2017 IEEE 35th International Conference on Computer Design

M2S-CGM: A Detailed Architectural Simulator for Coherent CPU-GPU Systems

Christopher E. Giles
Department of Electrical and Computer Engineering
University of Central Florida, FL, USA
christopher.e.giles@knights.ucf.edu

Mark A. Heinrich
Department of Computer Science
University of Central Florida, FL, USA
heinrich@cs.ucf.edu

***Abstract*—We introduce M2S-CGM a detailed architectural simulator that models the interactions between CPUs and GPUs operating in coherent heterogeneous compute environments. M2S-CGM extends an existing and established x86 CPU model and Southern Islands GPU model, adds a new custom-built memory system model and switching fabric called CGM, and incorporates a well-known SDRAM model. The CGM memory system simulator provides configurable entire system simulation and can support a range of non-coherent and coherent CPU-GPU configurations. M2S-CGM supports the runtime for OpenCL-based benchmarks in addition to traditional multithreaded CPU benchmarks and can run benchmarks from established heterogeneous benchmark collections. This allows us to experiment with different coherent CPU-GPU configurations and propose effective future improvements in these systems. We present the makeup of M2S-CGM's software architectural design, provide a validation of the simulator, and provide coherent CPU-GPU execution results. Our validation results show average differences between our physical test system and M2S-CGM, of 10.4%, 22%, and 6.4% for 2 threaded, 4 threaded, and heterogeneous benchmark runs respectively. Our coherent CPU-GPU experimental results show an average speedup of 2.8 for our benchmarks over the baseline noncoherent system.**

I. Introduction

Recently, GPUs are being included on die with the CPU in a much more tightly-integrated environment and are being positioned to support future coherent execution with the CPU [1]. This new design space poses interesting research questions, such as how to best utilize shareable resources in a coherent CPU-GPU environment and what changes could be made to improve the heterogeneous system programming model? For the exploration of the CPU-GPU coherent design space we introduce M2S-CGM as a new and novel simulation tool. M2S-CGM provides end-to-end simulation of the system elements required to simulate non-coherent and coherent CPU-GPU heterogeneous workloads. M2S-CGM extends the multicore out-of-order x86 CPU model and multi-Compute Unit (CU) Southern Islands GPU model found in Multi2Sim [2]. Multi2Sim's x86 CPU model and Southern Islands GPU models [3] have previously been established, supported by the community, and proven to provide reasonably accurate CPU and GPU emulation and timing [4], [5].

We enhance the Multi2Sim package by completely removing the existing Multi2Sim memory system and replacing it with our own detailed memory system called CGM. We also make emulation and timing modifications to the Multi2Sim CPU and GPU models that correct certain modeling issues, enhance simulation fidelity, model runtime interactions between the CPU and GPU, and integrate with CGM. CGM provides coherence protocols and execution-driven discrete models of configurable CPU and GPU cache structures, directories, virtual memory mechanisms, switching fabrics, a system agent, and a memory controller. CGM also models occupancy and contention for all memory system structures within its scope. Main memory simulation is provided by DRAMSim2, which provides a cycle-accurate model of a SDRAM memory controller, the SDRAM modules of the main memory system, and the physical memory's internal buses [6].

In this paper, we introduce M2S-CGM and explain its advantages over existing tools. We cover aspects of the software architecture of M2S-CGM and discuss details regarding its heterogeneous simulation capability. We also provide validations of M2S-CGM's multicore and GPGPU capabilities and provide baseline results for noncoherent and coherent CPU-GPU benchmark executions. We show that M2S-CGM includes a high level of configurability that supports several forms of heterogeneous system experimentation. This provides researchers the ability to conduct a range of experiments with varying degrees of configurability in the memory system for both the CPU and GPU. After comparison to our physical test system, we establish that M2S-CGM provides good correlation to modern computing systems and that information ascertained from experimentation is reliable and can be used for trade-off decisions in proposed architectural implementations.

To our knowledge, M2S-CGM is novel in several ways:

- M2S-CGM provides baseline results for coherent CPU-GPU heterogeneous executions in a fully modeled system with and without shared lower level caches between the CPU and GPU. Our experimental results show the potential for coherent heterogeneous systems and provides new directions for future research.
- M2S-CGM includes a significantly more detailed memory system for both the CPU and GPU. We find that related work utilizes simpler memory systems with only one or two levels of cache, lacks shared resources, or does not include other pertinent architectural elements, such as, a system agent or GPU hub and IOMMU.
- M2S-CGM provides modeling of intra CPU-GPU system functionality that would otherwise typically require a full

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.84

system simulator. For system calls that are of importance, the x86 emulator produces an appropriate approximation of the GPU's driver and OS's kernel code for use by the CPU and memory system timing models. The result is the accurate simulation of system behavior, but without the need or complexity of developing or modifying a full system simulator that can boot a relatively recent OS kernel. This provides speed in heterogeneous system simulation and maintains a high level of reliability in the simulated results.

- M2S-CGM implements a modeling methodology that is unlike other mainstream simulator methodologies. M2S-CGM implements a simulated hardware signaling based methodology that more closely represents the makeup of the hardware for all the discrete elements in the system. This methodology is free from predetermined events and fixed timings for each event. Additionally, system occupancy and contention are implicitly modeled via this modeling methodology.

The rest of this paper is organized as follows. Section II provides a background in heterogeneous systems and the motivation for research in this area. Sections III and IV provide a software architectural overview and validation of M2S-CGM. Section V discusses our heterogeneous system implementation methodology. Sections VI and VII provide our coherent CPU-GPU heterogeneous system experimental results and observations. Finally, sections VIII and IX summarize related work to our own and conclude the paper.

II. Background and Motivation

The start of the GPGPU era served as a tipping point in the mainstream use of GPUs as co-processing elements to the CPU—the crux of a CPU-GPU heterogeneous system. GPGPU applications are designed to offload extremely parallelizable code segments onto the GPU where the GPU's Single Instruction Multiple Data (SIMD) architecture can provide significant speedup over a CPU's multi-threaded equivalent implementation. Currently, there are two mainstream forms of GPU-based co-processing: (1) a traditional approach where one or more GPUs are located on discrete graphics cards connected through one of several peripheral interconnect types and (2) a more recent approach where the GPU is colocated with the CPU on die and is connected to the rest of the system via the on-die switching fabric [7], [8]. Despite the differences between the two approaches, the processing model has remained similar—the GPU is treated as a separate system element that operates independently in its own memory address space.

In the GPGPU model the user must endeavor to partition the execution of the application between the CPU and GPU such that the overall application provides higher performance to the equivalent multithreaded version. The performance of the GPGPU version of the application, as compared to a multithreaded equivalent, is subjective and depends on several system variables, such as, number of CPU cores, number of GPU compute units, memory system latency, contention, and the number of interactions between the CPU and GPU. As a rule of thumb, research shows that fewer interactions between the CPU and GPU and larger problem sizes for the GPU will yield higher levels of speedup over applications with more interactions between the CPU and GPU and smaller problem sizes. However, it remains that some GPGPU implementations do not perform better than their multithread equivalents [9].

With discrete graphics cards, the application that is running on the CPU must first configure and setup the GPU's execution code and data elements prior to the execution of a selected kernel on the GPU. Subsequently, at the end of GPU kernel execution the application running on the CPU must copy the resultant data back from the GPU's memory address space to the CPU's address space so that the application can make use of the computed result. The GPU's configuration and data movement is accomplished via a series of system calls that invoke several OS and GPU driver interactions. This approach is required because the GPU is treated as a physically disparate I/O device, unequal to the CPU, with a different instruction set architecture (ISA) and memory structure.

This processing approach is still employed in modern processors that include both a CPU and GPU on die together. However, with the inclusion of the GPU on die with the CPU new heterogeneous hardware and software design spaces can be explored and new levels of parallel system performance are theoretically achievable. This serves as the motivation for producing M2S-CGM. M2S-CGM provides the foundational infrastructure required to study system architectural interactions between two processing elements with two different ISAs and very different processing capabilities. M2S-CGM allows for exploration of changes supporting performance improvements for heterogeneous workload executions and the study of the trade-offs those design choices impose. M2S-CGM allows us to experiment with both the programming model and its supporting hardware design spaces allowing for higher levels of hardware and software co-design.

III. M2S-CGM Architectural Overview

For our CPU-GPU heterogeneous system experiments, we show a typical configuration of the M2S-CGM simulator in Fig. 1 where we model a realistic processor and its ancillary components. The makeup of the system includes an x86 multicore CPU, a Southern Islands GPU, a detailed multi-level cache memory system, virtual memory mechanisms, switching fabric, system agent, memory controller, and SDRAM. The system's architecture is configured similarly to the Intel Haswell architecture where L2 caches, L3 caches, the GPU, and system agent sit on a network with a ring bus topology.

A. x86 CPU model

M2S-CGM extends the x86 CPU model found in Multi2-Sim. The x86 model includes a x86 ISA, disassembler, x86 system emulator, and a general purpose out-of-order pipelined CPU timing model. The x86 CPU can be configured as a multicore and multithreaded CPU with a highly configurable pipeline. During execution, applications are first loaded from

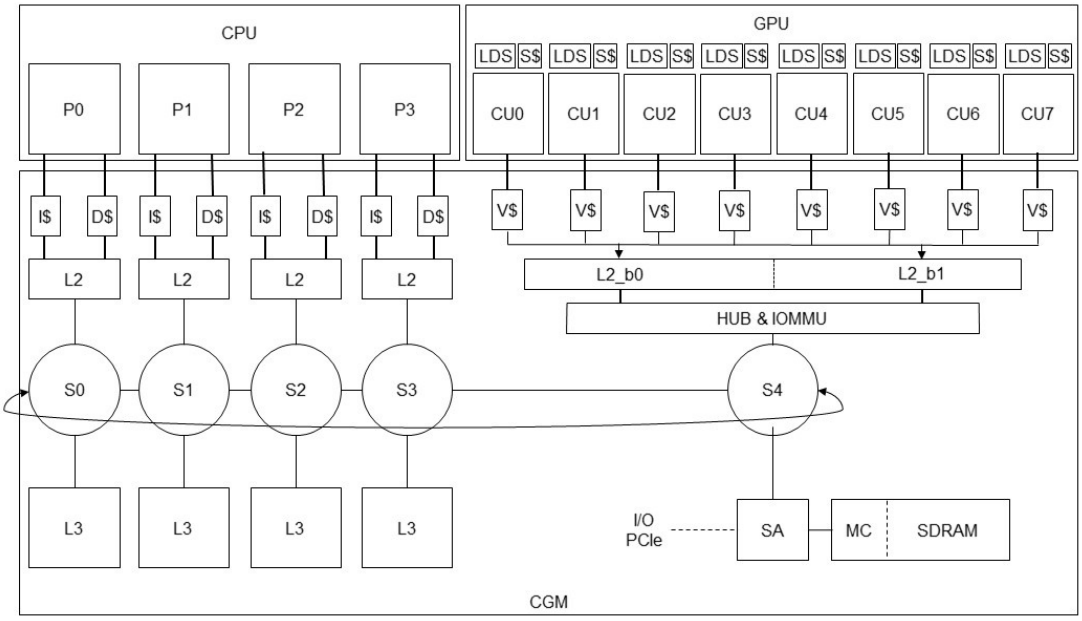


Figure 1: Simulated System Node Architectural Block Diagram

their native binary files, where macro instructions are subsequently fetched. The fetched macro instructions are then passed to a x86 emulator, executed, turned into pipeline uops, and passed to a x86 timing model. The CPU timing model interfaces with CGM at the fetch and issue stages and is dependent on CGM's modeled memory system latency. Addresses are looked up in a translation lookaside buffer and on hit are translated from virtual to physical prior to accessing the respective L1 cache. Memory system accesses are execution driven and latency varies based on memory system contention and occupancy. Access latency within the memory system, saturation of memory system elements, or lack of free miss status handling registers may result in any combination of an empty fetch queue, full reorder buffer, and full issue buffer which effectively stalls the CPU.

B. Southern Islands GPU Model

M2S-CGM also extends the Southern Islands GPU model found in Multi2Sim. The Southern Islands GPU model includes a Southern Islands ISA, disassembler, emulator, and an in-order GPU timing model. The GPU comprises a number of CUs and each CU in turn comprises a front end, SIMD lanes, a scalar unit, a branch unit, a vector memory unit, and a Local Data Share (LDS) unit. For the CPU to make use of the GPU, GPU kernel executions must be prepared by the CPU in advance as a part of the user's application. This requires the use of an OpenCL library, runtime, and the GPU's drivers. During execution, the CPU traps to the OS via a series of system calls that set up and load the GPU kernel on the GPU. Instructions are fetched by the CU Front End and are issued to their respective processing unit. During execution there are three types of memory access in the GPU. These include accesses to the LDS, scalar cache, and vector cache.

The GPU timing model interfaces with CGM at these access points where CGM provides latency for these internal memory elements and external memory system accesses. In addition to the caches and crossbar, the GPU memory system model includes a hub and IOMMU (see Fig. 1). The hub multiplexes memory system accesses between the GPU's L2 cache banks and the switching fabric. The IOMMU performs address translations for the GPU and can alternately preform both forward and reverse address translations if utilizing virtual addressing within the GPU's caches. CGM coalesces vector memory accesses at the system interface prior to memory system access. Upon L1 cache input queue saturation the vector pipeline is stalled until there is space to issue the next round of memory system accesses. Additionally, memory system accesses destined for the external memory system include a configurable virtual or physical address schema. For our experimentation purposes we assume that GPU virtual to physical address translation can occur at either the interface between each CU and the first level of cache or within the GPU's IOMMU. GPU TLB misses result in an intervention by the CPU. An efficient design of the GPU's hardware that enables the GPU to share a virtual address space with the CPU is still an open research question.

C. CGM Memory System Model

CGM is a new and custom memory system model that replaces Multi2Sim's existing memory system model. CGM provides models of configurable cache structures, cache directories, translation lookaside buffers, page table walkers,

switching fabrics, cross bars, a system agent, memory controller, and other discrete system elements such as the GPU's hub and IOMMU. For the CPU and GPU, CGM currently implements a detailed MESI and MEI coherence protocol that supports request forwarding (3-way hops), joins, upgrades, and is nearly NACK-free. Coherence protocol actions within the GPU are optimized for the GPU in both non-coherent and coherent modes of operation. CGM models occupancy and contention for all memory system structures and queues within CGM's scope. Caches are configurable and incorporate coalescing and miss status handling for flow control and support asynchronous data transfer and virtual lanes for message packet access routing. Switches support 4 ports, utilize an internal crossbar, and are asynchronously linked with other elements in the memory system via bidirectional links. Internally, the switches include multiple lanes for request, replies, and other coherence related message traffic. The system agent currently provides interfaces to the SDRAM controller and is responsible for providing interfaces to other I/O related system elements and other CPU nodes.

CGM's timing model and implementation is based on a simulated hardware signaling methodology for all simulated memory system elements in its scope. In this approach individual elements are advanced (signaled) when given work to perform. On advance elements wake up and try to process their assigned work until success. Once complete, the next element is given work to perform and it is in turn advanced by the previous element. On each advance elements assess a latency for the performance of the work performed and pause until that latency has expired. The assessed latency provides the modeled occupancy of that element. Contention is modeled as resources begin to fill or saturate creating element stalls and longer access latency. In comparison to the Multi2Sim existing memory system and GEM5 timing model, the hardware signaling methodology does not rely on a predetermined event and event execution time for the start of a modeled element's task. Thus the hardware signaling methodology more accurately models physical system elements, element interactions, and system-wide occupancy and contention.

D. DRAMSim 2

For the simulation of our main memory modules we include DRAMSim2 [6]. DRAMSim2 provides detailed timing models for several state-of-the-art SDRAM and DDR memory modules. DRAMSim2 is connected to CGM through CGM's memory controller. DRAMSim2 models contention and occupancy within the SDRAM modules and includes selectable scheduling paradigms.

IV. M2S-CGM VALIDATION

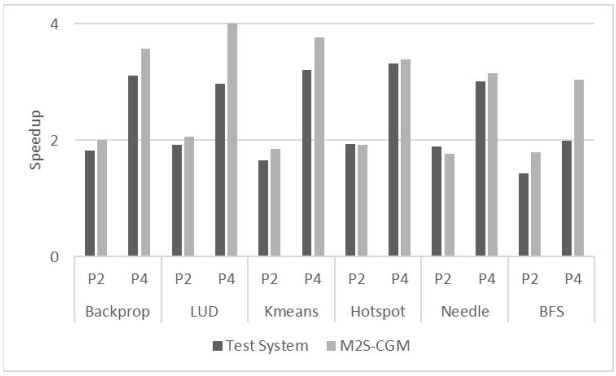
As mentioned earlier, previous work has introduced and established the CPU and GPU models of Multi2Sim [4], [5]. This leaves the validation of M2S-CGM with its detailed memory system model. For our tests we compare our simulated benchmark results to the benchmark results from our target test system. The test system comprises an Intel Core i7-4790K Devil's Canyon Quad-Core CPU and AMD Radeon HD 7990 64 CU GPU. We configure M2S-CGM as shown in Fig. 1 and to match our test system's hardware profile. M2S-CGM and test system frequencies for the CPU, GPU, and memory system are 4 GHz, 1 GHz, and 2 GHz respectively. For M2S-CGM, the system-wide cache block size is 64B and we assume an 8 byte header on all memory system messages. CPU L1, L2, and L3 cache sizes are configured as 32KB, 256KB, and 2 MB respectively with L3 caches in a striped configuration. GPU vector and L2 cache sizes are configured as 16KB and 64KB respectively. Main memory is configured as dual channel with 4GB SDRAM.

For validation we select the Rodinia OpenMP and OpenCL Backprop (BP), Lower Upper Decomposition (LUD), Kmeans, Hotspot, NeedlemanWunsch (NW), and Breadth First Search (BFS) benchmarks. The selected benchmarks provide a good spectrum of processor intercommunication aggressiveness for OpenMP executions and a good spectrum of inter-CPU-GPU-communication through GPU kernel invocations and memory copies for OpenCL executions. For all benchmarks we take measurements across the benchmark's parallel section. For OpenCL benchmarks, we define the beginning of the parallel section to be the first OpenCL-related memory buffer creation and the ending of the parallel section to be the completion of the final memory copy to the host device. We select benchmark problem sizes based on the maximum obtainable speedup in the simulated system where problem sizes range from medium to large.

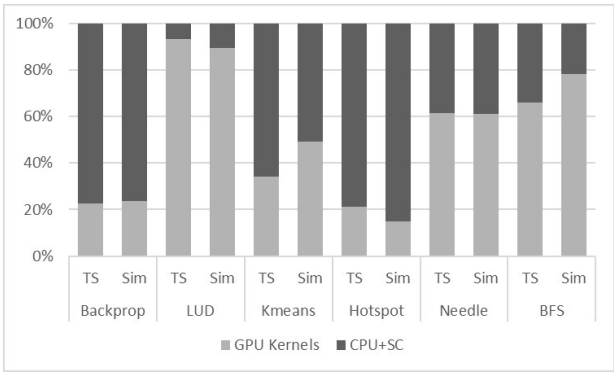
In validating M2S-CGM, we compare measured speedup on the test system to measured speedup on M2S-CGM for the Rodinia OpenMP benchmarks as shown in Fig. 2a. We also compare heterogeneous-workload percentage breakdown for the Rodinia OpenCL benchmarks as shown in Fig. 2b. We do not seek to compare absolute total cycles between the target test system and simulator because the simulated CPU and GPU are generalized and represent a wide range of possible processor configurations. Instead, by correctly modeling and observing system behavioral results we draw conclusions on the influence of system level design changes that can be applied to more than just a single processor's architecture.

In Fig. 2a we measure the speedup for 2 and 4 threads for the Rodinia OpenMP benchmarks on our test system and on M2S-CGM. The results show good correlation between the test system and M2S-CGM and highlight expected performance differences. For the OpenMP benchmarks, simulated execution had an average difference of 10.4% for the two threaded runs and 22% for the four threaded runs. These differences are expected and show correct simulation behavior as compared to a physical system that is running many other system processes in addition to the benchmarks themselves. These results also highlight the inherent parallelism and correctness of the memory system's MESI protocol.

Results for OpenCL benchmarks are shown in Fig. 2b. We measure heterogeneous workload execution time and provide breakdowns of GPU kernel time, CPU time, and OpenCL related system call time. In the results we combine CPU



(a) Rodinia OpenMP Benchmark Results



(b) Rodinia OpenCL Benchmark Results

Figure 2: Validation Results

time and OpenCL related system call time and denote this as "CPU+SC". Again, the results show close correlation between the test system and M2S-CGM and highlight the delicate simulation of the interplay between the CPU and GPU over the parallel section. For the OpenCL benchmarks, simulated execution time breakdown between the CPU and GPU is within 6.4% on average. We also note that the Rodinia OpenCL benchmarks themselves do not make use of the CPU for processing during the execution of the benchmark. Instead, the CPU only performs setup and management of problem data and GPU execution through a series of OpenCL calls and ultimately OS system calls. During GPU invocation the CPU goes dormant until the GPU completes and signals the CPU to wake up.

Based on the results shown here, and by successful comparison of M2S-CGM's simulated results to our test system, we establish that M2S-CGM provides a valid and realistic multicore and heterogeneous system model and therefore can serve as a strong platform for our heterogeneous system research.

V. Heterogeneous System Implementation Methodology

We Configure M2S-CGM in the following four configurations for our heterogeneous workload executions:

- A traditional GPGPU configuration, denoted as NC-MC, with the CPU and GPU operating in disparate virtual address spaces and disparate memory systems.
- A modified traditional GPGPU configuration, denoted as NC-L3, with the CPU and GPU operating in disparate virtual address spaces, but with shared lower level caches.
- A potential half CPU-GPU heterogeneous configuration, denoted as C-MC, with the CPU and GPU operating in a shared virtual address space, but disparate memory system.
- A potential full CPU-GPU heterogeneous configuration, denoted as C-L3, with the CPU and GPU operating in both a shared virtual address space and with shared lower level caches.

In our configuration nomenclature, "NC" and "C" stand for noncoherent and coherent. "MC" and "L3" stand for memory controller and L3 cache and represent the GPU memory request destination when entering the memory system from the GPU.

Our traditional and modified GPGPU configurations serve as our heterogeneous execution baseline and model a current GPGPU system. In these configurations the GPU is treated as a disparate device from the CPU and must work in its own address space. This requires the CPU to copy memory back and forth to and from the GPU prior to and after kernel executions through a series of OS systemcalls and GPU driver executions. Additionally, the OS must manage the state of the memory system, and ensure that CPU and GPU caches are fully flushed to main memory prior to a memory system read at the beginning and end of GPU kernel execution. In our modified GPGPU configuration we extend the memory system to allow for sharing of the L3 caches between the CPU and GPU which effectively gives the GPU a larger and faster third level of cache.

Our system changes that enable our half and full CPU-GPU heterogeneous configurations concern the entire software stack and the underlying hardware itself. We configure the OS, GPU driver, and OpenCL runtime such that the CPU and GPU can operate in a shared virtual address space and share data. We make the assumption that, for now, the CPU is still responsible for explicitly coordinating the activities of the GPU. Thus, the CPU must first allocate and prepare all application memory and GPU kernels prior to GPU executions. We configure our OpenCL benchmarks and runtime to pass memory by pointer to the GPU from the CPU. This eliminates the explicit requirement for memory copies between the CPU and GPU and allows the GPU to simply pull in the data it needs by standard memory system requests.

Several hardware changes are required to enable the CPU-GPU heterogeneous environment. First, we introduce memory system coherency between the CPU and GPU. We implement a MEI protocol for the GPU's memory system. This protocol is optimized for the streaming nature of GPU memory system

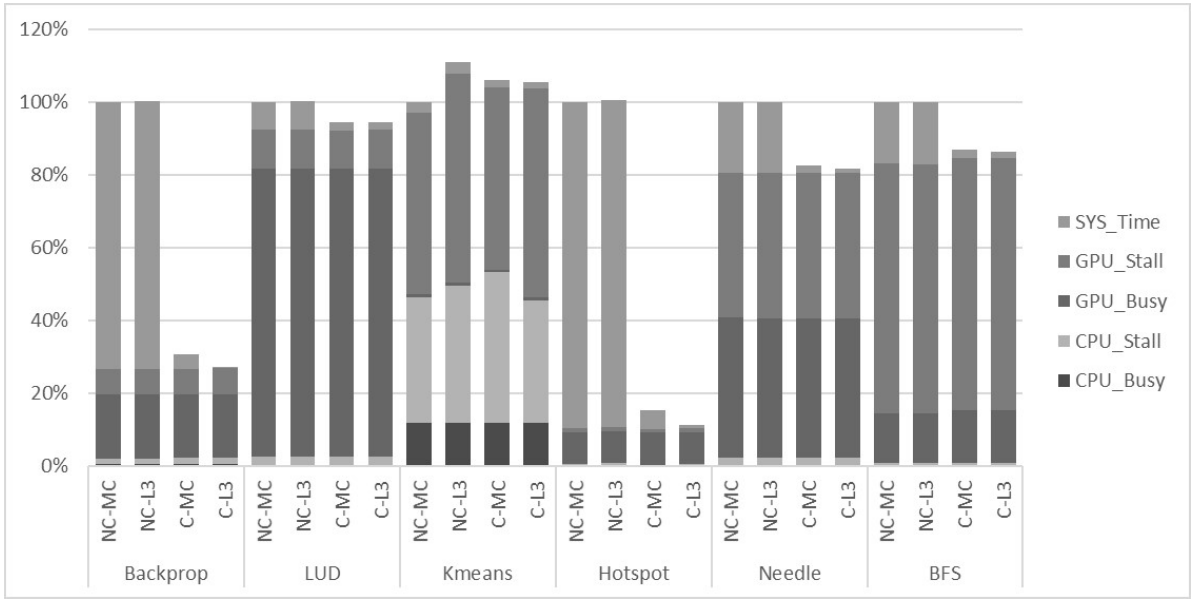


Figure 3: CPU-GPU Heterogeneous System Experimental Results

requests and readily connects with the CPU's MESI protocol. Coherence protocol request forwarding (3-way hops) is supported between the L3 and L2 caches by both the CPU and GPU. Second, memory system coherence directories are extended to account for the entire GPU as an nth core. Third, the hardware mechanisms necessary to support virtual memory are built into the GPU's memory system. This includes GPU specific TLBs, PTWs, and reverse address translation functionality which is incorporated in our modeled GPU IOMMU. We assume that GPU address translation occurs within the CU or at the IOMMU on a GPU TLB miss. A best approach to GPU led address translation in a coherent fully heterogeneous system is still an open research question. For our experiments, we wish to establish an ideal performance baseline and thus do not charge a penalty for GPU TLB misses. We plan to explore this topic in a subsequent paper, where we use our established ideal performance baseline as the point of comparison.

VI. Experimental Results

For our heterogeneous system experiments we maintain the system wide configuration parameters and OpenCL parallel section definition as outlined in Sec. IV. Experimental results for our selected Rodinia OpenCL Benchmark executions on each system configuration are shown in Fig. 3. For each OpenCL benchmark all results are normalized to the benchmark's NC-MC configuration results. The results show execution breakdowns for CPU Busy, CPU Stall, GPU Busy, GPU Stall, and System Time (SYS Time) in percentage of cycles. CPU and GPU busy time is the time the CPU and GPU performed work over the parallel section. CPU and GPU stall time is the time the CPU and GPU were stalled while waiting on outstanding memory system requests. System time is the time spent trapped to the OS while performing an OpenCL related system call or memory system related CPU interrupt, such as a cache flush. Results show that our modeled half coherent CPU-GPU heterogeneous system achieves speedups of 3.27, 1.06, 0.94, 6.51, 1.21, and 1.19 and our fully coherent CPU-GPU heterogeneous system achieves speedups of 3.67, 1.06, 0.95, 8.83, 1.23, and 1.40 for Backprop, LUD, Kmeans, Hotspot, Needleman, and BFS respectively.

The measured system time comprises all of the overhead associated with executing workloads in the CPU-GPU heterogeneous environment and includes all of the intercommunication required between the CPU and GPU. From comparison of the results between noncoherent and coherent executions, it is apparent that the extent to which speedup is achievable is dependent on the amount of overhead incurred by the application. Thus applications that require significant CPU intervention, like Backprop and Hotspot, see significant improvements and others, like LUD, Needleman, and BFS see slight improvement. However, all benchmarks with the exception of Kmeans did show improvement in speedup. Kmeans is an exception because the parallel section in Kmeans contains a significant amount of CPU setup between kernel executions which effectively defeats the purpose of parallelizing the benchmark over the GPU. This is evident in the nearly negligible GPU busy time for Kmeans.

CPU and GPU busy and stall time show the benchmark's utilization of the CPU and GPU together. After inspection of the Rodinia OpenCL Benchmark's source code and the experimental results it is apparent that allocation of work to the CPU is non existent. The current Rodinia OpenCL Benchmarks do not fully exploit the complete level of parallelism available between the CPU and GPU. As shown in the results, the Rodinia OpenCL benchmarks effectively delegate all processing to the GPU while the CPU mostly remains idle

where it is relegated to only performing coordinating functions with the GPU between kernel executions. In an ideal system the CPU would effectively share 50% of the total busy time with the GPU. Performance gains in these measurements rely on better utilization of the CPU and GPU. Despite this, CPU and GPU busy time remains consistent between configuration experiments which is expected and correct.

In the noncoherent configuration the results show that sharing the L3 caches between the CPU and GPU is ineffective and actually slightly hurts overall performance. This is due to the streaming nature of the GPU and low temporal reuse of memory system blocks. Thus, forwarding memory access request to L3 resulted in a predominance of L3 cache misses, which are then subsequently sent to the memory controller. In a noncoherent configuration it is better to directly forward the memory request to the memory controller and bypass the latency of the L3 cache access. However, in the heterogeneous configuration shared L3 caches can provide a measurable performance boost. This is due to the coherence protocol, where GPU cache flushes are no longer required on account of the supported request forwarding between the CPU and GPU.

VII. Observations

Drawing on the experiences gained from the implementation of M2S-CGM and the generation of our baseline experimental results we have gleamed several important insights related to the nature of coherent CPU-GPU heterogeneous workloads.

- The CPU and GPU should share a single virtual address space. By sharing a single virtual address space underlying mechanisms like memory copies between the CPU and GPU are no longer required and higher levels of parallelism can be obtained through traditional synchronization and coherency mechanisms.
- GPU address translation mechanisms need to be researched more. The current IOMMU approach incurs significant overhead as we must trap back to the CPU to solve address translation issues. The GPU should be capable of resolving address translation issues on it own.
- Current benchmarks do not fully exploit the complete level of parallelism available between the CPU and GPU. The Rodinia OpenCL benchmarks effectively delegate all processing to the GPU while the CPU mostly remains idle. In an ideal system the CPU would effectively be load balanced with the GPU and share 50% of the runtime's busy time. New benchmarks that take advantage of shared data in a fully coherent CPU-CPU heterogeneous environment are needed.

VIII. Related Work

Related work in CPU-GPU heterogeneous system research is diverse and covers many aspects over the breadth of both the software stack and hardware layer. Recent software-based approaches include techniques that manage system coherence such as hypervisor layers and modified programming models [10], [11], [12]. Higher level software-based approaches that utilize the current underlying hardware and software may provide a boost in speedup and efficacy of programming, however significant system changes proposed require a more robust software and hardware co-design where the programming model and underlying hardware are both changed to evoke a substantive improvement in processing performance. For this paper, we limit our related work summary to computer architectural simulators and recent research in hardware based CPU-GPU coherency.

Examples of directly related work in CPU-GPU heterogeneous computer architectural simulators includes Gem5-GPU [13] and FusionSim [14]. Gem5-GPU combines the Gem5 CPU model, the Ruby memory system model, and the GPGPU-Sim GPU model into a cohesive system that allows for simulation of GPGPU workloads under full system or system call emulation modes. With Gem5-GPU researchers can run CUDA and OpenCL benchmarks utilizing the predominantly NVIDIA-based ISA for GPU simulation provided by GPGPU-Sim's GPU model. FusionSim, another heterogeneous system simulator, also makes use of GPGPU-Sim and includes an x86 CPU model. In comparison to Gem5-GPU and FusionSim, M2S-CGM includes similar functionality, but makes different implementation choices. M2S-CGM allows for OpenCL-based benchmarks to be run on an AMD-based Southern Islands ISA and includes a CPU and GPU memory system model that provides higher detail over the memory system models included in Gem5-GPU and FusionSim. M2S-CGM's memory system model, includes elements such as a detailed MESI protocol, configurable cache structures, cache directories, translation lookaside buffers, page table walkers, switching fabrics, system agents (Intel termed uncore), memory controllers, GPU hub and IOMMU, and SDRAM model. M2S-CGM does not implement a full system simulation mode, however M2S-CGM does provide emulation and timing models for the functional interactions that occur by the OS as a result of application evoked system calls. This modeling methodology provides fidelity comparable to a full system simulator without the development overhead of implementing and supporting a full system simulation mode and kernel boot.

Another example of related work in CPU-GPU heterogeneous computer architectural simulators includes Multi2-Sim [2] from which M2S-CGM inherits its CPU and GPU model from. Multi2Sim allows for researches to run the AMD APP SDK sample OpenCL benchmarks in a non CPU-GPU coherent environment, includes simple memory system models for the CPU and GPU, and utilizes averaged flat latency timings for memory system interactions and for main memory latency [15]. M2S-CGM retains the ability to run the AMD APP SDK sample OpenCL benchmarks, but also includes extensions to the CPU and GPU models that correct timing issues and allows M2S-CGM to run the Rodinia OpenMP benchmarks and, currently, most of the Rodinia OpenCL benchmarks [16]. In contrast to Multi2Sim, M2S-CGM can be configured for both non-coherent and coherent executions of heterogeneous workloads in both disparate and shared virtual memory address spaces. Also as mentioned before, M2S-CGM replaces the existing Multi2Sim memory system model with

CGM. In lieu of Multi2Sim's averaged flat latency timings, CGM provides timings for memory requests, system coherence messages, and DRAM accesses while taking a complete picture of system-wide occupancy and contention into account. We do not make direct comparisons between M2S-CGM and Multi2Sim because Multi2Sim's native memory system does not support modeling of the differences outlined in Sec. I.

Examples of directly related research in hardware based CPU-GPU coherency include a timestamp-based protocol called Temporal Coherence [17] and a directory-based region coherence protocol called Heterogeneous System Coherence [18]. Temporal Coherence provides a low overhead coherence mechanism for memory systems internal to the GPU. In comparison to Temporal Coherence, Our work focuses on directory based full system coherence to include hardware based coherence internally to the GPU as well as between the CPU and GPU. Heterogeneous System Coherence proposes the addition of region buffers to the CPU and GPU L2 caches. In this context the region buffer acts as a course-grain filter for memory system message traffic. Thus, if the CPU or GPU controls a region of memory it then does not need to consult with other processing elements before preforming reads or writes to main memory. In contrast, our work focuses on a protocol based approach that relies on memory system block forwarding between the CPU and GPU. Our approach minimizes impacts to the general purpose CPU and does not introduce the possibility of region thrashing in the ideal heterogeneous system, where, the CPU and GPU jointly perform work within the same region. We note that comparisons between our experimental results and the experimental results derived via Heterogeneous System Coherence reconcile.

IX. Conclusion

In this paper we introduced M2S-CGM, a detailed architectural simulator that models the interactions between CPUs and GPUs operating in heterogeneous compute environments. We presented the motivation and need for M2S-CGM and provide in-depth details about its software architectural makeup. We provided a validation of M2S-CGM's multithreaded and heterogeneous system simulation capabilities via the comparison of executions of select Rodinia OpenMP and OpenCL Benchmarks on a test system and the simulator. Finally, we utilized the Rodinia OpenCL Benchmarks and conducted experiments over four noncoherent and coherent heterogeneous system configurations.

Our validation results show that M2S-CGM provides an accurate simulation model of a modern multicore and heterogeneous system with differences ranging from 10.4% and 22% for two and four threaded OpenMP runs and 6.4% for OpenCL runs. Our noncoherent and coherent benchmark executions show that added coherency between the CPU and GPU can provide significant performance gains. Results show that our modeled half coherent CPU-GPU heterogeneous system achieves speedups of 3.27, 1.06, 0.94, 6.51, 1.21, and 1.15 and our fully coherent CPU-GPU heterogeneous system achieves speedups of 3.67, 1.06, 0.95, 8.83, 1.23, and 1.16 for Backprop, LUD, Kmeans, Hotspot, Needleman, and BFS respectively over the noncoherent equivalent.

A standalone version of CGM and the modifications made to Multi2Sim, that enable our heterogeneous simulations, are made available for public use as free software for future research purposes. Current versions of CGM and M2S-CGM can be found on GitHub.

References

[1] AMD, "Amd graphics cores next (cgn) architecture," AMD, Tech. Rep., June 2012.

[2] NEU, "Multi2sim a heterogeneous system simulator," accessed: Jan-23-2017. [Online]. Available: http://www.multi2sim.org/

[3] AMD, "Amd radeon hd 7000 series graphics cards," accessed: Jan-1-2016. [Online]. Available: http://www.amd.com/en-us/products/processors/desktop/a-series-apu

[4] D. Schaa and R. Ubal, "Multi-architecture isa-level simulation of opencl," 2013, international Workshop on OpenCL. [Online]. Available: http://www.multi2sim.org/publications.html?id=iwocl-2013

[5] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 335–344.

[6] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "Dramsim: A memory system simulator," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 100–107, Nov. 2005.

[7] AMD, "Amd a-series desktop apus," accessed: Jan-23-2017. [Online]. Available: http://www.amd.com/en-us/products/graphics/desktop/7000

[8] Intel, "The compute architecture of intel processor graphics gen9," Intel, Tech. Rep., August 2015.

[9] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010.

[10] P. Rogers, "Heterogeneous system architecture overview," 2013, symposium on High Performance Chips. [Online]. Available: http://www.hotchips.org/archives/2010s/hc25/

[11] OpenACC-standard.org, "Openacc," accessed: May-3-2017. [Online]. Available: http://www.openacc.org/

[12] I. Gelado, J. E. Stone, J. Cabezas, S. Patel, N. Navarro, and W.-m. W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 347–358.

[13] 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.

[14] V. Zakharenko, T. Aamodt, and A. Moshovos, "Characterizing the performance benefits of fused cpu/gpu systems using fusionsim," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013, pp. 685–688.

[15] AMD, "Amd app sdk," accessed: Jan-1-2016. [Online]. Available: http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2009, pp. 44–54.

[17] I. Singh, A. Shriraman, W. W. L. Fung, M. O'Connor, and T. M. Aamodt, "Cache coherence for gpu architectures," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 578–590.

[18] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous system coherence for integrated cpu-gpu systems," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 457–467.