MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability

Akhil Arunkumar[‡] Evgeny Bolotin[†] Benjamin Cho[∓] Ugljesa Milic[+] Eiman Ebrahimi[†]

Oreste Villa[†] Aamer Jaleel[†] Carole-Jean Wu[‡] David Nellans[†]

Arizona State University[‡] NVIDIA[†] University of Texas at Austin[∓]
Barcelona Supercomputing Center / Universitat Politecnica de Catalunya[+]

{akhil.arunkumar,carole-jean.wu}@asu.edu,{ebolotin,eebrahimi,ovilla,ajaleel,dnellans}@nvidia.com
bjcho@utexas.edu,ugljesa.milic@bsc.es

ABSTRACT

Historically, improvements in GPU-based high performance computing have been tightly coupled to transistor scaling. As Moore's law slows down, and the number of transistors per die no longer grows at historical rates, the performance curve of single monolithic GPUs will ultimately plateau. However, the need for higher performing GPUs continues to exist in many domains. To address this need, in this paper we demonstrate that package-level integration of multiple GPU modules to build larger logical GPUs can enable continuous performance scaling beyond Moore's law. Specifically, we propose partitioning GPUs into easily manufacturable basic GPU Modules (GPMs), and integrating them on package using high bandwidth and power efficient signaling technologies. We lay out the details and evaluate the feasibility of a basic Multi-Chip-Module GPU (MCM-GPU) design. We then propose three architectural optimizations that significantly improve GPM data locality and minimize the sensitivity on inter-GPM bandwidth. Our evaluation shows that the optimized MCM-GPU achieves 22.8% speedup and 5x inter-GPM bandwidth reduction when compared to the basic MCM-GPU architecture. Most importantly, the optimized MCM-GPU design is 45.5% faster than the largest implementable monolithic GPU, and performs within 10% of a hypothetical (and unbuildable) monolithic GPU. Lastly we show that our optimized MCM-GPU is 26.8% faster than an equally equipped Multi-GPU system with the same total number of SMs and DRAM bandwidth.

CCS CONCEPTS

• **Computing methodologies** → **Graphics processors**; • **Computer systems organization** → **Parallel architectures**; *Single instruction, multiple data*;

KEYWORDS

Graphics Processing Units, Multi-Chip-Modules, NUMA Systems, Moore's Law

ACM Reference format:
Akhil Arunkumar[‡] Evgeny Bolotin[†] Benjamin Cho[∓] Ugljesa Milic[+] Eiman Ebrahimi[†] Oreste Villa[†] Aamer Jaleel[†] Carole-Jean Wu[‡] David Nellans[†] .2017.MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 13 pages.
https://doi.org/10.1145/3079856.3080231

ISCA '17, June 24-28, 2017, Toronto, ON, Canada

ACM ISBN 978-1-4503-4892-8/17/06…$15.00
https://doi.org/10.1145/3079856.3080231

1 INTRODUCTION

GPU-based compute acceleration is the main vehicle propelling the performance of high performance computing (HPC) systems [12, 17, 29], machine learning and data analytics applications in large-scale cloud installations, and personal computing devices [15, 17, 35, 47]. In such devices, each computing node or computing device typically consists of a CPU with one or more GPU accelerators. The path forward in any of these domains, either to exascale performance in HPC, or to human-level artificial intelligence using deep convolutional neural networks, relies on the ability to continuously scale GPU performance [29, 47]. As a result, in such systems, each GPU has the maximum possible transistor count at the most advanced technology node, and uses state-of-the-art memory technology [17]. Until recently, transistor scaling improved single GPU performance by increasing the Streaming Multiprocessor (SM) count between GPU generations. However, transistor scaling has dramatically slowed down and is expected to eventually come to an end [7, 8]. Furthermore, optic and manufacturing limitations constrain the reticle size which in turn constrains the maximum die size (e.g. $\approx 800mm^2$ [18, 48]). Moreover, very large dies have extremely low yield due to large numbers of irreparable manufacturing faults [31]. This increases the cost of large monolithic GPUs to undesirable levels. Consequently, these trends limit future scaling of single GPU performance and potentially bring it to a halt.

An alternate approach to scaling performance without exceeding the maximum chip size relies on multiple GPUs connected on a PCB, such as the Tesla K10 and K80 [10]. However, as we show in this paper, it is hard to scale GPU workloads on such "multi-GPU" systems, even if they scale very well on a single GPU. This is due

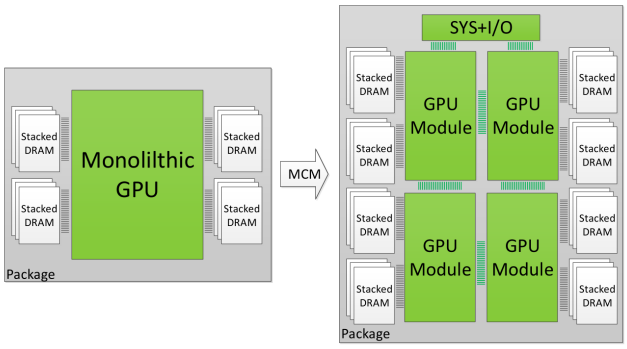


Figure 1: MCM-GPU: Aggregating GPU modules and DRAM on a single package.

to multiple unsolved challenges related to work partitioning, load balancing, and data sharing across the slow on-board interconnection network [20, 23, 33, 36]. However, due to recent advances in packaging [30] and signaling technologies [45], package-level integration provides a promising integration tier that lies between the existing on-chip and on-board integration technologies.

Leveraging this new integration tier, we propose a novel **Multi-Chip Module GPU** (MCM-GPU) architecture that enables continued GPU performance scaling despite the slowdown of transistor scaling and photoreticle limitations. Our proposal aggregates multiple GPU Modules (GPMs) within a single package as illustrated in Figure 1. First, we detail the basic MCM-GPU architecture that leverages NVIDIA's state-of-the-art Ground Reference Signaling (GRS) [45]. We then optimize our proposed MCM-GPU design using three architectural innovations targeted at improving locality and minimizing inter-GPM communication: (i) hardware caches to capture remote traffic in the local GPM, (ii) distributed and batched co-operative thread array (CTA) scheduling to better leverage inter-CTA locality within a GPM, and (iii) first touch page allocation policy to minimize inter-GPM traffic. Overall, this paper makes the following contributions:

- We motivate the need for more powerful GPUs by showing that many of today's GPU applications scale very well with increasing number of SMs. Given future GPUs can no longer continue their performance scaling using today's monolithic architectures, we propose the MCM-GPU architecture that allows performance and energy efficient scaling beyond what is possible today.
- We present a modular MCM-GPU with 256 SMs and discuss its memory system, on-package integration, and signaling technology. We show its performance sensitivity to inter-GPM bandwidth both analytically and via simulations. Our evaluation shows that since inter-GPM bandwidth is lower than a monolithic GPU's on-chip bandwidth, an on-package non-uniform memory access (NUMA) architecture is exposed in the MCM-GPU.
- We propose a locality-aware MCM-GPU architecture, better suited to its NUMA nature. We use architectural enhancements to mitigate the penalty introduced by non-uniform memory accesses. Our evaluations show that these optimizations provide an impressive 5x inter-GPM bandwidth reduction, and result in a 22.8% performance speedup compared to the baseline MCM-GPU. Our optimized MCM-GPU architecture achieves a 44.5% speedup over the largest possible monolithic GPU (assumed as a 128 SMs GPU), and comes within 10% of the performance of an unbuildable similarly sized monolithic GPU.
- Finally, we compare our MCM-GPU architecture to a multi-GPU approach. Our results confirm the intuitive advantages of the MCM-GPU approach.

	Fermi	Kepler	Maxwell	Pascal
SMs	16	15	24	56
BW (GB/s)	177	288	288	720
L2 (KB)	768	1536	3072	4096
Transistors (B)	3.0	7.1	8.0	15.3
Tech. node (nm)	40	28	28	16
Chip size (mm2)	529	551	601	610

Table 1: Key characteristics of recent NVIDIA GPUs.

2 MOTIVATION AND BACKGROUND

Modern GPUs accelerate a wide spectrum of parallel applications in the fields of scientific computing, data analytics, and machine learning. The abundant parallelism available in these applications continually increases the demands for higher performing GPUs. Table 1 lists different generations of NVIDIA GPUs released in the past decade. The table shows an increasing trend for the number of streaming multiprocessors (SMs), memory bandwidth, and number of transistors with each new GPU generation [14].

2.1 GPU Application Scalability

To understand the benefits of increasing the number of GPU SMs, Figure 2 shows performance as a function of the number of SMs on a GPU. The L2 cache and DRAM bandwidth capacities are scaled up proportionally with the SM count, i.e., 384 GB/s for a 32-SM GPU and 3 TB/s for a 256-SM GPU[1]. The figure shows two different performance behaviors with increasing SM counts. First is the trend of applications with limited parallelism whose performance plateaus with increasing SM count (Limited Parallelism Apps). These applications exhibit poor performance scalability (15 of the total 48 applications evaluated) due to the lack of available parallelism (i.e. number of threads) to fully utilize larger number of SMs. On the other hand, we find that 33 of the 48 applications exhibit a high degree of parallelism and fully utilize a 256-SM GPU. Note that such a GPU is substantially larger (4.5×) than GPUs available today. For these High-Parallelism Apps, 87.8% of the linearly-scaled theoretical performance improvement can potentially be achieved if such a large GPU could be manufactured.

Unfortunately, despite the application performance scalability with the increasing number of SMs, *the observed performance gains are unrealizable with a monolithic single-die GPU design*. This is because the slowdown in transistor scaling [8] eventually limits the number of SMs that can be integrated onto a given die area. Additionally, conventional photolithography technology limits the maximum possible reticle size and hence the maximum possible

[1] See Section 4 for details on our experimental methodology

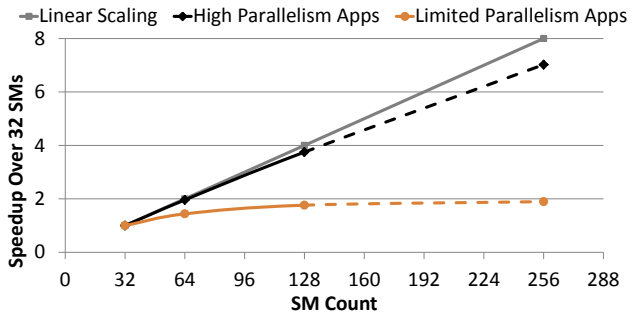


Figure 2: Hypothetical GPU performance scaling with growing number of SMs and memory system. 48 applications are grouped into 33 that have enough parallelism to fill a 256 SMs GPU, and 15 that do not.

die size. For example, $\approx 800mm^2$ is expected to be the maximum possible die size that can be manufactured [18, 48]. For the purpose of this paper we assume that GPUs with greater than 128 SMs are not manufacturable on a monolithic die. We illustrate the performance of such an unmanufacturable GPU with dotted lines in Figure 2.

2.2 Multi-GPU Alternative

An alternative approach is to stop scaling single GPU performance, and increase application performance via board- and system-level integration, by connecting multiple maximally sized monolithic GPUs into a multi-GPU system. While conceptually simple, multi-GPU systems present a set of critical challenges. For instance, work distribution across GPUs cannot be done easily and transparently and requires significant programmer expertise [20, 25, 26, 33, 42, 50]. Automated multi-GPU runtime and system-software approaches also face challenges with respect to work partitioning, load balancing, and synchronization [23, 49].

Moreover, a multi-GPU approach heavily relies on multiple levels of system interconnections. It is important to note that the data movement and synchronization energy dissipated along these interconnects significantly affects the overall performance and energy efficiency of such multi-GPU systems. Unfortunately, the quality of interconnect technology in terms of available bandwidth and energy per bit becomes progressively worse as communication moves off-package, off-board, and eventually off-node, as shown in Table 2 [9, 13, 16, 32, 46]. While the above integration tiers are an essential part of large systems (e.g. [19]), it is more desirable to reduce the off-board and off-node communication by building more capable GPUs.

2.3 Package-Level Integration

Recent advances in organic package technology are expected to address today's challenges and enable on-package integration of active components. For example, next generation packages are expected to support a $77mm$ substrate dimension [30], providing enough room to integrate the MCM-GPU architecture described in this paper. Furthermore, advances in package level signaling technologies such as NVIDIA's Ground-Referenced Signaling (GRS), offer the necessary high-speed, high-bandwidth signaling for organic package substrates.

	Chip	Package	Board	System
BW	10s TB/s	1.5 TB/s	256 GB/s	12.5 GB/s
Energy	80 fJ/bit	0.5 pJ/bit	10 pJ/bit	250 pJ/bit
Overhead	Low	Medium	High	Very High

Table 2: Approximate bandwidth and energy parameters for different integration domains.

GRS signaling can operate at 20 Gb/s while consuming just 0.54 pJ/bit in a standard 28nm process [45]. As this technology evolves, we can expect it to support up to multiple TB/s of on-package bandwidth. This makes the on-package signaling bandwidth eight times larger than that of on-board signaling.

The aforementioned factors make package level integration a promising integration tier, that qualitatively falls in between chip- and board-level integration tiers (See Table 2). In this paper, we aim to take advantage of this integration tier and set the ambitious goal of exploring how to manufacture a 2× more capable GPU, comprising 256 or more SMs within a single GPU package.

3 MULTI-CHIP-MODULE GPUS

The proposed Multi-Chip Module GPU (MCM-GPU) architecture is based on aggregating multiple GPU modules (GPMs) within a single package, as opposed to today's GPU architecture based on a single monolithic die. This enables scaling single GPU performance by increasing the number of transistors, DRAM, and I/O bandwidth per GPU. Figure 1 shows an example of an MCM-GPU architecture with four GPMs on a single package that potentially enables up to 4× the number of SMs (chip area) and 2× the memory bandwidth (edge size) compared to the largest GPU in production today.

3.1 MCM-GPU Organization

In this paper we propose the MCM-GPU as a collection of GPMs that share resources and are presented to software and programmers as a single monolithic GPU. Pooled hardware resources, and shared I/O are concentrated in a shared on-package module (the SYS + I/O module shown in Figure 1). The goal for this MCM-GPU is to provide the same performance characteristics as a single (unmanufacturable) monolithic die. By doing so, the operating system and programmers are isolated from the fact that a single logical GPU may now be several GPMs working in conjunction. There are two key advantages to this organization. First, it enables resource sharing of underutilized structures within a single GPU and eliminates hardware replication among GPMs. Second, applications will be able to transparently leverage bigger and more capable GPUs, without any additional programming effort.

Alternatively, on-package GPMs could be organized as multiple fully functional and autonomous GPUs with very high speed interconnects. However, we do not propose this approach due to its drawbacks and inefficient use of resources. For example, if implemented as multiple GPUs, splitting the off-package I/O bandwidth across GPMs may hurt overall bandwidth utilization. Other common architectural components such as virtual memory management, DMA engines, and hardware context management would also be private rather than pooled resources. Moreover, operating systems and programmers would have to be aware of potential load imbalance

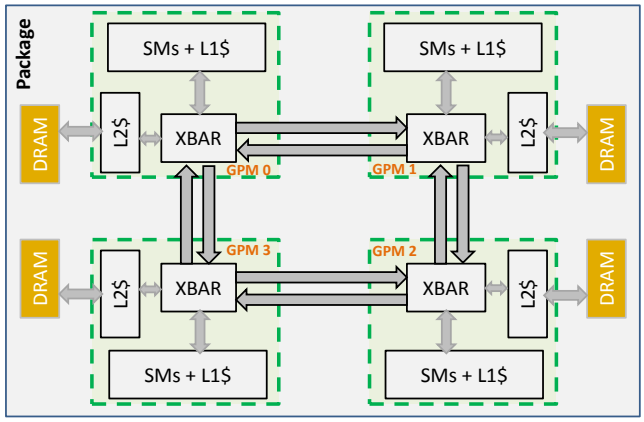


Figure 3: Basic MCM-GPU architecture comprising four GPU modules (GPMs).

and data partitioning between tasks running on such an MCM-GPU that is organized as multiple independent GPUs in a single package.

3.2 MCM-GPU and GPM Architecture

As discussed in Sections 1 and 2, moving forward beyond 128 SM counts will almost certainly require at least two GPMs in a GPU. Since smaller GPMs are significantly more cost-effective [31], in this paper we evaluate building a 256 SM GPU out of four GPMs of 64 SMs each. This way each GPM is configured very similarly to today's biggest GPUs. Area-wise each GPM is expected to be 40% - 60% smaller than today's biggest GPU assuming the process node shrinks to 10nm or 7nm. Each GPM consists of multiple SMs along with their private L1 caches. SMs are connected through the GPM-Xbar to a GPM memory subsystem comprising a local memory-side L2 cache and DRAM partition. The GPM-Xbar also provides connectivity to adjacent GPMs via on-package GRS [45] inter-GPM links.

Figure 3 shows the high-level diagram of this 4-GPM MCM-GPU. Such an MCM-GPU is expected to be equipped with 3TB/s of total DRAM bandwidth and 16MB of total L2 cache. All DRAM partitions provide a globally shared memory address space across all GPMs. Addresses are fine-grain interleaved across all physical DRAM partitions for maximum resource utilization. GPM-Xbars route memory accesses to the proper location (either the local or a remote L2 cache bank) based on the physical address. They also collectively provide a modular on-package ring or mesh interconnect network. Such organization provides spatial traffic locality among local SMs and memory partitions, and reduces on-package bandwidth requirements. Other network topologies are also possible especially with growing number of GPMs, but a full exploration of inter-GPM network topologies is outside the scope of this paper. The L2 cache is a memory-side cache, caching data only from its local DRAM partition. As such, there is only one location for each cache line, and no cache coherency is required across the L2 cache banks. In the baseline MCM-GPU architecture we employ a centralized CTA scheduler that schedules CTAs to MCM-GPU SMs globally in a round-robin manner as SMs become available for execution, as in the case of a typical monolithic GPU.

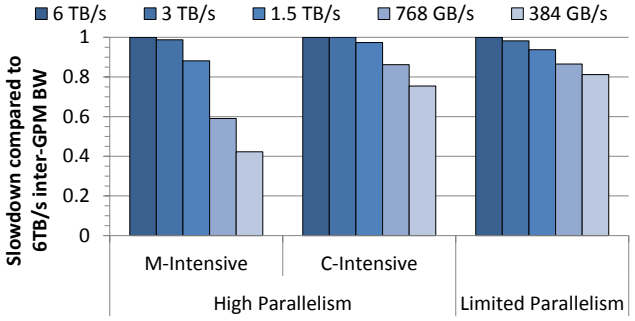


Figure 4: Relative performance sensitivity to inter-GPM link bandwidth for a 4-GPM, 256SM MCM-GPU system.

The MCM-GPU memory system is a Non Uniform Memory Access (NUMA) architecture, as its inter-GPM links are not expected to provide full aggregated DRAM bandwidth to each GPM. Moreover, an additional latency penalty is expected when accessing memory on remote GPMs. This latency includes data movement time within the local GPM to the edge of the die, serialization and deserialization latency over the inter-GPM link, and the wire latency to the next GPM. We estimate each additional inter-GPM hop latency, for a potentially multi-hop path in the on-package interconnect as 32 cycles. Each additional hop also adds an energy cost compared to a local DRAM access. Even though we expect the MCM-GPU architecture to incur these bandwidth, latency, and energy penalties, we expect them to be much lower compared to off-package interconnects in a multi-GPU system (see Table 2).

3.3 On-Package Bandwidth Considerations

3.3.1 Estimation of On-package Bandwidth Requirements. We calculate the required inter-GPM bandwidth in a generic MCM-GPU. The basic principle for our analysis is that on-package links need to be sufficiently sized to allow full utilization of expensive DRAM bandwidth resources. Let us consider a 4-GPM system with an aggregate DRAM bandwidth of $4b$ units (3TB/s in our example), such that b units of bandwidth (768 GB/s in our example) are delivered by the local memory partition directly attached to each GPM. Assuming an L2 cache hit-rate of $\sim 50\%$ for the average case, $2b$ units of bandwidth would be supplied from each L2 cache partition. In a statistically uniform address distribution scenario, $2b$ units of bandwidth out of each memory partition would be equally consumed by all four GPMs. Extending this exercise to capture inter-GPM communication to and from all memory partitions results in the total inter-GPM bandwidth requirement of the MCM-GPU. A link bandwidth of $4b$ would be necessary to provide $4b$ total DRAM bandwidth. In our 4-GPM MCM-GPU example with 3TB/s of DRAM bandwidth ($4b$), link bandwidth settings of less than 3TB/s are expected to result in performance degradation due to NUMA effects. Alternatively, inter-GPM bandwidth settings greater than 3TB/s are not expected to yield any additional performance.

3.3.2 Performance Sensitivity to On-Package Bandwidth. Figure 4 shows performance sensitivity of a 256 SM MCM-GPU system as we decrease the inter-GPM bandwidth from an abundant 6TB/s per link all the way to 384GB/s. The applications are

grouped into two major categories of high- and low-parallelism, similar to Figure 2. The scalable high-parallelism category is further subdivided into memory-intensive and compute-intensive applications (For further details about application categories and simulation methodology see Section 4).

Our simulation results support our analytical estimations above. Increasing link bandwidth to 6TB/s yields diminishing or even no return for an entire suite of applications. As expected, MCM-GPU performance is significantly affected by the inter-GPM link bandwidth settings lower than 3TB/s. For example, applications in the memory-intensive category are the most sensitive to link bandwidth, with 12%, 40%, and 57% performance degradation for 1.5TB/s, 768GB/s, and 384GB/s settings respectively. Compute-intensive applications are also sensitive to lower link bandwidth settings, however with lower performance degradations. Surprisingly, even the non-scalable applications with limited parallelism and low memory intensity show performance sensitivity to the inter-GPM link bandwidth due to increased queuing delays and growing communication latencies in the low bandwidth scenarios.

3.3.3 On-Package Link Bandwidth Configuration.

NVIDIA's GRS technology can provide signaling rates up to 20 Gbps per wire. The actual on-package link bandwidth settings for our 256 SM MCM-GPU can vary based on the amount of design effort and cost associated with the actual link design complexity, the choice of packaging technology, and the number of package routing layers. Therefore, based on our estimations, an inter-GPM GRS link bandwidth of 768 GB/s (equal to the local DRAM partition bandwidth) is easily realizable. Larger bandwidth settings such as 1.5 TB/s are possible, albeit harder to achieve, and a 3TB/s link would require further investment and innovations in signaling and packaging technology. Moreover, higher than necessary link bandwidth settings would result in additional silicon cost and power overheads. Even though on-package interconnect is more efficient than its on-board counterpart, it is still substantially less efficient than on-chip wires and thus we *must* minimize inter-GPM link bandwidth consumption as much as possible.

In this paper we assume a low-effort, low-cost, and low-energy link design point of 768GB/s and make an attempt to bridge the performance gap due to relatively lower bandwidth settings via architectural innovations that improve communication locality and essentially eliminate the need for more costly and less energy efficient links. The rest of the paper proposes architectural mechanisms to capture data-locality within GPM modules, which eliminate the need for costly inter-GPM bandwidth solutions.

4 SIMULATION METHODOLOGY

We use an NVIDIA in-house simulator to conduct our performance studies. We model the GPU to be similar to, but extrapolated in size compared to the recently released NVIDIA Pascal GPU [17]. Our SMs are modeled as in-order execution processors that accurately model warp-level parallelism. We model a multi-level cache hierarchy with a private L1 cache per SM and a shared L2 cache. Caches are banked such that they can provide the necessary parallelism to saturate DRAM bandwidth. We model software based cache coherence in the private caches, similar to state-of-the-art GPUs. Table 3 summarizes baseline simulation parameters.

Number of GPMs	4
Total number of SMs.	256
GPU frequency	1GHz
Max number of warps	64 per SM
Warp scheduler	Greedy then Round Robin
L1 data cache	128 KB per SM, 128B lines, 4 ways
Total L2 cache	16MB, 128B lines, 16 ways
Inter-GPM interconnect	768GB/s per link, Ring, 32 cycles/hop
Total DRAM bandwidth	3 TB/s
DRAM latency	100ns

Table 3: Baseline MCM-GPU configuration.

Benchmark	Abbr.	Memory Footprint (MB)
Algebraic multigrid solver	AMG	5430
Neural Network Convolution	NN-Conv	496
Breadth First Search	BFS	37
CFD Euler3D	CFD	25
Classic Molecular Dynamics	CoMD	385
Kmeans clustering	Kmeans	216
Lulesh (size 150)	Lulesh1	1891
Lulesh (size 190)	Lulesh2	4309
Lulesh unstructured	Lulesh3	203
Adaptive Mesh Refinement	MiniAMR	5407
Mini Contact Solid Mechanics	MnCtct	251
Minimum Spanning Tree	MST	73
Nekbone solver (size 18)	Nekbone1	1746
Nekbone solver (size 12)	Nekbone2	287
SRAD (v2)	Srad-v2	96
Shortest path	SSSP	37
Stream Triad	Stream	3072

Table 4: The high parallelism, memory intensive workloads and their memory footprints[2].

We study a diverse set of 48 benchmarks that are taken from four benchmark suites. Our evaluation includes a set of production class HPC benchmarks from the CORAL benchmarks [6], graph applications from Lonestar suite [43], compute applications from Rodinia [24], and a set of NVIDIA in-house CUDA benchmarks. Our application set covers a wide range of GPU application domains including machine learning, deep neural networks, fluid dynamics, medical imaging, graph search, etc. We classify our applications into two categories based on the available parallelism — high parallelism applications (parallel efficiency $\geq 25\%$) and limited parallelism applications (parallel efficiency $< 25\%$). We further categorize the high parallelism applications based on whether they are memory-intensive (M-Intensive) or compute-intensive (C-Intensive). We classify an application as memory-intensive if it suffers from more than 20% performance degradation if the system memory bandwidth is halved. In the interest of space, we present the detailed per-application results for the M-Intensive category workloads and present only the average numbers for the C-Intensive and limited-parallelism workloads. The set of M-Intensive benchmarks, and their memory footprints are detailed in Table 4. We simulate all our benchmarks for one billion warp instructions, or to completion, whichever occurs first.

[2]Other evaluated compute intensive and limited parallelism workloads are not shown in Table 4.

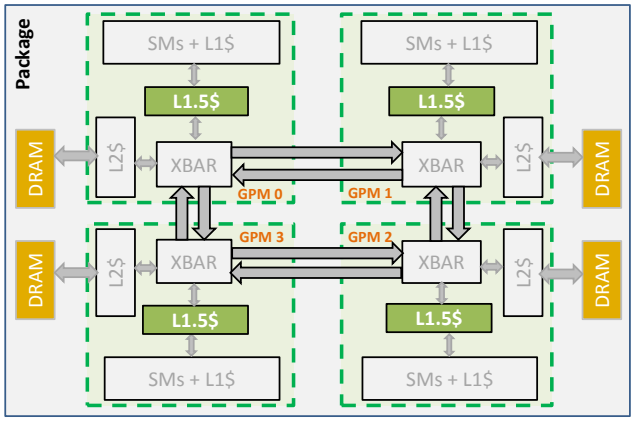


Figure 5: MCM-GPU architecture equipped with L1.5 GPM-side cache to capture remote data and effectively reduce inter-GPM bandwidth and data access latency.

5 OPTIMIZED MCM-GPU

We propose three mechanisms to minimize inter-GPM bandwidth by capturing data locality within a GPM. First, we revisit the MCM-GPU cache hierarchy and propose a GPM-side hardware cache. Second, we augment our architecture with distributed CTA scheduling to exploit inter-CTA data locality within the GPM-side cache and in memory. Finally, we propose data partitioning and locality-aware page placement to further reduce on-package bandwidth requirements. The three mechanisms combined significantly improve MCM-GPU performance.

5.1 Revisiting MCM-GPU Cache Architecture

5.1.1 *Introducing L1.5 Cache.*

The first mechanism we propose to reduce on-package link bandwidth is to enhance the MCM-GPU cache hierarchy. We propose to augment our baseline GPM architecture in Figure 3 with a GPM-side cache that resides between the L1 and L2 caches. We call this new cache level the L1.5 cache as shown in Figure 5. Architecturally, the L1.5 cache can be viewed as an extension of the L1 cache and is shared by all SMs inside a GPM. We propose that the L1.5 cache stores remote data accesses made by a GPM partition. In other words, all local memory accesses will bypass the L1.5 cache. Doing so reduces both remote data access latency and inter-GPM bandwidth. Both these properties improve performance and reduce energy consumption by avoiding inter-GPM communication.

To avoid increasing on-die transistor overhead for the L1.5 cache, we add it by rebalancing the cache capacity between the L2 and L1.5 caches in an iso-transistor manner. We extend the GPU L1 cache coherence mechanism to the GPM-side L1.5 caches as well. This way, whenever an L1 cache is flushed on a synchronization event such as reaching a kernel execution boundary, the L1.5 cache is flushed as well. Since the L1.5 cache can receive multiple invalidation commands from GPM SMs, we make sure that the L1.5 cache is invalidated only once for each synchronization event.

5.1.2 *Design Space Exploration for the L1.5 Cache.*

We evaluate MCM-GPU performance for three different L1.5 cache capacities: an 8MB L1.5 cache where half of the memory-side L2 cache capacity is moved to the L1.5 caches, a 16MB L1.5 cache where almost all of the memory-side L2 cache is moved to the L1.5 caches[3], and finally a 32MB L1.5 cache, a non iso-transistor scenario where in addition to moving the entire L2 cache capacity to the L1.5 caches we add an additional 16MB of cache capacity. As the primary objective of the L1.5 cache is to reduce the inter-GPM bandwidth consumption, we evaluate different cache allocation policies based on whether accesses are to the local or remote DRAM partitions.

Figure 6 summarizes the MCM-GPU performance for different L1.5 cache sizes. We report the average performance speedups for each category, and focus on the memory-intensive category by showing its individual application speedups. We observe that performance for the memory-intensive applications is sensitive to the L1.5 cache capacity, while applications in the compute-intensive and limited-parallelism categories show very little sensitivity to various cache configurations. When focusing on the memory-intensive applications, an 8MB iso-transistor L1.5 cache achieves 4% average performance improvement compared to the baseline MCM-GPU. A 16MB iso-transistor L1.5 cache achieves 8% performance improvement, and a 32MB L1.5 cache that doubles the transistor budget achieves an 18.3% performance improvement. We choose the 16MB cache capacity for the L1.5 and keep the total cache area constant.

Our simulation results confirm the intuition that the best allocation policy for the L1.5 cache is to only cache remote accesses, and therefore we employ a remote-only allocation policy in this cache. From Figure 6 we can see that such a configuration achieves the highest average performance speedup among the two iso-transistor configurations. It achieves an 11.4% speedup over the baseline for the memory-intensive GPU applications. While the GPM-side L1.5 cache has minimal impact on the compute-intensive GPU applications, it is able to capture the relatively small working sets of the limited-parallelism GPU applications and provide a performance speedup of 3.5% over the baseline. Finally, Figure 6 shows that the L1.5 cache generally helps applications that incur significant performance loss when moving from a 6TB/s inter-GPM bandwidth setting to 768GB/s. This trend can be seen in the figure as the memory-intensive applications are sorted by their inter-GPM bandwidth sensitivity from left to right.

In addition to improving MCM-GPU performance, the GPM-side L1.5 cache helps to significantly reduce the inter-GPM communication energy associated with on-package data movements. This is illustrated by Figure 7 which summarizes the total inter-GPM bandwidth with and without L1.5 cache. Among the memory-intensive workloads, inter-GPM bandwidth is reduced by as much as 39.9% for the `SSSP` application and by an average of 16.9%, 36.4%, and 32.9% for memory-intensive, compute-intensive, and limited-parallelism workloads respectively. On average across all evaluated workloads, we observe that inter-GPM bandwidth utilization is reduced by 28% due to the introduction of the GPM-side L1.5 cache.

5.2 CTA Scheduling for GPM Locality

In a baseline MCM-GPU similar to monolithic GPU, at kernel launch, a first batch of CTAs are scheduled to the SMs by a centralized scheduler in-order. However during kernel execution, CTAs are

[3]A small cache capacity of 32KB is maintained in the memory-side L2 cache to accelerate atomic operations.

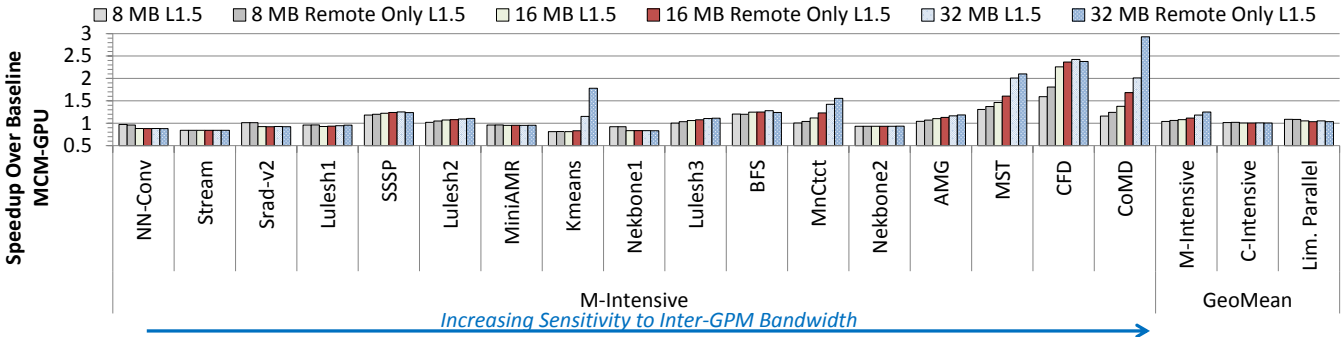


Figure 6: Performance of 256 SM, 768 GB/s inter-GPM BW MCM-GPU with 8MB (iso-transistor), 16 MB (iso-transistor), and 32 MB (non-iso-transistor) L1.5 caches. The M-Intensive applications are sorted by their sensitivity to inter-GPM bandwidth.

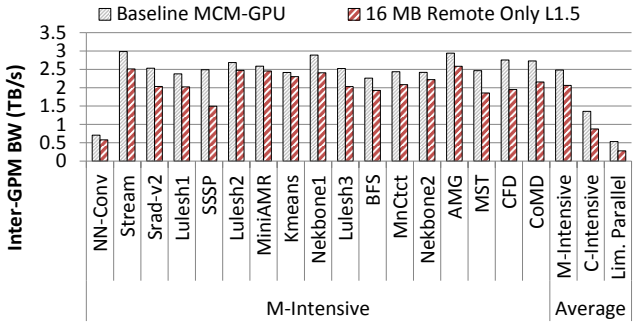
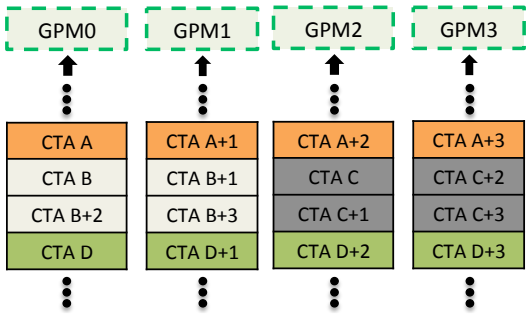


Figure 7: Total inter-GPM bandwidth in baseline MCM-GPU architecture and with a 16MB *remote-only* L1.5 cache.

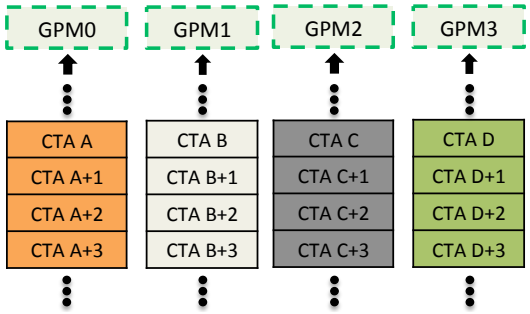
allocated to SMs in a round-robin order based on the availability of resources in the SMs to execute a given CTA. In steady state application execution, this could result in consecutive CTAs being scheduled on SMs in different GPMs as shown in Figure 8(a). The colors in this figure represent four groups of contiguous CTAs that could potentially enjoy data locality if they were scheduled in close proximity and share memory system resources. While prior work has attempted to exploit such inter-CTA locality in the private L1 cache [37], here we propose a CTA scheduling policy to exploit this locality across all memory system components associated with GPMs due to the NUMA nature of the MCM-GPU design.

To this end, we propose using a distributed CTA scheduler for the MCM-GPU. With the distributed CTA scheduler, a group of contiguous CTAs are sent to the same GPM as shown in Figure 8(b). Here we see that all four contiguous CTAs of a particular group are assigned to the same GPM. In the context of the MCM-GPU, doing so enables better cache hit rates in the L1.5 caches and also reduces inter-GPM communication. The reduced inter-GPM communication occurs due to contiguous CTAs sharing data in the L1.5 cache and avoiding data movement over the inter-GPM links. In the example shown in Figure 8, the four groups of contiguous CTAs are scheduled to run on one GPM each, to potentially exploit inter-CTA spatial data locality.

We choose to divide the total number of CTAs in a kernel equally among the number of GPMs, and assign a group of contiguous CTAs



(a) Centralized CTA Scheduling in an MCM-GPU



(b) Distributed CTA Scheduling in an MCM-GPU

Figure 8: An example of exploiting inter-CTA data locality with CTA scheduling in MCM-GPU.

to a GPM. Figures 9 and 10 show the performance improvement and bandwidth reduction provided by our proposal when combined with the L1.5 cache described in the previous section. On average, the combination of these proposals improves performance by 23.4% / 1.9% / 5.2% on memory-intensive, compute-intensive, and limited-parallelism workloads respectively. In addition, inter-GPM bandwidth is reduced further by the combination of these proposals. On average across all evaluated workloads, we observe that inter-GPM bandwidth utilization is reduced by 33%.

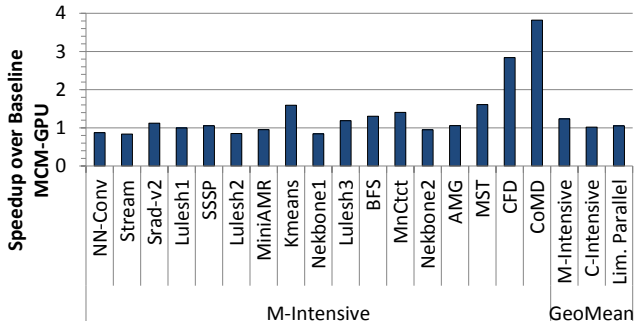


Figure 9: Performance of MCM-GPU system with a distributed scheduler.

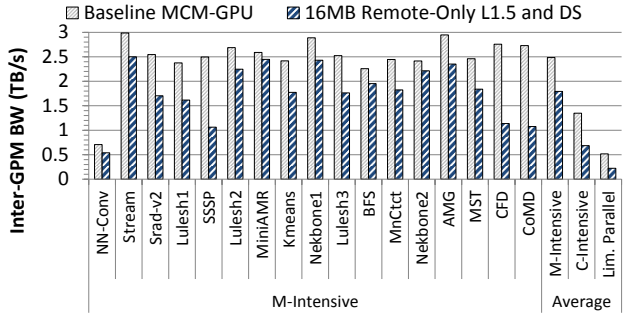


Figure 10: Reduction in inter-GPM bandwidth with a distributed scheduler compared to baseline MCM-GPU architecture.

For workloads such as `Srad-v2`, and `Kmeans`, the combination of distributed scheduling and remote-only caching provides significant performance improvement while remote-only caching does not improve performance in isolation (Figure 6). This is due to the improved inter-CTA data reuse in the L1.5 cache when distributed scheduling is applied. Although distributed scheduling provides significant additional performance benefit for a number of evaluated workloads, we observe that it causes some applications to experience degradation in performance. Such workloads tend to suffer from the coarse granularity of CTA division and may perform better with a smaller number of contiguous CTAs assigned to each GPM. A case for a dynamic mechanism for choosing the group size could be made. While we do not explore such a design in this paper, we expect a dynamic CTA scheduler to obtain further performance gain.

5.3 Data Partitioning for GPM Locality

Prior work on NUMA systems focuses on co-locating code and data by scheduling threads and placing pages accessed by those threads in close proximity [27, 39, 53]. Doing so limits the negative performance impact of high-latency low-bandwidth inter-node links by reducing remote accesses. In an MCM-GPU system, while the properties of inter-GPM links are superior to traditional inter-package links assumed in prior work (i.e., the ratio of local memory bandwidth compared to remote memory bandwidth is much greater and latency much lower for inter-package links), we revisit page placement policies to reduce inter-GPM bandwidth.

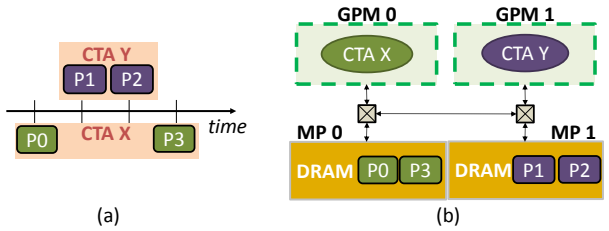


Figure 11: First Touch page mapping policy: (a) Access order. (b) Proposed page mapping policy

To improve MCM-GPU performance, special care is needed for page placement to reduce inter-GPM traffic when possible. Ideally, we would like to map memory pages to physical DRAM partitions such that they would incur as many local memory accesses as possible. In order to maximize DRAM bandwidth utilization and prevent camping on memory channels within the memory partitions, we will still interleave addresses at a fine granularity across the memory channels of each memory partition (analogous to the baseline described in Section 3.2).

Figure 11 shows a schematic representation of the first touch (FT) page mapping policy we employ in the MCM-GPU. When a page is referenced for the first time in the FT policy, the page mapping mechanism checks which GPM the reference is from and maps the page to the local memory partition (MP) of that GPM. For example, in the figure, page P0 is first accessed by CTA-X which is executing on GPM0. This results in P0 being allocated in MP0. Subsequently, pages P1 and P2 are first accessed by CTA-Y executing on GPM1, which maps those pages to MP1. Following this, page P3 is first accessed by CTA-X, which maps the page to MP0. This policy results in keeping DRAM accesses mostly local. Regardless of the referencing order, if a page is first referenced from CTA-X in GPM0, then the page will be mapped to the MP0, which would keep accesses to that page local and avoid inter-GPM communication. This page placement mechanism is implemented in the software layer by extending current GPU driver functionality. Such driver modification is transparent to the OS, and does not require any special handling from the programmer.

An important benefit that comes from the first touch mapping policy is its synergy with our CTA scheduling policy described in Section 5.2. We observe that inter-CTA locality exists across multiple kernels and within each kernel at a page granularity. For example, the same kernel is launched iteratively within a loop in applications that contain convergence loops and CTAs with the same indices are likely to access the same pages. Figure 12 shows an example of this. As a result of our distributed CTA scheduling policy and the first touch page mapping policy described above, we are able to exploit inter-CTA locality across the kernel execution boundary as well. This is enabled due to the fact that CTAs with the same indices are bound to the same GPM on multiple iterative launches of the kernel, therefore allowing the memory pages brought to a GPM's memory partition to continue to be local across subsequent kernel launches. Note that this locality does not show itself without the first touch page mapping policy as it does not increase L1.5 cache hit rates since the caches are flushed at kernel boundaries. However,

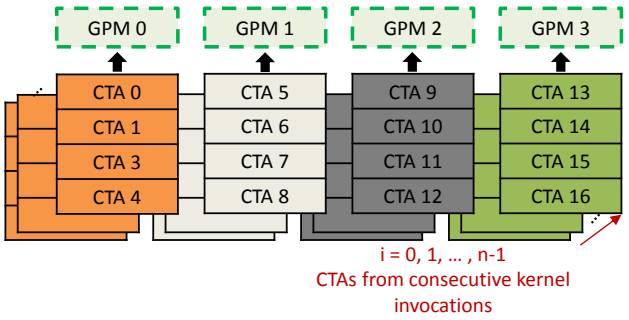


Figure 12: Exploiting cross-kernel CTA locality with First Touch page placement and distributed CTA scheduling

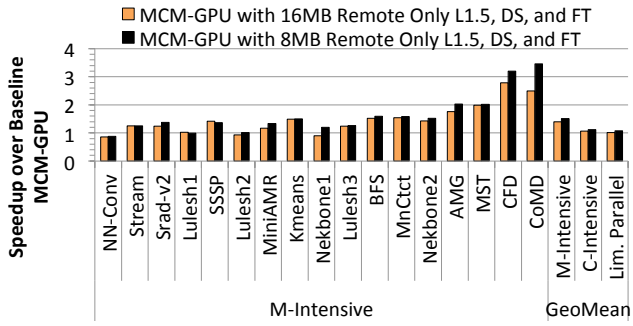


Figure 13: Performance of MCM-GPU with First Touch page placement

we benefit significantly from more local accesses when distributed scheduling is combined with first touch mapping.

FT also allows for much more efficient use of the cache hierarchy. Since FT page placement keeps many accesses local to the memory partition of a CTA's GPM, it reduces pressure on the need for an L1.5 cache to keep requests from going to remote memory partitions. In fact using the first touch policy shifts the performance bottleneck from inter-GPM bandwidth to local memory bandwidth. Figure 13 shows this effect. In this figure, we show two bars for each benchmark — FT with DS and 16MB remote-only L1.5 cache, and FT with DS and 8MB remote-only L1.5 cache. The 16MB L1.5 cache leaves room for only 32KB worth of L2 cache in each GPM. This results in sub-optimal performance as there is insufficient cache capacity that is allocated to local memory traffic. We observe that in the presence of FT, an 8MB L1.5 cache along with a larger 8MB L2 achieves better performance. The results show that with this configuration we can obtain 51% /11.3% / 7.9% performance improvements compared to the baseline MCM-GPU in memory-intensive, compute-intensive, and limited parallelism applications respectively. Finally Figure 14 shows that with FT page placement a multitude of workloads experience a drastic reduction in their inter-GPM traffic, sometimes almost eliminating it completely. On average our proposed MCM-GPU achieves a 5× reduction in inter-GPM bandwidth compared to the baseline MCM-GPU.

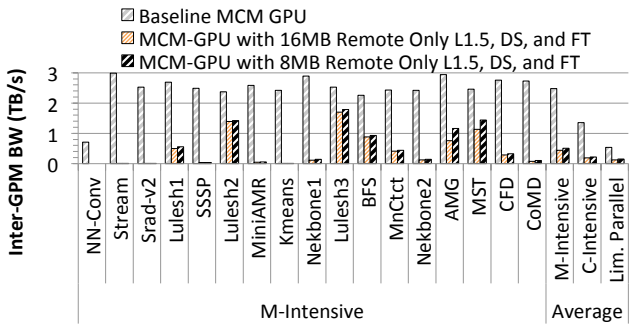


Figure 14: Reduction in inter-GPM bandwidth with First Touch page placement

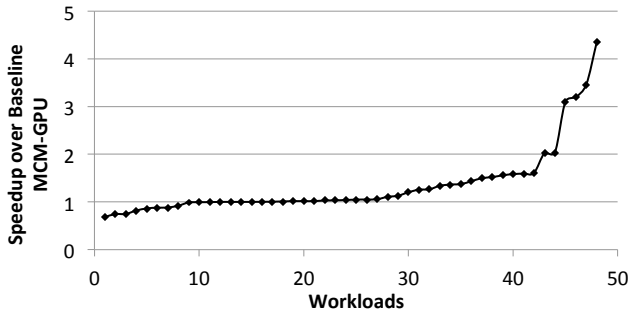


Figure 15: S-curve summarizing the optimized MCM-GPU performance speedups for all workloads.

5.4 MCM-GPU Performance Summary

Figure 15 shows the s-curve depicting the performance improvement of MCM-GPU for all workloads in our study. Of the evaluated 48 workloads, 31 workloads experience performance improvement while 9 workloads suffer some performance loss. M-Intensive workloads such as `CFD`, `CoMD` and others experience drastic reduction in inter-GPM traffic due to our optimizations and thus experience significant performance gains of up to 3.2× and 3.5× respectively. Workloads in the C-Intensive and limited parallelism categories that show high sensitivity to inter-GPM bandwidth also experience significant performance gains (e.g. 4.4× for `SP` and 3.1× for `XSBench`). On the flip side, we observe two side-effects of the proposed optimizations. For example, for workloads such as `DWT` and `NN` that have limited parallelism and are inherently insensitive to inter-GPM bandwidth, the additional latency introduced by the presence of the L1.5 cache can lead to performance degradation by up to 14.6%. Another reason for potential performance loss as observed in `Streamcluster` is due to the reduced capacity of on-chip writeback L2 caches[4] which leads to increased write traffic to DRAM. This results in performance loss of up to 25.3% in this application. Finally, we observe that there are workloads (two in our evaluation set) where different CTAs perform unequal amount of work. This leads to workload imbalance due to

[4]L1.5 caches are set up as write-through to support software based GPU coherence implementation

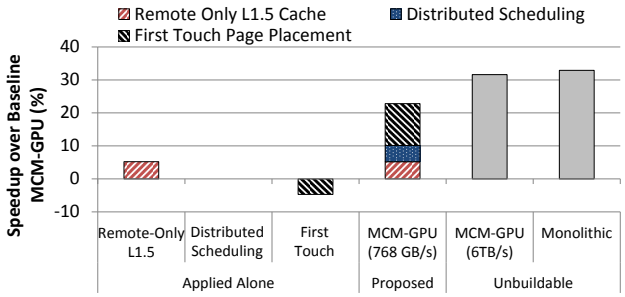


Figure 16: Breakdown of the sources of performance improvements of optimized MCM-GPU when applied alone and together. Three proposed architectural improvements for MCM-GPU almost close the gap with unbuildable monolithic GPU.

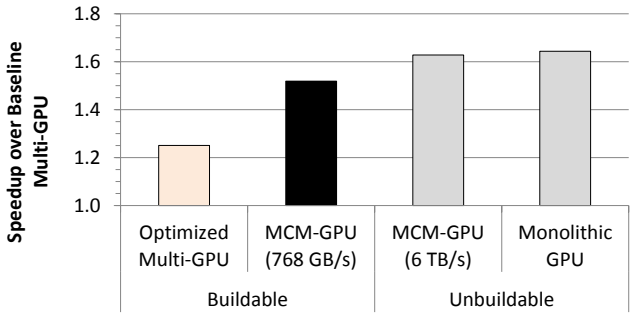


Figure 17: Performance comparison of MCM-GPU and Multi-GPU.

the coarse-grained distributed scheduling. We leave further optimizations of the MCM-GPU architecture that would take advantage of this potential opportunity for better performance to future work.

In summary, we have proposed three important mircroarchitectural enhancements to the baseline MCM-GPU architecture: (i) a remote-only L1.5 cache, (ii) a distributed CTA scheduler, and (iii) a first touch data page placement policy. It is important to note that these independent optimizations, work best when they are combined together. Figure 16 shows the performance benefit of employing the three mechanisms individually. The introduction of the L1.5 cache provides a 5.2% performance. Distributed scheduling and first touch page placement on the other hand, do not improve performance at all when applied individually. In fact they can even lead to performance degradation, e.g., -4.7% for the first touch page placement policy.

However, when all three mechanisms are applied together, we observe that the optimized MCM-GPU, achieves a speedup of 22.8% as shown in Figure 16. We observe that combining distributed scheduling with the remote-only cache improves cache performance and reduces the inter-GPM bandwidth further. This results in an additional 4.9% performance benefit compared to having just the remote-only cache while also reducing inter-GPM bandwidth by an additional 5%. Similarly, when first touch page placement is employed in conjunction with the remote-only cache and distributed scheduling, it provides an additional speedup of 12.7% and reduces inter-GPM bandwidth by an additional 47.2%. These results demonstrate that our proposed enhancements not only *exploit* the currently available data locality within a program but also *improve* it. Collectively, all three locality-enhancement mechanisms achieve a 5× reduction in inter-GPM bandwidth. These optimizations enable the proposed MCM-GPU to achieve a 45.5% speedup compared to the largest implementable monolithic GPU and be within 10% of an equally equipped albeit *unbuildable* monolithic GPU.

6 MCM-GPU VS MULTI-GPU

An alternative way of scaling GPU performance is to build multi-GPU systems. This section compares performance and energy efficiency of the MCM-GPU and two possible multi-GPU systems.

6.1 Performance vs Multi-GPU

A system with 256 SMs can also be built by interconnecting two maximally sized discrete GPUs of 128 SMs each. Similar to our MCM-GPU proposal, each GPU has a private 128KB L1 cache per SM, an 8MB memory-side cache, and 1.5 TB/s of DRAM bandwidth. We assume such a configuration as a maximally sized future monolithic GPU design. We assume that two GPUs are interconnected via the next generation of on-board level links with 256 GB/s of aggregate bandwidth, improving upon the 160 GB/s commercially available today [17]. We assume the multi-GPU to be fully transparent to the programmer. This is accomplished by assuming the following two features: (i) a unified memory architecture between two peer GPUs, where both GPUs can access local and remote DRAM resources with load/store semantics, (ii) a combination of system software and hardware which automatically distributes CTAs of the same kernel across GPUs.

In such a multi-GPU system the challenges of load imbalance, data placement, workload distribution and interconnection bandwidth discussed in Sections 3 and 5, are amplified due to severe NUMA effects from the lower inter-GPU bandwidth. Distributed CTA scheduling together with the first-touch page allocation mechanism (described respectively in Sections 5.2 and 5.3) are also applied to the multi-GPU. We refer to this design as a *baseline multi-GPU* system. Although a full study of various multi-GPU design options was not performed, alternative options for CTA scheduling and page allocation were investigated. For instance, a fine grain CTA assignment across GPUs was explored but it performed very poorly due to the high interconnect latency across GPUs. Similarly, round-robin page allocation results in very low and inconsistent performance across our benchmark suite.

Remote memory accesses are even more expensive in a multi-GPU when compared to MCM-GPU due to the relative lower quality of on-board interconnect. We optimize the multi-GPU baseline by adding GPU-side hardware caching of remote GPU memory, similar to the L1.5 cache proposed for MCM-GPU. We have explored various L1.5 cache allocation configurations, and observed the best average performance with a half of the L2 cache capacity moved to the L1.5 caches that are dedicated to caching remote DRAM accesses, and another half retained as the L2 cache for caching local DRAM accesses. We refer to this as the *optimized multi-GPU*.

Figure 17 summarizes the performance results for different buildable GPU organizations and unrealizable hypothetical designs, all normalized to the baseline multi-GPU configuration. The optimized multi-GPU which has GPU-side caches outperforms the baseline multi-GPU by an average of 25.1%. Our proposed MCM-GPU on

the other hand, outperforms the baseline multi-GPU by an average of 51.9% mainly due to higher quality on-package interconnect.

6.2 MCM-GPU Efficiency

Besides enabling performance scalability, MCM-GPUs are energy and cost efficient. MCM-GPUs are energy efficient as they enable denser integration of GPU modules on a package that alternatively would have to be connected at a PCB level as in a multi-GPU case. In doing so, MCM-GPUs require significantly smaller system footprint and utilize more efficient interconnect technologies, e.g., 0.5 pJ/b on-package vs 10 pJ/b on-board interconnect. Moreover, if we assume almost constant GPU and system power dissipation, the performance advantages of the MCM-GPU translate to additional energy savings. In addition, superior transistor density achieved by the MCM-GPU approach allows to lower GPU operating voltage and frequency. This moves the GPU to a more power-efficient operating point on the transistor voltage-frequency curve. Consequently, it allows trading off ample performance (achieved via abundant parallelism and number of transistors in package) for better power efficiency.

Finally, at a large scale such as HPC clusters the MCM-GPU improves performance density and as such reduces the number of GPUs per node and/or number of nodes per cabinet. This leads to a smaller number of cabinets at the system level. Smaller total system size translates to smaller number of communicating agents, smaller network size and shorter communication distances. These result in lower system level energy dissipation on communication, power delivery, and cooling. Similarly, higher system density also leads to total system cost advantages and lower overheads as described above. Moreover, MCM-GPUs are expected to result in lower GPU silicon cost as they replace large dies with medium size dies that have significantly higher silicon yield and cost advantages.

7 RELATED WORK

Multi-Chip-Modules are an attractive design point that have been extensively used in the industry to integrate multiple heterogeneous or homogeneous chips in the same package. On the homogeneous front, IBM Power 7 [5] integrates 4 modules of 8 cores each, and AMD Opteron 6300 [4] integrates 2 modules of 8 cores each. On the heterogeneous front, the IBM z196 [3] integrates 6 processors with 4 cores each and 2 storage controller units in the same package. The Xenos processor used in the Microsoft Xbox360 [1] integrates a GPU and an EDRAM memory module with its memory controller. Similarly, Intel offers heterogeneous and homogeneous MCM designs such as the Iris Pro [11] and the Xeon X5365 [2] processors respectively. While MCMs are popular in various domains, we are unaware of any attempt to integrate homogeneous high performance GPU modules on the same package in an OS and programmer transparent fashion. To the best of our knowledge, this is the first effort to utilize MCM technology to scale GPU performance.

MCM package level integration requires efficient signaling technologies. Recently, Kannan et al. [31] explored various packaging and architectural options for disintegrating multi-core CPU chips and studied its suitability to provide cache-coherent traffic in an efficient manner. Most recent work in the area of low-power links has focused on differential signaling because of its better noise immunity and lower noise generation [40, 44]. Some contemporary MCMs, like those used in the Power 6 processors, have over 800 single-ended links, operating at speeds of up to 3.2 Gbps, from a single processor [28]. NVIDIA's Ground-Referenced Signaling (GRS) technology for organic package substrates has been demonstrated to work at 20 Gbps while consuming just 0.54pJ/bit in a standard 28nm process [45].

The MCM-GPU design exposes a NUMA architecture. One of the main mechanisms to improve the performance of NUMA systems is to preserve locality by assigning threads in close proximity to the data. In a multi-core domain, existing work tries to minimize the memory access latency by thread-to-core mapping [21, 38, 51], or memory allocation policy [22, 27, 34]. Similar problems exist in MCM-GPU systems where the primary bottleneck is the inter-GPM interconnection bandwidth. Moreover, improved CTA scheduling has been proposed to exploit the inter-CTA locality, higher cache hit ratios, and memory bank-level parallelism [37, 41, 52] for monolithic GPUs. In our case, distributed CTA scheduling along with the first-touch memory mapping policy exploits inter-CTA localities both within a kernel and across multiple kernels, and improves efficiency of the newly introduced GPM-side L1.5 cache.

Finally, we propose to expose the MCM-GPU as a single logical GPU via hardware innovations and extensions to the driver software to provide programmer- and OS-transparent execution. While there have been studies that propose techniques to efficiently utilize multi-GPU systems [20, 23, 33, 36], none of the proposals provide a fully transparent approach suitable for MCM- GPUs.

8 CONCLUSIONS

Many of today's important GPU applications scale well with GPU compute capabilities and future progress in many fields such as exascale computing and artificial intelligence will depend on continued GPU performance growth. The greatest challenge towards building more powerful GPUs comes from reaching the end of transistor density scaling, combined with the inability to further grow the area of a single monolithic GPU die. In this paper we propose MCM-GPU, a novel GPU architecture that extends GPU performance scaling at a package level, beyond what is possible today. We do this by partitioning the GPU into easily manufacturable basic building blocks (GPMs), and by taking advantage of the advances in signaling technologies developed by the circuits community to connect GPMs on-package in an energy efficient manner.

We discuss the details of the MCM-GPU architecture and show that our MCM-GPU design naturally lends itself to many of the historical observations that have been made in NUMA systems. We explore the interplay of hardware caches, CTA scheduling, and data placement in MCM-GPUs to optimize this architecture. We show that with these optimizations, a 256 SMs MCM-GPU achieves 45.5% speedup over the largest possible monolithic GPU with 128 SMs. Furthermore, it performs 26.8% better than an equally equipped discrete multi-GPU, and its performance is within 10% of that of a hypothetical monolithic GPU that cannot be built based on today's technology roadmap.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their insightful feedback. This work is supported in part by the National Science Foundation (Grant #CCF-1618039).

REFERENCES

[1] 2005. Xenos: XBOX360 GPU. (2005). http://fileadmin.cs.lth.se/cs/Personal/Michael_Doggett/talks/eg05-xenos-doggett.pdf Accessed: 2016-08-19.

[2] 2007. The Xeon X5365. (2007). http://ark.intel.com/products/30702/Intel-Xeon-Processor-X5365-8M-Cache-3_00-GHz-1333-MHz-FSB Accessed: 2016-08-19.

[3] 2011. IBM zEnterprise 196 Technical Guide. (2011). http://www.redbooks.ibm.com/redbooks/pdfs/sg247833.pdf Accessed: 2016-08-19.

[4] 2012. AMD Server Solutions Playbook. (2012). http://www.amd.com/Documents/AMD_Opteron_ServerPlaybook.pdf Accessed: 2016-08-19.

[5] 2012. IBM Power Systems Deep Dive. (2012). http://www-05.ibm.com/cz/events/febannouncement2012/pdf/power_architecture.pdf Accessed: 2016-08-19.

[6] 2014. CORAL Benchmarks. (2014). https://asc.llnl.gov/CORAL-benchmarks/

[7] 2015. Intel Delays 10nm to 2017. (2015). http://www.extremetech.com/computing/210050-intel-confirms-10nm-delayed-to-2017-will-introduce-kaby-\lake-at-14nm-to-fill-gap

[8] 2015. International Technology Roadmap for Semiconductors 2.0. (2015). http://www.itrs2.net/itrs-reports.html

[9] 2015. Switch-IB 2 EDR Switch Silicon - World's First Smart Switch. (2015). http://www.mellanox.com/related-docs/prod_silicon/PB_SwitchIB2_EDR_Switch_Silicon.pdf Accessed: 2016-06-20.

[10] 2015. TESLA K80 GPU ACCELERATOR . (2015). https://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf Accessed: 2016-06-20.

[11] 2015. The Compute Architecture of Intel Processor Graphics Gen8. (2015). https://software.intel.com Accessed: 2016-08-19.

[12] 2015. TOP500 Shows Growing Momentum for Accelerators. (2015). http://insidehpc.com/2015/11/top500-shows-growing-momentum-for-accelerators/ Accessed: 2016-06-20.

[13] 2016. ConnectX-4 VPI Single and Dual Port QSFP28 Adapter Card User Manual. (2016). http://www.mellanox.com/related-docs/user_manuals/ConnectX-4_VPI_Single_and_Dual_QSFP28_Port_Adapter_Card_User_Manual.pdf Accessed: 2016-06-20.

[14] 2016. Inside Pascal: NVIDIA's Newest Computing Platform. (2016). https://devblogs.nvidia.com/parallelforall/inside-pascal Accessed: 2016-06-20.

[15] 2016. NVIDIA cuDNN, GPU Accelerated Deep Learning. (2016). https://developer.nvidia.com/cudnn Accessed: 2016-11-17.

[16] 2016. NVIDIA NVLink High-Speed Interconnect. (2016). http://www.nvidia.com/object/nvlink.html Accessed: 2016-06-20.

[17] 2016. The New NVIDIA Pascal Architecture. (2016). http://www.nvidia.com/object/gpu-architecture.html Accessed: 2016-06-20.

[18] 2016. The TWINSCAN NXT:1950i Dual-Stage Immersion Lithography System. (2016). https://www.asml.com/products/systems/twinscan-nxt/twinscan-nxt1950i/en/s46772?dfp_product_id=822 Accessed: 2016-11-18.

[19] 2016. Titan : The world's #1 Open Science Super Computer. (2016). https://www.olcf.ornl.gov/titan/

[20] Tal Ben-Nun, Ely Levy, Amnon Barak, and Eri Rubin. 2015. Memory Access Patterns: The Missing Piece of the multi-GPU Puzzle. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*. ACM, New York, NY, USA, 19:1–19:12. https://doi.org/10.1145/2807591.2807611

[21] Sergey Blagodurov, Alexandra Fedorova, Sergey Zhuravlev, and Ali Kamali. 2010. A case for NUMA-aware contention management on multicore systems. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT '10)*. IEEE, Vienna, Austria, 557–558. https://doi.org/10.1145/1854273.1854350

[22] William L. Bolosky, Robert P. Fitzgerald, and Michael L. Scott. 1989. Simple but Effective Techniques for NUMA Memory Management. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles (SOSP '89)*. ACM, New York, NY, USA, 19–31. https://doi.org/10.1145/74850.74854

[23] Javier Cabezas, Lluís Vilanova, Isaac Gelado, Thomas B. Jablin, Nacho Navarro, and Wen-mei W. Hwu. 2015. Automatic Parallelization of Kernels in Shared-Memory Multi-GPU Nodes. In *Proceedings of the 29th ACM on International Conference on Supercomputing (ICS '15)*. ACM, New York, NY, USA, 3–13. https://doi.org/10.1145/2751205.2751218

[24] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC '09)*. IEEE, Washington, DC, USA, 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[25] Long Chen, Oreste Villa, and Guang R. Gao. 2011. Exploring Fine-Grained Task-Based Execution on Multi-GPU Systems. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '11)*. IEEE, Washington, DC, USA, 386–394. https://doi.org/10.1109/CLUSTER.2011.50

[26] Long Chen, Oreste Villa, Sriram Krishnamoorthy, and Guang R. Gao. 2010. Dynamic load balancing on single- and multi-GPU systems. In *Proceedings of the IEEE International Symposium on Parallel Distributed Processing (IPDPS '10)*. IEEE, Atlanta, GA, USA, 1–12. https://doi.org/10.1109/IPDPS.2010.5470413

[27] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. 2013. Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. ACM, New York, NY, USA, 381–394. https://doi.org/10.1145/2451116.2451157

[28] Daniel Dreps. 2007. The 3rd generation of IBM's elastic interface on POWER6. In *Proceedings of the IEEE Hot Chips 19 Symposium (HCS '19)*. IEEE, 1–16. https://doi.org/10.1109/HOTCHIPS.2007.7482489

[29] Michael Feldman, Christopher G. Willard, and Addison Snell. 2015. HPC Application Support for GPU Computing. (2015). http://www.intersect360.com/industry/reports.php?id=131

[30] Mitsuya Ishida. 2014. Kyocera APX - An Advanced Organic Technology for 2.5D Interposers. (2014). https://www.ectc.net Accessed: 2016-06-20.

[31] Ajaykumar Kannan, Natalie Enright Jerger, and Gabriel H. Loh. 2015. Enabling Interposer-based Disintegration of Multi-core Processors. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. ACM, New York, NY, USA, 546–558. https://doi.org/10.1145/2830772.2830808

[32] Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. 2011. GPUs and the Future of Parallel Computing. *IEEE Micro* 31, 5 (Sept. 2011), 7–17. https://doi.org/10.1109/MM.2011.89

[33] Jungwon Kim, Honggyu Kim, Joo Hwan Lee, and Jaejin Lee. 2011. Achieving a Single Compute Device Image in OpenCL for Multiple GPUs. In *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP '11)*. ACM, New York, NY, USA, 277–288. https://doi.org/10.1145/1941553.1941591

[34] Richard P. LaRowe Jr., James T. Wilkes, and Carla S. Ellis. 1991. Exploiting Operating System Support for Dynamic Page Placement on a NUMA Shared Memory Multiprocessor. In *Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP '91)*. ACM, New York, NY, USA, 122–132. https://doi.org/10.1145/109625.109639

[35] Andrew Lavin and Scott Gray. 2016. Fast Algorithms for Convolutional Neural Networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR '16)*. IEEE, Las Vegas, NV, USA, 4013–4021. https://doi.org/10.1109/CVPR.2016.435

[36] Janghaeng Lee, Mehrzad Samadi, Yongjun Park, and Scott Mahlke. 2013. Transparent CPU-GPU Collaboration for Data-parallel Kernels on Heterogeneous Systems. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques (PACT '13)*. IEEE, Piscataway, NJ, USA, 245–256. http://dl.acm.org/citation.cfm?id=2523721.2523756

[37] Minseok Lee, Seokwoo Song, Joosik Moon, John Kim, Woong Seo, Yeongon Cho, and Soojung Ryu. 2014. Improving GPGPU resource utilization through alternative thread block scheduling. In *Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture (HPCA '14)*. IEEE, Orlando, FL, USA, 260–271. https://doi.org/10.1109/HPCA.2014.6835937

[38] Hui Li, Sudarsan Tandri, Michael Stumm, and Kenneth C. Sevcik. 1993. Locality and Loop Scheduling on NUMA Multiprocessors. In *Proceedings of the International Conference on Parallel Processing - Volume 02 (ICPP '93)*. IEEE, Washington, DC, USA, 140–147. https://doi.org/10.1109/ICPP.1993.112

[39] Zoltan Majo and Thomas R. Gross. 2012. Matching Memory Access Patterns and Data Placement for NUMA Systems. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization (CGO '12)*. ACM, New York, NY, USA, 230–241. https://doi.org/10.1145/2259016.2259046

[40] Mozhgan Mansuri, James E. Jaussi, Joseph T. Kennedy, Tzu-Chien Hsueh, Sudip Shekhar, Ganesh Balamurugan, Frank O'Mahony, Clark Roberts, Randy Mooney, and Bryan Casper. 2013. A scalable 0.128-to-1Tb/s 0.8-to-2.6pJ/b 64-lane parallel I/O in 32nm CMOS. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC '13)*. IEEE, San Francisco, CA, USA, 402–403. https://doi.org/10.1109/ISSCC.2013.6487788

[41] Mengjie Mao, Wujie Wen, Xiaoxiao Liu, Jingtong Hu, Danghui Wang, Yiran Chen, and Hai Li. 2016. TEMP: Thread Batch Enabled Memory Partitioning for GPU. In *Proceedings of the 53rd Annual Design Automation Conference (DAC '16)*. ACM, New York, NY, USA, Article 65, 6 pages. https://doi.org/10.1145/2897937.2898103

[42] Takuji Mitsuishi, Jun Suzuki, Yuki Hayashi, Masaki Kan, and Hideharu Amano. 2016. Breadth First Search on Cost-efficient Multi-GPU Systems. *SIGARCH Comput. Archit. News* 43, 4 (April 2016), 58–63. https://doi.org/10.1145/2927964.2927975

[43] Molly A. O'Neil and Martin Burtscher. 2014. Microarchitectural performance characterization of irregular GPU kernels. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC '14)*. IEEE, Raleigh, NC, USA, 130–139. https://doi.org/10.1109/IISWC.2014.6983052

[44] John Poulton, Robert Palmer, Andrew M. Fuller, Trey Greer, John Eyles, William J. Dally, and Mark Horowitz. 2007. A 14-mW 6.25-Gb/s Transceiver in 90-nm CMOS. *IEEE Journal of Solid-State Circuits* 42, 12 (Dec 2007), 2745–2757. https://doi.org/10.1109/JSSC.2007.908692

[45] John W. Poulton, William J. Dally, Xi Chen, John G. Eyles, Thomas H. Greer, Stephen G. Tell, John M. Wilson, and C. Thomas Gray. 2013. 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* 48, 12 (Dec 2013), 3206–3218. https://doi.org/10.1109/JSSC.2013.2279053

[46] Debendra D. Sharma. 2014. PCI Express 3.0 Features and Requirements Gathering for beyond. (2014). https://www.openfabrics.org/downloads/Media/Monterey_2011/Apr5_pcie%20gen3.pdf Accessed: 2016-06-20.

[47] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints* (Sept. 2014). arXiv:cs.CV/1409.1556

[48] Bruce W. Smith and Kazuaki Suzuki. 2007. *Microlithography: Science and Technology, Second Edition*. https://books.google.com/books?id=_hTLDCeIYxoC

[49] Jeff A. Stuart and John D. Owens. 2009. Message Passing on Data-parallel Architectures. In *Proceedings of the IEEE International Symposium on Parallel&Distributed Processing (IPDPS '09)*. IEEE, Washington, DC, USA, 1–12. https://doi.org/10.1109/IPDPS.2009.5161065

[50] Jeff A. Stuart and John D. Owens. 2011. Multi-GPU MapReduce on GPU Clusters. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS '11)*. IEEE, Washington, DC, USA, 1068–1079. https://doi.org/10.1109/IPDPS.2011.102

[51] David Tam, Reza Azimi, and Michael Stumm. 2007. Thread Clustering: Sharing-aware Scheduling on SMP-CMP-SMT Multiprocessors. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07)*. ACM, New York, NY, USA, 47–58. https://doi.org/10.1145/1272996.1273004

[52] Jin Wang, Norm Rubin, Albert Sidelnik, and Sudhakar Yalamanchili. 2016. LaPerm: Locality Aware Scheduler for Dynamic Parallelism on GPUs. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE, Piscataway, NJ, USA, 583–595. https://doi.org/10.1109/ISCA.2016.57

[53] Kenneth M. Wilson and Bob B. Aglietti. 2001. Dynamic Page Placement to Improve Locality in CC-NUMA Multiprocessors for TPC-C. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC '01)*. ACM, New York, NY, USA, 33–33. https://doi.org/10.1145/582034.582067