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

Christopher Giles, Mark Heinrich

Abstract—We introduce M2S-CGM a detailed architectural simulator that models the interactions between CPUs and GPUs operating in 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 end-to-end 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 select 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 and validate the simulator with preliminary evaluations. We also discuss our current research efforts where we plan to utilize M2S-CGM.

Index Terms—GPGPU, HSA, Rodinia benchmark suite, hardware architectural simulators.

1 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 [3]. 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 heterougeneous system programming model? For our exploration of the CPU-GPU coherent design space we introduce M2S-CGM. M2S-CGM provides end-to-end simulation of all 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 [8]. Multi2Sim's x86 CPU model and Southern Islands [4] GPU model have previously been established, supported by the community, and proven to provide reasonably accurate CPU and GPU emulation and timing models [11, 12]. 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 enhance simulation fidelity, model runtime interactions between the CPU and GPU, and support interactions with CGM. CGM provides protocols and execution-driven discrete models of configurable cache structures, directories, switching fabrics, a system agent, and a memory controller. CGM models occupancy and contention for all memory system structures. We also integrate DRAMSim2, which simulates physical memory modules [13]. DRAMSim2 provides a cycle-accurate model of a SDRAM memory controller, the SDRAM modules of the main memory system, and the internal main memory bus.

In comparison to other existing heterogeneous system simulators M2S-CGM has several unique characteristics. Gem5-gpu [9] 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 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 [14]. M2S-CGM includes similar functionality and fidelity to gem5-gpu and FusionSim, but makes different implementation choices. M2S-CGM allows for OpenCL-based benchmarks to be run on an AMD-based Southern Islands GPU model with a custom and detailed GPU memory system model. M2S-CGM also models the interactions between the CPU and GPU that occur as a result of OpenCL related system calls without the overhead of supporting a Full System Simulation mode. Multi2Sim, from which M2S-CGM inherits its CPU and GPU model from, allows for researches to run the AMD APP SDK sample OpenCL benchmarks in a non-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 [1]. M2S-CGM retains the ability to run the AMD APP SDK sample OpenCL benchmarks, but includes extensions to the CPU and GPU models that correct timing issues and allows M2S-CGM to run the Rodinia OpenMP benchmarks and most of the Rodinia OpenCL benchmarks [5]. In addition, M2S-CGM can be configured for both non-coherent and coherent executions of heterogeneous workloads in both disparate and shared memory address spaces. M2S-CGM replaces the existing Multi2Sim memory system model with CGM; our detailed and customized memory system model. CGM provides a detailed CPU and GPU memory system model, including elements such as a detailed MESI protocol, configurable cache structures, cache directories, switching fabrics, system agents, memory controllers, a hub-iommu, switching fabric, and SDRAM model. Timings for memory requests, system messages, and DRAM accesses are modeled taking into account configurable latencies and system-wide occupancy and contention.

We designed and implemented M2S-CGM for our near-term research purposes. We plan to make CGM a standalone simulator with the intent to provide modular, highly configurable, and highly detailed simulation of system-side and memory system elements. Researchers implementing their own processor simulations or reusing currently available processor simulations, regardless of type, size and quantity, can easily utilize CGM as the integrating factor at the system level. The standalone version of CGM will be made available for public use as free software for future research purposes. In addition, the modifications made to Multi2Sim that enable our heterogeneous simulations will also be made available for public use as free software upon completion of our near-term research efforts.

2 The Heterogeneous System Era

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

- *Christopher Giles is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL. Contact via E-mail christopher.e.giles@knights.ucf.edu.*
- *Mark Heinrich is with the Department of Computer Science, University of Central Florida, Orlando, FL. Contact via E-mail heinrich@cs.ucf.edu.*

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 GPUs are located on a discrete graphics card and connected through one of several peripheral interconnect types and (2) a more recent approach where GPUs are colocated with the CPU on die and are connected to the rest of the system via the on die switching fabric [2, 6]. Despite the differences between the two approaches, the processing model has remained similar—the GPU is treated as a separate system element and operates independently in its own memory address space.

In the GPGPU model an application 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 a physically disparate device, unequal to the CPU, with a different instruction set architecture (ISA) and memory structure.

With the recent 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. The Heterogeneous System Architecture Intermediate Language (HSAIL) is one example of a recent software based-approach [10]. HSAIL provides an intermediate layer that abstracts separate ISAs into a single instruction type. An HSAIL virtual machine manages the execution of the application and automatically constructs executables for the target ISAs in a cohesive environment. However, despite the level of flexibility offered by HSAIL and other predecessor languages, like OpenCL, the architecture of the underlying hardware remains critical to overall application performance. Effective exploitation of the system's hardware in a heterogeneous manner requires that the underlying computational architecture supports a shared processing environment. This touches many system-level design areas including elements such as the memory system, OS, compiler, runtime, drivers, and to an extent the co-processor(s) themselves.

M2S-CGM provides the foundational infrastructure required to study system architectural interactions between two processing elements with two different ISAs. 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 level of hardware and software co-design. In the following sections we describe the composition of M2S-CGM.

3 M2S-CGM System Architecture

For our current and near-term heterogeneous system experiments, we show a realistic example of a typical configuration of the M2S-CGM system in Fig. 1. Our system architectural design and implementation includes the elements necessary to run CPU-GPU heterogeneous workloads and experiment with architectural changes and draw sound conclusions on performance results. The makeup of the system includes an x86 multicore CPU, a Southern Islands GPU, a multi-level cache memory system, a switching fabric, system agent, memory controller, and SDRAM.

3.1 x86 CPU model

M2S-CGM extends the 32-bit x86 CPU model found in Multi-2Sim. 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 their native binary files, where, macro instructions are subsequently fetched. The fetched macro instructions are then passed to the x86 emulator, executed, turned into pipeline uops, and passed to the 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 latencies. Addresses are translated from virtual to physical prior to accessing the respective L1 cache. Memory system accesses are execution driven and latencies vary 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.

3.2 Southern Islands GPU Model

M2S-CGM 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 points where CGM provides latencies for internal memory elements like the LDS and scalar 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. Memory system accesses are by virtual address within the first two levels of GPU cache. The GPU memory system model includes a hub-iommu (see Fig. 1) that multiplexes memory system accesses between the GPU's L2 cache banks and the switching fabric. Additionally, the hub-iommu performs forward and reverse address translations on behalf of the GPU.

3.3 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, switching fabrics, system agents, memory controllers, and other

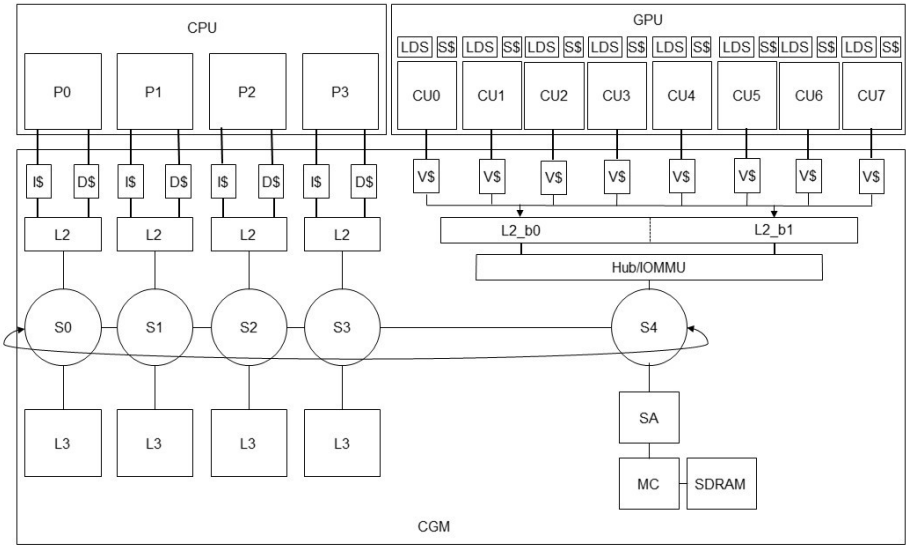

Fig. 1: Simulated System Architectural Block Diagram

discrete system elements such as the GPU's hub-iommu. CGM currently implements a detailed MESI coherence protocol that supports request forwarding (3-way hops), joins, upgrades and is nearly NACK-free. In addition, 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 can potentially provide interfaces to other I/O system elements and other CPU nodes.

CGM builds on ideas found in related memory system simulators, such as, the Multi2Sim existing memory system and GEMS [7], but makes different implementation choices in its timing model and adds new fidelity to the GPU memory system model. CGM's timing model is based on an "advance-and-process" approach for all of its simulated memory system elements. In this approach individual elements are advanced when given work to perform. On advancement 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 latencies. In comparison to the Multi2Sim existing memory system and GEMS timing model, the "advance-and-process" approach does not rely on a predetermined future wake up time and method for the start of element work and thus more accurately models physical system elements, element interactions, and system-wide occupancy and contention.

3.4 DRAMSim 2

For the simulation of our main memory modules we include DRAMSim2 [13]. DRAMSim2 provides detailed timing models for several state-of-the-art memory modules, such as, SDRAM, and DDR memory types. DRAMSim2 is connected to CGM through CGM's modeled memory controller. DRAMSim2 models contention and occupancy within the SDRAM modules and includes selectable scheduling paradigms. Optionally, DRAMSim2 can be removed from the overall simulation via CGM's configuration files, however, the inclusion of DRAMSim2 provides better simulation fidelity over the pipelined fixed latency alternative.

4 M2S-CGM Validation

As mentioned earlier, previous work has introduced and established the CPU and GPU models of Multi2Sim [11, 12]. 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. The M2S-CGM CPU and GPU models have configurable core and CU counts up to 64 each. Simulation 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 caches sizes are configured as 16KB and 64KB respectively. Main memory is configured as dual channel with 4GB SDRAM.

For validation we select Rodinia OpenMP and OpenCL Backprop (BP), Lowe Upper Decomposition (LUD), Kmeans, Hotspot, NeedlemanWunsch (NW), and Breadth First Search (BFS) benchmarks. The selected OpenMP and OpenCL benchmarks provide a good spectrum of processor intercommunication aggressiveness for OpenMP executions and inter-CPU-GPU-communication and GPU kernel invocations for OpenCL executions. For all benchmarks we take measurements across the benchmark's parallel section. We define the parallel section of an OpenCL benchmark to be the region of code that starts with the first OpenCL-related memory buffer creation and ends with 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 speed up on M2S-CGM for the Rodinia OpenMP benchmarks as shown in Fig. 2. We also compare heterogeneous-workload percentage breakdown for the Rodinia OpenCL benchmarks as shown in Fig. 3.

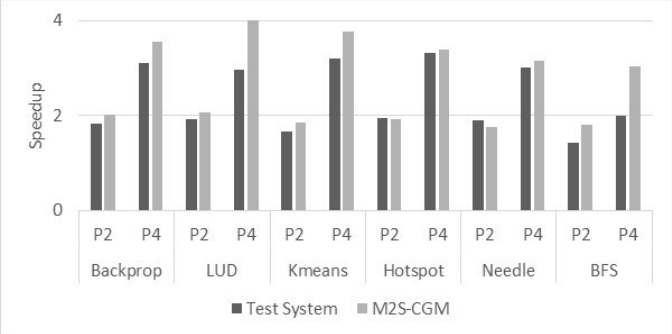

Fig. 2: Rodinia OpenMP Benchmark Results

In Fig. 2 we measure the speedup for 2 and 4 threads over the selected 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. The differences are expected and show correct simulation behavior as compared to a physical system running many other system processes in addition to the benchmarks themselves.

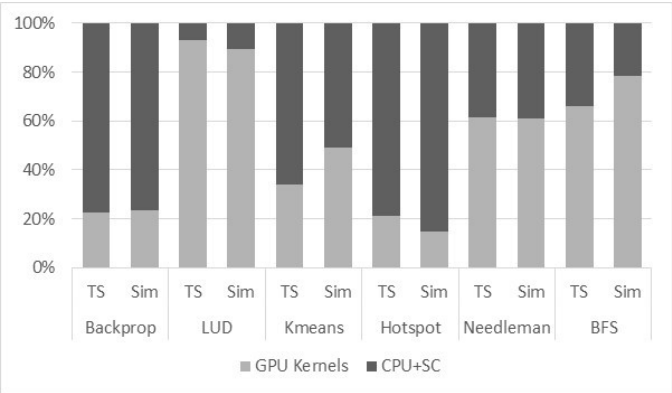

Fig. 3: Rodinia OpenCL Benchmark Results

In Fig. 3 we measure heterogeneous workload execution time and provide breakdowns of GPU kernel time, CPU time, and OpenCL related system call time. We combine CPU 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. For the OpenCL benchmarks, simulated execution time breakdown between the CPU and GPU is within 6.4% on average. 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.

5 Conclusion

In this paper we introduce M2S-CGM, a detailed architectural simulator that models the interactions between discrete CPUs and GPUs operating in heterogeneous compute environments, and provide a validation of it's multithreaded and heterogeneous system simulation capabilities. We design and implement M2S-CGM for our near-term research purposes and plan to utilize M2S-CGM in our exploration of the heterogeneous system design space. We first plan to classify select Rodinia OpenCL benchmarks in terms of their inherent heterogeneity to aid in determining how best to use them in the design of heterogeneous system experiments. We then plan to configure our select benchmarks for shared memory workloads and to conduct heterogeneous system experiments in a configurable environment where the CPU and GPU share various memory system elements and operate in a single address space.

References

[1] AMD. *AMD APP SDK*. Accessed: Jan-1-2016. URL: http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/.

[2] AMD. *AMD A-Series Desktop APUs*. Accessed: Jan-23-2017. URL: http://www.amd.com/en-us/products/graphics/desktop/7000.

[3] AMD. *AMD Graphics Cores Next (CGN) Architecture*. Tech. rep. AMD, June 2012.

[4] AMD. *AMD Radeon HD 7000 Series Graphics Cards*. Accessed: Jan-1-2016. URL: http://www.amd.com/en-us/products/processors/desktop/a-series-apu.

[5] S. Che et al. "Rodinia: A benchmark suite for heterogeneous computing". In: *2009 IEEE International Symposium on Workload Characterization (IISWC)*. Oct. 2009, pp. 44–54.

[6] Intel. *The Compute Architecture of Intel Processor Graphics Gen9*. Tech. rep. Intel, Aug. 2015.

[7] Milo M. K. Martin et al. "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset". In: *SIGARCH Comput. Archit. News* 33.4 (Nov. 2005), pp. 92–99.

[8] NEU. *Multi2Sim A Heterogeneous System Simulator*. Accessed: Jan-23-2017. URL: http://www.multi2sim.org/.

[9] J. Power et al. "gem5-gpu: A Heterogeneous CPU-GPU Simulator". In: *IEEE Computer Architecture Letters* 14.1 (Jan. 2015), pp. 34–36.

[10] Phil Rogers. "Heterogeneous System Architecture Overview". Symposium on High Performance Chips. 2013. URL: http://www.hotchips.org/archives/2010s/hc25/.

[11] Dana Schaa and Rafael Ubal. "Multi-Architecture ISA-Level Simulation of OpenCL". International Workshop on OpenCL. 2013. URL: http://www.multi2sim.org/publications.html?id=iwocl-2013.

[12] Rafael Ubal et al. "Multi2Sim: A Simulation Framework for CPU-GPU Computing". In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. PACT '12. Minneapolis, Minnesota, USA: ACM, 2012, pp. 335–344.

[13] David Wang et al. "DRAMsim: A Memory System Simulator". In: *SIGARCH Comput. Archit. News* 33.4 (Nov. 2005), pp. 100–107.

[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)*. Mar. 2013, pp. 685–688.