The gem5 Simulator: Version 20.0+*

A new era for the open-source computer architecture simulator

(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:
. 2016. The gem5 Simulator: Version 20.0+. 1, 1, Article 1 (January 2016), 26 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 THE GEM5 SIMULATOR

There is "a new golden age for computer architecture" [**? ?**] driven by changes in technology (e.g., the slowdown of Moore's Law and Dennard Scaling) and ever increasing computational needs. One of the first steps in research and development of new hardware architectures is software-based modeling and simulation. The gem5 simulator [9] is currently one of the most popular academic-focused computer architecture simulation frameworks. Since its publication in 2011, the gem5 paper has been cited over 3600 times[1], and every year many papers published in the top computer architecture venues use gem5 as their main evaluation technique.

The gem5 simulator [9] 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[2], documentation[3], and common resources[4] that enable computer architecture research.

The gem5 project is governed by a meritocratic, consensus-based community governance document[5] with a goal to provide a tool to further the state of the art in computer architecture. gem5 can be used for (but is not limited to) computer-architecture research, advanced development, system-level performance analysis and design-space exploration, hardware-software co-design, and low-level software performance analysis. Another goal of gem5 is to be a common framework for computer architecture. A common framework in the academic community makes it easier for other researchers to share workloads as well as models and to compare and contrast with other architectural techniques.

The gem5 community strives to balance the needs of its three categories of users: academic researchers, industry researchers, and students learning computer architecture. For instance, the gem5 community gem5 strives to balance

*gem5 is the result of the merger of the GEMS project started in 1999, and the m5 project started in 2003. Development of gem5 has been active for about 20 years, and this version is being published in 2020. Thus, "gem5-20".
[1] https://scholar.google.com/scholar?q=gem5
[2] https://gem5.googlesource.com/public/gem5
[3] https://www.gem5.org/
[4] https://gem5.googlesource.com/public/gem5-resources
[5] https://www.gem5.org/governance/

adding new features (important to researchers) and a stable code base (important for students). Specific user needs important to the community are enumerated below:

- Effectively and efficiently emulate the behavior of modern processors in a way that balances simulation performance and accuracy
- Serve as a malleable baseline infrastructure that can easily be adapted to emulate the desired behaviors
- Provide a core set of APIs and features that remain relatively stable
- Incorporate features that make it easy for companies and research groups to stay up to date with the tip and continue contributing to the project
- Additionally, the gem5 community is committed to openness, transparency, and inclusiveness.

In this paper, we discuss the current state of gem5. We first discuss the past, present and future of gem5. Then, we give an overview of gem5's main features available today and describe how to become a member of the gem5 community for researchers, students, and teachers. Finally, we include a significant section which describes the major changes in gem5 in the past nine years since the initial gem5 release. Each subsection was contributed by at least one of the people who developed that part of the simulator as noted in the footnotes.

It has taken a huge number of people to make gem5 what it is today. One of the goals of this paper is to recognize the hard work on this community infrastructure which is often overlooked. We have tried to include everyone who has contributed and documented all of the major changes.

1.1 The past, present, and future of gem5

The gem5 simulator was born when the m5 simulator [](https://www.computer.org/csdl/magazine/mi/2006/04/m4052/13rRUxYIMRJ) created at University of Michigan merged with the GEMS simulator [](Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset) from University of Wisconsin. These were two academic-oriented simulators, neither of which had an open development community (both simulators had their source available for free[6][7], but did not have a community-oriented development process). Both of these simulators were quite popular on their own. The GEMS paper has been cited over 1800 times and the m5 paper has been cited over 1000 times.

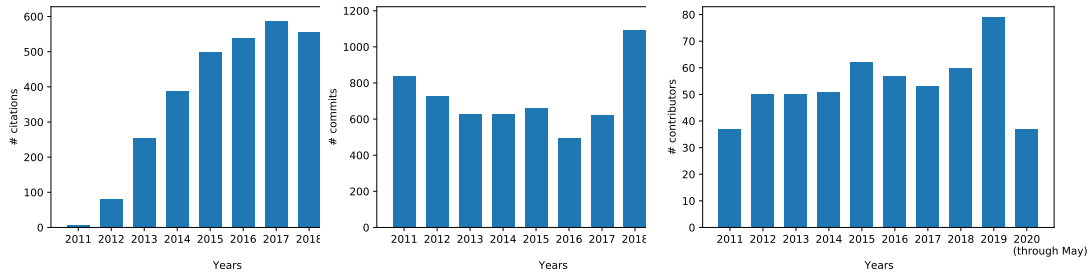
Since its initial release nine years ago the gem5 simulator has been wildly successful. In this time, the use of gem5 has exploded. Although not a perfect metric, the gem5 paper has received over 3600 citations according to Google Scholar (see Figure 1a).

At the same time, the contributor community has also grown. Figures 1b shows the number of commits per year and Figure 1c shows the number of unique contributors per year. These figures show that since the initial release of gem5 in 2011, development has been significantly accelerating.

With this acceleration in use and development of gem5 came growing pains [24]. The gem5 community was going through a shift from a small project with most contributors from one or two academic labs to a project with worldwide-distributed contributors. Additionally, given the growing user base, we could no longer assume that all gem5 users were also going to be main developers.

To solve the problems brought up by the gem5 community, the gem5 project has made major changes in the past nine years. We now have a formal governance structure, we have improved our documentation (see Section **??**), we have moved to a better distributed development platform, and improved our community outreach.

[6] https://sourceforge.net/projects/m5sim/
[7] https://research.cs.wisc.edu/gems/home.html



(a) Number of citations (b) Number of commits (c) Number of contributors

Fig. 1. Number of gem5 citations, commits and contributors from 2011 to May 2020.

To institute a formal governance model, we followed the best practices from other successful open source projects. We chose to institute a meritocratic governance model where anyone with an interest in the project can join the community, contribute to the project design and participate in the decision-making process. The governance structure also defines the roles and responsibilities of members of the community including users, contributors, and maintainers. We also formed a project management committee (PMC) to help ensure smooth running of the project. Importantly, members of the PMC do not have significant authority over other members of the community or of the direction of the project.

To simplify the contribution process, we have instituted many industry-standard development methodologies including providing a `CONTRIBUTING` document in the gem5 source. In the past, gem5 code contributions were managed with a number of esoteric software packages. Now, all gem5 code is stored in a git repository[8], code review is managed on Gerrit[9], we have continuous integration support (see Section 2.2), our website is implemented with jekyll and markdown[10], and we have a Jira-based issue tracker[11].

After transitioning to these more well known tools and improved development practices, we have seen a rise in the number of community contributors and using gem5 has become less frustrating. Continuous integration enables us to test every single changeset before it is committed. This allows us to catch bugs *before* they are committed into the mainline repository which makes gem5 more stable. Similarly, by implementing a bug tracking system, we can track issues that effect gem5. In the first six months of using a bug tracker we have closed over 250 issues.

The future of gem5. The future of gem5 is bright. We will be continuing to work with the community to define the roadmap for gem5 development for gem5 version 20.1 and beyond. In the short term, we are excited to be improving the underlying infrastructure of gem5 with better testing, refactoring aging code (some of gem5's code is over 20 years old!), and adding well-defined stable APIs. By defining stable APIs, we will make it easier for the community to build off of gem5. For instance, the inter-simulator interface is currently being defined so that gem5 can be used in conjunction with other simulators (e.g., SST [], SystemC **??**, and many others). We are also working on improving the interconnect model **??** and adding support for non-volatile memory controllers **??**.

[8] googlesource
[9] urlgem5-review
[10] gem5-website-repo
[11] https://gem5.atlassian.net/

One of the most exciting features coming to gem5 is that we will provide the community with a set of *publicly validated* models which will model current architectural system components including CPU cores, GPU compute units (CUs), caches, main memory systems, and devices. Past research [3, 4, 7, 10, 14, 15, 20, 26, 34, 35] has shown that some gem5 models can be imprecise. We will strive for accuracy compared to real systems; however, since most systems are proprietary and complex, accuracy for all workloads will be difficult. Thus, we will broadly advertise the relative performance, power, and other metrics when providing these models so users can make an informed decision when choosing their baseline configurations. This will reduce the researcher's time spent on configuring baselines and allow them to concentrate more effort on analyzing and developing their novel research ideas. The first step towards this goal of validated baselines is the gem5 resources repository described in Section **??**.

Finally, we are planning to publish an online *Learning gem5* course based on a largely expanded version of the *Learning gem5* material **??**. This course will cover how to get started using gem5, how to develop new models to be used in gem5, and the details of gem5's software architecture. In addition to the online version of the course, we will continue to conduct tutorials and workshops at computer architecture and computer systems conferences.

1.2 gem5's main features

The gem5 simulator is a cycle-level computer system simulation environment. At its core, gem5 contains an event-driven simulation engine. On top of this simulation engine, gem5 implements a large number of models for system components from CPUs (out-of-order designs, in-order designs, and others), memories (such as DDR3/4, GDDR5, HBM, and HMC), on-chip interconnects, coherent caches, I/O devices, and many others. Many of these different models are shown in Figure 2 The gem5 project also contains tests to help find bugs, a complex and feature rich statistics database, and a Python scripting interface to describe systems under test and run simulations.

The gem5 simulator has modular support for multiple ISAs (see Figure 2 ①). gem5 currently supports Arm, GPU ISAs, MIPS, Power, RISC-V, SPARC, and x86. These ISAs not only include the details to execute each instruction, but also the system-specific devices necessary for full system simulation. There is robust full system support for Linux on Arm and x86 and many other ISAs have some level of full system support.

All of these ISAs can be used with any of gem5's CPU models as the CPU models are designed to be ISA-agnostic (Figure 2 ②). gem5 includes four different CPU models which span the fidelity performance continuum. The highest performance CPU model is based on the kernel virtual machine (KVM) and leverages hardware virtualization to execute *at native speeds* [30]. Although the KVM CPU model can execute at native speed, it does not model the timing of execution or memory requests. gem5 also contains "simple" CPU models which are single-cycle models that can be used for memory system studies or other studies that do not require high-fidelity execution models. Finally, gem5 contains a detailed in-order CPU model (the "minor" CPU) and an out-of-order CPU model (the "O3" CPU).

Underlying the memory system model in gem5 is a modular port interface which allows any component that implements the port API to be connected to any other component implementing that API (Figure 2 ③). This allows models designed for one system to be easily used in other system designs.

There are two different cache systems in gem5: Ruby (Figure 2 ④), which models cache coherence protocols with high fidelity, and the "classic" caches (Figure 2 ⑤) which lack cache coherence fidelity and flexibility. Ruby enables user-defined cache coherence protocols, though gem5 includes many protocols out of the box. Users can also choose to use a simple network model or the detailed Garnet model [2] when using Ruby caches which offers cycle-level detail for the on-chip network.

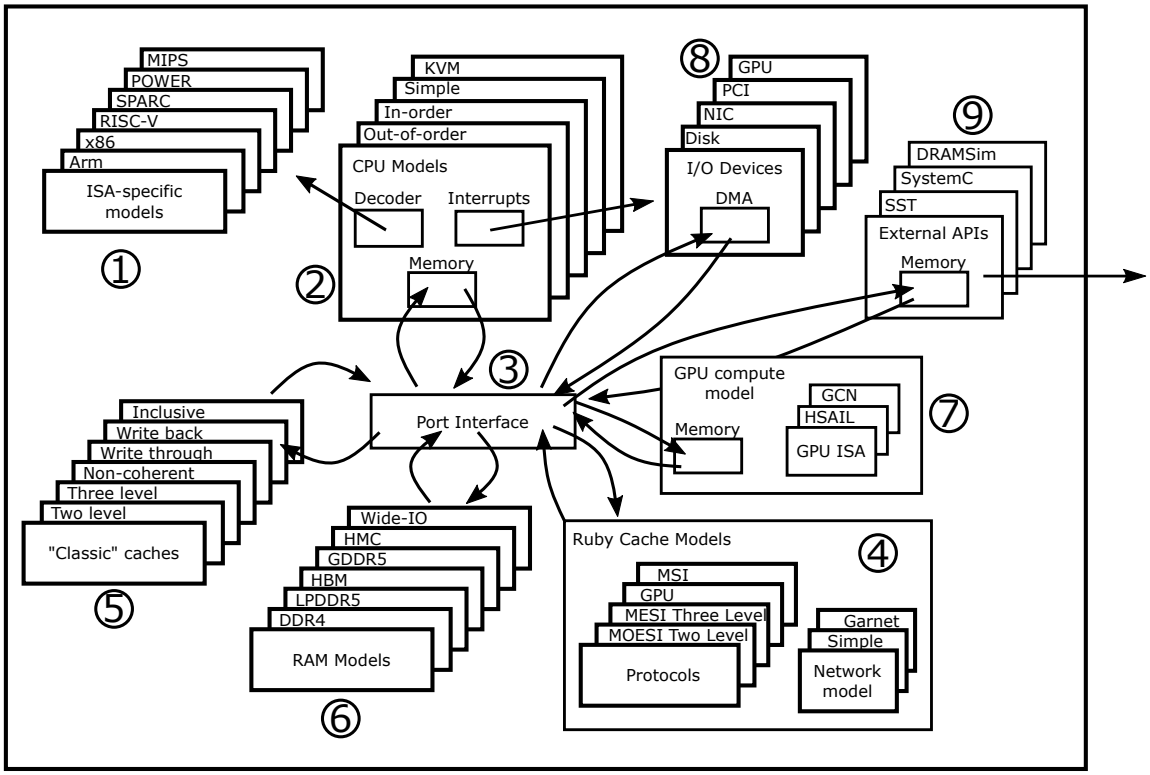


Fig. 2. An overview of gem5's architecture. Its modular components allow any of each model type to be used in system configuration via Python scripts. Users can choose the fidelity of the memory system, CPU model, etc. while being able to select any ISA, devices, etc. The port interface allows any memory component to be connected to any other memory component as specified by the Python script. Details of each of these simulator components are discussed in Section 1.2

The classic caches have a single hard-coded hierarchical MOESI coherence protocol. However, this cache model is easily composable allowing users to construct hierarchical cache topologies without worrying about the correctness of the coherence protocol. Both Ruby and the classic caches can be used with any CPU model, any ISA, and any memory controller model.

The gem5 simulator also includes an event-driven DRAM model (Figure 2 ⑥). This DRAM model is easily configurable with the timing parameters for a variety of different DRAM controllers including DDR3, DDR4, GDDR, HBM, HMC, LPDDR4, LPDDR5, and others. Although this is not a cycle-accurate DRAM model like DRAMSim [23, 29, 36] or Ramulator [22], it is nearly as accurate while providing more flexibility and higher performance [**?**]

In addition to CPU models, gem5 also includes a cycle-level compute-based GPU [] (Figure 2 ⑦). This GPU model does not support graphics applications, but supports many compute applications based on the heterogeneous system architecture (HSA) and ROCm runtime. The GPU model is based on AMD's Graphics Core Next (GCN) architecture [](AMD. AMD Graphics Core Next (GCN) Architecture. https://www.techpowerup.com/gpu-specs/docs/amd-gcn1-architecture.pdf, June 2012). The GPU model has a modular ISA similar to the CPU model in gem5, and can be extended to support other GPU ISAs in the future.

An important component to full system simulation is supporting I/O and other devices (Figure 2 ⑧). gem5 supports many system-agnostic devices such as disk controllers, PCI components, ethernet controllers, etc. and system-specific devices such as the Arm GIC and SMMU and x86 PC devices. Additionally, gem5 contains support for a functional-only GPU model to enable simulating applications that depend on graphics APIs but do not depend on graphics performance [13].

Finally, gem5 has been integrated with other computer architecture simulator systems to enable users with models in other simulator systems to use gem5's features (Figure 2 ⑨). For instance, gem5 has been integrated with the Structural Simulation Toolkit (SST) [16] which uses gem5's detailed CPU models in conjunction with SST's multi-node modeling capabilities. The IEEE standard SystemC API [25] has also been integrated with gem5 to enable users with SystemC models to use them as gem5 components.

Although there are many computer architecture simulators, and many of these are open source with features that overlap with gem5, gem5 is a unique simulation infrastructure.

- gem5 is *dynamically configurable* through a robust Python-based scripting interface. Most other simulators are configured statically with flat text files (e.g., json) or at compilation time. On the other hand, gem5 allows users to simulate complex systems much more easily by using object-oriented Python scripts to compose simpler systems into more complex ones.
- gem5 is *extensible* through a clean model API. gem5 has over 300 models and adding new models is straightforward and well documented.
- gem5 is a *full system* simulator. Its high-fidelity models can support booting unmodified operating systems and running modified applications with cycle-level statistics.
- gem5 is a *community-driven* and *frequently updated* project. The gem5 community is thriving. Since its original release nine years ago, there have been over 250 unique contributors and over 7500 commits. Even in the last six months, gem5 has had over 850 commits and 50 unique contributors.

1.3 Becoming part of the gem5 community

As a reader of this paper, you are already becoming part of the gem5 community! Anyone who uses gem5 or contributes in any way is part of the gem5 community. Contributing can be as simple as sending a question on the gem5 mailing list[12] or as complex as adding a new model to the upstream codebase. Below, we discuss some of the common ways to use gem5 and become part of the community.

1.3.1 For researchers. Currently, the most common use case for gem5 is in computer architecture research. In this case, researchers download the gem5 source, build the simulator, and then add their own device models on top of the models included in upstream gem5. This use case requires deep knowledge of the core simulation frameworks of gem5. However, we are working to make it easier to get started developing and researching with gem5 through efforts like the *Learning gem5* materials and online course **??**.

After using gem5 in their research, we encourage these users to contribute their improvements and fixes to the gem5 back to the mainline codebase. Not only does this improve gem5 for others, but it also makes reproducing research results easier. Rather than managing many local changes and trying to keep up with new releases of gem5, when code is contributed upstream it is the responsibility of *others in the community* to ensure that the code stays up to date.

[12] http://www.gem5.org/mailing_lists/

Additionally the gem5 project employs a permissive BSD license to lower the barrier of contribution for both academics and industry researchers.

1.3.2 For students and teachers. The gem5 simulator can be used as a tool for teaching computer architecture as well. Historically, there has been a very steep learning curve for using gem5 even for simple experiments. However, with the improved documentation, this learning curve is coming down.

We will be continuing to improve gem5 with the goal of making it easier for both students and teachers to use to learn and teach computer architectures concepts. For example, the new *Learning gem5* material created for the online course will include a set of example exercises that we hope can be used in both undergraduate and graduate computer architecture courses. Additionally, we are working to develop a new GUI-based frontend for gem5 and to develop known-good models that do not required deep knowledge of simulator internals to configure and use.

2 MAJOR CHANGES IN GEM5-20

In addition to the systematic changes in project management discussed in Section **??** there has also been innumerable improvements to the codebase. 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[13]

The gem5 simulator has a steep learning curve. Most of the time, using gem5 in research means *modifying* the simulator to change or add new models. 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. 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 [24]. 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[14] 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" initially offered summer of 2020. This "summer school" will mainly be an online class (e.g., Coursera) with all videos available on the gem5 YouTube channel[15], 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.

[13] By Jason Lowe-Power
[14] http://www.gem5.org/documentation/learning_gem5/introduction/
[15] https://www.youtube.com/channel/UCCpCGEj_835WYmbB0g96lZw

2.2 Testing in gem5[16]

Heard back, waiting for the text.

2.3 Updating Guest↔Simulator APIs[17]

It is sometimes helpful or necessary for gem5 to interact with the software running inside the simulation in some non-architectural way. For instance, gem5 might want to intervene and adjust the guest's behavior to skip over some uninteresting function, like one that sets all of physical memory to zeroes, or which uses a loop to measure CPU speed or implement a delay. It might also want to monitor guest behavior to know when something important like a kernel panic has happened. Guest software might also want to purposefully request some behavior from gem5. It could, for instance, request that gem5 exit, record the current value of the simulation statistics, take a checkpoint, read or write a file on the host, etc.

One way gem5 reacts to guest behavior is by requesting a callback when the guest executes a certain program counter (PC). The PC would generally come from the table of symbols loaded with, for instance, an OS kernel, and would let gem5 detect when certain kernel functions were about to execute. This mechanism has been improved to make it easier for different types of CPU models to implement. These include the CPU models which use KVM and the ARM Fast Model based CPUs.

The gem5↔guest interaction might also be triggered by the guest itself. One common way to use these mechanisms from within the guest is to use the "m5" utility which parses command line arguments and then triggers whatever gem5 behavior was requested. This utility is in the process of being revamped so that support is consistent across ISAs, along with many other improvements including supporting all the back end mechanisms described above.

Because it is not possible to universally predict what PCs correspond to requests from the guest, a different signaling mechanism is necessary. Traditional gem5 CPU models redefined unused opcodes from the target ISA for that purpose. However, this mechanism is not universal. For instance, when using the KVM-based CPU model instructions behave like they would on real hardware since they are running on real hardware. In these special cases, we require other APIs.

Finally, the gem5 simulator code must be able to decipher the calling convention of guest code. Historically this was done in several different ways. These were somewhat redundant, inconsistent, incomplete, and difficult to maintain.

We have implemented a new system of templates to pull apart a function's signature and marshal arguments from within the guest automatically. Those arguments are then used to call an arbitrary function in gem5. Once the function finishes, it can optionally return a value into the guest if it wants to override or just observe guest behavior.

For instance, suppose we had the function shown in Figure 3a. If we wanted to call it from within the guest using calling convention AAPCS32, once gem5 had detected the call (as described above), it could call `foo()` with arguments from the guest as shown in Figure 3b.

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[18], 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

[16] by Sean Wilson and Robert R. Bruce
[17] By Gabriel Black
[18] https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html

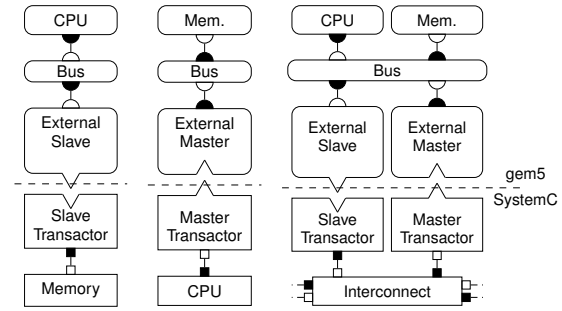
```
int
foo(char bar, float baz)
{
    return (baz < 0) ? bar : bar + 1;
}
```

(a) Example guest↔function.

```
invokeSimcall<Aapcs32>(tc, foo);
```

(b) Example gem5 code.

Fig. 3. Example use of new Guest↔Simulator APIs



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

Fig. 4. Possible scenarios for binding gem5 and SystemC.

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[19]. 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 4. Menard et al. documented our approach and showed that the transaction between gem5 and TLM only

[19]By Chistian Menard, Matthias Jung, Abdul Mutaal Ahmad, and Jeronimo Castrillon

introduces a low overhead of about 8% [25]. The source code as well as basic usage examples can be found in `util/tlm` of the gem5 repository.

2.4.2 SystemC in gem5[20]. Alternatively, gem5 also has its own built in SystemC kernel and TLM implementation, and can run models natively as long as they are recompiled with gem5's SystemC header files. These models can then use gem5's configuration mechanism and be controlled from Python, and, by using modified versions of the bridges developed to run gem5 within a SystemC simulation, TLM sockets can be connected to gem5's native ports.

This approach integrates models into gem5 more cleanly and fully since they are now first class gem5 models with access to all of gem5's APIs. Existing models and `c_main` implementations can generally be used as-is without any source level modifications; they just need to be recompiled against gem5's SystemC headers and linked into a gem5 binary.

While some parts of gem5's SystemC implementation are taken from the open source reference implementation (most of the data structure library and TLM), the core implementation is new and based off of the SystemC standard. This means that code which depends on nonstandard features, behaviors, and implementation specific details of the reference implementation may not compile or work properly within gem5. That said, gem5's SystemC kernel passes almost all of the reference implementation's test suite. The few exceptions are tests that are broken, explicitly test for implementation specific behavior, or test for deprecated and undocumented features.

2.5 Cache Replacement Policies and New Compression Support[21]

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 [19]. These policies can be used with both the classic caches and Ruby caches. 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 [31] 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 [27], C-Pack [12], and FPCD [5] 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.

[20] By Gabriel Black
[21] By Daniel Carvalho

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 unifying the Ruby and classic cache models. 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[22]. 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.

Additionally, a new three level coherence protocol (MESI_Three_Level) 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 instructions and data. The first and the second levels do not distinguish between instructions and data. Levels 0 and 1 are private to each CPU core while the second level is shared across cores, all of them or possibly a subset.

2.6.2 GPU Coherence Protocols[23]. Haven't heard anything, yet.

2.6.3 Arm Support in Ruby Coherence Protocols[24]. Until recently, configurations combining Ruby and multicore Arm systems were not properly supported. We have revamped the `MOESI_CMP_directory` protocol and made it the default when building gem5 for Arm. Several issues that resulted in protocol deadlocks (especially when scaling up to many-core configurations) were fixed. Other fixes include support for functional accesses, DMA bugs, and improved modeling of cache and directory latencies. Additionally, support for load-locked/store-conditional (LL/SC) operations was added to the `MESI_Three_Level` protocol, which enables it to be used with Arm as well.

2.7 Garnet Network Model[25]

The interconnection system within gem5 is modeled in various levels of detail and provides extensive flexibility in terms of modeling modern systems. The interconnect models are present within the cache-coherent ruby memory system of gem5. It provides the ability to create arbitrary topologies – thereby constructing both homogeneous and heterogeneous systems. There are two major variants of network models available within the ruby memory system today: simple and garnet. The Simple network models the routers, links, and the latencies involved with minimal detailing. This is great for simulations that can sacrifice interconnect detailing for faster simulation. The Garnet model

[22] by Nilay Vaish
[23] by Blake Hectman
[24] by Tiago Mück
[25] By Srikant Bharadwaj and Tushar Krishna

adds detailed router microarchitecture with cycle-level buffering, resource-contention and flow control mechanisms [2]. This model is suitable for studies that focus on interconnection units and data flow patterns.

gem5 currently implements an upgraded Garnet 2.0 model which provides custom routing algorithms, routers/links that support heterogenous latencies, and standalone network simulation support. These features allow detailed studies of on-chip networks as well as support for highly flexible topologies. Garnet is moving to version 3.0 with the release of HeteroGarnet which is underway. HeteroGarnet revamps Garnet to support the modern heterogenous systems such as 2.5D integration systems, MCM based architectures, and futuristic interconnect designs such as optical networks [8]. We are also working to include support for recent work on routerless NoCs [](https://ieeexplore.ieee.org/abstract/document/8327032, https://ieeexplore.ieee.org/abstract/document/9065600).

2.8 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 [37]. In that time, the number of RISC-V users 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.

2.8.1 General RISC-V ISA Implementation[26] *[28, 32].* 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 are not detailed enough and RTL simulation is too time consuming for these methods to be feasible for architectural parameter exploration. With FPGA emulation, it is difficult to retrieve performance information without modifying both the RTL design and the system software. 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 this 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. 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. 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

[26] By Alec Roelke

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. Additionally, gem5's version of the RISC-V test-suite[27] has been updated to the latest version and several corner cases in gem5 have been fixed, so that now most of the tests are working correctly.

2.8.2 RISC-V Full System Support[28]. To simulate complete operating systems as well, full system simulation was added for RISC-V. More specifically, Sv39 paging according to the privileged ISA 1.11[29] with a 39-bit virtual address space, a page-table walker performing a three-level translation, and a translation lookaside buffer has been added. The page-table walker code has been based on the existing gem5 code for x86 due to the structural similarities. While a few steps are still missing to run Linux, general support to run a complete RISC-V operating system on gem5 is available now.

2.9 Branch Predictor Improvements[30]

In gem5, multiple branch prediction models are available, many of which were added since the initial release of gem5. Currently, gem5 supports 5 different branch prediction techniques including the well-known TAGE predictor besides standard predictors such as bi-mode, tournament, etc. This list can easily be expanded to cover different variants of these well-known branch predictors. Besides, the support for loop predictor and indirect branch predictor is also available.

Furthermore, the modularity of the implementation of different branch predictors allows ease of inclusion of secondary or side predictors into the prediction mechanism of primary predictors. For example, TAGE can be seamlessly augmented with a loop predictor to predict loops with constant iteration numbers. Indirect branch predictor can be made to use complex TAGE-like scheme instead of simple history-based predictors with only a few hours of development effort. In addition to this, these different predictors can be configured with different sizes of history registers and table-like structures. For example, TAGE predictor can be configured to run with different sizes of the history register and consequently different number of predictor tables, allowing users to carry research on the effects of different predictor sizes in various performance metrics.

Future development is planned to include the support of neural branch predictors (e.g., perceptron branch predictor, etc.) and different variants of TAGE and perceptron predictors that have demonstrated significant improvement in branch misses in recent years.

2.10 GPU Compute Model[31]

Heard back, waiting for text.

2.10.1 Autonomous Data-Race-Free GPU Tester[32]. 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

[27] https://github.com/riscv/riscv-tests
[28] By Nils Asmussen
[29] https://riscv.org/specifications/privileged-isa/
[30] by Dibakar Gope
[31] by Anthony Gutierrez
[32] by Tuan Ta

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 [33].

2.11 Syscall Emulation Improvements[33]

System call emulation mode (SE mode) allows gem5 to execute user-mode binaries without executing the kernel-mode system calls of a real operating system. Basic functionality existed in the original gem5 release [9], but major improvements have been made in the past few years. Recent additions improve the usability and increase the variety of workloads which may run in SE mode.

2.11.1 Dynamic Executables. For many years, gem5 supported only statically linked executables. This limitation prevented evaluation of workloads which require dynamic linking and loading. To support these workloads, the SE mode infrastructure was modified to support dynamic executables using ELF [](ELF spec).

At a high level, the internal ELF loader was altered to detect the requirement of an interpreter—the tool responsible for handling dynamic loaded libraries. If the interpreter is required, the ELF loader will load the both the interpreter and the workload into the process address space within the simulator. The ELF loader will also initialize stack variables to help the interpreter and the workload find each other. With the interpreter in the address space, the workload will delegate lookups (function bindings) to the interpreter which will fixup function call invocation points on behalf of the workload.

Several system calls were modified to enable the functionality. Most of the changes were related to file handling and memory mappings.

This dynamic executable support can be combined with the virtual file system described below in Section 2.11.3 to enable cross-platform compatibility. With this support, users can run dynamically linked SE mode binaries for any ISA on any host ISA as long as the dynamic linker, loader, and libraries are present on the host machine.

2.11.2 Threading Library Support. With dynamic executable support, users encounter issues with libraries which depend on pthreads. Many common libraries have a dependency on the pthread library. This results in a transitive

[33] by Brandon Potter

dependency from the perspective of the workload. To meet the dependency, we decided to directly support usage of native threading libraries. There are currently two options to support threading in SE mode: m5threads and native threading libraries (i.e. pthreads).

The first option, m5threads, is a custom threading library originally conceived to provide threading support specifically for gem5. The library predates dynamic executable support and therefore is statically linked against the workloads which require threading. The m5threads library is an incomplete threading library; it does not implement the entire API supported by a library like pthreads.

The second option, native threading libraries, utilizes dynamic executable support to make standard system libraries like pthreads available to the workload. To use this feature, the user must ensure that enough thread contexts have been allocated in their configuration script to support all threads.

The threading library support required changes to the SE mode infrastructure. Specifically, the clone system call required support for many new options and the futex system call required significant work.

2.11.3 Virtual File System. In SE mode, many system call implementations rely on functionality provided by the host machine. For example, a workload's invocation of the "open" system call will cause the gem5 CPU model to hand control over to the simulator. The SE mode "open" implementation will then call the glibc open function on the host machine (which in-turn uses the host machine's open system call). Effectively, the system call is passed from the simulated process space down to the host machine. This pass-through method has been utilized since SE mode has existed.

There are several reasons to employ pass-through:

(1) It avoids reimplementing complicated operating system features.
(2) It promotes code reuse by not specializing the system call implementation for each ISA.
(3) It allows the host resources to be utilized directly from the simulated process.

There are several drawbacks stemming from pass-through:

(1) It creates API mismatches for system calls which rely on glibc library implementations. Specifically, a system call's options may differ for simulated ISA and the host ISA.
(2) It creates ABI mismatches for system calls which directly call into the host system call without interpreting system call parameters.
(3) It creates issues when utilizing some host resources.

The virtual file system provides a solution for the third drawback specifically for filesystem handling. When files are touched by the simulated process, the results of the accesses or modifications pass-through to the host filesystem. For some cases, this causes problems. For example, reading the contents of "/proc/cpuinfo" will report back results which differ from the simulated system's configuration. In another example, the workload might try to open "/dev/thing" for device access.

To obviate these problem, the virtual file system provides a level of indirection to catch filesystem path evaluations and modify them before pass-through occurs. Any path on the simulator can be redirected to any location on the host similar to mounting volumes in docker. The `key:value` strings for path redirection can be set via the Python configuration files.

2.11.4 AMD ROCm v1.6. At the time of publication, a dated version of the ROCm software stack can be used with x86 ISA builds. The ROCm v1.6 libraries can be loaded and used on both RHEL6 and Ubuntu 16.04 operating systems.

We distribute a set of docker containers and dockerfiles to help users get started using this specific version of ROCm with gem5.

2.12 Arm Improvements

2.12.1 Armv8 Support[34]. The Armv8-A architecture introduced two different architectural states: AArch32, supporting the A32 and T32 instruction sets (backward-compatible with Armv7-A's Arm and Thumb instruction sets, respectively), and AArch64, a new state offering support for 64-bit addressing via the A64 instruction set. gem5 currently supports all of the above instruction sets, and the interworking between them. In addition, on top of the user-level features, several contributions have targeted important system-level extensions, e.g. the Security (aka TrustZone [6]) and virtualization extensions [**?**], thus opening up new avenues for architectural and micro-architectural research.

While Armv8-A was a major iteration of the architecture, there have been several smaller iterations introduced by Arm with a yearly cadence, and various contributors have implemented some of the main features from those extensions, up to Armv8.3-A.

2.12.2 Support for the Arm Scalable Vector Extension (SVE)[35]. In 2016, Arm introduced their Scalable Vector Extension (SVE) [**?**], a novel approach to vector instruction sets. Instead of having fixed-size vector registers, SVE operates on registers that can be anywhere between 128 to 2048 bit long (in 128-bit increments). SVE code is arranged in a way that is agnostic to the underlying vector length (Vector Length Agnostic Programming), and a single SVE instruction will perform its operation on as many elements as the vector register can fit, depending on its length. On top of the 32 variable-length vector registers, SVE also adds 16 variable length predicate registers for predicated execution. These registers store one bit per byte (the minimum element size) in the vector register, and can be used to select specific elements in your vector for operation [**?**].

In order to support SVE, gem5 implements register storage and register access as two separated classes, a container and an interface, decoupling one from the other. The vector registers can be of any arbitrary size and be accessed as vectors of elements of any particular type, depending on the operand types of each instruction. This not only facilitates handling variable size registers, it also abstracts the nuances of handling predicate registers, where the values stored have to be grouped and interpreted differently depending on the operand type.

This design provides enough flexibility to support any vector instruction sets with arbitrarily large vector registers.

2.12.3 Trusted Firmware Support[36]. Trusted Firmware (TF-A) is Arm's reference implementation of Secure World software for A-profile architectures. It enables Secure Boot flow models, and provides implementations for the Secure Monitor executing at Exception Level 3 (EL3) as well as for several Arm low-level software interface standards, including System Control and Management Interface (SCMI) driver for accessing System Control Processors (SCP), Power State Coordination Interface (PSCI) library support for power management, and Secure Monitor Call (SMC) handling.

TF-A is supported on multiple Platforms, each of them characterized by its set of hardware components and their location in the memory map. From the list of ADPs (Arm Development Platforms), it is worth mentioning the Juno ADP and the Fixed Virtual Platforms (FVP) ADP family. However, the Arm reference platforms in gem5 are part of the `VExpress_GEM5_Base` family. These are loosely based on a Versatile Express RS1 platform with a slightly modified memory map. TF-A implementations are provided for both Juno and FVPs, however not for `VExpress_GEM5_Base`.

[34] by Giacomo Gabrielli, Javier Setoain, and Giacomo Travaglini
[35] by Giacomo Gabrielli, Javier Setoain, and Giacomo Travaglini
[36] by Adrian Herrera

Towards unifying Arm's platform landscape, we now provide a `VExpress_GEM5_Foundation` platform as part of gem5's `VExpress_GEM5_Base` family. This is based on and compatible with FVP Foundation, meaning all Foundation software may run unmodified in gem5, including but not limited to TF-A. This allows for simulating boot flows based on UEFI implementations (U-boot, EDK II), and brings us a step closer to Windows support in gem5.

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[37]. 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.

2.13.2 Python 3[38]. One of the main features which separates gem5 from other architectural simulators is its robust support for scripting. The main interface to configuring and running gem5 simulations is Python scripts. 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

[37] by Andreas Sandberg
[38] by Andreas Sandberg and Giacomo Travaglini

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 added 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.

2.13.3 Asynchronous Modeling in gem5[39]. The difficulties of writing a complex device/hw model within gem5 is that your model needs to be able to work and be representative of the simulated hardware in both atomic and timing mode.

For simple devices which only respond to requests, this is usually not a concern. The situation gets worse when the device can send requests and response or has DMA capabilities. A method generating and forwarding a read packet needs to differentiate between atomic and timing behavior by handling the first with a blocking operation (the read returns the value as soon as the forwarding method returns) and the second with a non-blocking call: the value will be returned later in time. The situation becomes dramatic in timing mode if multiple sequential DMAs are stacked so that any read operation depends on previous ones; this is the case for page table walks for example.

This software design problem has been elegantly solved using coroutines. Coroutines allow you to execute your task, checkpoint it, and resume it later from where you stopped. To be more specific to our use case, you can tag your DMA packets with the coroutine itself, and you could resume the coroutine once the device receives the read response.

While waiting for coroutines to be fully supported in C++20, we've implemented a coroutine library within gem5 that allows developers to use coroutines to generate asynchronous models. The coroutine class is built on top of a "Fiber" class, which was a pre-existing symmetric coroutine implementation, and it provides boost-like [] APIs to the user.

At the moment coroutines are used by the SMMUv3 model developed and the GICv3 ITS model (Interrupt Translation Service). There are many other use cases for this API in other gem5 models, and we are planning on updating those models in the future.

2.14 Off-Chip Memory System Models[40]

gem5 can model a large number of configurations in the off-chip memory system. Its memory controller handles requests from the on-chip memory system and issues read and write commands for the actual memory device []() and models their timing behavior. Over the years a number of contributions have added features that allow modeling of emerging new technologies and features.

2.14.1 New memory controller features[41]. The gem5 DRAM controller provides the interface to external memory, which is traditionally DRAM. The controller consists of 2 main components: the memory controller and the DRAM interface. The DRAM interface contains media specific information, defining the architecture and timing parameters of the DRAM as well as the functions that manage the media specific operations like activation, precharge, refresh and low power modes.

[39] by Giacomo Travaglini
[40] by Nikos Nikoleris
[41] by Wendy Elsasser

2.14.2 Low-power DDR[42]*.* LPDDR5 is currently in mass production for use in multiple markets including mobile, automotive, AI, and 5G. This technology is expected to become the mainstream Flagship Low-Power DRAM by 2021 with anticipated longevity due to proposed speed grade extensions. The specification defines a flexible architecture and multiple options to optimize across different use cases, trading off power, performance, reliability and complexity. To evaluate these tradeoffs, we have updated the memory controller to support the new features and added LPDDR5 configurations.

While these changes have been incorporated for LPDDR5, some of them could be applicable to other memory technologies as well. The gem5 changes incorporate new timing parameters, support of multi-cycle commands and support of interleaved bursts. These features require new checks and optimizations in gem5 to ensure the model integrity when comparing to real hardware. For example, support for multi-cycle commands along with the changes to LPDDR5 clocking motivated a new check in gem5 to verify command bandwidth. Previously, the DRAM controller did not verify contention on the command bus and assumed adequate command bandwidth, but with the evolution of new technologies this assumption is not always valid.

2.14.3 Quality of Service Extensions[43]*.* The coexistence of heterogeneous tasks/workloads on a single computer system is common practice in modern systems, from the automotive to the high-performance computing use-case. It allows the system to minimize costs by improving the resources utilization and improving the efficiency of data sharing across workloads. This, however, comes at the cost of potential severe performance degradation due to interference on shared resources, and increased uncertainty in terms of workload performance predictability.

To compensate for these shortcomings, we stress the need to introduce a mechanism for predictively and deterministically managing such systems resources, i.e., providing Quality of Service (QoS). The concept and challenges of QoS itself are not new and achieving QoS is generally hard in complex systems, as we learned from Computer Networking. We therefore define here QoS in Systems on Chips on the following two principles:

(1) *QoS is resource access arbitration*: a QoS-enabled resource (e.g., memory) guarantees certain Levels of Service (memory access bandwidth and latency, compute time, peripheral access, etc.) to its serviced users (e.g., software threads)
(2) *QoS is quantifiable and predictable:* the set of guarantees a QoS-enabled resource can provide are known a priori and characteristic of the implemented QoS arbitration schemes. The level of service that a QoS enabled resource will guarantee to its users must therefore be predictable to a certain extent given a specific set of policies and their configurations.

Quality of Service is the ability of a system to provide differential treatment to its clients, in a quantifiable and predictable way.

The contribution involved the definition of a QoS-aware memory controller in gem5, and the definition of basic (example) policies modelling the prioritization algorithm of the memory controller. Those are the Fixed priority policy (every master in the system has a fixed priority assigned) and the Proportional Fair policy (where the priority of a master is dynamically adjusted at runtime based on utilization). The DRAM controller in gem5 had been rewritten to include the QoS changes; with the framework in place a user can write its own policy and seamlessly plug it into a real memory controller model to unlock system wide explorations under its own arbitration algorithm.

[42] by Wendy Elsasser
[43] by Matteo Andreozzi

2.14.4 DRAMPower and DRAM Power-Down Modes[44]. 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 [11] 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[45], 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 [18] 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.

2.14.5 Future Improvements to Off Chip Memory Models[46]. With the advent of SCM (storage class memory), emerging NVM could also exist on a memory interface, potentially alongside DRAM. To enable support of NVM and future memory interfaces, a systematic approach was chosen to refactor the DRAM controller. The DRAM interface was pulled out of the controller and moved to a separate DRAM interface object. In parallel, an NVM interface was created to model an agnostic interface to emerging memory.

The DRAM interface and the NVM Interface have configurable address ranges allowing flexible heterogeneous memory configurations. For example, single memory controller can have a DRAM interface, an NVM interface, of both interfaces defined. Other configurations are feasible, providing a flexible framework to study new memory topologies and evaluate the placement of emerging NVM in the memory sub-system.

2.15 Virtualized Fast Forwarding[47]

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

[44] by Matthias Jung, Wendy Elsasser, Radhika Jagtap, Subash Kannoth, Omar Naji, Éder F. Zulian, Andreas Hansson, Christian Weis, and Norbert Wehn
[45] https://www.micron.com/support/tools-and-utilities/power-calc
[46] by Wendy Elsasser
[47] by Andreas Sandberg

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[48]

Haven't head back, yet.

2.17 Memory Traces and Traffic Generator[49]

Haven't heard back, yet.

2.18 Classic Caches Improvements[50]

The classic memory system implements a snooping MOESI-like coherence protocol that allows for flexible, configurable cache hierarchies. The coherence protocol is primarily implemented in the Cache and the CoherentXBar and the SnoopFilter implements a common optimization to reduce unnecessary coherence traffic.

Over the years, the components of the classic memory system have received significant contributions with a primary focus is adding support for future technologies and enhancing its accuracy.

2.18.1 Non-Coherent Cache. The cache model in gem5 implements the full coherence protocol and as a result can be used in any level of the coherent memory subsystem (e.g., L1 data cache or instruction cache, last-level cache). The non-coherent cache is a stripped down version of the cache design to be used below the point-of-coherence (closer to memory). Below the point-of-coherence, the non-coherent cache receives only requests for fetches and writebacks and itself send requests for fetches and writebacks to memory below. As opposed to the regular cache, the non-coherent cache will not send any snoops to invalidate or fetch data from caches above. As such the non-coherent cache is a greatly simplified version in terms of handling the coherence protocol compared to the regular cache while otherwise supporting the same flexibility (e.g., configurable tags, replacement policies, inclusive or exclusive, etc.).

The non-coherent cache can model large system-level caches which are used by the CPUs and other devices in the system.

2.18.2 Write Streaming Optimizations. Write streaming is a common access pattern which is typically encountered when software initializes or copies large memory buffers (e.g., memset, memcpy). When executed, the core issues a

[48] by Cutis Dunham
[49] by Andreas Hanson
[50] by Nikos Nikoleris

large number of write requests to the data cache. The data cache receives these write requests and issues requests for exclusive copies of the corresponding cache lines. To get an exclusive copy, it has to invalidate copies of that line and fetch a copy of the data (e.g., from off-chip memory). As soon as it receives data, it performs all writes for that line and often will overwrite it completely. As a result, the data cache unnecessarily fetches data only to overwrite it shortly after. Often these write buffers are large in size and also trash the data cache.

Common optimizations [21] coalesce writes to form full cache line writes, avoid unnecessary data fetches and achieve significant reduction in memory bandwidth. In addition, when the written memory buffer is large, we can also avoid thrashing the data cache by bypassing allocation.

We have implemented a simple mechanism to detect write streaming access patterns and enable coalescing and bypassing. The mechanism attaches to the data cache and analyses incoming write requests. When the number of sequential writes reaches a first threshold, it enables write coalescing and when a second threshold is reached, in addition, the cache will bypass allocation for the writes in the stream.

2.18.3 Cache Maintenance Operations. Typically, the contents of the cache are handled by the coherence protocol. For most user-level code, caches are invisible. This greatly simplifies programming and ensures software portability. However, when interfacing with devices or persistent memory, the effect of caching becomes visible to the programmer. In such cases, a user might have to trigger a writeback which propagates all the way to the device or the persistent memory. In other cases, a cache invalidation will ensure that a subsequent load will fetch the newest version of the data from a buffer of the main memory.

Cache maintenance operations (CMOs) are now supported in gem5 in a way that can deal with arbitrary cache hierarchies. An operation can either clean and/or invalidate a cache line. A clean operation will find the dirty copy and trigger a writeback and an invalidate operation will find all copies of the cache line and invalidate them and the combined operation will perform both actions. The effects of CMOs are defined with reference to a configurable point in the system. For example, a clean and invalidate to the point-of-coherence will find all copies of the block above the point-of-coherence, invalidate them, and if any of them is dirty also trigger a writeback to the memory below the point-of-coherence.

2.18.4 Snooping Support and Snoop Filtering. In large systems, broadcasting snoop messages is slow; costs energy and time; and can cause significant scalability bottlenecks. Therefore, directories and filters are used to keep track of which caches / nodes are keeping a copy of a particular cached line. We added a snoop filter to gem5 which is a distributed component that keeps track of the coherence state of all lines cached "above" it, similar to the AMD Probe Filter [] (Conway, Pat, Nathan Kalyanasundharam). For example, if the snoop filter sits next to (in front of) the L3 cache, it knows about all lines in the L2 and L1 caches that are connected to that L3 cache.

Using the snoop filter, we can reduce the amount of messages from $O(N^2)$ to $O(N)$ with N concurrent masters in the system. Modelling the snoop filter / directory separately from the cache allows us to use different organizations for the filter and the cache, and distributing area between shared caches vs coherence tracking filters / directories. We also model the effect of limited filter capacity through back-invalidations that remove cache entries if the filter becomes full; allowing for more realistic cache performance metrics. Finally, the more centralized coherence tracking in the filter allows for more tight checking of correct function of the distributed coherence protocol in the classic memory system.

2.19 dist-gem5: Support for Distributed System Modeling[51]

Designing distributed systems requires careful analysis of the complex interplay between processor microarchitecture, memory subsystem, inter-node network, and software layers. However, simulating a multi-node computer system using one gem5 process can take an eon. Responding to the need for efficient simulation of multi-node computer systems, dist-gem5 enables parallel, distributed simulation of a hierarchical, computer cluster using multiple gem5 processes. dist-gem5 spawns several gem5 processes, in which each of them can simulate one or several computer systems (i.e., compute node) or a scale-out network topology (i.e., network node). dist-gem5 automatically launches these gem5 processes, forward simulated packets between them through real TCP connections, and performs quantum-based synchronization to ensure correct and deterministic simulation.

More specifically, dist-gem5 consists of the following three main components:

Packet forwarding: dist-gem5 establishes a TCP socket connection between each compute node and a corresponding port of the network node to (i) forward simulated packets between compte nodes through the simulated network topology and (ii) exchange synchronization messages. Within each gem5 process, dist-gem5 launches a receiver thread that runs in parallel with the main simulation thread to free the main simulation thread from polling on the TCP connections.

Synchronization: In addition to network topology simulation, the network node implements a synchronization barrier for performing quantum-based synchronization. dist-gem5 schedules a global sync event every quantum in each gem5 process. The process() method of the global sync event in compute nodes sends out a "sync request" message through the TCP connection to the network node and waits for the reception of a "sync ack" to start the next quantum simulation. On the other hand, the `process()` method of the network node waits for the reception of "sync requests" from all compute nodes and then sends out "sync acks" to each compute node.

Distributed checkpointing: dist-gem5 supports distributed checkpointing by capturing the external, inter gem5 process states, including the in-flight packets inside the network node. To ensure that no in-flight message exists between gem5 processes when the distributed checkpoint is taken, dist-gem5 only initiates checkpoints at a periodic global sync event.

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[52]

Haven't heard back, yet.

2.21 Runtime Power Modeling and DVFS Support[53]

Virtually all processing today needs to consider not just aspects of performance, but also that of energy and power consumption. Systems are either constrained on power or thermal run conditions (mobile devices, boosting of desktop systems), or need to operate as energy efficiently as possible (in HPC and data centers). We have added support to gem5 to faithfully model power-relevant silicon structures (voltage and frequency domains) enabling DVFS (dynamic voltage and frequency scaling), devices that allow for DVFS control by operating system governors and autonomous control, and activity-based power modelling that measures key micro-architectural events, voltage, and frequency and allows

[51] by Mohammad Alian
[52] by Andrew Bardsley
[53] by Stephan Diestelhorst

detailed aggregation of power consumed over time. We have reported on that work in earlier material [] (V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth and S. Kaxiras, Introducing DVFS-Management in a Full-System Simulator, 2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, 2013, pp. 535-545, doi: 10.1109/MASCOTS.2013.75.), and have extended the flexibility of the power equation models and available activities since, for example including power consumption caused by the activity of the SVE vector units.

2.22 Elastic Traces[54]

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 [17]. 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 ACKNOWLEDGEMENTS

The development of gem5 is distributed. The contributions to the gem5 community go beyond just the source code, and many people who have contributed to the broader gem5 community are not acknowledged here. Note that until this paper is submitted, please contact Jason Lowe-Power if you have any questions about the contributions.

We would like to specially acknowledge the late Nathan Binkert. Nate was a driving force behind the creation of gem5 and without his vision and his dedication to code quality this open-source community infrastructure would not be the success that it is today.

The gem5 project management committee consists of Bradford Beckmann, Gabriel Black, Anthony Gutierrez, Jason Lowe-Power (chair), Steven Reinhardt, Ali Saidi, Andreas Sandberg, Matthew Sinclair, Giacomo Travaglini, and David Wood. Previous members include Nathan Binkert, and Andreas Hansson. The project management committee manages the administration of the project and ensures that the gem5 community runs smoothly.

This work is supported in part by the National Science Foundation (CNS-1925724, CNS-1925485, CNS-1850566) and Brookhaven National Laboratory.

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

[54]by Radhika Jagtap, Matthias Jung, Stephan Diestelhorst, Andreas Hansson, Thomas Grass, and Norbert Wehn

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] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*. IEEE, 33–42.

[3] Ayaz Akram and Lina Sawalha. 2016. x86 Computer Architecture Simulators: A Comparative Study. In *IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 638–645.

[4] Ayaz Akram and Lina Sawalha. 2019. Validation of the gem5 Simulator for x86 Architectures. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 53–58.

[5] 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.

[6] Tiago Alves and Don Felton. 2004. TrustZone: Integrated Hardware and Software Security. *Information Quarterly* (2004), 18–24.

[7] Mochamad Asri, Ardavan Pedram, Lizy K John, and Andreas Gerstlauer. 2016. Simulator Calibration for Accelerator-Rich Architecture Studies. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS),*. IEEE, 88–95.

[8] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna. 2020. Kite: A Family of Heterogeneous Interposer Topologies Enabled via Accurate Interconnect Modeling. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*.

[9] 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

[10] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. 2012. Accuracy Evaluation of GEM5 Simulator System. In *IEEE 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*. York, UK, 1–7.

[11] Karthik Chandrasekar, Christian Weis, Yonghui Li, Benny Akesson, Omar Naji, Matthias Jung, Norbert Wehn, and Kees Goossens. 2014. DRAMPower: Open-source DRAM power & energy estimation tool. http://www.drampower.info.

[12] 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.

[13] R. de Jong and A. Sandberg. 2016. NoMali: Simulating a realistic graphics driver stack using a stub GPU. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 255–262.

[14] Fernando A Endo, Damien Couroussé, and Henri-Pierre Charles. 2014. Micro-architectural Simulation of In-Order and Out-of-Order ARM Microprocessors with gem5. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*. IEEE, 266–273.

[15] Anthony Gutierrez, Joseph Pusdesris, Ronald G Dreslinski, Trevor Mudge, Chander Sudanthi, Christopher D Emmons, Mitchell Hayenga, and Nigel Paver. 2014. Sources of Error in Full-System Simulation. In *IEEE International Symposium on Performance Analysis of Systems and Software*. Monterey, CA, 13–22.

[16] Mingyu Hsieh, Kevin Pedretti, Jie Meng, Ayse Coskun, Michael Levenhagen, and Arun Rodrigues. 2012. SST + Gem5 = a Scalable Simulation Infrastructure for High Performance Computing. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques* (Desenzano del Garda, Italy) *(SIMUTOOLS '12)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 196–201.

[17] Radhika Jagtap, Stephan Diestelhorst, Andreas Hansson, Matthias Jung, and Norbert Wehn. 2016. Exploring System Performance using Elastic Traces: Fast, Accurate and Portable. In *IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS), July, 2016, Samos Island, Greece*.

[18] Radhika Jagtap, Matthias Jung, Wendy Elsasser, Christian Weis, Andreas Hansson, and Norbert Wehn. 2017. Integrating DRAM Power-Down Modes in gem5 and Quantifying their Impact. In *International Symposium on Memory Systems (MEMSYS17)*.

[19] 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

[20] Jae-Eon Jo, Gyu-Hyeon Lee, Hanhwi Jang, Jaewon Lee, Mohammadamin Ajdari, and Jangwoo Kim. 2018. DiagSim: Systematically Diagnosing Simulators for Healthy Simulations. *ACM Transactions on Architecture and Code Optimization (TACO)* 15, 1 (2018), 4.

[21] Norman P. Jouppi. 1993. Cache Write Policies and Performance. *SIGARCH Comput. Archit. News* 21, 2 (May 1993), 191–201. https://doi.org/10.1145/173682.165154

[22] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Comput. Archit. Lett.* 15, 1 (Jan. 2016), 45–49. https://doi.org/10.1109/LCA.2015.2414456

[23] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob. 2020. DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator. *IEEE Computer Architecture Letters* (2020), 1–1.

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

[25] 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.

[26] Tony Nowatzki, Jaikrishnan Menon, Chen-Han Ho, and Karthikeyan Sankaralingam. 2015. Architectural simulators considered harmful. *IEEE Micro* 35, 6 (2015), 4–12.

[27] 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.

[28] Alec Roelke and Mircea R. Stan. 2017. RISC5: Implementing the RISC-V ISA in gem5. In *Proceedings of Computer Architecture Research with RISC-V*.

[29] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters* 10, 1 (2011), 16–19.

[30] A. Sandberg, N. Nikoleris, T. E. Carlson, E. Hagersten, S. Kaxiras, and D. Black-Schaffer. 2015. Full Speed Ahead: Detailed Architectural Simulation at Near-Native Speed. In *2015 IEEE International Symposium on Workload Characterization*. 183–192.

[31] 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.

[32] Tuan Ta, Lin Cheng, and Christopher Batten. 2018. Simulating Multi-Core RISC-V Systems in gem5. In *Proceedings of Computer Architecture Research with RISC-V*.

[33] 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

[34] Teruo Tanimoto, Takatsugu Ono, and Koji Inoue. 2017. Dependence Graph Model for Accurate Critical Path Analysis on Out-of-Order Processors. *Journal of Information Processing* 25 (2017), 983–992.

[35] Matthew Walker, Sascha Bischoff, Stephan Diestelhorst, Geoff Merrett, and Bashir Al-Hashimi. 2018. Hardware-Validated CPU Performance and Energy Modelling. In *Performance Analysis of Systems and Software (ISPASS), 2018 IEEE International Symposium on*. IEEE, 44–53.

[36] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. 2005. DRAMsim: A Memory System Simulator. *SIGARCH Comput. Archit. News* 33, 4 (Nov. 2005), 100–107. https://doi.org/10.1145/1105734.1105748

[37] 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