compression algorithm field in the packet header (see Fig. 9) to identify a coalesced response packet from a regular ones. It also uses 3 bit of the header to determine the number of responses it contains. Note that the header bits are not fully utilized, hence can carry more info.

4.2 Remote Data Caching

DualOpt optimizes cache-friendly workloads by employing RDMA Cache–a local cache dedicated for retaining remote data. Traditionally, remotely accessed data is locally cached in a small private L1 cache. The L1 cache serves both local and remote requests from thousands of threads in GPUs. As a result, locality in remote accesses has not be exploited well due to the contention at L1 caches. DualOpt puts RDMA Cache prior to the RDMA to filter out outgoing remote memory requests as shown in Fig. 7(a). The RDMA Cache exploits locality in remote memory accesses of cache-friendly workloads to reduce transfer of data via slow remote interconnect. Thus, avoiding costly remote memory accesses.

4.3 Decision Engine

As described above, DualOpt performs two-independent, locality-driven optimizations. DualOpt proposes a decision engine to accurately categorize the locality behavior of a workload and declare

an optimization. First, the decision engine identifies if a workload is local or remote-dominated. Then, it uses the spatio-temporal locality (see §3.2) as a metric to further classify remote-dominated workloads. A decision engine has two phases: monitoring and optimization phase. During a monitoring phase, it tracks memory access requests arriving at the L1 cache within a small window of execution. Specifically, the ratio of remote accesses, cache block utilization of remote reads, and the L1 cache hit rate of remote accesses are collected. As shown in Fig. 10, cache insensitive workloads have low cache line utilization and hit rate, while cache friendly workloads have good cache line utilization or good hit rate or both. These execution statistics of a workload, in combination with a predefined thresholds (*remote_thresh, block_thresh, and hit_thresh*), are fed to the decision engine to determine the category of a workload. Fig. 11 shows the operation flow chart of a decision engine. Based on the flow chart, *remote access coalescing* is declared if a workload has a cache block utilization and hit rate below the *block_thresh and hit_thresh*, respectively. Otherwise, *remote data caching* is declared. Then the declared optimization is applied to a workload during the optimization phase.

However, to widen the optimization phase, a decision engine should be able to quickly complete the monitoring phase. This entails accurately determining the class of a workload in as small a window of execution time as possible. This is possible due to the fairly consistent behavior of workloads throughout execution (see Fig. 10). Fig. 12 shows the accuracy of the narrow-window estimate of a decision engine as compared with a golden model. One can observe that a decision engine is able to accurately classify the workloads. Thus, a decision engine monitors, identifies and declares an optimization at the beginning of execution. In order to maintain good decision accuracy, each workload is monitored for as small as 20K remote memory accesses. However, handling of time-varying statistics such as multi-tenant workloads is outside the scope of DualOpt. In the future, we plan to explore the adoption of a reconfigurable decision engine, which has been studied well in the context of reconfigurable hardware [14].

4.4 Implications on Memory Coherence

In DualOpt, we introduce an RDMA Cache to retain a local copy of remotely accessed data for cache-friendly workloads. To maintain software coherence, RDMA Cache invalidates its cache entries and flushes any dirty data back to remote GPU memory on kernel ends (synchronization points). On the other hand, cache insensitive workloads immediately propagate all remote updates due to cache bypassing. Hence, coherence is maintained as no local copy of remote data exist. In a similar fashion, globally coherent instructions such as atomics are handled the conventional way. The adoption of DualOpt is also compatible with the current IO-coherent GPUs.

4.5 Handling other Inter-GPU Transactions

Any non-read remote transactions, including *sys-scoped* and *non-sys-scoped* writes are not considered for coalescing in DualOpt. Non-read remote transactions are handled the conventional way, since they do not offer coalescing opportunity. Our design choice ensures backward compatibility of DualOpt with a strong GPU memory consistency model.

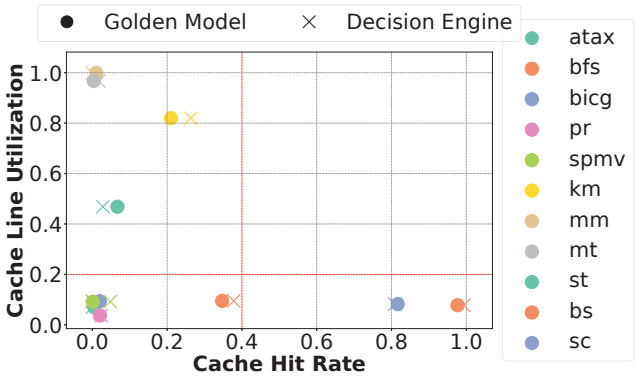

Figure 12: 2D spatio-temporal locality of workloads (Decision Engine vs Golden Model). The Golden Model uses stats collected from end-to-end execution of a workload. Cache line utilization is normalized. Cache-insensitive workloads are located in the lower left quadrant, and cache-friendly workloads occupy in the remaining portion of the plot.

GPU Parameters	
SEs	16
CUs per SE	4
CTA policy	partition
L1$ (Vector)	1 per CU, 16KB 4-way
L1$ (Scalar)	1 per SE, 16KB 4-way
L2 Cache	2MB 16-way, writeback
MSHR	32 entries
Cache Line	64 bytes
DRAM	4GB HBM
Inter-GPU Fabric Parameters	
BW	64 GB/s
Flit Size	16 Bytes

DualOpt Parameters	
BGU	2-byte addr. extension OR & divider logic
SCBU	2 addr. registers comparator
MSHR	2-byte bit-mask per entry bit-mask comp.
RDMA	3 64-entry buffer buffer timeout = 30 comp. & adder
RDMA$	1.5MB 16-way, writeback
L2 Cache	0.5MB 16-way, writeback
Decision Engine	*remote_thresh* = 2% *block_thresh* = 0.2 *hit_thresh* = 0.4

Table 1: Parameter specification of the modeled architecture.

5 EXPERIMENTAL METHODOLOGY

Modeled System. We use a cycle-accurate model of multi-GPU system with 4 AMD GPUs [3]. DualOpt can also be implemented on other architectures (*e.g.,* NVIDIA GPUs). Table 1 shows the architectural parameters of the system. The GPUs are interconnected via PCIe-like interconnect (4th generation). Each GPU has 16 SEs. Each SE has 4 CUs and a scalar L1 cache. Each CU has its own private L1 vector caches, while the L2 cache is shared within GPU.
Simulation Infrastructure. We use MGPUSim [36] (version 2.0.1), a multi-GPU simulator that accurately models AMD systems. The additional hardware modules in DualOpt are accurately modeled in MGPUSim. We use CACTI [28] to determine the timing and power consumption of the additional MSHR entries. We design and synthesize other DualOpt components (BGU, cache extensions, and RDMA CoDec) using commercial 28nm CMOS technology.
Evaluated Workloads. We use a diverse set of workloads collected from the widely used benchmarks such as AMDAPPSDK [1], SHOC [12], Hetero-Mark [37], and Polybench [44].
Simulation Methodology. We run all workloads till completion. We follow the same evaluation for all configurations. In bfs, we select the vertex with maximum outgoing edge as the source vertex.

Class	Workload	Description
Cache Insensitive	atax [44]	matrix transpose and vector multiplication
	bfs [12]	breadth first search
	bicg [44]	biconjugate gradient stabilized
	pr [37]	page rank
	spmv [12]	sparse matrix vector multiplication
Cache-Friendly	km [37]	kmeans clustering
	mm [1]	matrix multiplication
	mt [1]	matrix transpose
	st [12]	stencil 2D
	bs [1]	bitonic sort
	sc [1]	simple convolution
Local Dominated	aes [37]	advanced encryption standard
	fir [37]	finite impulse response

Table 2: Workload characteristics evaluated in DualOpt.

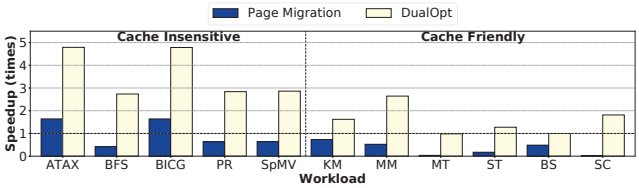

Figure 13: Performance comparison of DualOpt, Page Migration normalized to Baseline (DCA).

This enforces a wider graph traversal and a longer execution time. Since working version of pr is available in CUDA, we convert and re-implement it in OpenCL. Our decision engine starts operating after a warm up period for good accuracy. Then, it collects statistics for the first 20K remote memory accesses before declaring a decision. This takes negligible time compared to the total execution time.

6 EVALUATION

In this section, we present the overall performance improvement of DualOpt. We also show the inter-GPU traffic and memory access latency impact of DualOpt. We also show its sensitivity to transfer granularity, buffer timeout, CTA scheduling policy, interconnect configurations, and number of GPUs.

6.1 Performance Analysis

Speedup. We report the performance improvement of DualOpt normalized to `Baseline` (transfers inter-GPU data at cache line granularity) in Fig. 13. Our baseline adopts runtime of MGPUSIM [36] to distribute data in a unified multi-GPU system. We also compare with `Page Migration`, which is a unified memory based system that initially distribute pages via page migration. Overall, DualOpt improves performance by 2.5× on average. In cache insensitive workloads, DualOpt transfers remotely accessed data at a finer granularity, which are then coalesced for better interconnect utilization. As described in §3.2, these workloads offer high coalescing opportunities since the majority of remote reads utilize a fraction ($\frac{1}{16}$th) of a cache line. Therefore, DualOpt utilizes this opportunity and offers 3.6× speedup for cache insensitive workloads. We also observe that workloads such as atax and bicg work well with `Page Migration` than `Baseline`. This is due to serialization of outgoing remote memory accesses in DCA because of cache stalls induced by column-wise accesses common in atax and bicg.

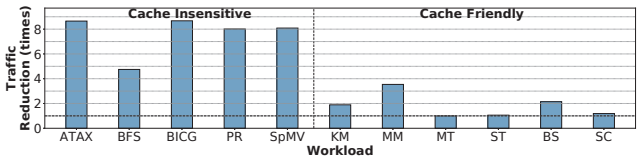

Figure 14: Inter-GPU traffic reduction of DualOpt normalized to Baseline (DCA).

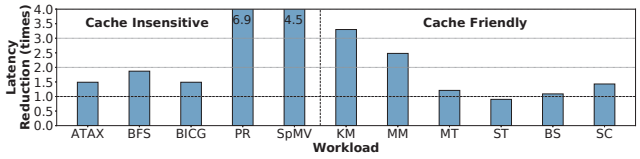

Figure 15: Memory access latency improvement of DualOpt normalized to 4-GPU baseline.

Cache-friendly workloads use RDMA Cache to serve remote memory accesses at the RDMA. The benefits of RDMA Cache stem from two reasons: 1) cache lines evicted from the small L1 caches can be retained at RDMA Cache, and 2) RDMA Cache allows sharing of data among CUs in a GPU. For instance, mm has much unexploited locality (shown in Fig. 6) that translates to significant performance improvement (2.6×). The majority of remote accesses in mm are to read-only data, while updates are stored to local memory. Thus, RDMA Cache can retain and reuse matrix tiles (within and across CUs), significantly cutting down on global memory accesses. On the other hand, workloads such as mt, where the majority of loads are to the local address space and stores dominate remote accesses [36], show minimal performance improvement. Overall, DualOpt improves the performance of cache-friendly workloads by an average of 1.6×.

Inter-GPU Traffic. We further look into the impact of DualOpt on the inter-GPU traffic traversing the slow remote links. As shown in Fig. 14, DualOpt reduces inter-GPU traffic across data points with noticeable improvements in 8 out of 11 workloads. Note that the reduction in traffic depends on the transaction types (*e.g.* load, stores) and shared page types (*e.g.* read-only, read-write) present in a workload. Workloads such as mm with significant traffic reductions have the majority of their remote transactions to read-only data.

Memory Latency. Fig. 15 presents the memory latency of DualOpt normalized to `Baseline`. We use the average latency of the memory requests as a metric for memory latency. DualOpt reduces remote accesses traversing inter-GPU links. Because remote accesses comprise the bulk of memory accesses and they are significantly slower than local accesses, DualOpt improves the overall memory latency by 2.4× (average).

Performance Breakdown of Remote Access Coalescing: As described in §4, Remote Access Coalescing has three steps: *L1 cache bypass*, *fine-grained access*, and *coalescing*. Here, we investigate and quantify the contribution of each to the overall performance in Fig. 16. We note that *L1 cache bypass* incurs minimal performance loss in most workloads except atax and bicg. Bypassing caches improves the performance of atax and bicg by 3.2×, on

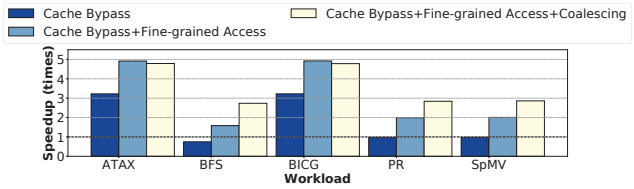

Figure 16: Performance improvements over the baseline of cache insensitive workloads achieved by cache bypassing, fine-grained access, and coalescing.

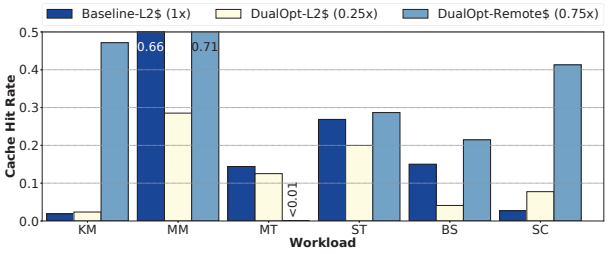

Figure 17: RDMA Cache hit rate of DualOpt in comparison with L2 cache hit rate of Baseline and DualOpt.

average. As described earlier, the column-wise accesses in these workloads incur frequent cache set conflicts. However, the bypassing of caches evades the cache stalls and allows more outstanding memory requests. In the case of bfs, bypassing sacrifices the 33% of L1 cache hits (see Fig. 6), which translates to 25% performance loss. *Fine-grained access*, on the other hand, consistently improves upon *L1 cache bypass* across workloads by 1.7×, on average. These small-sized accesses can be carried by small packets, so it lowers interconnect congestion. Finally, *coalescing* of fine-grained accesses further boosts the overall performance by 1.2×. This is due to the ample coalescing opportunity created by the outstanding fine-grained accesses bypassed by the L1 cache. However, in atax and bicg, coalescing shows no benefit as the buffering delay outweighs its benefit.

RDMA Cache Effectiveness. To exploit locality of cache-friendly workloads, DualOpt employs RDMA Cache to store remotely accessed data at the RDMA (see §4.2). To maintain iso-area comparison, $\frac{3}{4}$ of L2 cache is sliced off in DualOpt. Hence, DualOpt reduces the latency of costly remote memory accesses at the expense of L2 cache hits. In order to quantify the efficacy of RDMA Cache, we show the change in cache hit rates of RDMA Cache and L2 cache. Fig. 17 shows that RDMA Cache saves on average 35% of remote accesses from traversing the slow remote links. This is achieved at the expense of the L2 cache hit rate that reduces from 21% to 13%. Generally, we observe a lower L2 cache hit rates in all workloads except in sc. In sc, serving remote memory requests at RDMA Cache can potentially relieve congestion at the remote L2 cache. As a result, local memory accesses are able to enjoy the L2 cache without contention from remote accesses.

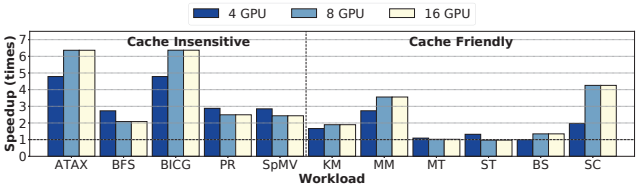

Figure 18: Performance of DualOpt with 4, 8, and 16 GPUs normalized to Baseline with 4, 8, and 16 GPUs.

6.2 Sensitivity Analysis

Transfer Granularity. Here, we evaluate the performance impact of varying the transfer granularity. So far, the transfer granularity is set to 4 bytes. A small transfer granularity offers good coalescing ratio, while large transfer granularity incurs less metadata overhead per transfer. Thus, we evaluate the effect of increasing the transfer granularity from 4 to 8 to 16 bytes on performance. Our evaluation shows that $4-byte$ offers the best traffic reduction. It improves upon $8-byte$ and $16-byte$ by 3% and 15%, respectively. Since 99% of accesses in cache insensitive workloads are 4 byte accesses (see Fig. 5), increasing transfer granularity above 4 bytes incur unwanted data transfer and less coalescing ratio.

Buffer Timeout. We also study the impact of varying the buffer timeout to show the trade-off between spending more time to actively coalesce newer requests and aggressively issue remote requests. We execute DualOpt with a buffer timeout of 10, 30, and 50 cycles. Increasing the timeout from 10 to 30 cycles improves performance by up to 4%. However, further increasing the timeout to 50 cycles starts to lower performance (up to −2%) as requests spend more time and block other outgoing requests.

CTA Scheduling Policy. We investigate the performance benefit of DualOpt under different CTA scheduling policy, *i.e.,* round-robin and greedy. The average performance improvement in round-robin and greedy systems is 2.5× and 2.7×, respectively. In a nutshell, DualOpt is general and works in different CTA scheduling policy.

Interconnect Bandwidth. In order to solve the inter-GPU bandwidth bottleneck, recent works [15, 29] have offered high bandwidth interconnect. Thus, we evaluate DualOpt in high-bandwidth inter-GPU fabric similar to PCIev5 and PCIev6. DualOpt outperforms the `Baseline` by 2.5× (on average) in both cases.

Large Flit Sizes. Irrespective of their limitations, such as high hardware costs [22], large flit sizes can reduce inter-GPU congestion. Thus, we evaluate how DualOpt performs with large flit sizes. Our result shows that, DualOpt is able to improve a `Baseline` with a 32 and 64 byte flit by an average of 2× and 1.8×, respectively.

Scalability. Finally, we evaluate the performance of DualOpt on 8 and 16 GPUs. Fig. 18 presents that DualOpt improves the performance of a multi-GPU with 8 and 16 GPUs by 2.8× and 2.9×, respectively. In summary, this demonstrates that DualOpt is able to deliver performance with more GPUs in the system.

6.3 Comparison With Prior Works

We compare DualOpt with Griffin [5], a state-of-the-art page migration based scheme. On average, DualOpt outperforms Griffin by 4× as shown in Fig. 19. Specifically, DualOpt unlocks high

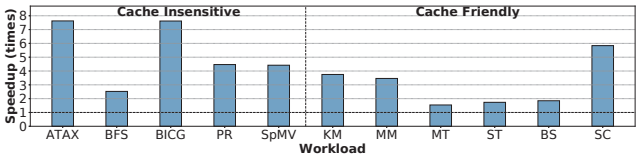

Figure 19: Performance of DualOpt normalized to a state-of-the-art prior work Griffin [5].

	Program-mability	Transfer Granularity	No Replication Cost	HW Complexity
CARVE [42]	✓	Cache line	✗	✗
Griffin [5]	✓	Page/Cache line	✓	✗
Proact [27]	✗	Fine-grained	✗	✓
GPS [26]	✗	Fine-grained	✗	✗
DualOpt	✓	Fine-grained/Cache line	✓	✓

Table 3: Comparison of DualOpt with related works.

Component	BGU	SCBU	MSHR	CODEC
Area (μm^2)	964.99	202.94	697	37234
Power (mW)	0.43	0.27	0.1	31.9

Table 4: Hardware overhead of DualOpt.

performance in atax and bicg by evading cache-related stalls via cache bypassing. In addition, TLBs in our Baseline [36] can cache translations of remote memory address space, boosting their performance. We also qualitatively compared DualOpt with other prior works [5, 26, 27, 42] in Table 3.

6.4 Resource Overheads

Additional hardware units added in DualOpt are: (1) a BGU per CU that generates the 2-byte bit masks used for fine-grained addressing, (2) an SCBU augmented at each cache controller which selectively identifies and bypasses remote memory accesses, (3) MSHR extensions to store the additional 2-byte entry of the fine-grained addresses, and (4) the CoDeC unit at RDMA that takes care of the coalescing and de-coalescing of remotely accessed data. Table 4 shows the resource requirement of each component. Overall, DualOpt incurs a 0.032% hardware overhead.

7 RELATED WORK

7.1 Inter-GPU Communication

Numerous works have improved the performance of multi-GPU systems [4, 5, 7, 20, 21, 25–27, 39, 42, 46]. Among these, GPS and CARVE [26, 42] store remote data locally to reduce the inter-GPU communication bottleneck. CARVE [42] identifies and caches remote data in local DRAM, while GPS [26] replicates shared pages in the local memory of GPUs. However, GPS requires programmer effort to selectively replicate pages across GPUs. Moreover, data replication in both works might degrade the performance of workloads with large data footprints. Unlike these approaches, DualOpt preserves programmability and requires no in-DRAM storage. DualOpt uses a *hardware-only* architecture to automatically deliver locality-aware optimization.

7.2 Intra-GPU Communication

Prior works [23, 24, 32–34, 45] have proposed a diverse memory subsystem optimization for cache usage and interconnect. Some of these techniques date as far back as the IBM 360 in the 1960's [24]. Prior works [23, 32] have proposed sub-cache line (32 byte) data transfer and storage. There are also works [45] that leverage cache bypassing techniques to reduce cache thrashing. In contrast, DualOpt uses a fine-grained transfer (as small as 4 byte) of remotely accessed data in the context of multi-GPU systems. Moreover, coalescing of these fine-grained remote data is applied to further reduce interconnect traffic. There are also works that use inter-core communication to implement a shared L1 cache and L1 TLB in GPUs [6, 10, 13, 17, 38, 40]. Intra-GPU communication schemes are complementary to DualOpt, which targets closing the local to remote memory bandwidth gap in multi-gpu systems.

7.3 NUMA-Aware Caching

Different NUMA-aware caching techniques has been extensively explored in both CPUs and GPUs [11, 16, 18, 19, 25, 31, 41, 43]. The earliest works have used a portion of the last level cache (LLC) to store remotely accessed data [25]. To overcome the size limitation of the LLC, other works have re-purposed DRAM as temporary storage [42]. However, in case of memory over-subscription, the benefit from DRAM caching may not be enough to compensate for the costly memory accesses to the CPU. This is particularly frequent in irregular workloads. In contrast, DualOpt exploits caching at the RDMA only when it is profitable; we carefully assess the profit based on the locality of the workload. If the workload has good cache locality (cache-friendly), we will locally store a remotely accessed data. Otherwise, DualOpt bypasses remote accesses and dedicates all cache capacity for local accesses.

8 CONCLUSION

This paper explored the impact of locality to optimize the costly remote memory accesses in multi-GPU systems. Based on our workload characterization, we demonstrated the need for locality-aware optimizations instead of a one-size-fits-all approach. Thus, we proposed DualOpt, a *hardware-only* design that deliver locality-aware optimizations to reduce memory access latency. DualOpt optimized cache-insensitive workloads by implementing fine-grained remote accesses instead of traditional cache line transfers. These fine-grained remote data are then coalesced to save the scarce inter-GPU bandwidth, DualOpt optimized cache-friendly workloads by allocating an RDMA Cache to retain remote data locally. DualOpt uses a decision engine to automatically identify and deliver optimizations catered to the locality of the workload. Our evaluation on representative workloads showed that DualOpt reduces inter-GPU traffic by 4.4×. This translates to a 2.5× performance improvement on a 4-GPU system, with a hardware overhead of 0.032%.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insightful feedback. We also like to thank Tarunesh Verma and Armand Behroozi for their feedback on the drafts of this article.

REFERENCES

[1] AMD. 2015. AMD APP SDK OpenCL Optimization Guide.

[2] AMD. 2020. AMD CrossfireTM Technology. https://www.amd.com/en/technologies/crossfire. last accessed on 11/7/2021.

[3] AMD. 2020. AMD Radeon™ Pro V520 Graphics.

[4] Akhil Arunkumar, Evgeny Bolotin, Benjamin Cho, Ugljesa Milic, Eiman Ebrahimi, Oreste Villa, Aamer Jaleel, Carole-Jean Wu, and David Nellans. 2017. MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 320–332. https://doi.org/10.1145/3079856.3080231

[5] Trinayan Baruah, Yifan Sun, Ali Tolga Dinçer, Saiful A. Mojumder, José L. Abellán, Yash Ukidave, Ajay Joshi, Norman Rubin, John Kim, and David Kaeli. 2020. Griffin: Hardware-Software Support for Efficient Page Migration in Multi-GPU Systems. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 596–609. https://doi.org/10.1109/HPCA47549.2020.00055

[6] Trinayan Baruah, Yifan Sun, Saiful A. Mojumder, José L. Abellán, Yash Ukidave, Ajay Joshi, Norman Rubin, John Kim, and David Kaeli. 2020. Valkyrie: Leveraging Inter-TLB Locality to Enhance GPU Performance *(PACT '20)*. Association for Computing Machinery, New York, NY, USA, 455–466. https://doi.org/10.1145/3410463.3414639

[7] Tal Ben-Nun, Michael Sutton, Sreepathi Pai, and Keshav Pingali. 2020. Groute: Asynchronous Multi-GPU Programming Model with Applications to Large-Scale Graph Processing. 7, 3, Article 18 (June 2020), 27 pages. https://doi.org/10.1145/3399730

[8] P. Bright. 2016. Moore's law really is dead this time. https://arstechnica.com/%20information-technology/2016/02/moores-law-really-is-dead-this-time/. last accessed on 11/7/2021.

[9] Niladrish Chatterjee, Mike O'Connor, Gabriel H. Loh, Nuwan Jayasena, and Rajeev Balasubramonia. 2014. Managing DRAM Latency Divergence in Irregular GPGPU Applications. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 128–139. https://doi.org/10.1109/SC.2014.16

[10] Kyoshin Choo, William Panlener, and Byunghyun Jang. 2014. Understanding and Optimizing GPU Cache Memory Performance for Compute Workloads. In *2014 IEEE 13th International Symposium on Parallel and Distributed Computing*. 189–196. https://doi.org/10.1109/ISPDC.2014.29

[11] Chiachen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. 2016. CANDY: Enabling coherent DRAM caches for multi-node systems. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13. https://doi.org/10.1109/MICRO.2016.7783738

[12] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. 2010. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units* (Pittsburgh, Pennsylvania, USA) *(GPGPU-3)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1735688.1735702

[13] Saumay Dublish, Vijay Nagarajan, and Nigel Topham. 2016. Cooperative Caching for GPUs. *ACM Trans. Archit. Code Optim.* 13, 4, Article 39 (Dec. 2016), 25 pages. https://doi.org/10.1145/3001589

[14] Siying Feng, Jiawen Sun, Subhankar Pal, Xin He, Kuba Kaszyk, Dong-hyeon Park, Magnus Morton, Trevor Mudge, Murray Cole, Michael O'Boyle, Chaitali Chakrabarti, and Ronald Dreslinski. 2021. CoSPARSE: A Software and Hardware Reconfigurable SpMV Framework for Graph Analytics. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 949–954. https://doi.org/10.1109/DAC18074.2021.9586114

[15] Denis Foley and John Danskin. 2017. Ultra-Performance Pascal GPU and NVLink Interconnect. *IEEE Micro* 37, 2 (2017), 7–17. https://doi.org/10.1109/MM.2017.37

[16] Cheng-Chieh Huang, Rakesh Kumar, Marco Elver, Boris Grot, and Vijay Nagarajan. 2016. C3D: Mitigating the NUMA bottleneck via coherent DRAM caches. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783739

[17] Mohamed Assem Ibrahim, Hongyuan Liu, Onur Kayiran, and Adwait Jog. 2019. Analyzing and Leveraging Remote-Core Bandwidth for Enhanced Performance in GPUs. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 258–271. https://doi.org/10.1109/PACT.2019.00028

[18] Djordje Jevdjic, Gabriel H. Loh, Cansu Kaynak, and Babak Falsafi. 2014. Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache. *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture* (2014), 25–37.

[19] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. 2013. Die-Stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (Tel-Aviv, Israel) *(ISCA '13)*. Association for Computing Machinery, New York, NY, USA, 404–415. https://doi.org/10.1145/2485922.2485957

[20] Gwangsun Kim, Minseok Lee, Jiyun Jeong, and John Kim. 2014. Multi-GPU System Design with Memory Networks. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 484–495. https://doi.org/10.1109/MICRO.2014.55

[21] Hyojong Kim, Jaewoong Sim, Prasun Gera, Ramyad Hadidi, and Hyesoon Kim. 2020. Batch-Aware Unified Memory Management in GPUs for Irregular Workloads. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1357–1370. https://doi.org/10.1145/3373376.3378529

[22] Junghee Lee, Chrysostomos Nicopoulos, Sung Joo Park, Madhavan Swaminathan, and Jongman Kim. 2013. Do we need wide flits in Networks-on-Chip?. In *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2–7. https://doi.org/10.1109/ISVLSI.2013.6654614

[23] Bingchao Li, Jizeng Wei, Jizhou Sun, Murali Annavaram, and Nam Sung Kim. 2019. An Efficient GPU Cache Architecture for Applications with Irregular Memory Access Patterns. *ACM Trans. Archit. Code Optim.* 16, 3, Article 20 (June 2019), 24 pages. https://doi.org/10.1145/3322127

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

[25] Ugljesa Milic, Oreste Villa, Evgeny Bolotin, Akhil Arunkumar, Eiman Ebrahimi, Aamer Jaleel, Alex Ramirez, and David Nellans. 2017. Beyond the Socket: NUMA-Aware GPUs. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 123–135.

[26] Harini Muthukrishnan, Daniel Lustig, David Nellans, and Thomas Wenisch. 2021. GPS: A Global Publish-Subscribe Model for Multi-GPU Memory Management. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 46–58. https://doi.org/10.1145/3466752.3480088

[27] Harini Muthukrishnan, David Nellans, Daniel Lustig, Jeffrey A. Fessler, and Thomas F. Wenisch. 2021. Efficient Multi-GPU Shared Memory via Automatic Optimization of Fine-Grained Transfers. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 139–152. https://doi.org/10.1109/ISCA52012.2021.00020

[28] N. P. Jouppi N. Muralimanohar, R. Balasubramonian†. 2009. CACTI 6.0: A Tool to Understand Large Caches. In *HP laboratories*.

[29] NVIDIA. [n. d.]. NVLINK AND NVSWITCH The Building Blocks of Advanced Multi-GPU Communication. last accessed on 11/7/2021.

[30] NVIDIA. 2020. NVIDIA DGX Systems. https://www.nvidia.com/en-us/data-center/dgx-systems/. last accessed on 11/7/2021.

[31] Moinuddin K. Qureshi and Gabe H. Loh. 2012. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 235–246. https://doi.org/10.1109/MICRO.2012.30

[32] Minsoo Rhu, Michael Sullivan, Jingwen Leng, and Mattan Erez. 2013. A locality-aware memory hierarchy for energy-efficient GPU architectures. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 86–98.

[33] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2012. Cache-Conscious Wavefront Scheduling. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. 72–83. https://doi.org/10.1109/MICRO.2012.16

[34] Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2013. Divergence-Aware Warp Scheduling. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 99–110.

[35] Seunghee Shin, Guilherme Cox, Mark Oskin, Gabriel H. Loh, Yan Solihin, Abhishek Bhattacharjee, and Arkaprava Basu. 2018. Scheduling Page Table Walks for Irregular GPU Applications. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 180–192. https://doi.org/10.1109/ISCA.2018.00025

[36] Yifan Sun, Trinayan Baruah, Saiful A. Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, Harrison Barclay, Amir Kavyan Ziabari, Zhongliang Chen, Rafael Ubal, José L. Abellán, John Kim, Ajay Joshi, and David Kaeli. 2019. MGPUSim: Enabling Multi-GPU Performance Modeling and Optimization. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 197–209.

[37] Yifan Sun, Xiang Gong, Amir Kavyan Ziabari, Leiming Yu, Xiangyu Li, Saoni Mukherjee, Carter McCardwell, Alejandro Villegas, and David R. Kaeli. 2016. Hetero-mark, a benchmark suite for CPU-GPU collaborative computing. *2016 IEEE International Symposium on Workload Characterization (IISWC)* (2016), 1–10.

[38] David Tarjan and Kevin Skadron. 2010. The Sharing Tracker: Using Ideas from Cache Coherence Hardware to Reduce Off-Chip Memory Traffic with Non-Coherent Caches. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–10. https://doi.org/10.1109/SC.2010.54

[39] Nandita Vijaykumar, Eiman Ebrahimi, Kevin Hsieh, Phillip B. Gibbons, and Onur Mutlu. 2018. The Locality Descriptor: A Holistic Cross-Layer Abstraction to Express Data Locality In GPUs. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 829–842. https://doi.org/10.1109/ISCA.2018.00074

[40] Jianfei Wang, Li Jiang, Jing Ke, Xiaoyao Liang, and Naifeng Jing. 2019. A Sharing-Aware L1.5D Cache for Data Reuse in GPGPUs. In *Proceedings of the 24th Asia*

and South Pacific Design Automation Conference (Tokyo, Japan) *(ASPDAC '19)*. Association for Computing Machinery, New York, NY, USA, 388–393. https://doi.org/10.1145/3287624.3287633

[41] Vinson Young, Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. 2018. ACCORD: Enabling Associativity for Gigascale DRAM Caches by Coordinating Way-Install and Way-Prediction. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 328–339. https://doi.org/10.1109/ISCA.2018.00036

[42] Vinson Young, Aamer Jaleel, Evgeny Bolotin, Eiman Ebrahimi, David Nellans, and Oreste Villa. 2018. Combining HW/SW Mechanisms to Improve NUMA Performance of Multi-GPU Systems. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture* (Fukuoka, Japan) *(MICRO-51)*. IEEE Press, 339–351. https://doi.org/10.1109/MICRO.2018.00035

[43] Vinson Young, Prashant J. Nair, and Moinuddin K. Qureshi. 2017. DICE: Compressing DRAM Caches for Bandwidth and Capacity. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 627–638. https://doi.org/10.1145/3079856.3080243

[44] Tomofumi Yuki and Louis-Noël Pouchet. 2015. Polybench 4.0.

[45] Zhong Zheng, Zhiying Wang, and Mikko Lipasti. 2015. Adaptive Cache and Concurrency Allocation on GPGPUs. *IEEE Computer Architecture Letters* 14, 2 (2015), 90–93. https://doi.org/10.1109/LCA.2014.2359882

[46] Amir Kavyan Ziabari, Yifan Sun, Yenai Ma, Dana Schaa, José L. Abellán, Rafael Ubal, John Kim, Ajay Joshi, and David Kaeli. 2016. UMH: A Hardware-Based Unified Memory Hierarchy for Systems with Multiple Discrete GPUs. 13, 4, Article 35 (Dec. 2016), 25 pages. https://doi.org/10.1145/2996190