Evaluating the Effect of Last-Level Cache Sharing on Integrated GPU-CPU Systems with Heterogeneous Applications

Víctor García[1,2] Juan Gómez-Luna[3] Thomas Grass[1,2] Alejandro Rico[4] Eduard Ayguade[1,2] Antonio J. Peña[2]
[1]Universitat Politècnica de Catalunya [2]Barcelona Supercomputing Center [3]Universidad de Córdoba [4]ARM Inc.

Heterogeneous systems are ubiquitous in the field of High-Performance Computing (HPC). Graphics processing units (GPUs) are widely used as accelerators for their enormous computing potential and energy efficiency; furthermore, on-die integration of GPUs and general-purpose cores (CPUs) enables unified virtual address spaces and seamless sharing of data structures, improving programmability and softening the entry barrier for heterogeneous programming. Although on-die GPU integration seems to be the trend among the major microprocessor manufacturers, there are still many open questions regarding the architectural design of these systems. This paper is a step forward towards understanding the effect of on-chip resource sharing between GPU and CPU cores, and in particular, of the impact of last-level cache (LLC) sharing in heterogeneous computations. To this end, we analyze the behavior of a variety of heterogeneous GPU-CPU benchmarks on different cache configurations.

We perform an evaluation of the popular Rodinia benchmark suite modified to leverage the unified memory address space. We find such GPGPU workloads to be mostly insensitive to changes in the cache hierarchy due to the limited interaction and data sharing between GPU and CPU. We then evaluate a set of heterogeneous benchmarks specifically designed to take advantage of the fine-grained data sharing and low-overhead synchronization between GPU and CPU cores that these integrated architectures enable. We show how these algorithms are more sensitive to the design of the cache hierarchy, and find that when GPU and CPU share the LLC execution times are reduced by 25% on average, and energy-to-solution by over 20% for all benchmarks.

I. Introduction

Resource sharing within heterogeneous GPU-CPU systems has gotten some attention in the literature over the last few years. The ubiquity of these systems integrating GPU and CPU cores on the same die has prompted researchers to analyze the effect of sharing resources such as the last-level cache (LLC) [1], [2], the memory controller [3]–[5], or the network-on-chip [6]. Most of these works start with the premise that GPU and CPU applications exhibit different characteristics (spatial vs. temporal locality) and have different requirements (high bandwidth vs. low latency), and therefore careful management of the shared resources is necessary to guarantee fairness and maximize performance. To evaluate their proposals, the authors use multiprogrammed workloads composed of a mix of GPU and CPU applications running concurrently.

The tight integration of CPU and GPU cores enables a unified virtual address space and hardware-managed coherence, increasing programmer productivity by eliminating the need for explicit memory management. Devices can seamlessly share data structures and perform fine-grained synchronization via atomic operations. In this manner, algorithms can be divided in smaller steps that can be executed on the device they are best suited for (i.e., data parallel regions on the GPU or serial/low data parallelism regions on the CPU). Using multiprogrammed workloads to evaluate resource sharing can give insights into the challenges of GPU-CPU integration, but we believe it is not representative of future HPC applications that will fully leverage the capabilities of such systems.

This paper analyzes the impact of LLC sharing in heterogeneous computations where GPU and CPU collaborate to reach a solution, leveraging the unified virtual address space and low overhead synchronization. Our intent is to present guidelines for the cache configuration of future integrated architectures, as well as for applications to best benefit from these.

We first evaluate a set of kernels from the Rodinia benchmark suite modified to use the unified virtual address space, and show that on average sharing the LLC can reduce execution time by 13%, mostly due to better resource utilization. Since these benchmarks do not take full advantage of the characteristics of integrated heterogeneous systems, we then analyze a set of heterogeneous benchmarks that use cross-device atomics to synchronize and work together on shared data. We evaluate the effect of LLC sharing on these workloads and find it reduces execution times in average by 25% and up to 53% on the most sensitive benchmark. We also find reductions in energy-to-solution by more than 30% in 9 out of 11 benchmarks. To the best of our knowledge, this is the first effort analyzing the effect of cache sharing in integrated GPU-CPU systems with strongly cooperative heterogeneous applications. The key conclusions from this study are that LLC sharing:

- Has a limited, albeit positive, impact on traditional GPGPU workloads such as those from the Rodinia suite.
- Allows for faster communication between GPU and CPU cores and can greatly improve the performance of computations that rely on fine-grained synchronization.
- Can reduce memory access latencies by keeping shared data closer and avoiding coherency traffic.
- Improves resource utilization by allowing GPU and CPU cores to fully use the available cache.

The rest of the paper is organized as follows. Section II gives some background on heterogeneous computing and presents the motivation behind this work. Section III reviews related work. Section IV discusses the methodology we have used and the benchmarks analyzed. In Section V we present and discuss the experimental results obtained. Section VI presents the conclusions we extract from this work.

978-1-5090-3896-1/16/$31.00 ©2016 IEEE

II. Background and Motivation

This section serves as an overview of heterogeneous architectures and the memory consistency and cache coherence models followed by GPUs and CPUs. We describe the new computational paradigm these systems make possible and present the motivation that led to the evaluation we present in this work.

A. Heterogeneous Computing

GPUs and other accelerators have traditionally been discrete devices connected to a host machine by a high bandwidth interconnect such as PCI Express (PCIe). Host and device have different memory address spaces, and data must be copied back and forth between the two memory pools explicitly by the programmer. In this traditional model, the host allocates and copies the data to the device, launches a computation kernel, waits for its completion and copies back the result. This offloading model leverages the characteristics of the GPU to exploit data parallel regions, but overlooks the capabilities of the general-purpose CPU. Deeply pipelined out-of-order cores can exploit high instruction level parallelism (ILP), and are better suited for regions of code where GPUs typically struggle, such as those with high control flow or memory divergence.

On-die integration of GPU and CPU cores provides multiple benefits: a shared memory pool avoids explicit data movement and duplication; communication through the network-on-chip instead of a dedicated high bandwidth interconnect (PCIe) saves energy and decreases latency; reduced latency enables efficient fine-grained data sharing and synchronization, allowing programmers to distribute the computation between devices, each contributing by executing the code regions they are more efficient on. For further details we refer the reader to the literature, where the benefits of integrated heterogeneous architectures have already been explored [7]–[13].

In this work we focus on architectures used in the field of HPC. Most HPC systems are built out of hundreds or thousands of nodes, each composed of general purpose cores and specialized accelerators. In many cases these accelerators are GPUs, and the computation follows the master - worker scheme described above. While technically it can be considered heterogeneous computing, we believe the trend will continue to be tighter integration and better utilization of all computing resources. Figure 1 shows an example of a heterogeneous system integrating CPU cores and GPU Streaming Multiprocessors (SMs) on the same die. Two different LLC configurations are shown. Configuration a) has separate L3 caches for GPU and CPU; memory requests from one can only go to the other through the directory. Configuration b) has a unified L3 cache that both GPU and CPU can access directly and in equal condition. We evaluate the effect that sharing the LLC as in configuration b) has for heterogeneous computations where GPU and CPU collaborate and share data.

B. Heterogeneous Programming Models and Architectures

Heterogeneous System Architecture (HSA) [14], [15] is a multi-company effort led by companies such as AMD, ARM and Samsung to create an open standard for heterogeneous computing. The HSA specification allows for integrated GPU-CPU systems sharing data in a unified virtual memory space and synchronizing via locks and barriers like traditional multi-core processors. The OpenCL programming model [16] offers support for Shared Virtual Memory (SVM) since the 2.0 specification. On a system supporting SVM features, the same pointer can be used indistinctly by the CPU and GPU, and coherence is maintained by the hardware as in Symmetric Multi-core Processors (SMP). The OpenCL 2.0 specification provides a series of atomic operations that can be used to guarantee race-free code when sharing data through SVM [16]. These atomic operations allow for fine-grained synchronization among compute units, opening the door for heterogeneous applications that work on shared data structures and coordinate via low-overhead synchronization.

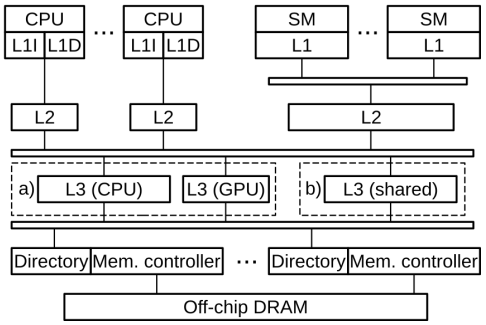

Fig. 1. Heterogeneous architecture with: a) Private LLCs b) Shared LLC.

Unified Virtual Addressing (UVA) was introduced in NVIDIA devices on CUDA 4 [17]. UVA allows the GPU to directly access pinned host memory through the high-bandwidth interconnect. This zero-copy memory allows application programmers to use the same pointers in the device and the host, improving programmability. However, on systems with discrete GPUs, the interconnect (usually PCI Express) is optimized for bandwidth and not latency, and therefore it is seldom useful to rely on zero-copy memory. Integrated systems such as the Tegra X1 [18] do not suffer from this problem, and zero-copy memory can be used to avoid data movement and replication. With CUDA 6 NVIDIA introduced Unified Memory (UM). UM builds on top of UVA to further simplify the task of GPU programming. On systems with discrete GPUs, UM has the CUDA driver move data on-demand to and from the device, removing the burden from the programmer. UM is widely believed to be the next step towards a fully unified coherent memory space between CPUs and GPU.

All the major vendors currently provide integrated heterogeneous systems fully supporting the capabilities described above. AMD has a number of Accelerated Processing Units (APUs) in the market with full system coherence between GPU and CPU, and is expected to continue the push for heterogeneous computing. Intel began integrating GPUs on-chip with their Sandy Bridge processors. Both companies' current products tightly integrate GPU and CPU and support the Shared Virtual Memory (SVM) features specified in OpenCL 2.0 [19], [20].

NVIDIA integrates general purpose ARM cores with their graphics units in their Tegra line of products [18]. Mobile and embedded chip vendors have been designing integrated system-on-chips (SoCs) for years now, and most current mobile chips integrate on the same die one or more general purpose CPU cores and a GPU [21], [22].

C. Memory Consistency and Cache Coherence

Memory consistency models guarantee a specified ordering of all load and store instructions. In general terms, the semantics of a strict consistency model simplify programmability at the cost of performance. Relaxed consistency models allow compilers and hardware to perform memory reordering, increasing performance. This complicates the task of the programmer, since memory instructions may need to be protected with fences or operated on with atomics.

The x86 ISA follows a variant of Sequential Consistency (SC) [23] called Total Store Order (TSO) [24]. In this strict model, loads following a store (in program order) can be executed before the store if they are to a different memory address. Although there is not much public information describing the memory consistency models followed by GPUs from the major vendors, they have been largely inferred to be relaxed models. One of such models is Release Consistency (RC) [25]. RC enables many memory optimizations that maximize throughput, but is strict enough to allow programmers to reason about data race conditions. RC is the consistency model defined in the HSA standard [14], and it is followed in GPUs by vendors such as ARM [26] and AMD [27].

Cache coherence protocols are pivotal to maintaining memory consistency within a system. They guarantee that all sharers of a datum always obtain the latest value written and not stale data. Regardless of the protocol itself (i.e. MESI, MOESI, etc.), x86-based SMPs follow the coherence model Read For Ownership (RFO). In an RFO machine, cores must obtain a block in an exclusive state before writing to it. This scheme is effective for workloads that exhibit temporal locality and data reuse, where the cost of exclusively requesting blocks and the associated invalidations is amortized over time. GPUs have traditionally exhibited a different memory access behavior, streaming through data with little data reuse. In addition, the high memory traffic generated by the large number of threads running concurrently exerts a high pressure in the memory subsystem, and any additional coherence traffic would only aggravate the problem. Because of this, GPUs implement very simple coherence mechanisms with private write-through write-combining L1 caches that can contain stale data [17], [28].

On the other hand, recent work shows that that the choice of consistency model minimally impacts the performance of GPUs [27]. While stricter consistency models and system coherence does not come for free, researchers are already working on solutions to solve the challenges faced [29]. We believe integrated systems will change the way we understand heterogeneous programming and change the characteristics of heterogeneous workloads. Stricter consistency models across a heterogeneous system will improve programmability and allow programmers to maintain the memory semantics they are used to on traditional SMPs. Therefore, in this work we perform our evaluation of heterogeneous computations on a system that implements a TSO consistency model with RFO coherence across all computing elements. For future work we plan to also evaluate different consistency models and measure their impact.

III. Related Work

Lee and Kim analyzed the impact of LLC sharing between GPU and CPU [1]. They find that the multi-threaded nature of GPUs allows them to hide large off-chip latencies by switching to different threads on memory stalls. In addition, the data-streaming nature of GPU workloads shows little data reuse. Therefore they conclude that caching is barely useful for such workloads, and argue that cache management policies in heterogeneous systems should take this into consideration. They propose TAP, a cache management policy that detects when caching is beneficial for the GPU application, and favors CPU usage of the LLC when it is not. Although the policy could be useful even on the heterogeneous computations we are evaluating, their premise of cache insensitivity on GPUs does not hold for computations where fine grained synchronization is performed between compute units, as we will show in our evaluation. Mekkat et al. build on the same premise [2]. They use set dueling [30] to measure CPU and GPU sensitivity to caching during time intervals. With this information, they dynamically set a thread-level parallelism (TLP) threshold for each interval. The threshold determines after what amount of TLP the GPU's memory requests start bypassing the LLC. The goal again is to prevent the GPU from taking over most of the LLC space and depriving the cache-sensitive CPU of it.

Other works have explored the challenges of resource sharing within GPU-CPU systems. Ausavarungnirun et al. focus their study on the memory controller [4]. They find the high memory traffic generated by the GPU can interfere with the requests from the CPU, violating fairness and reducing performance. They propose a new application-aware memory scheduling scheme that can efficiently serve both the bursty, bandwidth-intensive GPU workloads and the time-sensitive CPU requests. Kayiran et al. consider the effects on the network-on-chip and memory controller [5]. They monitor memory system congestion and if necessary limit the amount of concurrency the GPU is allowed. By reducing the amount of active warps in the GPU, they are able to improve CPU performance in the presence of GPU-CPU interference.

All these works analyze resource sharing within integrated GPU-CPU systems, but they perform their evaluation on multi-programmed workloads where GPU and CPU are executing different unrelated benchmarks. This methodology can shed light on some of the problems associated with resource sharing in heterogeneous architectures, but it is not able to provide any insight about the effect such sharing has in heterogeneous computations where GPU and CPU cores collaborate and share data. The goal of our work is to analyze how these heterogeneous algorithms are affected by sharing the LLC.

IV. Methodology

In this section we present the methodology we follow in our work. First, we discuss how we simulate our different target hardware configurations. Next, we introduce the set of benchmarks we leverage for our evaluation in Section V.

A. Hardware Simulation

We use gem5-gpu [31] to simulate an integrated heterogeneous GPU-CPU system. gem5-gpu is a cycle-level simulator that merges gem5 [32] and GPGPU-Sim [33]. We use gem5's full-system mode running the Linux operating system. The simulator provides full system coherence between GPU and CPU using the MESI coherence protocol and follows the TSO consistency model. For more details regarding the rationale behind our choice of memory model, see Section II-C. Table I lists the configuration parameters of the system. We simulate a four core CPU and an integrated GPU composed of four Fermi-like Streaming Multiprocessors (SMs) grouped in two clusters of two. Considering NVIDIA Tegra X1 is composed of two SMs [18], this configuration is our best guess as to how the next generation of heterogeneous systems will be.

Figure 1 shows the topology of the simulated system. The CPU's L1 and L2 caches are private per CPU core. Each GPU SM has a private L1, connected through a crossbar to the shared L2, which is itself attached to the global crossbar. In configuration a) the system has two L3 caches private to GPU and CPU; configuration b) shows a unified L3 that can be used by both. The LLC size listed in Table I refers to the shared configuration; see Section V for details on how it is split for the private LLC configuration. In both cases, the LLC(s) run in the same clock domain as the CPU. This allows us to present a fair comparison by setting the same access latency on both configurations, albeit providing a conservative estimation of the benefits of LLC sharing. All caches are write-back and inclusive with a least recently used (LRU) replacement policy; the cache line size is 128 bytes. The network-on-chip (NoC) is modeled with gem5's detailed Garnet model [34]. Flit size is 16 bytes for all links; data message size is equal to the cache line size and fits within 9 flits (1 header + 8 payload flits); control messages fit within 1 flit. In order to focus on the interactions between GPU and CPU we select a region of interest for all benchmarks, skipping initialization (memory allocation, input file reading, etc.) and clean-up phases. The results shown in Section V correspond only to this region. The power results from section V-B were obtained with CACTI version 6.5 [35] configured with the parameters shown in Table I.

TABLE I
Simulation Parameters

CPU	
Cores	4 @ 2 Ghz
L1D Cache	64kB - 4 way - 1ns lat.
L1I Cache	32kB - 4 way - 1ns lat.
L2 Cache	512kB - 8 way - 4ns lat.
GPU	
SMs	4 - 32 lanes per SM @ 1.4 Ghz
L1 Cache	16kB + 48kB shared mem. - 4 way - 22ns lat.
L2 Cache	512kB - 16 way - 4 slices - 63ns lat.
LLC and DRAM	
LLC	8MB - 4 banks - 32 way - 10ns lat.
DRAM	4 channels - 2 ranks - 16 banks @ 1200 MHz
RAS/RCD/CL/RP	32 / 14.16 / 14.16 / 14.16 ns
RRD/CCD/WR/WTR	4.9 / 5 / 15 / 5ns

B. Benchmarks

We evaluate the effect of LLC sharing in two sets of heterogeneous benchmarks. First we look at Rodinia GPU [36], a well-known benchmark suite used to evaluate GPUs. Initially designed to run on discrete GPUs, the benchmarks explicitly copy data to and from the device. We use a modified version of the kernels provided by the gem5-gpu developers that removes data movement and makes use of pointers, leveraging the shared address space. The benchmarks analyzed are listed in Table II.

However, Rodinia benchmarks have little interaction between CPU and GPU, as they were developed for heterogeneous systems with discrete GPUs, and a general recommendation for this kind of platforms is avoiding memory transfers between CPU and GPU. Another possible set of benchmarks to perform the kind of evaluation we are interested in could be the Valar [37] benchmark suite. Valar is a set of benchmarks for heterogeneous systems that focuses on the interaction of CPU and GPU cores. However, the benchmarks are designed for old AMD APUs that lack new characteristics such as memory coherence and cross-device atomics. We therefore need a set of heterogeneous benchmarks that allows us to explore the possibilities and challenges of new features such as GPU-CPU memory coherence and cross-device atomic operations. To this end, we have prepared our own collection of benchmarks that exploits the most recent features in heterogeneous integrated systems. They present different heterogeneous computation patterns and are summarized in Table III.

Four benchmarks (DSP, DSC, IH, and PTTWAC) deploy concurrent CPU-GPU collaboration patterns. In these benchmarks, the input workload is dynamically distributed among CPU threads and GPU blocks. DSP and DSC utilize an adjacent synchronization scheme, which allows CPU threads and/or GPU blocks working on adjacent input data chunks to synchronize. Each CPU thread or GPU block has an associated flag that is read and written atomically with cross-device atomics. Both DSP and DSC are essentially memory-bound algorithms, as they perform data shifting in memory. DSC deploys reduction and prefix-sum operations, in order to calculate the output position of the compacted elements. IH carries out an intensive use of atomic operations on a set of common memory locations (i.e., a histogram). Chunks of image pixels are statically assigned in a cyclic manner to CPU threads and GPU blocks. These update the histogram bins atomically using cross-device atomic additions. PTTWAC performs a partial transposition of a matrix. It works in-place; thus, each matrix element has to be saved (to avoid overwriting) and then shifted to the output location. As each of these elements is assigned to a CPU thread

TABLE II
RODINIA BENCHMARKS

Benchmark	Short Name	Dataset
Backprop	RBP	256K nodes
Breadth-First Search	RBF	256K nodes
Gaussian	RGA	512 × 512 matrix
Hotspot	RHP	512 × 512 data points
LavaMD	RLA	10 boxes per dimension
LUD	RLU	2K × 2K matrix
NN	RNN	1024K data points
NW	RNW	8K × 8K data points
Particlefilter	RPF	10K particles
Pathfinder	RPA	100K × 10K data points
Srad	RSR	512 × 512 data points

or a GPU block, these need to coordinate through a set of atomically updated flags.

In BFS the computation switches between CPU threads and GPU blocks in a coarse-grain manner. Depending on the amount of work of each iteration of the algorithm, CPU threads or GPU blocks are chosen. CPU and GPU threads share global queues in shared virtual memory. At the end of each iteration, they are globally synchronized using cross-device atomics. LCAS and UCAS are two kernels from the same AMD SDK sample. First, a CPU thread creates an array which represents a linked list to hold IDs of all GPU threads. Then, in the first kernel (LCAS) each GPU thread inserts in lock-free manner their respective IDs into the linked list using atomic compare-and-swap (CAS). In the second kernel (UCAS) the GPU threads unlink or delete them one-by-one atomically using CAS. RANSAC implements a fine-grain switching scheme of this iterative method. One CPU thread computes a mathematical model for each iteration, which is later evaluated by one GPU block. As iterations are independent, several threads and blocks are working at the same time. TQ is a dynamic task queue system, where the work to be processed by the GPU is dynamically identified by the CPU. Several queues are allocated in shared virtual memory. CPU threads and GPU blocks access them by atomically updating three variables per queue that represent the number of enqueued tasks, the number of consumed tasks, and the current number of tasks in the queue. The applicability of this task queue system is illustrated by a real-world kernel: histogram calculation of frames from a video sequence.

V. EVALUATION

We evaluate the effect of sharing the last-level cache by running a set of heterogeneous benchmarks with two cache configurations. For the private LLC configuration we split the cache by giving 1/8 to the GPU and 7/8 to the CPU. We follow current products from Intel and AMD where the ratio of GPU-to-CPU cache size is between 1/8 and 1/16 [28], [44]. We evaluated different split ratios from 1/2 to 1/16 and saw similar trends among them. Unfortunately, splitting the LLC in this manner and directly comparing the results would not provide a fair evaluation. The additional cache space available to both CPU and GPU cores in the shared configuration may affect the results if the benchmarks are cache sensitive. To isolate the gains that are caused by faster communication and synchronization from those that are due to better utilization of the available cache space, we run all the benchmarks with an extremely large, 32-way associative LLC of 1 GB total aggregate size. Under this configuration the working set of most benchmarks fits in the private LLC, and therefore the gains cannot be attributable to the extra cache space.

A. Rodinia

We first analyze the Rodinia benchmark suite. As stated in Section IV-B, these benchmarks have minimal interaction between GPU and CPU. The one interaction all benchmarks share is in the allocation and initialization of data by the host prior to launching the computation kernel(s). Therefore, on a shared LLC configuration, if the working set of the benchmark fits within the LLC, GPU memory requests will hit in the LLC and avoid an extra hop to the CPU's private LLC with the corresponding coherence traffic.

Figure 2 shows speedup on the shared LLC configuration normalized to private LLCs. Of the 11 benchmarks, 7 show a speedup of over 10% with an 8MB LLC. Of those, RBF and RLU lose all the speedup with a 1GB cache; we can therefore attribute the gains to better resource utilization when sharing the LLC. RBF has a significant amount of branch and memory divergence and is largely constrained by global memory accesses [45]. Our results confirm this and show that the kernel benefits from caching due to data reuse. On the 1GB configuration the GPU is able to fit the whole working set in its private cache hierarchy; since there is no further GPU-CPU interaction after the initial loading of data, there is no performance benefit by sharing the LLC. We also observe a similar behavior in RLU.

RBP, RPA and RSR speedup is also reduced on the 1GB configuration, but still obtain 13%, 33% and 13% improvement respectively. RSR features a loop in the host code calling the GPU kernels a number of iterations. After each iteration, the CPU performs a reduction with the result matrix, and therefore the benchmark benefits from faster GPU-CPU communication. The speedup is reduced on the 1GB configuration because there is data reuse within the two GPU kernels, and the larger private LLC allows more data to be kept on-chip. On RBP the CPU performs computations on shared data before and after the GPU kernel; the benefit of sharing the LLC is two-fold: the GPU finds the data in the shared LLC at the start of the kernel, and the CPU obtains the result faster by avoiding an extra hop to the private GPU LLC. RPA sees the largest performance improvement although there is no further GPU-CPU communication past the initial loading of data; the gains are thus attributable to the GPU finding the data in the shared LLC at the start of the kernel. Both these benchmarks see non-negligible performance gains despite the limited GPU-CPU interaction. The reason is that the total execution time for both benchmarks is low, and the effect of the initial hits on the host-allocated data is magnified. We chose a small input set in order

TABLE III
SUMMARY OF HETEROGENEOUS BENCHMARKS

Benchmark	Short Name	Field	Computation Pattern	Dataset
Breadth-First Search [38]	BFS	Graphs	Coarse-grain switching	NY/NE graphs [39]
DS Padding [40]	DSP	Data manipulation	Concurrent collaboration	2K × 2K× 256 float
DS Stream Compaction [40]	DSC	Data manipulation	Concurrent collaboration	1M float
FineGrainSVMCAS link [41]	LCAS	Synthetic benchmark	Fine-grain linked list	4K elements
FineGrainSVMCAS unlink [41]	UCAS	Synthetic benchmark	Fine-grain linked list	4K elements
Image Histogram [42]	IH	Image processing	Concurrent collaboration	Random and natural images (1.5M pixels, 256 bins)
PTTWAC Transposition [38]	PTTWAC	Data manipulation	Concurrent collaboration	197 × 35588 doubles (tile size = 128)
Random Sample Consensus [43]	RANSAC	Image processing	Fine-grain switching	5922 input vectors
Task Queue System [38]	TQ	Work queue	Producer-consumer	128 frames

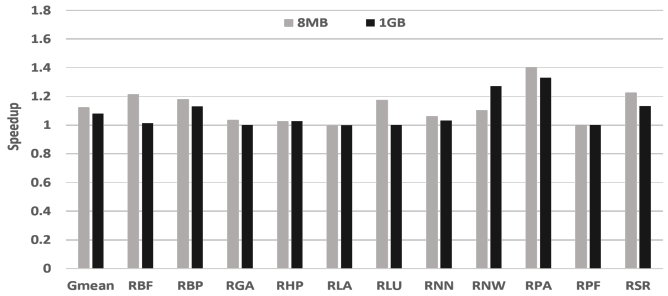

Fig. 2. Speedup of Rodinia benchmarks with a shared LLC normalized to a private LLC configuration.

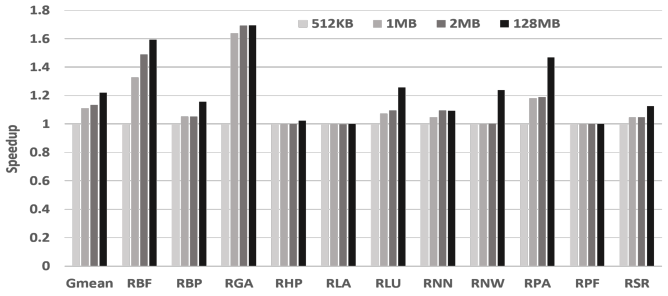

Fig. 3. Cache sensitivity of Rodinia benchmarks. Each column corresponds to the GPU LLC size in a configuration with private caches. Results are normalized to GPU LLC of size 512 KB.

to run our simulations within a reasonable time-frame. On large computations this benefit would be diminished over time, hence on a real hardware with larger input sets, it is likely the gains would be minimal.

RNW is the only benchmark where the performance gain of sharing the LLC actually goes up to 27% when increasing the LLC size to 1GB. The reason is the large input set used, with a heap usage of 512MB. The GPU LLC on the private configuration is not large enough to hold all the data; there is data reuse within the kernel, but due to the large working set, it is evicted out of the LLC before it is reused. On the shared configuration, the GPU benefits both from finding the data in the LLC and from being able to keep it there for reuse. In addition, after the GPU kernel completes, the CPU reads back the result matrix, benefiting from faster communication.

RNN experiences a small gain from sharing the LLC because it features some GPU-CPU interaction beyond the initial loading of data. When the GPU finishes computing distances, the CPU reads the final distance vector and searches for the closest value. The benchmark also benefits from the extra cache space, and thus the gains are reduced on the 1GB configuration where the 12MB input set fits in the LLC.

RGA, RHP, RLA, and RPF do not benefit from sharing the LLC. RGA launches a kernel many times to operate on two matrices and a vector. The benchmark benefits from caching due to all the data reuse, but once the matrices and vector are loaded, there is no further interaction with the CPU until the kernel completes. Although the CPU then reads the data and performs a final computation, this is just a small portion of the execution time and thus the benefit is minimal. In RLA, the kernel is optimized to access contiguous memory locations, allowing the GPU to coalesce a large amount of memory accesses and reduce the total memory traffic pushed into the cache hierarchy. This memory access pattern and a high data reuse produces close to 99% cache hit rate in the GPU L1 caches despite an input set size of 8MB. As a consequence, sharing the LLC provides no benefit beyond the initial loading of data. A similar behavior can be observed in RPF, where the small memory footprint of the kernel allows data to fit within the GPU L1 and L2 caches. RHP performs multiple iterations operating over the same three matrices, showing data reuse with a large reuse distance. With a 1GB LLC the whole working set is able to fit in the cache, but the kernel is mostly cache insensitive and gains little from the increased hit rate. There is no GPU-CPU communication, and the small benefit of initially hitting in the LLC is diminished over the total execution time.

These results show that sharing the LLC does not provide a significant benefit for computations such as the ones found in the Rodinia benchmark suite, with minimal GPU-CPU interaction and data sharing only at kernel boundaries. The geometric mean speedup for all benchmarks is 9% on the 1GB configuration and 13% on the 8MB configuration, and it is mostly due to the extra cache space available to the GPU. In order to measure the sensitivity of the benchmarks to GPU cache size, we run them with different LLC sizes of 4MB, 8MB, 16MB and 1GB on the private LLC configuration. Following the 1/8 ratio of GPU to CPU LLC, the GPU obtains 512KB, 1MB, 2MB, and 128MB respectively. We keep the same access latencies for all configurations in order to provide a fair comparison. Figure 3 shows speedup as we increase cache size, normalized to the 512KB GPU LLC. Confirming our previous findings, we see that RBF, RGA, RLU, RNW and RPA show sensitivity to cache size, obtaining over 20% performance increase with an ideal 128MB cache. RGA and

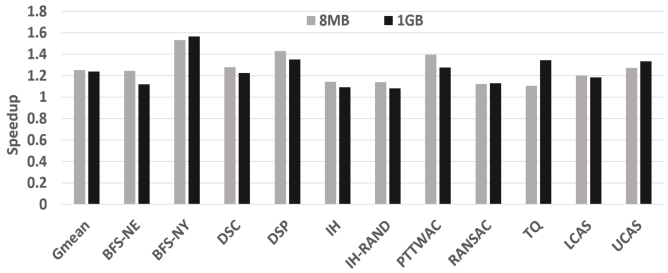

Fig. 4. Speedup of heterogeneous benchmarks with a shared LLC.

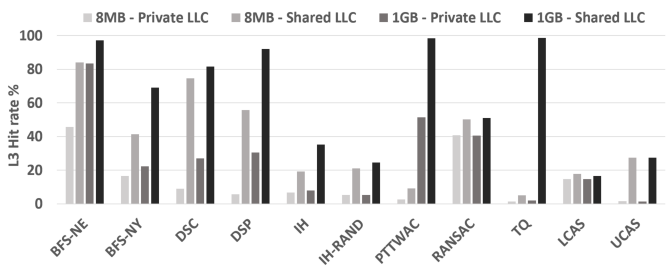

Fig. 5. LLC hit rates for private and shared configuration.

RBF show very high cache sensitivity, with up to 69% and 49% improvement respectively with a more realistic 2MB cache. RLA and RPF, as discussed earlier, make almost no use of the LLC and therefore do not benefit from a larger cache. RHP shows data reuse and sees minor gains with a 128MB LLC that is able to fit the whole working set; with a smaller LLC the number of cache misses increases, but the GPU is able to hide the extra latency and the benchmark is thus mostly cache insensitive. RBP and RSR show some sensitivity to cache size, confirming the loss of speedup shown in Figure 2 is due to the increased LLC size. RNN sees some improvements up to the 2MB configuration, after which the working set fits within the 16MB cache hierarchy. Although the 2MB of GPU LLC are not enough to hold all the data, the kernel features no data reuse and does not benefit from a larger LLC.

B. Heterogeneous Benchmarks

We run the heterogeneous benchmarks with two CPU worker threads, with the exception of LCAS and UCAS which use only one. As in Section V-A, we run the benchmarks also with an ideal 1GB LLC to isolate the gains that come from the additional cache space available on the shared configuration.

Figure 4 shows the speedup obtained with a shared LLC over the private LLC configuration. Of the 11 benchmarks, 6 show improvements of over 20% with a shared LLC. For BFS we choose two different input graphs; the smaller *NY* graph has variable amount of work per iteration, switching often between GPU and CPU computation. The larger *NE* graph has many iterations with a large amount of work, and therefore executes mostly in the GPU, switching less often between GPU and CPU. The performance gain for BFS-NY is higher than for BFS-NE, achieving as much as a 56% speedup on the 1GB configuration. This is reduced to 11% on BFS-NE with the 1GB LLC because with limited GPU-CPU communication, the benefit comes mostly from additional cache space.

In order to understand how sharing the LLC affects cache hit rates, we show in Figure 5 the L3 hit rates on both the private and shared configurations. With private LLCs we calculate the aggregated L3 hit rate as: $(Hits_{CPU} + Hits_{GPU})/(Access_{CPU} + Access_{GPU}) * 100$. We see that BFS-NE obtains 39% more hits in the private LLC configuration by increasing the size from 8MB to 1GB. The computation is mostly performed by the GPU, where only 1MB of LLC is available on the private 8MB configuration. Being able to use the remaining 7MB provides a significant gain.

The speedup observed in BFS-NY supports the relevance of fast GPU-CPU communication on workloads making an extensive use of atomic synchronizations. In gem5-gpu atomics are implemented with read-modify-write (RMW) instructions. A RMW instruction is composed of two parts, an initial load (LD) of the cache block with exclusive state, and a following write (WR) with the new value. Once a thread successfully completes the LD, no other thread can modify that memory location until the WR is finished, guaranteeing atomicity.

Figure 6 shows the average access time to perform the LD part of the RMW on a shared LLC configuration, normalized to private LLCs. We can see how BFS performs the operation 40% and 45% faster with a shared LLC for the NE and NY graphs respectively. On the other hand, DSC and DSP perform slower RMW LDs with a shared LLC, and nevertheless show speedups of 27% and 42% respectively. The average time to perform the RMW LD is higher because there are more L1 misses when the exclusive LD is attempted. This is a side-effect of the faster GPU-CPU communication. GPU and CPU cores compete for the cache blocks holding the array of synchronization flags, invalidating each other. The shorter the latency to reach the current owner of the block, the more likely it is for a core to have relinquished ownership of the block by the time it is reused. The reason the benchmarks obtain a speedup with a shared LLC is that the overall memory access time for all accesses is lower. In particular for the GPU, the average latency for all the threads of a warp to complete a load instruction is 65% and 40% lower for DSC and DSP on the 8MB configuration. Figure 6 shows how both benchmarks go from lower than 10% LLC hit rates with a private configuration to above 80% when sharing the LLC. The benchmarks are memory bound and the reduced memory latency caused by hitting in the shared LLC compensates the higher miss rate when performing the atomics.

Figure 7 shows the average GPU and CPU instructions per cycle (IPC) with a shared LLC configuration normalized to private LLCs. DSC and DSP achieve up to 30% and 47% higher GPU IPC by sharing the LLC. The more latency-sensitive CPU cores see a large increase of up to 49% on the 1GB configuration when the whole working set fits in the cache.

IH performs a histogram on an input image. We configure the benchmark with 256 bins, which fit within 9 cache blocks

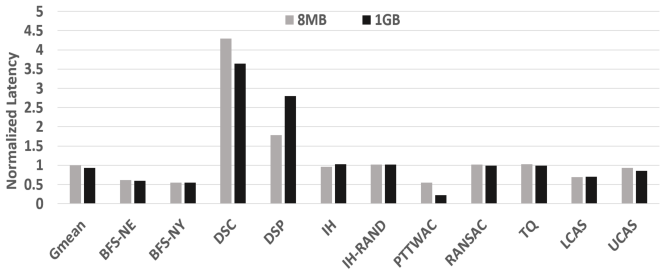

Fig. 6. Average latency to perform a RMW LD normalized to private LLC.

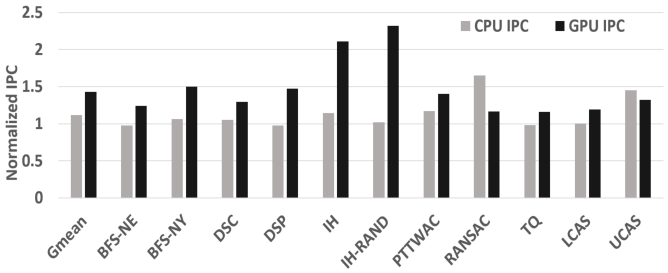

Fig. 7. Normalized IPC with a shared 8MB LLC.

(8 if aligned to block size); our intuition was that these blocks would be highly contended and the benchmark would benefit from faster atomic operations. Interestingly we see only a relatively small speedup of 14% and 8% on the 8MB and 1GB configurations, respectively. In Figure 6 we see that sharing the LLC does not reduce the time to perform a RMW LD. The speedup is small because in the end, the CPU is the bottleneck. The GPU benefits from multiple bins falling on the same cache line, as threads from the same warp can increment multiple bins in a fast manner. That, on the other hand, causes false sharing on the CPU caches. The work is statically partitioned, so the GPU completes its part while the CPU takes 10x longer to finish 1/8 of the image. We observe how the GPU does indeed benefit from sharing the LLC; the average latency for all the threads of a warp to finish a LD operation is reduced by 63% and 59% with a shared LLC. After the GPU finishes computing its part, the CPU remains computing, and eventually all the lines with the bins are loaded into the CPU caches, obtaining no benefit from the shared LLC. Figure 7 clearly depicts this; the IPC of the GPU increases over 2x on the 8MB configuration, while the CPU sees barely any improvement.

One of the consequences of using an image as the input is that we observe less conflict than expected for the cache blocks holding the bins. Images usually have similar adjacent pixels, and it is likely that after obtaining a block in exclusive to perform the atomic increment, the following pixels require incrementing a bin in the same block. In order to evaluate the shared LLC with a different memory access pattern, we also run the benchmark with a randomized pixel distribution (IH-RAND). This input reduces the number of RMW LD hits, indicating there is more contention for the lines holding the bins. Ultimately, however, the number of cache hits reduced is low, as with 32 out of 256 bins per cache block it is still likely that the next atomic increment falls on a bin in the same block.

PTTWAC performs a partial matrix transposition in-place. The input matrix requires 53 MB of memory, hence not fitting in the cache hierarchy on the 8MB configuration. Sharing an 8MB LLC with such a large input barely increases L3 hit rate, but provides a 39% speedup. Figure 6 shows that this is due to a significant reduction on the average latency to perform the RMW LD. On the 1GB configuration the latency reduction is even larger, but the speedup is down to 27%. In this case the cache is large enough to fit the matrix, and the benchmark only benefits from faster atomics.

In RANSAC the CPU first performs a fitting stage on a sample of random vectors. When finished, it signals the GPU to proceed with the evaluation stage where all the outliers are calculated. This process is repeated until a convergence threshold is reached. We see that sharing the LLC improves performance by 12% for both cache sizes without speeding the atomic operations. Both GPU and CPU threads spin reading the synchronization flag when their counterpart is computing; therefore there is no contention for the block once it is read. The hit rate when performing a RMW LD is 72% for both shared and private LLC configurations, and thus the average access time is already low. The speedup in this case is not produced by faster synchronization, but from sharing the vector array. The memory footprint of the array is small, increasing L3 hit rate only by a 10%. Nevertheless, Figure 7 shows this 10% has a large effect on the latency-sensitive CPU, that achieves over 60% higher IPC with a shared LLC. The GPU also finds the vector array in the LLC on the first iteration, and gains a more modest 17% IPC. The total execution time of this benchmark is low, and thus the impact of initially hitting in the LLC is magnified. As with Rodinia, on a longer executing benchmark the gain would diminish.

LCAS uses a CPU thread to traverse half of a linked list while the GPU threads traverse the other half, inserting in each position an identifier and atomically updating the head of the list. The cache block containing the head is highly contended, causing the atomic operations to be the bottleneck. UCAS traverses the list resetting the identifiers to 0. The difference lies in the order they access the elements. Although the data structure containing the identifiers is conceptually a linked list, it is implemented as an array where the first position contains the array index of the next element. On LCAS the CPU inserts identifiers in consecutive array positions and GPU threads update the array position matching their thread identification number. Hence, threads from the same warp update contiguous positions. On UCAS the order in which the elements are accessed is the reverse order in which the linked list was updated on LCAS, i.e. the reverse order in which the threads were able to perform the atomic operation. This difference causes the observed speedup variation. On UCAS the scattered access pattern causes many blocks to be moved back and forth between GPU and CPU, and is reflected by the near 0% L3 hit rate seen in Figure 5. In this case the data migration latency from GPU to CPU is also an important factor.

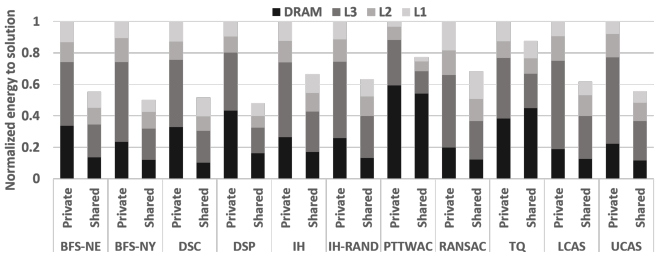

Fig. 8. Energy-to-solution normalized to private LLC.

The results show that although UCAS achieves on average a lower latency reduction to perform a RMW LD with a shared LLC, the benefit of faster data movement actually renders a higher speedup compared to LCAS.

In TQ the CPU is in charge of inserting 128 frames in several queues. GPU blocks dequeue individual frames and generate their histogram. As the histogram of each frame is only accessed by one GPU block, it will be kept in L1 ensuring low latency for RMW operations already on the private LLC configuration. Additionally, the number of atomics on the more contended control variables of the queues is very small compared to the amount of atomic operations on the histograms. Thus, the average latency to perform the RMW is not reduced by sharing the LLC. However, the LLC hit rate is higher in the shared configuration, because the GPU blocks will eventually read frames previously cached by the CPU thread, when enqueuing them. This explains the 10% speedup on the 8MB configuration. The improvement is much higher on the 1GB configuration (34%) because the larger cache can keep the entire pool of 128 frames (54MB) and the queues.

Figure 8 shows energy-to-solution with an 8MB LLC normalized to private LLCs. All benchmarks show that sharing the LLC decreases energy-to-solution, over 30% on 9 of the 11. For all benchmarks static power is reduced due to shorter execution times. The other major reduction comes from lower L3 dynamic power. A shared LLC increases hit rates and avoids the extra requests and coherence traffic caused by a cache miss. This can lead to a significant reduction as in BFS, RSC, RANSAC, LCAS and UCAS. The third reduction comes from DRAM dynamic power. The results we present in this section are of the benchmark's region of interest; the data has already been allocated an initialized; in most cases the data is already on-chip, therefore the total number of off-chip accesses is already low. Sharing the LLC improves resource utilization and allows for data to stay longer in the hierarchy, reducing even further off-chip traffic. The exceptions are PTTWAC and TQ. Both benchmarks have a working set size far larger than the cache hierarchy, and must still load data from DRAM. The shared LLC minimally reduces this in PTTWAC, and slightly increases it in TQ. The reason is that the frames sometimes evict the queues from the shared LLC, causing off-chip write-backs and subsequent reloads. On the private configuration the queues are able to stay in the CPU's LLC.

VI. Conclusions

This paper was motivated by the lack of efforts focusing on the effects of resource sharing in heterogeneous computations. We believe the tighter integration of CPU cores with GPUs and other accelerators will change the way we understand heterogeneous computing in the same way the advent of multi-core processors changed how we think about algorithms.

Our results show that the Rodinia benchmark suite with coarse-grained GPU-CPU communication experiences an average 13% speedup using an 8MB shared LLC versus a private LLC configuration. We have shown that the gain is mainly due to the extra cache space available to the GPU or due to short execution times. We have additionally analyzed a set of heterogeneous benchmarks featuring stronger CPU and GPU collaboration. These show that computations that leverage the shared virtual address space and fine-grained synchronization achieve an average speedup of 25% and of up to 53% with an 8MB shared LLC. In addition, energy-to-solution is reduced for all benchmarks due to lower static power and L3 and DRAM dynamic power consumption.

Our results indicate that sharing the LLC in an integrated GPU-CPU system is desirable for heterogeneous computations. The first benefit we observed is due to the faster synchronization between GPU and CPU. In applications where fine-grained synchronization via atomic operations is used and many actors contend to perform the atomics, accelerating this operation provides considerable speedups. The second benefit is due to data sharing; if GPU and CPU operate on shared data structures, sharing the LLC will often reduce average memory access time and dynamic power. We have observed this effect both with read-only and exclusive read-write data. The third benefit we observed is due to better utilization of on-chip resources. A cache hierarchy where the LLC is partitioned will often underuse the cache space available, while sharing it guarantees full utilization if needed.

Nevertheless, resource sharing between such disparate computing devices entails additional problems. We have seen an increase of conflict misses specially with large input sets. In these cases the benefits of sharing the LLC offsets the drawbacks. However, we are only focusing on computations that fully leverage the characteristics of integrated heterogeneous architectures. In the last few years researchers have shown and proposed solutions for the challenges of resource sharing with other types of workloads, and further investigation is required if the trend of GPU-CPU integration is to continue.

Summing up, the benefits we have listed encourage a rethinking of heterogeneous computing. In an integrated heterogeneous system, computation can be divided into smaller chunks; each chunk can be executed on the computing device that is best suited for, seamlessly sharing data structures between compute units and synchronizing via fine-grained atomic operations. Sharing on-chip resources such as the last-level cache can provide performance gains if the algorithms fully leverage the capabilities of these integrated systems.

ACKNOWLEDGMENTS

This work has been supported by the Spanish Ministry of Science and Innovation (contract TIN2015-65316-P) and by the BSC/UPC NVIDIA GPU Center of Excellence.

REFERENCES

[1] J. Lee and H. Kim, "Tap: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture," in *IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA)*, 2012.

[2] V. Mekkat, A. Holey, P.-C. Yew, and A. Zhai, "Managing shared last-level cache in a heterogeneous multicore processor," in *International Conference on Parallel Architectures and Compilation Techniques*, 2013.

[3] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver, "A QoS-aware memory controller for dynamically balancing GPU and CPU bandwidth use in an MPSoC," in *Design Automation Conference (DAC)*, 2012, pp. 850–855.

[4] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems," in *39th International Symposium on Computer Architecture (ISCA)*, 2012, pp. 416–427.

[5] O. Kayiran, N. C. Nachiappan, A. Jog, R. Ausavarungnirun, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, "Managing GPU concurrency in heterogeneous architectures," in *47th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014, pp. 114–126.

[6] J. Lee, S. Li, H. Kim, and S. Yalamanchili, "Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 4, pp. 48:1–48:28, 2013.

[7] M. Daga, A. M. Aji, and W. Feng, "On the efficacy of a fused CPU+GPU processor (or APU) for parallel computing," in *Symposium on Application Accelerators in High-Performance Computing*, 2011.

[8] M. Daga and M. Nutter, "Exploiting coarse-grained parallelism in B+ tree searches on an APU," in *SC Companion: High Performance Computing, Networking Storage and Analysis (SCC)*, 2012.

[9] M. Daga, M. Nutter, and M. Meswani, "Efficient breadth-first search on a heterogeneous processor," in *IEEE International Conference on Big Data*, Oct. 2014, pp. 373–382.

[10] M. C. Delorme, T. S. Abdelrahman, and C. Zhao, "Parallel radix sort on the amd fusion accelerated processing unit," in *42nd International Conference on Parallel Processing*, Oct. 2013, pp. 339–348.

[11] J. He, M. Lu, and B. He, "Revisiting co-processing for hash joins on the coupled CPU-GPU architecture," *Proc. VLDB Endow.*, vol. 6, no. 10, pp. 889–900, Aug. 2013.

[12] J. Hestness, S. W. Keckler, and D. A. Wood, "GPU computing pipeline inefficiencies and optimization opportunities in heterogeneous CPU-GPU processors," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2015, pp. 87–97.

[13] W. Liu and B. Vinter, "Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors," *Parallel Comput.*, vol. 49, no. C, pp. 179–193, Nov. 2015.

[14] *Heterogeneous System Architecture: A Technical Review*, AMD, 2012. [Online]. Available: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/hsa10.pdf

[15] W. Hwu, *Heterogeneous system architecture*. Morgan Kaufman, 2016.

[16] *The OpenCL Specification v2.0*, Khronos OpenCL Working Group, 2015. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf

[17] *CUDA C Programming Guide*, NVIDIA Corporation, 2014. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[18] NVIDIA. (2015) NVIDIA Tegra X1. [Online]. Available: http://www.nvidia.com/object/tegra-x1-processor.html

[19] *Compute Cores. Whitepaper*, AMD, 2014. [Online]. Available: https://www.amd.com/Documents/Compute_Cores_Whitepaper.pdf

[20] Intel Corporation. (2015) The compute architecture of Intel processor graphics Gen9. [Online]. Available: https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

[21] Qualcomm. (2013) Snapdragon S4 processors: System on chip solutions for a new mobile age. Whitepaper. [Online]. Available: https://www.qualcomm.com/documents/snapdragon-s4-processors-system-chip-solutions-new-mobile-age

[22] *Exynos 5. Whitepaper*, Samsung, 2012. [Online]. Available: http://www.samsung.com/global/business/semiconductor/minisite/Exynos/data/Enjoy_the_Ultimate_WQXGA_Solution_with_Exynos_5_Dual_WP.pdf

[23] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Trans. Comput.*, vol. 28, no. 9, pp. 690–691, Sep. 1979.

[24] S. Owens, S. Sarkar, and P. Sewell, "A better x86 memory model: X86-TSO," in *22Nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, 2009, pp. 391–407.

[25] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *Computer*, vol. 29, no. 12, pp. 66–76, Dec. 1996.

[26] J. Goodacre and A. N. Sloss, "Parallelism and the ARM instruction set architecture," *Computer*, vol. 38, no. 7, pp. 42–50, Jul. 2005.

[27] B. A. Hechtman and D. J. Sorin, "Exploring memory consistency for massively-threaded throughput-oriented processors," in *40th International Symposium on Computer Architecture (ISCA)*, 2013, pp. 201–212.

[28] *GNC Architecture. Whitepaper*, AMD, 2012. [Online]. Available: https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf

[29] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous system coherence for integrated CPU-GPU systems," in *46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 457–467.

[30] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *34th International Symposium on Computer Architecture (ISCA)*, 2007, pp. 381–391.

[31] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: A heterogeneous CPU-GPU simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34–36, Jan. 2015.

[32] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[33] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *International Symposium on Performance Analysis of Systems and Software*, 2009.

[34] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *International Symposium on Performance Analysis of Systems and Software*, 2009.

[35] N. Muralimanohar and R. Balasubramonian, "Cacti 6.0: A tool to understand large caches," 2007.

[36] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *International Symposium on Workload Characterization*, Oct. 2009.

[37] P. Mistry, Y. Ukidave, D. Schaa, and D. Kaeli, "Valar: A benchmark suite to study the dynamic behavior of heterogeneous systems," in *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units (GPGPU-6)*. ACM, 2013, pp. 54–65.

[38] W.-M. Hwu, *Heterogeneous System Architecture: A new compute platform infrastructure*. Morgan Kaufmann, 2015.

[39] University of Rome "La Sapienza", "9th DIMACS Implementation Challenge," 2014, http://www.dis.uniroma1.it/challenge9/index.shtml.

[40] J. Gómez Luna, L.-W. Chang, I.-J. Sung, W.-M. Hwu, and N. Guil, "In-place data sliding algorithms for many-core architectures," in *44th International Conference on Parallel Processing (ICPP)*, Sep. 2015.

[41] AMD, "AMD accelerated parallel processing (APP) software development kit (SDK) 3.0," http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/, 2016.

[42] J. Gómez-Luna, J. M. González-Linares, J. I. Benavides, and N. Guil, "An optimized approach to histogram computation on GPU," *Machine Vision and Applications*, vol. 24, no. 5, pp. 899–908, 2013.

[43] J. Gómez-Luna, H. Endt, W. Stechele, J. M. González-Linares, J. I. Benavides, and N. Guil, "Egomotion compensation and moving objects detection algorithm on GPU," in *Applications, Tools and Techniques on the Road to Exascale Computing*, ser. Advances in Parallel Computing, vol. 22. IOS Press, 2011, pp. 183–190.

[44] Intel Corporation. (2013) Products (formerly Haswell). [Online]. Available: http://ark.intel.com/products/codename/42174/Haswell

[45] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *IEEE International Symposium on Workload Characterization (IISWC)*, Dec. 2010.