The gem5 Simulator: Version 20.0+

A new era for the open-source computer architecture simulator

JASON LOWE-POWER, University of California, Davis
BOBBY R. BRUCE, University of California, Davis

(A sentence about gem5 at a high level.) (A sentence about all the features of gem5.) The gem5 simulator has been under active development over the last nine years since the original gem5 paper was published. In this time, there have been over 7500 commits to the codebase from over 250 unique contributors which have improved the simulator by adding new features, fixing bugs, and increasing the code quality. Due to its popularity, the gem5 community has instituted a new meritocratic governance model to encourage community-centric contributions. (Come back to this.)

ACM Reference format:
Jason Lowe-Power and Bobby R. Bruce. 2016. The gem5 Simulator: Version 20.0+. 1, 1, Article 1 (January 2016), 9 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

This is a placeholder for the introduction text.

Dummy citation. Having at least one citation keeps bibtex happy [3].

1 MAJOR CHANGES IN GEM5-20

This section contains descriptions of some of the major changes to gem5. There are too many changes to list. There were XXXX commits since gem5 was released. It is likely that major changes are missing.

1.1 Learning gem5[1]

The gem5 simulator has a steep learning curve. Not only do new users have to navigate the 100s of different models, but they also have to understand the core of the simulation framework. Most of the time, using gem5 in research means *modifying* the simulator to change or add new models. We found that this steep learning curve was one of the biggest impediments to productively using gem5. There was anecdotal evidence that it would take new users *years* to learn to use gem5 effectively [6]. Additionally, the only way to learn parts of gem5 was to work with a senior graduate student or to intern at a company and pick up the knowledge "on the job". Many parts of gem5 were not documented except as the source code.

Learning gem5 reduces the knowledge gap between new users and experienced gem5 developers. Learning gem5 takes a bottom up approach to teaching new users the internals of gem5. There are currently three parts of Learning gem5, "Getting Started", "Modifying and Extending", and "Modeling Cache Coherence with Ruby". Each part walks the reader through a step-by-step coding example starting from the simplest possible design up to a more realistic example. By explaining the thought process behind each step, the

[1] By Jason Lowe-Power

reader gets a similar experience to working alongside an experienced gem5 developer. Learning gem5 includes documentation on the gem5 website[2] and source code in the gem5 repository for these simple ground-up models.

Looking forward, we will be significantly expanding the areas of the simulator covered by Learning gem5 and creating a gem5 "summer school". This "summer school" will mainly be an online class (e.g., Coursera), but we hope to have in-person versions of the class as well. These classes will also be the basis of gem5 Tutorials held with major computer architecture and other related conferences.

1.2 Testing in gem5[3]

Heard back, waiting for the text.

1.3 Updating Guest¡-¿Simulator APIs[4]

Haven't heard anything back, yet.

1.4 SystemC Integration

While the open and configurable architecture of gem5 is of particular interest in academia, the industry's main tool for virtual prototyping is SystemC Transaction Level Modelling (TLM) [1]. Many hardware vendors provide SystemC TLM models of their IP and there are tools, such as Synopsys Platform Architect[5], that assist in building a virtual system and analyzing it. Also, many research projects use SystemC TLM, as they benefit from the rich ecosystem of accurate of-the-shelf models of real hardware components. However, there is a lack of accurate and modifiable CPU models in SystemC since the model providers want to protect their IP. This makes the combination of gem5 with SystemC very attractive.

1.4.1 gem5 to SystemC Bridge[6]. SystemC TLM and gem5 were developed around the same time and are based on similar underlying ideas. As a consequence, the hardware model used by TLM is surprisingly close to the model of gem5. In both approaches, the system is organized as a set of components that communicate by exchanging data packets via a well defined protocol. The protocol abstracts over the physical connection wires that would be used in a register transfer level (RTL) simulation and thereby significantly increases simulation speed. In gem5, components use *master* and *slave* ports to communicate to other components, whereas in SystemC TLM, connections are established via *initiator* and *target* sockets. Also, the three protocols *atomic*, *timing* and *functional* provided by gem5 find their equivalent in the *blocking*, *non-blocking* and *debug* protocols of TLM. The major difference in both protocols is the treatment of backpressure, which is implemented by a retry phase in gem5 and with the exclusion rule of TLM.

The similarity of the two approaches enabled us to create a light-weight compatibility layer. In our approach, co-simulation is achieved by hosting the gem5 simulation on top of a SystemC simulation. For this, we replaced the gem5 discrete event kernel with a SystemC process that is managed by the SystemC kernel. A set of transactors further enables communication between the two simulation domains by translating

[2] http://www.gem5.org/documentation/learning_gem5/introduction/
[3] by Sean Wilson and Robert R. Bruce
[4] By Gabriel Black
[5] https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html
[6] By Chistian Menard, Jeronimo Castrillon, and Matthias Jung

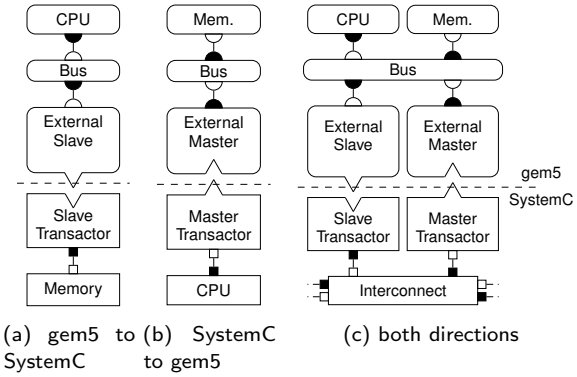

Fig. 1. Possible scenarios for binding gem5 and SystemC.

between the two protocols as is shown in Figure 1. This work was published in [7] where we documented our approach and showed that the transaction between gem5 and TLM only introduces a low overhead of about 8%. The source code as well as basic usage examples can be found in `util/tlm` of the gem5 repository.

1.4.2 SystemC in gem5[7]. Haven't heard anything back, yet.

1.5 Cache Replacement Policies and New Compression Support[8]

In general, hardware components frequently contain tables, whose contents are managed by replacement policies. In gem5, multiple replacement policies are available, which can paired with any table-like structure, allowing users to carry research on the effects of different replacement algorithms in various hardware units. Currently, gem5 supports 13 different replacement policies including several standard policies such as LRU, FIFO, and Pseudo-LRU, and various RRIPs [5]. This list is easily expandable to cover schemes with greater complexity as well.

The simulator also supports cache compression by providing several state-of-the-art compression algorithms [9] and a default compression-oriented cache organization. This basic organization scheme is derived from accepted approaches in the literature: adjacent blocks share a tag entry, yet they can only be co-allocated in a data entry if each of them compresses to at least a specific percentage of the cache line size. Currently, only BDI [8], C-Pack [4], and FPCD [2] are implemented, but the modularity of the compressors allows for simple implementation of other dictionary-based and pattern-based compression algorithms (e.g., only a few hours of development effort for a developer familiar with the code).

These replacement policies are a great example of gem5's modularity and how code developed for one purpose can be reused in many other parts of the simulator. Current and future development is planned to increase the use of these flexible replacement policies. For instance, we are planning to extend the TLB and other cache structures beyond the data caches to take advantage of the same replacement policies. Additionally, although the aforementioned cache compression policies have only been applied to the classic

[7] By Gabriel Black
[8] By Daniel Carvalho

caches, we are planning to use the same modular code to enable cache compression for the Ruby caches as well.

1.6 Ruby Cache Model Improvements

1.6.1 General Improvements[9]. Haven't heard anything yet.

1.6.2 GPU Coherence Protocols[10]. Haven't heard anything, yet.

1.6.3 ARM Support in Ruby Coherence Protocols[11]. Haven't heard anything, yet.

1.7 RISC-V ISA Support

RISC-V is a new ISA which has quickly gained popularity since its creation in 2010, only one year before the initial gem5 release [11]. In that time, the number of users RISC-V has grown significantly, especially in the computer architecture research community. Thus, the addition of RISC-V as a supported ISA for gem5 is one of the main new features in the past nine years.

1.7.1 General RISC-V ISA Implementation[12]. **TODO: CITE (https://carrv.github.io/2017/papers/roelke-risc5-carrv2017.pdf) and (https://carrv.github.io/2018/papers/CARRV_2018_paper_3.pdf).**

The motivation for implementing the RISC-V ISA into gem5 stemmed from needing a way to explore architectural parameters for RISC-V designs. At the time of implementation, the only means of simulating RISC-V was using spike (its simplified, single-cycle RTL simulator), QEMU, or full RTL simulation or emulation on FPGA. Spike and QEMU arent detailed enough and RTL simulation is too time consuming for these methods to be feasible for architectural parameter exploration, and FPGA emulation is difficult to retrieve performance information from without modifying both the RTL design and executed software to track and present it. The gem5 simulator provides an easy means of performing this type of analysis through its detailed hardware models that do not require software modification and allows for variable levels of detail. By adding RISC-V to gem5, this type of analysis is enabled for the rapidly-growing ISA.

The implementation was done by following the divisions of the instruction set into its base ISA and extensions, beginning with the 32-bit integer base set, RV32I. It was modeled off of the existing gem5 code for MIPS and Alpha, which are also RISC instruction sets that share many of the same operations as RISC-V. Including support for 64-bit addresses and data (RV64) and for the multiply (M) extension mainly involved adding the new instructions and changing some parameters to expand register and data path widths. The next two extensions, atomic (A) and floating point (F and D for single- and double-precision, respectively), were more complicated. The A extension includes both load-reserved/store-conditional (LR/SC) sequence of instructions for performing complex atomic operations on memory and a set of read-modify-write instructions for performing simple ones. The former two instructions had analogues in MIPS, but, at the time of implementation, gem5 did not have support for single instructions that both read and wrote memory atomically. These instructions were implemented as a pair of micro-ops that acted like an LR/SC pair with

[9] by Nilay Vaish
[10] by Blake Hectman
[11] by Tiago Mück
[12] By Alec Rokle

one of the pair additionally performing the specified operation. Floating-point instructions required many special cases to ensure correct error handling and reporting, and we were not able to implement one of the five possible rounding modes (round away from zero) RISC-V specifies for inexact calculations due to the fact that C++ does not support it. Finally, support for the non-standard compressed (C) extension, which adds 16-bit versions of high-usage instructions, was added when it was discovered that this extension was included by default in many RISC-V software toolchains. Its implementation required the creation of a state machine in the instruction decoder to keep track of whether the current instruction is compressed or not, to increment the PC by the correct amount based on the size of the instruction, and to handle cases where a full-length instruction crosses a 32-bit word boundary.

With this implementation, most RISC-V Linux programs are supported for execution in system-call-emulation mode. Future work by others would then go on to improve the implementation of atomic instructions, including actual atomic read-modify-write accesses in a single instruction and steps toward support for full-system simulation.

1.7.2 RISC-V Full System Support[13]. Haven't heard anything back, yet.

1.8 Predictor Improvements

I haven't heard anything back, yet.

1.9 GPU Compute Model[14]

Heard back, waiting for text. This may be a bit late due to having to run it past the lawyers.

1.9.1 Autonomous Data-Race-Free GPU Tester[15]. The Ruby coherence protocol tester is designed for CPU-like memory systems that implement relatively strong memory consistency models (e.g., TSO) and hardware-based coherence protocols (e.g., MESI). In such systems, once a processor sends a memory request to memory, the request appears globally to the rest of the system. Without knowing implementation details of target memory systems, the tester can rely on the issuing order of reads and writes to determine the current state of shared memory. However, existing GPU memory systems are often based on weaker consistency models (e.g., sequential consistency for data-race-free) and implement software-directed cache coherence protocols (e.g., VIPER requiring explicit cache flushes and invalidations from software to maintain cache coherence). The order in which reads and writes appear globally can be different from the order they are issued from GPU cores. Therefore, the previous CPU-centric Ruby tester is not applicable to testing GPU memory systems.

The gem5 simulator currently supports an autonomous random data-race-free testing framework to validate GPU memory systems. The tester works by randomly generating and injecting sequences of data-race-free reads and writes that are well synchronized by proper atomic operations and memory fences to a target memory system. By maintaining the data-race freedom of all generated sequences, the tester is able to validate responses from the system under test. The tester is also able to periodically check for forward progress of the system and report possible deadlock and livelock issues. Once encountering a failure,

[13] By Nils Asmussen
[14] by Anthony Gutierrez
[15] by Tuan Ta

the tester generates an event log that captures only related memory transactions related to the failure, which significantly eases the debugging process. Tuan Ta et al. showed how the tester effectively detected bugs in the implementation of VIPER protocol in gem5 [10].

1.10 Syscall Emulation Improvements[16]

Heard back, waiting for text. This may be a bit late due to having to run it past the lawyers.

1.11 ARM Improvements[17]

Heard back, waiting for text.

Note: May want to add something about https://community.arm.com/developer/ip-products/system/b/soc-design-blog/posts/simplifying-workload-modelling-with-amba-atp-engine

1.11.1 ARMv8 Support[18]. Heard back, waiting for text.

1.12 Vector Instructions Extensions[19]

1.13 Internal gem5 Improvements and Features

1.13.1 HDF5 Support[20]. Haven't heard back, yet.

1.13.2 Python 3[21]. Haven't heard back, yet.

1.14 Flexible DRAM Controller[22]

Heard back, waiting for text.

1.14.1 [23]. Across applications, DRAM is a significant contributor to the overall system power. For example, the DRAM access energy per bit is up to three orders of magnitude higher compared to an on-chip memory access. Therefore, an accurate and fast power estimation is crucial for an efficient design space exploration. DRAMPower (cite: DRAMPower: Open-source DRAM Power & Energy Estimation Tool Karthik Chandrasekar, Christian Weis, Yonghui Li, Sven Goossens, Matthias Jung, Omar Naji, Benny Akesson, Norbert Wehn, and Kees Goossens URL: http://www.drampower.info) is an open source tool for fast and accurate power and energy estimation for several DRAM memories based on JEDEC standards. It supports unique features like power-down, bank-wise power estimation, per bank refresh, partial array self-refresh, and many more.

In contrast to Microns DRAM Power estimation spread sheet (cite Micron. DDR3 SDRAM System Power Calculator), which estimates the power from device manufacturers data sheet and workload specifications (e.g. Rowbuffer-Hit-Rate or Read-Write-Ratio), DRAMPower uses the actual timings from the memory transactions, which leads to a much higher accuracy in power estimation. Furthermore, the DRAMPower tool

[16] by Brandon Potter
[17] by Lots of People
[18] by Giacomo Gabrielli
[19] by Javier Setoain
[20] by Andreas Sandberg
[21] by Andreas Sandberg and Giacomo Travaglini
[22] by Wendy Elsasser, Matthais Jung, and others
[23] by Matthias Jung, Subash Kannoth, Omar Naji, Eder F. Zulian, Christian Weis, ...

performs DRAM command trace analysis based on memory state transitions and hence, avoids cycle-by-cycle evaluation, thus speeding up simulations.

For the efficient integration of DRAMPower into gem5, we changed the tool from a standalone simulator to a library that could be used in discrete event-based simulators for calculating the power consumption online during the simulation. Furthermore, we integrate the power-down modes into the DRAM controller model of gem5 (cite: 3. Integrating DRAM Power-Down Modes in gem5 and Quantifying their Impact R. Jagtap, M. Jung, W. Elsasser, C. Weis, A. Hansson, N. Wehn. ACM International Symposium on Memory Systems (MEMSYS 2017), October, 2017, Washington, DC, USA) in order to provide the research community a tool for power-down analysis for a breadth of use cases. We further evaluated the model with real HPC workloads, illustrating the value of integrating low power functionality into a full system simulator.

1.14.2 Quality of Service Extensions[24]. Heard back, waiting for text. It may be a bit late.

1.15 Virtualized Fast Forwarding[25]

I haven't heard anything back, yet. Note: Cite paper "Full Speed Ahead: Detailed Architectural Simulation at Near-Native Speed"

1.16 gem5 and SST Integration[26]

Haven't head back, yet.

1.17 Memory Traces and Traffic Generator[27]

Haven't heard back, yet.

1.18 Classic Caches Improvements[28]

Haven't heard back, yet.

1.18.1 Snooping Support and Snoop Filtering[29]. Heard back, waiting for text.

1.19 dist-gem5: Support for Distributed Computing[30]

Haven't heard back, yet.

Note: I should probably reach out to Mohammad Alian.

Cite "dist-gem5: Distributed Simulation of Computer Clusters" and "pd-gem5: Simulation Infrastructure for Parallel/Distributed Computer Systems"

1.20 The Minor In-Order CPU Model[31]

Haven't heard back, yet.

[24] by Matteo Andreozzi
[25] by Andreas Sandberg
[26] by Cutis Dunham
[27] by Andreas Hanson
[28] by Nikos Nikoleris and Andreas Hanson
[29] by Stephan Diestelhorst
[30] by Gabor Dozsa
[31] by Andrew Bardsley

1.21 Power Modeling and DVFS Support[32]

Haven't heard back, yet.

1.22 Elastic Traces[33]

Detailed execution-driven CPU models, like gem5, offer high accuracy, but at the cost of simulation speed. Therefore, trace-driven simulations are widely adopted to alleviate this problem, especially for studies focusing on memory-system exploration. However, traces with fixed time stamps always include the implicit behavior of the simulated memory system with which they were recorded. If the memory system is changed during exploration this will lead to wrong simulation results, since an out-of-order core would react differently on the new memory system.

Ideally, trace-driven core models will mimic out-of-order processors executing full-system workloads to enable computer architects to evaluate modern systems. Therefore, we proposed the concept of elastic traces in which we accurately capture data and load/store order dependencies by instrumenting a detailed out-of-order processor model (cite: 4. Exploring System Performance using Elastic Traces: Fast, Accurate and Portable R. Jagtap, S. Diestelhorst, A. Hansson, M. Jung and N. Wehn. IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS), July, 2016, Samos Island, Greece). In contrast to existing work, we do not rely on offline analysis of timestamps, and instead use accurate dependency information tracked inside the processor pipeline. We thereby account for the effects of speculation and branch misprediction resulting in a more accurate trace playback compared to fixed time traces. We integrated a trace player in gem5 that honors the dependencies and thus adapts its execution time to memory-system changes, as would the actual CPU. Compared to the detailed CPU model, our trace player achieves a speed-up of 6-8 times while maintaining a high simulation accuracy (83-93%), achieving fast and accurate system performance exploration.

2 OTHER WORK BUILDING OFF OF GEM5

Note: Come up with a better name.

Accelerated simulated fault injection testing - https://ieeexplore.ieee.org/document/8109288/

A Framework for Non-intrusive Trace-driven Simulation of Manycore Architectures with Dynamic Tracing Configuration - https://link.springer.com/chapter/10.1007/978-3-030-03769-7_28

gem5-gpu Power et al.

gem5-aladdin Shao et al.

Many others.

3 ACKNOWLEDGEMENTS

The development of gem5 is distributed. It is likely we have missed someone that should be acknowledged.

NSF CCRI. Brookhaven.

List all contributors that did not reply to the message about this paper.

[32] by Akash Bagdia, Stephan Diestelhorst, David Guillen-Fandos, and Anouk Van Laer
[33] by Radhika Jagtap

REFERENCES

[1] 2012. IEEE Standard for Standard SystemC Language Reference Manual. *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)* (Jan 2012). https://doi.org/10.1109/IEEESTD.2012.6134619

[2] Alaa R Alameldeen and Rajat Agarwal. 2018. Opportunistic compression for direct-mapped DRAM caches. In *Proceedings of the International Symposium on Memory Systems*. ACM, 129–136.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 17. https://doi.org/10.1145/2024716.2024718

[4] Xi Chen, Lei Yang, Robert P Dick, Li Shang, and Haris Lekatsas. 2010. C-pack: A high-performance microprocessor cache compression algorithm. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 18, 8 (2010), 1196–1208.

[5] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, and Joel Emer. 2010. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (Saint-Malo, France) *(ISCA 10)*. Association for Computing Machinery, New York, NY, USA, 6071. https://doi.org/10.1145/1815961.1815971

[6] Jason Lowe-Power. 2015. gem5 Horrors and what we can do about it. In *Second gem5 User Workshop with ISCA 2015*.

[7] Christian Menard, Jeronimo Castrillon, Matthias Jung, and Norbert Wehn. 2017. System Simulation with gem5 and SystemC: The Keystone for Full Interoperability. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 62–69.

[8] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. 2012. Base-delta-immediate compression: practical data compression for on-chip caches. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 377–388.

[9] Somayeh Sardashti, Angelos Arelakis, Per Stenström, and David A Wood. 2015. A primer on compression in the memory hierarchy. *Synthesis Lectures on Computer Architecture* 10, 5 (2015), 1–86.

[10] Tuan Ta, Xianwei Zhang, Anthony Gutierrez, and Bradford M. Beckmann. 2019. Autonomous Data-Race-Free GPU Testing. In *IEEE International Symposium on Workload Characterization, IISWC 2019, Orlando, FL, USA, November 3-5, 2019*. IEEE, 81–92. https://doi.org/10.1109/IISWC47752.2019.9042019

[11] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanovi. 2011. *The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA*. Technical Report UCB/EECS-2011-62. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.html