The gem5 Simulator

Nathan Binkert[1], Bradford Beckmann[2],
Gabriel Black[3], Steven K. Reinhardt[2], Ali Saidi[4], Arkaprava Basu[5], Joel Hestness[6],
Derek R. Hower[5], Tushar Krishna[7], Somayeh Sardashti[5], Rathijit Sen[5], Korey Sewell[8],
Muhammad Shoaib[5], Nilay Vaish[5], Mark D. Hill[5], and David A. Wood[5]

http://gem5.org

Abstract

The gem5 simulation infrastructure is the merger of the best aspects of the M5 [4] and GEMS [9] simulators. M5 provides a highly configurable simulation framework, multiple ISAs, and diverse CPU models. GEMS complements these features with a detailed and flexible memory system, including support for multiple cache coherence protocols and interconnect models. Currently, gem5 supports most commercial ISAs (ARM, ALPHA, MIPS, Power, SPARC, and x86), including booting Linux on three of them (ARM, ALPHA, and x86).

The project is the result of the combined efforts of many academic and industrial institutions, including AMD, ARM, HP, MIPS, Princeton, MIT, and the Universities of Michigan, Texas, and Wisconsin. Over the past ten years, M5 and GEMS have been used in hundreds of publications and have been downloaded tens of thousands of times. The high level of collaboration on the gem5 project, combined with the previous success of the component parts and a liberal BSD-like license, make gem5 a valuable full-system simulation tool.

1 Introduction

Computer architecture researchers commonly use software simulation to prototype and evaluate their ideas. As the computer industry continues to advance, the range of designs being considered increases. On one hand, the emergence of multicore systems and deeper cache hierarchies has presented architects with several new dimensions of exploration. On the other hand, researchers need a flexible simulation framework that can evaluate a wide diversity of designs and support rich OS facilities including IO and networking.

Computer architecture researchers also need a simulation framework that allows them to collaborate with their colleagues in both industry and academia. However, a simulator's licensing terms and code quality can inhibit that collaboration. Some open source software licenses can be too restrictive, especially in an industrial setting, because they require publishing any simulator enhancements. Furthermore, poor code quality and the lack of modularity can make it difficult for new users to understand and modify the code.

The gem5 simulator overcomes these limitations by providing a flexible, modular simulation system that is capable of evaluating a broad range of systems and is widely available to all researchers. This infrastructure provides flexibility by offering a diverse set of CPU models, system execution modes, and memory system models. A commitment to modularity and clean interfaces allows researchers to focus on a particular aspect of the code without understanding the entire code base. The BSD-based license makes the code available to all researchers without awkward legal restrictions.

This paper provides a brief overview of gem5's goals, philosophy, capabilities, and future work along with pointers to sources of additional information.

2 Overall Goals

The overarching goal of the gem5 simulator is to be a community tool focused on architectural modeling. Three key aspects of this goal are flexible modeling to appeal to a broad range of users, wide availability and utility to

---

[1]Hewlett-Packard Labs, Palo Alto, Cal.
[2]Advanced Micro Devices, Inc., Bellevue, Wash.
[3]Google, Inc., Mountain View, Cal.
[4]ARM, Inc., Austin, Tex.
[5]University of Wisconsin, Madison, Wisc.
[6]University of Texas, Austin, Tex.
[7]Massachusetts Institute of Technology, Cambridge, Mass.
[8]University of Michigan, Ann Arbor, Mich.

Processor		Memory System		
CPU Model	System Mode	Classic	Ruby	
			Simple	Garnet
Atomic Simple	SE	Speed		
	FS			
Timing Simple	SE			
	FS			
In-Order	SE			
	FS			
O3	SE			Accuracy
	FS			

Figure 1: **Speed vs. Accuracy Spectrum.**

the community, and high level of developer interaction to foster collaboration.

2.1 Flexibility

Flexibility is a fundamental requirement of any successful simulation infrastructure. For instance, as an idea evolves from a high-level concept to a specific design, architects need a tool that can evaluate systems at various levels of detail, balancing simulation speed and accuracy. Different types of experiments may also require different simulation capabilities. For example, a fine-grain clock gating experiment may require a detailed CPU model, but modeling multiple cores is unnecessary. Meanwhile, a highly scalable interconnect model may require several CPUs, but those CPUs don't need much detail. Also, by using the same infrastructure over time, an architect will be able to get more done more quickly with less overhead.

The gem5 simulator provides a wide variety of capabilities and components which give it a lot of flexibility. These vary in multiple dimensions and cover a wide range of speed/accuracy trade offs as shown in Figure 1. The key dimensions of gem5's capabilities are:

- **CPU Model.** The gem5 simulator currently provides four different CPU models, each of which lie at a unique point in the speed-vs.-accuracy spectrum. *AtomicSimple* is a minimal single IPC CPU model, *TimingSimple* is similar but also simulates the timing of memory references, *InOrder* is a pipelined, in-order CPU, and *O3* is a pipelined, out-of-order CPU model. Both the O3 and InOrder models are "execute-in-execute" designs [4].

- **System Mode.** Each execution-driven CPU model can operate in either of two modes. *System-call Emulation (SE)* mode avoids the need to model devices or an operating system (OS) by emulating most system-level services. Meanwhile, *Full-System (FS)* mode executes both user-level and kernel-level instructions and models a complete system including the OS and devices.

- **Memory System.** The gem5 simulator includes two different memory system models, *Classic* and *Ruby*. The Classic model (from M5) provides a fast and easily configurable memory system, while the Ruby model (from GEMS) provides a flexible infrastructure capable of accurately simulating a wide variety of cache coherent memory systems.

The gem5 simulator can also execute workloads in a number of ISAs, including today's most common ISAs, x86 and ARM. This significantly increases the number of workloads and configurations gem5 can simulate.

Section 4 provides a more detailed discussion of these capabilities.

2.2 Availability

There are several types of gem5 user; each has different goals and requirements. These include academic and corporate researchers, engineers in industry, and undergraduate and graduate students. We want the gem5 simulator to be broadly available to each of these types of user. The gem5 license (based on BSD) is friendly both to corporate users, since businesses need not fear being forced to reveal proprietary information, and to to academics, since they retain their copyright and thus get credit for their contributions.

2.3 High level of collaboration

Full-system simulators are complex tools. Dozens of person-years of effort have gone into the gem5 simulator, developing both the infrastructure for flexible modeling and the numerous detailed component models. By being an open source, community-led project, we can leverage the work of many researchers, each with different specialties. The gem5 community is very active and leverages a number of collaborative technologies to foster gem5 use and development, including mailing lists, a wiki, web-based patch reviews, and a publicly accessible source repository.

3 Design Features

This section focuses on a few key aspects of gem5's implementation: pervasive object orientation, Python integration, domain-specific languages, and use of standard

interfaces. While most of these features are simply good software engineering practice, they are all particularly useful for designing simulators.

3.1 Pervasive Object-Oriented Design

Flexibility is an important goal of the gem5 simulator and key aspect of its success. Flexibility is primarily achieved through object-oriented design. The ability to construct configurations from independent, composable objects leads naturally to advanced capabilities such as multi-core and multi-system modeling.

All major simulation components in the gem5 simulator are SimObjects and share common behaviors for configuration, initialization, statistics, and serialization (checkpointing). SimObjects include models of concrete hardware components such as processor cores, caches, interconnect elements and devices, as well as more abstract entities such as a workload and its associated process context for system-call emulation.

Every SimObject is represented by two classes, one in Python and one in C++ which derive from `SimObject` base classes present in each language. The Python class definition specifies the SimObject's parameters and is used in script-based configuration. The common Python base class provides uniform mechanisms for instantiation, naming, and setting parameter values. The C++ class encompasses the SimObject's state and remaining behavior, including the performance-critical simulation model.

3.2 Python Integration

The gem5 simulator derives significant power from tight integration of Python into the simulator. While 85% of the simulator is written in C++, Python pervades all aspects of its operation. As mentioned in Section 3.1, all SimObjects are reflected in both Python and C++. The Python aspect provides initialization, configuration, and simulation control. The simulator begins executing Python code almost immediately on start-up; the standard `main()` function is written in Python, and all command-line processing and startup code is written in Python.

3.3 Domain-Specific Languages

In situations that require significant flexibility in performing a specialized task, domain-specific languages (DSLs) provide a powerful and concise way to express a variety of solutions by leveraging knowledge and idioms common to that problem space. The gem5 environment provides two domain-specific languages, one for specifying instruction sets (inherited from M5) and one for specifying cache coherence protocols (inherited from GEMS).

ISA DSL. The gem5 ISA description language unifies the decoding of binary instructions and the specification of their semantics. The gem5 CPU models achieve ISA independence by using a common C++ base class to describe instructions. Derived classes override virtual functions like `execute()` to implement opcodes, such as `add`. Instances of these derived classes represent specific machine instructions, such as `add r1,r2,r3`. Implementing a specific ISA thus requires a set of C++ declarations for these derived classes, plus a function that takes a machine instruction and returns an instance of one of the derived classes that corresponds to that instruction.

The ISA description language allows users to specify this required C++ code compactly. Part of the language allows the specification of class templates (more general than C++ templates) that cover broad categories of instructions, such as register-to-register arithmetic operations. Another portion of the language provides for the specification of a decode tree that concisely combines opcode decoding with the creation of specific derived classes as instances of the previously defined templates.

While the original ISA description language targeted RISC architectures such as the Alpha ISA, it has been significantly extended to cope with complex variable-length ISAs, particularly x86, and ISAs with complex register semantics like SPARC. These extensions include a microcode assembler, a predecoder, and multi-level register index translation. These extensions are discussed in more detail in a recent book chapter [5].

Cache Coherence DSL. SLICC is a domain-specific language that gives gem5 the flexibility to implement a wide variety of cache coherence protocols. Essentially, SLICC defines the cache, memory, and DMA controllers as individual per-memory-block state machines that together form the overall protocol. By defining the controller logic in a higher-level language, SLICC allows different protocols to incorporate the same underlying state transition mechanisms with minimal programmer effort.

The gem5 version of SLICC is very similar to the prior GEMS version of SLICC [9]. Just like the prior version, gem5 SLICC defines protocols as a set of states, events, transitions, and actions. Within the specification files, individual transition statements define the valid combinations and actions within each transition specify the operations that must be performed. Also similar to the previous version, gem5 SLICC ties the state machine-specific logic to protocol-independent components such as cache memories and network ports.

While gem5 SLICC contains several similarities to its predecessor design, the language does include several enhancements. First, the language itself is now implemented in Python rather than C++, making it easier to read and edit. Second, to adhere to the gem5 SimObject structure, all configuration parameters are specified as input parameters and gem5 SLICC automatically generates the appropriate C++ and Python files. Finally, gem5 SLICC allows local variables to simplify programming and improve performance.

3.4 Standard Interfaces

Standard interfaces are fundamental to object-oriented design. Two central interfaces are the port interface and the message buffer interface.

Ports are one of the interfaces used to connect two memory objects together in gem5. In the Classic memory system, the ports interface connects all memory objects including CPUs to caches, caches to busses, and busses to devices and memories. Ports support three mechanisms for accessing data (timing, atomic, and functional) and an interface for things like determining topology and debugging. Timing mode is used to model the detailed timing of memory accesses. Requests are made to the memory system by sending messages, and responses are expected to return asynchronously via other messages. Atomic mode is used to get some timing information, but is not message-oriented. When an atomic call is made (via a function call), the state change for the operation is performed synchronously. This has higher performance but gives up some accuracy because message interactions are not modeled. Finally, functional accesses update the simulator state without changing any timing information. These are generally used for debugging, system-call emulation, and initialization.

Ruby utilizes the ports interface to connect to CPUs and devices, and adds message buffers to connect to Ruby objects internally. Message buffers are similar to ports in that they provide a standard communication interface. However, message buffers differ in some subtle ways with regards to message typing and storage. In the future, ports and message buffers may evolve into a unified interface.

4 Simulation Capabilities

The gem5 simulator has a wide range of simulation capabilities ranging from the selection of ISA, CPU model, and coherence protocol to the instantiation of interconnection networks, devices and multiple systems. This section describes some of the different options available in these categories.

ISAs. The gem5 simulator currently supports a variety of ISAs including Alpha, ARM, MIPS, Power, SPARC, and x86. The simulator's modularity allows these different ISAs to plug into the generic CPU models and the memory system without having to specialize one for the other. However, not all possible combinations of ISAs and other components are currently known to work. An up-to-date list can be found on the gem5 website.

Execution Modes. The gem5 simulator can operate in two modes: System-call Emulation (SE) and Full-System (FS). In SE mode, gem5 emulates most common system calls (e.g. `read()`). Whenever the program executes a system call, gem5 traps and emulates the call, often by passing it to the host operating system. There is currently no thread scheduler in SE mode, so threads must be statically mapped to cores, limiting its use with multi-threaded applications. The SPEC CPU benchmarks are often run in SE mode.

In FS mode, gem5 simulates a bare-metal environment suitable for running an OS. This includes support for interrupts, exceptions, privilege levels, I/O devices, etc. Because of the additional complexity and completeness required, not all ISAs current support FS mode.

Compared to SE mode, FS mode improves both the simulation accuracy and variety of workloads that gem5 can execute. While SPEC CPU benchmarks can be run in SE mode, running them in FS mode will provide more realistic interactions with the OS. Workloads that require many OS services or I/O devices may only be run in FS mode. For example, because a web server relies on the kernel's TCP/IP protocol stack and a network interface to send and receive requests and a web browser requires a X11 server and display adapter to visualize web pages these workloads must be run is FS mode.

CPU Models. The gem5 simulator supports four different CPU models: AtomicSimple, TimingSimple, InOrder, and O3. AtomicSimple and TimingSimple are non-pipelined CPU models that attempt to fetch, decode, execute and commit a single instruction on every cycle. The AtomicSimple CPU is a minimal, single IPC CPU which completes all memory accesses immediately. This low overhead makes AtomicSimple a good choice for simulation tasks such as fast-forwarding. Correspondingly, the TimingSimple CPU also only allows one outstanding memory request at a time, but the CPU does model the timing of memory accesses.

The InOrder model is an "execute-in-execute" CPU model emphasizing instruction timing and simulation ac-

curacy with an in-order pipeline. InOrder can be configured to model different numbers of pipeline stages, issue width, and numbers of hardware threads.

Finally, the O3 CPU is a pipelined, out-of-order model that simulates dependencies between instructions, functional units, memory accesses, and pipeline stages. Parameterizable pipeline resources such as the load/store queue and reorder buffer allow O3 to simulate superscalar architectures and CPUs with multiple hardware threads (SMT). The O3 model is also "execute-in-execute", meaning that instructions are only executed in the execute stage after all dependencies have been resolved.

Cache Coherence Protocols. SLICC enables gem5's Ruby memory model to implement many different types of invalidation-based cache coherence protocols, from snooping to directory protocols and several points in between. SLICC separates cache coherence logic from the rest of the memory system, providing the necessary abstraction to implement a wide range of protocol logic. Similar to its GEMS predecessor [9], SLICC performs all operations at a cache-block granularity. The word-level granularity required by update-based protocols is not currently supported. This limitation has not been a issue so far because invalidation-based protocols dominate the commercial market. Specifically, gem5 SLICC currently models a broadcast-based protocol based on the AMD Opteron^{TM} [7], as well as a CMP directory protocol [10].

Not only is SLICC flexible enough to model different types of protocols, but it also simulates them in sufficient depth to model detailed timing behavior. Specifically, SLICC allows specifying transient states within the individual state machines as cache blocks move from one base state to another. SLICC also includes separate virtual networks (a.k.a. network message classes) so message dependencies and stalls can be properly modeled. Using these virtual networks, the SLICC-generated controllers connect to the interconnection network.

Interconnection Networks. The Ruby memory model supports a vast array of interconnection topologies and includes two different network models. In essence, Ruby can create any arbitrary topology as long as it is composed of point-to-point links. A simple Python file declares the connections between components and shortest path analysis is used to create the routing tables. Once Ruby creates the links and routing tables, it can implement the resulting network in one of two ways.

The first Ruby network model is referred to as the *Simple* network. The Simple network models link and router latency as well as link bandwidth. However, the Simple network does not model router resource contention and flow control. This model is great for experiments that

require Ruby's detailed protocol modeling but that can sacrifice detailed network modeling for faster simulation.

The second Ruby network model is the *Garnet* network model [1]. Unlike the simple network, Garnet models the router micro-architecture in detail, including all relevant resource contention and flow control timing. This model is suitable for on-chip network studies.

Devices. The gem5 simulator supports several I/O devices ranging from simple timers to complex network interface controllers. Base classes are available that encapsulates common device interfaces such as PCI to avoid code duplication and simplify implementing new devices. Currently implemented models includes NICs, an IDE controller, a frame buffer, DMA engines, UARTs, and interrupt controllers.

Modeling Multiple Systems. Because of the simulator's object oriented design it also supports simulating multiple complete systems. This is done by instantiating another set of objects (CPU, memory, I/O devices, etc.). Generally, the user connects the systems via the network interfaces described above to create a client/server pair that communicate over TCP/IP. Since all the simulated systems are tightly coupled within gem5 the results of multi-system simulation is still deterministic.

5 Future Work

While gem5 is a highly capable simulator, there is always a desire for additional features and other improvements. A few of the efforts underway or under consideration include:

- *A first-class power model.* While external power models such as Orion [6] and McPAT [8] have been used with GEMS and M5, we are working on a more comprehensive, modular, and integrated power model for gem5.

- *Full cross-product ISA/CPU/memory system support.* The modularity and flexibility of gem5 enables a wide variety of combinations of ISAs, CPU models, and memory systems, as illustrated in Figure 1, each of which can be used in SE or FS mode. Because each component model must support the union of all features required by any ISA in any mode, particular component models do not always work in every conceivable circumstance. We continue to work to eliminate these inconsistencies.

- *Parallelization.* To address the inherent performance limitations of detailed simulation and leverage the ubiquity of multi-core systems, we have been refactoring

gem5's internal event system to support parallel discrete event simulation [11].

- *Checkpoint import.* Although gem5's simple CPU models are much faster than their detailed counterparts, they are still considerably slower than binary translation-based emulators such as QEMU [3] and SimNow[TM] [2]. Rather than duplicating the enormous effort of developing a binary translation capability within gem5, we plan to enable the transfer of state checkpoints from these emulators into gem5. Users will be able to fast-forward large workloads to interesting points using these high-performance alternatives, then simulate from those points in gem5. Even higher performance may be possible by using a hardware virtual machine environment such as KVM[1] rather than binary translation.

6   User Resources

All gem5 simulator documentation and information is available at the website http://www.gem5.org. The website includes instructions on how to check out, build, and run the gem5 simulator, as well as how to download supplemental support files like OS binaries and disk images.

The gem5 user community is active and communicates through three mailing lists: (1) the *announce* mailing list is used to announce significant modifications or achievements; (2) the *user* mailing list is used for general discussions about gem5 and for questions about how to use it; and (3) the *dev* mailing list is for discussions regarding mainline gem5 development.

Developers will find support resources in the form of systems for revision control, bug tracking, code reviews, and code browsing. All of these can be accessed through the main website.

We encourage you to visit the web site, subscribe to the mailing lists, and help us make gem5 a valuable community resource.

Acknowledgements

References

[1] Agarwal, N., Krishna, T., Peh, L.-S., and Jha, N. K. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (Apr. 2009), pp. 33–42.

[2] Barnes, B., and Slice, J. SimNow: A fast and functionally accurate AMD X86-64 system simulator. Tutorial at the IEEE International Workload Characterization Symposium, 2005.

[3] Bellard, F. QEMU, a fast and portable dynamic translator. In *Proceedings of the USENIX Annual Technical Conference* (2005), pp. 41–46.

[4] Binkert, N. L., Dreslinski, R. G., Hsu, L. R., Lim, K. T., Saidi, A. G., and Reinhardt, S. K. The M5 Simulator: Modeling Networked Systems. *IEEE Micro 26*, 4 (Jul/Aug 2006), 52–60.

[5] Black, G., Binkert, N., Reinhardt, S. K., and Saidi, A. *Processor and System-on-Chip Simulation.* Springer, 2010, ch. 5, "Modular ISA-Independent Full-System Simulation".

[6] Kahng, A. B., Li, B., Peh, L.-S., and Samadi, K. ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2009), pp. 423–428.

[7] Keltcher, C. N., McGrath, K. J., Ahmed, A., and Conway, P. The AMD Opteron Processor for

---

[1]http://linux-kvm.org

Multiprocessor Servers. *IEEE Micro 23*, 2 (Mar/Apr 2003), 66–76.

[8] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. Mc-PAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (2009), pp. 469–480.

[9] Martin, M. M. K., Sorin, D. J., Beckmann, B. M., Marty, M. R., Xu, M., Alameldeen, A. R., Moore, K. E., Hill, M. D., and Wood, D. A. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News 33*, 4 (2005), 92–99.

[10] Marty, M. R., Bingham, J. D., Hill, M. D., Hu, A. J., Martin, M. M. K., and Wood, D. A. Improving multiple-CMP systems using token coherence. In *Proceedings of the 11th Annual International Symposium on High-Performance Computer Architecture (HPCA)* (2005), pp. 328–339.

[11] Reinhardt, S. K., Hill, M. D., Larus, J. R., Lebeck, A. R., Lewis, J. C., and Wood, D. A. The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (1993), pp. 48–60.