# gem5-gpu: A Heterogeneous CPU-GPU Simulator

Jason Power, Joel Hestness, Marc S. Orr, Mark D. Hill, David A. Wood
University of Wisconsin–Madison
E-mail: {powerjg,hestness,morr,markhill,david}@cs.wisc.edu

**Abstract**—*gem5-gpu* is a new simulator that models tightly integrated CPU-GPU systems. It builds on gem5, a modular full-system CPU simulator, and GPGPU-Sim, a detailed GPGPU simulator. *gem5-gpu* routes most memory accesses through Ruby, which is a highly configurable memory system in gem5. By doing this, it is able to simulate many system configurations, ranging from a system with coherent caches and a single virtual address space across the CPU and GPU to a system that maintains separate GPU and CPU physical address spaces. *gem5-gpu* can run most unmodified CUDA 3.2 source code. Applications can launch non-blocking kernels, allowing the CPU and GPU to execute simultaneously. We present *gem5-gpu*'s software architecture and a brief performance validation. We also discuss possible extensions to the simulator. *gem5-gpu* is open source and available at gem5-gpu.cs.wisc.edu.

**Index Terms**—Modeling techniques, Simulators, Heterogeneous (hybrid) systems, General-purpose graphics processors

## 1 Introduction

Computer architecture is transitioning from the multicore era into the heterogeneous era [9]. Many systems are shipping with integrated CPUs and graphics processing units (GPUs) [9], [5], [8]. Increasing levels of integration pose new research questions.

Historically, computer architects have used cycle-level simulators to explore and evaluate new processor designs. We leverage two mature simulators, gem5 [3] and GPGPU-Sim [2]. gem5 is a multicore full-system simulator with multiple CPU, ISA, and memory system models. Object oriented design, flexible configuration support, and its maturity make gem5 a popular tool for investigating general purpose CPUs and multicore platforms. GPGPU-Sim is a detailed general-purpose GPU (GPGPU) simulator [2]. GPGPU-Sim models GPGPU compute units (CUs)—called streaming multiprocessors by NVIDIA—and the GPU memory system.

To explore the heterogeneous system design space, we introduce the *gem5-gpu* simulator which combines the CU model from GPGPU-Sim and the CPU and memory system models from gem5. *gem5-gpu* builds on ideas used in related CPU-GPU simulators but makes different design choices. It captures interactions with execution-driven simulation rather than well-partitioned trace-driven simulation, e.g., MacSim [1]. It uses a more-detailed—therefore slower—GPU component than MV5 [6] and does not rely on the deprecated m5 simulator. It supports more flexible memory hierarchy and coherence protocols than Multi2Sim [10] or FusionSim [11] at a possible increase in simulation time. *gem5-gpu* is the only simulator with all of the following advantages:

- Detailed cache coherence model,
- Full-system simulation,
- Checkpointing,
- Tightly integrated with the latest gem5 simulator, and
- Increased extensibility of GPGPU programming model and entire system architecture.

By integrating GPGPU-Sim's CU model into gem5, *gem5-gpu* can capture interactions between a CPU and a GPU in a heterogeneous processor. In particular, GPGPU-Sim CU memory accesses flow through gem5's Ruby memory system, which enables a wide array of heterogeneous cache hierarchies and coherence protocols. *gem5-gpu* also provides a tunable DMA engine to model data transfers in configurations with separate CPU and GPU physical address spaces. Through these features *gem5-gpu* can simulate both existing and future heterogeneous processors.

*gem5-gpu* is open source and available at gem5-gpu.cs.wisc.edu.

This paper first presents GPGPU and infrastructure background (Sections 2 and 3), the design of *gem5-gpu* (Section 4), a brief validation (Section 5), and future directions (Section 6).

## 2 Heterogeneous Computing

General-purpose GPU (GPGPU) computing is the practice of offloading computation to run on programmable GPUs. Applications commonly targeted to GPGPU computing include data-parallel image processing, scientific, and numerical algorithms, though

• *Manuscript submitted: 15-Nov-2013. Manuscript accepted: 12-Dec-2013. Final manuscript received: 18-Dec-2013*

there is a trend toward more irregularly parallel workloads, such as graph analysis.

Work units offloaded to the GPU are called kernels. Kernels can be structured to execute thousands of threads on the GPU in a single-instruction, multiple-thread (SIMT) fashion. In systems with separate CPU and GPU address spaces, such as discrete GPUs, data is explicitly copied between the GPU address space and the CPU address space.

Writing an application to take advantage of a GPU requires user-level calls to a GPGPU application programming interface (runtime), and this runtime interfaces with a kernel-level driver that controls the GPU device. Currently, the most popular GPGPU runtimes are CUDA and OpenCL.

There is a trend towards simplifying the programming model for GPGPU computing. The heterogeneous system architecture (HSA) foundation, a consortium of companies and universities, has announced future support for heterogeneous uniform memory access (hUMA) [9]. hUMA provides the programmer with a shared virtual address space between the CPU and GPU. Additionally, implementations of hUMA provide the CPU and GPU with a coherent view of the virtual address space. NVIDIA supports a similar construct with unified virtual addressing (UVA) [7]. A major goal in the development of *gem5-gpu* is to support these future programming models and architectural directions in a flexible and extensible way.

## 3 The Giant's Shoulders

### 3.1 gem5

The gem5 simulation infrastructure (gem5.org) is a community-focused, modular, system modeling tool, developed by numerous universities and industry research labs [3]. gem5 includes multiple CPU, memory system, and ISA models. It provides two execution modes, (1) system call emulation, which can run user-level binaries using emulated system calls, and (2) full-system, which models all necessary devices to boot and run unmodified operating systems. Finally, gem5 also supports checkpointing the state of the system, which allows simulations to jump to the region of interest.

Two specific features of gem5 make it particularly well-suited for developing a heterogeneous CPU-GPU simulator. First, gem5 provides several mechanisms for modular integration of new architectural components. When new components need to communicate through the memory system, they can leverage gem5's flexible port interface for sending and receiving messages. Additionally, the gem5 EXTRAS interface can be used to specify external code that is compiled into the gem5 binary. This interface makes it simple to add and remove complex components from gem5's infrastructure.

Second, gem5 includes the detailed cache and memory simulator, Ruby. Ruby is a flexible infrastructure built on the domain specific language, SLICC, which is used to specify cache coherence protocols. Using Ruby, a developer can expressively define cache hierarchies and coherence protocols, including those expected in emerging heterogeneous processors.

Currently, gem5 includes no model for GPUs.

### 3.2 GPGPU-Sim

GPGPU-Sim is a detailed GPGPU simulator (gpgpu-sim.org) backed by a strong publication record [2]. It models the compute architecture of modern NVIDIA graphics cards. GPGPU-Sim executes applications compiled to PTX (NVIDIA's intermediate instruction set) or disassembled native GPU machine code. GPGPU-Sim models the functional and timing portions of the compute pipeline including the thread scheduling logic, highly-banked register file, special function units, and memory system. GPGPU-Sim includes models for all types of GPU memory as well as caches and DRAM.

GPGPU applications can access a multitude of memory types. Global memory is the main data store where most data resides, similar to the heap in CPU applications. It is accessed with virtual addresses and is cached on chip. Other GPU-specific memory types include constant, used to handle GPU read-only data; scratchpad, a software-managed, explicitly addressed and low-latency in-core cache; local, mostly used for spilling registers; parameter, used to store compute kernel parameters; instruction, used to store the kernel's instructions; and texture, a graphics-specific, explicitly addressed cache.

GPGPU-Sim consumes mostly unmodified GPGPU source code that is linked to GPGPU-Sim's custom GPGPU runtime library. The modified runtime library intercepts all GPGPU-specific function calls and emulates their effects. When a compute kernel is launched, the GPGPU-Sim runtime library initializes the simulator and executes the kernel in timing simulation. The main simulation loop continues executing until the kernel has completed before returning control from the runtime library call. GPGPU-Sim is a functional-first simulator; it first functionally executes all instructions, then feeds them into the timing simulator.

GPGPU-Sim has some limitations when modeling heterogeneous systems:

- No host CPU timing model
- No timing model for host-device copies
- Rigid cache model
- No way to model host-device interactions

Because of these limitations, researchers interested in exploring a hybrid CPU-GPU chip as a heterogeneous compute platform cannot rely on GPGPU-Sim alone.

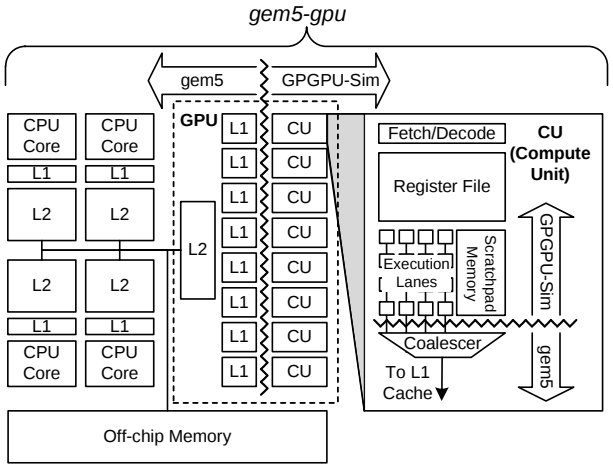


Fig. 1. Overview of *gem5-gpu* architecture with an example configuration.

## 4 *gem5-gpu* ARCHITECTURE

Figure 1 shows one example architecture *gem5-gpu* can simulate: a four core CPU and an eight CU GPU integrated on the same chip. The number of CPUs, CUs, and topology connecting them is fully configurable. Two on-chip topologies that *gem5-gpu* provides out of the box are a shared and a split memory hierarchy (i.e., integrated and discrete GPUs, respectively).

Many CUs make up the GPU, each of which has fetch/decode logic, a large register file, and many (usually 32 or 64) execution lanes. When accessing global memory, each lane sends its address to the coalescer, which merges memory accesses to the same cache block. The GPU may also contain a cache hierarchy that stores data from recent global memory accesses.

### 4.1 gem5 $\longleftrightarrow$ GPGPU-Sim Interface

One of our goals is to have a clean interface between gem5 and GPGPU-Sim. Although there are many possible options, we chose the memory interface, as shown in Figure 1. We add a single pseudo-instruction to gem5 to facilitate calls into the simulator for DMA engine and GPU functionality. Then, *gem5-gpu* routes general-purpose memory instructions—accesses to the global address space—from GPGPU-Sim to Ruby through gem5's port interface.

### 4.2 Memory System Modeling

*gem5-gpu* uses Ruby to model both the function and timing of most CU memory accesses. The load-store pipeline is modeled in gem5, including the coalescing, virtual address translation, and cache arbitration logic. By using the port interface in gem5, *gem5-gpu* has the flexibility to vary the number of execution lanes, number of CUs, cache hierarchy, etc. and incorporate other GPU models in the future.

Currently, GPGPU-Sim issues only general-purpose memory instructions to gem5, including accesses to global and constant memory. We leverage GPGPU-Sim to model memory operations to scratchpad and parameter memory. Texture and local memory are not currently supported although they require straightforward simulator augmentation.

*gem5-gpu* supports a shared virtual address space between the CPU and GPU (i.e. the GPU using the CPU page table for virtual to physical translations). Alternatively, through a configuration option, *gem5-gpu* models separate GPU and CPU physical address spaces.

### 4.3 Detailed Cache Coherence Models

*gem5-gpu* leverages gem5's cache coherence modeling language, SLICC. When configuring *gem5-gpu*, any cache coherence protocol can be used, including the multitude that are currently distributed with gem5. However, these included protocols assume a homogeneous cache topology.

To augment these for heterogeneous computing, *gem5-gpu* adds a family of heterogeneous cache coherence protocols: *MOESI_hsc* (heterogeneous system coherence with MOESI states). *MOESI_hsc* uses a MOESI protocol for all of the CPU caches included in the system. For the GPU caches, we add an L2 cache controller that provides coherence between the GPU and CPU L2 caches. *MOESI_hsc* models the GPU L1 cache similar to current GPU cache architectures: write-through and only valid and invalid states. Additionally, the GPU L1 cache may contain stale data that is flushed at synchronization points and kernel boundaries.

*gem5-gpu* also includes a split version of *MOESI_hsc* that models architectures with separate CPU and GPU physical address spaces. When this model is used, communication between the CPU and GPU requires explicit memory copies through the DMA engine.

In addition to providing detailed cache coherence models, *gem5-gpu* can use any of the network topologies provided by gem5 (e.g., mesh, torus, crossbar). *gem5-gpu*'s default configuration uses the cluster topology to divide the CPU and GPU into two clusters. Accesses from each cluster goes through a common interconnect to the directory and memory controllers.

### 4.4 Application Programming Interface

To avoid the complexity of implementing or interfacing existing GPGPU drivers and runtimes, *gem5-gpu* provides a slim runtime and driver emulation. Similar to GPGPU-Sim, *gem5-gpu* runs unmodified GPGPU applications by linking against the *gem5-gpu* GPGPU runtime library. When a user application calls a GPGPU runtime function, gem5 pseudo-instructions execute to make up-calls into the simulator. This organization is flexible and extensible, making it convenient to add and test new features and integrate new GPU models.

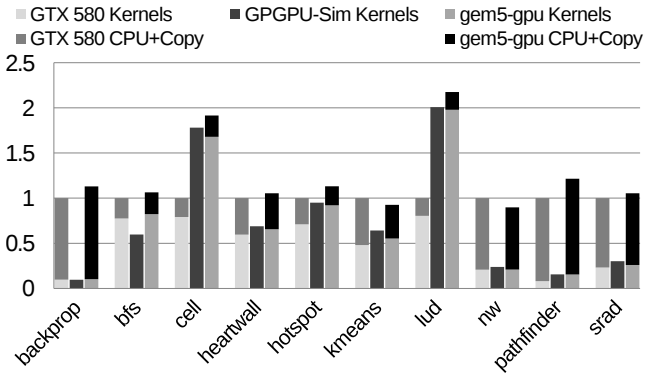


Fig. 2. Run times normalized to NVIDIA GTX 580.

## 5 Performance Validation

We performed *gem5-gpu* validation with a focus on global memory performance using memory microbenchmarks and a subset of the Rodinia benchmarks [4]. For these tests, we configure both GPGPU-Sim v3.0.2 and *gem5-gpu* to model the NVIDIA GTX 580 discrete GPU system parameters as closely as possible. For memory access microbenchmarks, including coalescing, cache, off-chip latency and bandwidth, we find that *gem5-gpu* performance closely matches the GTX 580 discrete GPU in all cases.

Figure 2 presents a region of interest (ROI) run time comparison for Rodinia benchmarks running in GPGPU-Sim and *gem5-gpu* normalized to the GTX 580. In addition to GPU kernel run time, the plot includes CPU execution and memory copy time between the CPU and the GPU (GPGPU-Sim does not provide a timing model for these). In most cases, *gem5-gpu* ROI run time is within 22% of the GTX 580.

In all cases, *gem5-gpu* kernel run time is highly correlated to stand-alone GPGPU-Sim. The small differences are attributed to different memory system models. The primary performance differences between these and the GTX 580 are due to GPGPU-Sim's CU model fidelity. In particular, benchmarks such as cell, lud, and pathfinder show increased kernel run time due to elevated latencies in the GPGPU-Sim CU for register handling and the use of the PTX intermediate representation instead of object code. These CU modeling issues have been addressed in more recent versions of GPGPU-Sim, and we plan to pull these changes into *gem5-gpu* in the future.

## 6 Future Work

Since the changes required to gem5 are minimal, we will work to more closely integrate with gem5 to ease the use of *gem5-gpu*. Additionally, as both GPGPU-Sim and gem5 evolve, *gem5-gpu* will take advantage of new features and bug fixes by leveraging these open source projects.

Although we have found *gem5-gpu* to be a helpful tool that is used both inside and outside our research group, there are some limitations we plan to address. First, *gem5-gpu* is currently limited to the x86 ISA. We plan to extend *gem5-gpu* to support the ARM ISA in the future. *gem5-gpu* currently only supports CUDA, NVIDIA's GPGPU runtime. OpenCL is closely related to CUDA and minimal modifications to *gem5-gpu* are required to enable support for OpenCL. Finally, we plan to support other GPU models beyond GPGPU-Sim in the future. The interface between *gem5-gpu* and GPGPU-Sim is extensible enabling other GPU models to be easily added.

## 7 Conclusions

In this paper we presented an overview of *gem5-gpu*, a unique heterogeneous simulator that combines a state-of-the-art full-system CPU simulator and a GPGPU simulator, gem5 and GPGPU-Sim, respectively. *gem5-gpu* provides architects a flexible system to experiment with both current and future heterogeneous architectures.

*gem5-gpu* is open source and available at gem5-gpu.cs.wisc.edu. More information can also be found by joining the *gem5-gpu* mailing list: gem5-gpu-dev@googlegroups.com. We look forward to working with the architecture community to improve *gem5-gpu* and incorporate additional features and ideas.

## 8 Acknowledgments

We thank both the gem5 and GPGPU-Sim communities. This work is supported in part with NSF grants CNS-1117280, CCF-1218323, and CNS-1302260. The views expressed herein are not necessarily those of the NSF. Professors Hill and Wood have significant financial interests in AMD.

## References

[1] "Macsim," https://code.google.com/p/macsim/.
[2] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *ISPASS*, 2009.
[3] 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," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, 2011.
[4] 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 *IISWC*, 2009, pp. 44–54.
[5] "OpenCL Programmability on 4th Generation Intel Core Processors," http://software.intel.com/sites/billboard/article/opencl-programmability-4th-generation-intel-core-processors, June 2013.
[6] J. Meng and K. Skadron, "A reconfigurable simulator for large-scale heterogeneous multicore architectures," in *ISPASS 2011*, 2011.
[7] NVIDIA, "NVIDIA CUDA C Programming Guide Ver. 4.0," 2011.
[8] "NVIDIA Brings Kepler, Worlds Most Advanced Graphics Architecture, to Mobile Devices," http://blogs.nvidia.com/blog/2013/07/24/kepler-to-mobile/, July 2013.
[9] P. Rogers, "Heterogeneous System Architecture Overview," in *Hot Chips 25*, 2013.
[10] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for CPU-GPU computing," in *PACT '12*, 2012.
[11] V. Zakharenko, T. Aamodt, and A. Moshovos, "Characterizing the performance benefits of fused CPU/GPU systems using fusionsim," in *DATE '13*, 2013.