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
MANY MANY OTHERS

(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, Bobby R. Bruce, and Many many others. 2016. The gem5 Simulator: Version 20.0+. 1, 1, Article 1 (January 2016), 12 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 THE GEM5 SIMULATOR

The gem5 simulator [3] is an open source community-supported computer architecture simulator system. It consists of a simulator core and models for everything from out of order processors, to DRAM, to network devices. The gem5 project consists of the gem5 simulator[1], documentation[2], and common resources[3] that enable computer architecture research.

The gem5 project is governed by a meritocratic, consensus-based community governance document[4] with a goal to provide a tool to further the state of the art in computer architecture.

- Give a high-level overview of gem5
- What are the main features
- What sets it apart from other architecture simulators
  - Governance and community-oriented
  - Used by multiple companies and by many academics

1.0.1 *gem5's main features.* ISA support.

CPU models that can be used with (almost) any ISA.

Memory system.

Two different cache models.

[1]https://gem5.googlesource.com/public/gem5
[2]https://www.gem5.org/
[3]https://gem5.googlesource.com/public/gem5-resources
[4]https://www.gem5.org/governance/

2016. Manuscript submitted to ACM

"Classic caches": I would like to come up with a better name than this. Maybe "Composable caches"?

Ruby allows for user-defined cache coherence protocols. We provide the following protocols.

Ruby also works with Garnet detailed network model.

Full system support.

1.1 The past nine years

The gem5 simulator has been wildly successful in the past nine years since the initial release of gem5.

- Talk about the citations to gem5. Lots of use.
- Very active development community
- Lots of commits
- Lots of committers

Show a figure with the number of commits per year.

Show a figure with the number of unique contributors per year.

Unfortunately, this success brought growing pains. Cite "gem5 Horrors" [6].

1.2 gem5 now

To solve these problems we

- Instituted a governance structure
- Improved documentation (cite Learning gem5)
- Worked to fund raise (CCRI)
- Moved development model to modern tools (git, gerrit)
- Created a contributing document to help people get started

1.2.1 Changes in gem5. In addition to these systematic changes there has also been lots of improvements to the codebase. Section 2 contains all of the details.

These include... (a long list).

1.2.2 Using gem5 in research and education. This talks about how to use gem5 in your research and your education.

Start with gem5.org.

1.3 The future of gem5

The future of gem5 is bright.

- Lots of new stuff coming in a new roadmap that's coming soon!
  - Improved python interface
  - New RAM models (including NVM)
  - Garnet 3.0
  - Models for ML accelerators
  - Known-good configurations
  - Many more.
- Here, we talk about how to contribute to gem5.

- We will do a better job recognizing contributions. The gem5 project is only successful with the community is vibrant. We have taken positive steps towards making it easier to contribute (like a governance document and contributing documentation), but we need to do more. We will encourage everyone to contribute.
- More educational material coming!
- More workshops, tutorials, etc.

2 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.

2.1 Learning gem5[5]

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 reader gets a similar experience to working alongside an experienced gem5 developer. Learning gem5 includes documentation on the gem5 website[6] 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.

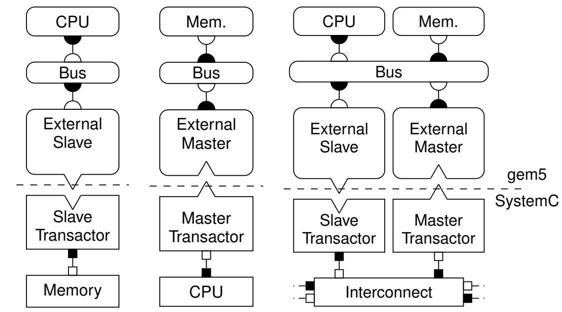
2.2 Testing in gem5[7]

Heard back, waiting for the text.

2.3 Updating Guest¡-¿Simulator APIs[8]

Haven't heard anything back, yet.

[5]By Jason Lowe-Power
[6]http://www.gem5.org/documentation/learning_gem5/introduction/
[7]by Sean Wilson and Robert R. Bruce
[8]By Gabriel Black



(a) gem5 to SystemC (b) SystemC to gem5 (c) both directions

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

2.4 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[9], 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.

2.4.1 gem5 to SystemC Bridge[10]. 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 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.

[9] https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html
[10] By Chistian Menard, Jeronimo Castrillon, and Matthias Jung

2.4.2 *SystemC in gem5*[11]. Haven't heard anything back, yet.

2.5 Cache Replacement Policies and New Compression Support[12]

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

2.6 Ruby Cache Model Improvements

The Ruby cache model, originally from the GEMS simulator [], is one of the key differentiating features of gem5. The domain-specific language SLICC allows users to define new coherence protocols with high fidelity. In mainline gem5, there are now 12 unique protocols including GPU-specific protocols [**?**], region-coherence protocols [**?**], research protocols like token coherence [**?**], and teaching protocols [**?**].

When gem5 was first released, Ruby had just been integrated into gem5. In the nine years since, Ruby and the SLICC protocols have become much more deeply integrated into the general gem5 memory system. Today, Ruby shares the same replacement protocols **??**, the same port system to send requests into and out of the cache system, and the same flexible DRAM controller models **??**.

Looking forward, we will be further integrating Ruby into gem5. Our goal is to one day have a unified cache model which has the composability and speed of the classic caches and the flexibility and fidelity of SLICC protocols.

2.6.1 *General Improvements*[13]. Ruby now supports state checkpointing and restoration with warm cache. This enables running simulations from regions of interest, rather than having to start fresh every time. To enable checkpoints, we support accessing the memory system functionally i.e. without any notion of time or events. The absence of timed events allows much higher simulation speeds.

[11] By Gabriel Black
[12] By Daniel Carvalho
[13] by Nilay Vaish

Additionally, a new three level coherence protocol has been added to gem5. For simplicity, this protocol was built on top of a prior two level protocol by adding an L0 cache at the CPU cores. At level 0, the protocol has separate caches for instruction and data. The first and the second levels do not distinguish between instruction and data. Levels 0 and 1 are private to the CPU core, which the second level is shared across (possibly a subset) cores.

2.6.2 *GPU Coherence Protocols*[14].

Haven't heard anything, yet.

2.6.3 *ARM Support in Ruby Coherence Protocols*[15].

Haven't heard anything, yet.

2.7 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.

2.7.1 *General RISC-V ISA Implementation*[16].

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 aren't 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 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.

[14] by Blake Hectman
[15] by Tiago Mück
[16] By Alec Rokle

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.

2.7.2 RISC-V Full System Support[17]. Haven't heard anything back, yet.

2.8 Predictor Improvements

I haven't heard anything back, yet.

2.9 GPU Compute Model[18]

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

2.9.1 Autonomous Data-Race-Free GPU Tester[19]. 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, 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].

2.10 Syscall Emulation Improvements[20]

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

[17] By Nils Asmussen
[18] by Anthony Gutierrez
[19] by Tuan Ta
[20] by Brandon Potter

2.11 ARM Improvements[21]

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

2.11.1 ARMv8 Support[22]*.* Heard back, waiting for text.

2.12 Vector Instructions Extensions[23]

2.13 Internal gem5 Improvements and Features

It is important to recognize not only all of the ground-breaking additions to the models in gem5, but also general improvements to the simulation infrastructure. Although these improvements do not always result in new research findings, they are a key *enabling factor* for the research conducted using gem5.

The simulator core of gem5 provides support for event-driven execution, statistics, and many other important functions. These parts of the simulator are some of the most stable components, and, as part of the gem5-20 release and in the subsequent releases, we will be defining stable APIs for these interfaces. By making these interfaces *stable* APIs, it will facilitate long-term support for integrating other simulators (e.g., SST **??** and SystemC **??**) and projects that build off of gem5 (e.g., gem5-gpu [], gem5-aladdin [], and many others.)

2.13.1 HDF5 Support[24]*.* A major change in the latest gem5 release is the new statistics API. While the driver for this API was to improve support for hierarchical statistics formats like HDF5 [], there are other more tangible benefits as well. Unlike the old API where all statistics live in the same namespace, the new API introduces a notion of statistics groups. In most typical use cases, statistics are bound to the current SimObject's group, which is then bound to its parent by the runtime. This ensures that there is a tree of statistics groups that match the SimObject graph. However, groups are not limited to SimObject. Behind the scenes, this reduces the amount of boiler plate code when defining statistics and makes the code far less error prone. The new API also brings benefits to simulation scripts. A feature many users have requested in the past has been the ability to dump statistics for a subset of the object graph. This is now possible by passing a SimObject to the stat dump call, which limits the statistics dump to that subtree of the graph.

With the new statistics API in place, it became possible to support hierarchical data formats like HDF5. Unlike gem5's traditional text-based statistics files, HDF5 stores data in a binary file format that resembles a file system. Unlike the traditional text files, HDF5 has a rich ecosystem of tools and official bindings for many popular languages, including Python and R. In addition to making analysis easier, the HDF5 backend is optimized for storing time series. HDF5 files internally store data as N-dimensional matrices. In gem5's implementation, we use one dimension for time and the remaining dimensions for the statistic we want to represent. For example, a scalar statistic is represented as a 1-dimensional vector. When analyzing such series using Python, the HDF5 backend imports such data sets as a standard NumPy array that can be used in common data analysis and visualization flows. The additional data needed to support filesystem-like structures inside the stat files introduces some storage overheads. However, these are quickly amortized when sampling statistics since the incremental storage needed for every sample is orders of magnitude smaller than the traditional text-based statistics format.

[21] by Lots of People
[22] by Giacomo Gabrielli
[23] by Javier Setoain
[24] by Andreas Sandberg

2.13.2 Python 3[25]. Even before m5 became gem5, the architecture of the built on a core written in C++ and configuration scripts using that core as a Python library. While the fundamental design has not changed, there have been many changes to the underlying implementation over the past years. The original implementation frequently suffered from bugs in the code generated by SWIG and usability was hampered by poor adherence to modern standards in SWIG's C++ parser. The move to PyBind11 [] greatly improved the reliability of the bindings by removing the need for a separate C++ parser, and made it easier to expose new functionality to Python in a reliable and type-safe manner.

The move away from SWIG to PyBind11 provided a good starting point for the more ambitious project of making gem5 Python 3 compatible. Making gem5 Python 3 compatible has not add any new features yet, but it ensures that the simulator will continue to run on Linux distributions that are released in 2020 and onwards. It does however enable exciting improvements under the hood. A couple of good examples are type annotations that can be used to enable better static code analysis and greatly improved string formatting. Our ambition is to completely phase out Python 2 support in the near future to benefit from these new features.

2.14 Flexible DRAM Controller[26]

Heard back, waiting for text.

2.14.1 [27]. 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 Micron's DRAM Power estimation spread sheet (cite Micron. DDR3 SDRAM System Power Calculator), which estimates the power from device manufacturer's 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 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.

[25]by Andreas Sandberg and Giacomo Travaglini
[26]by Wendy Elsasser, Matthais Jung, and others
[27]by Matthias Jung

2.14.2 Quality of Service Extensions[28]. Heard back, waiting for text. It may be a bit late.

2.15 Virtualized Fast Forwarding[29]

Support for hardware virtualization in gem5 might have seemed like a non-obvious enhancement to the simulator when we started to work on it in 2012. However, it has turned out to be a very useful feature for bring up, model development, testing, and novel simulation research [](Full Speed Ahead: Detailed Architectural Simulation at Near-Native Speed (http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Auu%3Adiva-220649) CoolSim: Statistical techniques to replace cache warming with efficient, virtualized profiling Directed Statistical Warming through Time Traveling). Work on the original implementation started in the summer 2012 in Arm Research and targeted the Arm Cortex A15 chip. Some of the most challenging parts of the development were the lack of a stable kernel API for KVM on Arm and the limited availability of production silicon. However, despite these challenges, we had a working prototype that booted Linux in autumn. This prototype was refined and merged into gem5 in April 2013, just one month after qemu gained support for Arm KVM. Support for x86 followed later that year. A good overview the KVM implementation can be found in the technical report by Sandberg et. al [] (Full Speed Ahead: Detailed Architectural Simulation at Near-Native Speed (http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Auu%3Adiva-220649)). The original full-system implementation was later extended to support syscall emulation mode on x86 [] (Alex Dutu in a gem5 workshop a few years ago).

Support for hardware virtualization in gem5 enabled research into novel ways of accelerating simulation. The original intention was to use KVM to generate checkpoints and later simulate those checkpoints in parallel. However, we quickly realized that the checkpointing step could be eliminated by cloning the simulator state at runtime. This led to the introduction of the fork call in gem5's Python API. Under the hood, this call drains the simulator to make sure everything is in a consistent state, it then uses the UNIX fork call to create a copy of the simulator. A typical use case uses a main process that generates samples that are simulated in parallel. More advanced use cases use fork semantics to simulate multiple outcomes of a sample to quantify the cache warming errors introduced by using KVM to fast-forward between samples [] (Full speed ahead).

2.16 gem5 and SST Integration[30]

Haven't head back, yet.

2.17 Memory Traces and Traffic Generator[31]

Haven't heard back, yet.

2.18 Classic Caches Improvements[32]

Haven't heard back, yet.

2.18.1 Snooping Support and Snoop Filtering[33]. Heard back, waiting for text.

[28] by Matteo Andreozzi
[29] by Andreas Sandberg
[30] by Cutis Dunham
[31] by Andreas Hanson
[32] by Nikos Nikoleris and Andreas Hanson
[33] by Stephan Diestelhorst

2.19 dist-gem5: Support for Distributed Computing[34]

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"

2.20 The Minor In-Order CPU Model[35]

Haven't heard back, yet.

2.21 Power Modeling and DVFS Support[36]

Haven't heard back, yet.

2.22 Elastic Traces[37]

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.

3 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.

[34] by Gabor Dozsa
[35] by Andrew Bardsley
[36] by Akash Bagdia, Stephan Diestelhorst, David Guillen-Fandos, and Anouk Van Laer
[37] by Radhika Jagtap

gem5-aladdin Shao et al.

Many others.

4 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.

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), 1–7. 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, 60–71. 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 Asanović. 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