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), 18 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 [3] 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 [3] is an open source community-supported computer architecture simulator system. It consists of a simulator core and models for everything from out of order processors, to DRAM, to network devices. The gem5 project consists of the gem5 simulator[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. The goal of gem5 is 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.

[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/

The gem5 community strives to balance the needs of its three categories of users: academic researchers, industry researchers, and students. For instance, gem5 strives to balance 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 give an overview of gem5's main features available today, and then quickly discuss the past, present, and future of gem5. Then, we include a significant section which describes the major changes in gem5 in the past nine years since the initial gem5 release.

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 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. 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. 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. 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* [**?**]. Although the KVM CPU model can execute at native speed, it does not model the timing of execution or memory. 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. This allows models designed for one system to be easily used in other system designs.

There are two different cache systems in gem5, Ruby which models cache coherence protocols with high fidelity, and the "classic" caches. Ruby enables user-defined cache coherence protocols, though gem5 includes many protocols

out of the box. There are now 12 unique protocols including GPU-specific protocols [**?**], research protocols like token coherence [**?**] and region-coherence protocols [**?**], and teaching protocols [**?**]. Users can also choose to use the detailed Garnet network mode [**?**] when using Ruby caches which offers cycle-level detail for the on-chip network.

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. 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 [] or Ramulator [], 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 []. This GPU model does not support graphics applications, but supports many compute applications based on OpenCL. (Should probably add a couple more sentences.)

An important component to full system simulation is supporting I/O and other devices. 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 "fake" graphics GPU to enable simulating applications that depend on graphics APIs.

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. For instance, gem5 has been integrated with the System Simulation Toolkit (SST) [] which uses gem5's detailed CPU models in conjunction with SST's multi-node modeling capabilities. The IEEE standard SystemC API [] 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 XXX 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.

1.2 The past nine years

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

At the same time, the contributor community has also grown. Figures **??** shows the number of commits per year and Figure **??** 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 [6]. 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.

1.3 gem5 now

To solve these problems the gem5 community has made major changes in the past nine years. We now have a formal governance structure, we have improved our documentation, we have moved to a better distributed development platform, and improved our community outreach. There are still many ways we can improve (see 1.4), but we have already come a long way!

To solve these problems we

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

1.4 The future of gem5

The future of gem5 is bright.

- Lots of new stuff coming in a new roadmap that's coming soon!
  - Improved python interface
  - New RAM models (including NVM)
  - Garnet 3.0
  - Models for ML accelerators
  - Known-good configurations
  - Many more.
- Here, we talk about how to contribute to gem5.
- We will do a better job recognizing contributions. The gem5 project is only successful with the community is vibrant. We have taken positive steps towards making it easier to contribute (like a governance document and contributing documentation), but we need to do more. We will encourage everyone to contribute.
- More educational material coming!
- More workshops, tutorials, etc.

2 MAJOR CHANGES IN GEM5-20

In addition to the systematic changes in project management discussed in Section 1.3 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[6]

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

Learning gem5 reduces the knowledge gap between new users and experienced gem5 developers. Learning gem5 takes a bottom up approach to teaching new users the internals of gem5. There are currently three parts of Learning gem5, "Getting Started", "Modifying and Extending", and "Modeling Cache Coherence with Ruby". Each part walks the reader through a step-by-step coding example starting from the simplest possible design up to a more realistic example. By explaining the thought process behind each step, the reader gets a similar experience to working alongside an experienced gem5 developer. Learning gem5 includes documentation on the gem5 website[7] and source code in the gem5 repository for these simple ground-up models.

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

2.2 Testing in gem5[8]

Heard back, waiting for the text.

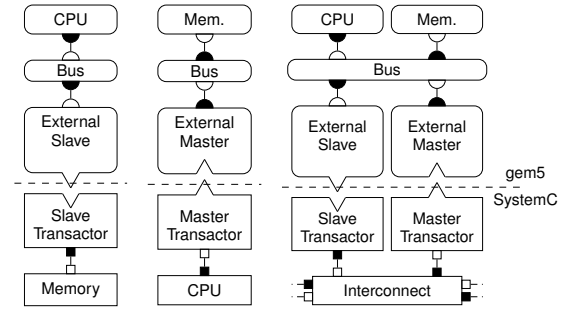
2.3 Updating Guest¡-¿Simulator APIs[9]

Haven't heard anything back, yet.

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[10], 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

[6] By Jason Lowe-Power
[7] http://www.gem5.org/documentation/learning_gem5/introduction/
[8] by Sean Wilson and Robert R. Bruce
[9] By Gabriel Black
[10] https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html



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

Fig. 1. 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[11]. SystemC TLM and gem5 were developed around the same time and are based on similar underlying ideas. As a consequence, the hardware model used by TLM is surprisingly close to the model of gem5. In both approaches, the system is organized as a set of components that communicate by exchanging data packets via a well defined protocol. The protocol abstracts over the physical connection wires that would be used in a register transfer level (RTL) simulation and thereby significantly increases simulation speed. In gem5, components use *master* and *slave* ports to communicate to other components, whereas in SystemC TLM, connections are established via *initiator* and *target* sockets. Also, the three protocols *atomic*, *timing* and *functional* provided by gem5 find their equivalent in the *blocking*, *non-blocking* and *debug* protocols of TLM. The major difference in both protocols is the treatment of backpressure, which is implemented by a retry phase in gem5 and with the exclusion rule of TLM.

The similarity of the two approaches enabled us to create a light-weight compatibility layer. In our approach, co-simulation is achieved by hosting the gem5 simulation on top of a SystemC simulation. For this, we replaced the gem5 discrete event kernel with a SystemC process that is managed by the SystemC kernel. A set of transactors further enables communication between the two simulation domains by translating between the two protocols as is shown in Figure 1. This work was published in [7] where we documented our approach and showed that the transaction between gem5 and TLM only introduces a low overhead of about 8%. The source code as well as basic usage examples can be found in `util/tlm` of the gem5 repository.

2.4.2 SystemC in gem5[12]. Haven't heard anything back, yet.

2.5 Cache Replacement Policies and New Compression Support[13]

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

[11] By Chistian Menard, Jeronimo Castrillon, and Matthias Jung
[12] By Gabriel Black
[13] By Daniel Carvalho

carry research on the effects of different replacement algorithms in various hardware units. Currently, gem5 supports 13 different replacement policies including several standard policies such as LRU, FIFO, and Pseudo-LRU, and various RRIPs [5]. This list is easily expandable to cover schemes with greater complexity as well.

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

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

2.6 Ruby Cache Model Improvements

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

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

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

2.6.1 General Improvements[14]. 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 has been added to gem5. For simplicity, this protocol was built on top of a prior two level protocol by adding an L0 cache at the CPU cores. At level 0, the protocol has separate caches for instruction and data. The first and the second levels do not distinguish between instruction and data. Levels 0 and 1 are private to the CPU core, which the second level is shared across (possibly a subset) cores.

2.6.2 GPU Coherence Protocols[15]. Haven't heard anything, yet.

[14] by Nilay Vaish
[15] by Blake Hectman

2.6.3 ARM Support in Ruby Coherence Protocols[16]. Haven't heard anything, yet.

2.7 RISC-V ISA Support

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

2.7.1 General RISC-V ISA Implementation[17]. **TODO: CITE (https://carrv.github.io/2017/papers/roelke-risc5-carrv2017.pdf) and (https://carrv.github.io/2018/papers/CARRV_2018_paper_3.pdf).**

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

The implementation was done by following the divisions of the instruction set into its base ISA and extensions, beginning with the 32-bit integer base set, RV32I. It was modeled off of the existing gem5 code for MIPS and Alpha, which are also RISC instruction sets that share many of the same operations as RISC-V. Including support for 64-bit addresses and data (RV64) and for the multiply (M) extension mainly involved adding the new instructions and changing some parameters to expand register and data path widths. The next two extensions, atomic (A) and floating point (F and D for single- and double-precision, respectively), were more complicated. The A extension includes both load-reserved/store-conditional (LR/SC) sequence of instructions for performing complex atomic operations on memory and a set of read-modify-write instructions for performing simple ones. The former two instructions had analogues in MIPS, but, at the time of implementation, gem5 did not have support for single instructions that both read and wrote memory atomically. These instructions were implemented as a pair of micro-ops that acted like an LR/SC pair with one of the pair additionally performing the specified operation. Floating-point instructions required many special cases to ensure correct error handling and reporting, and we were not able to implement one of the five possible rounding modes (round away from zero) RISC-V specifies for inexact calculations due to the fact that C++ does not support it. Finally, support for the non-standard compressed (C) extension, which adds 16-bit versions of high-usage instructions, was added when it was discovered that this extension was included by default in many RISC-V software toolchains. Its implementation required the creation of a state machine in the instruction decoder to keep track of whether the current instruction is compressed or not, to increment the PC by the correct amount based on the size of the instruction, and to handle cases where a full-length instruction crosses a 32-bit word boundary.

[16] by Tiago Mück
[17] By Alec Rokle

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

2.7.2 RISC-V Full System Support[18]. To simulate complete operating systems as well, full system simulation was added for RISC-V. More specifically, Sv39 paging with a 39-bit virtual address space, a three-level translation and a translation lookaside buffer has been added. Additionally, gem5's version of the [] (RISC-V test-suite (https://github.com/riscv/riscv-tests)) 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. 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.

2.8 Predictor Improvements

I haven't heard anything back, yet.

2.9 GPU Compute Model[19]

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

2.9.1 Autonomous Data-Race-Free GPU Tester[20]. The Ruby coherence protocol tester is designed for CPU-like memory systems that implement relatively strong memory consistency models (e.g., TSO) and hardware-based coherence protocols (e.g., MESI). In such systems, once a processor sends a memory request to memory, the request appears globally to the rest of the system. Without knowing implementation details of target memory systems, the tester can rely on the issuing order of reads and writes to determine the current state of shared memory. However, existing GPU memory systems are often based on weaker consistency models (e.g., sequential consistency for data-race-free) and implement software-directed cache coherence protocols (e.g., VIPER requiring explicit cache flushes and invalidations from software to maintain cache coherence). The order in which reads and writes appear globally can be different from the order they are issued from GPU cores. Therefore, the previous CPU-centric Ruby tester is not applicable to testing GPU memory systems.

The gem5 simulator currently supports an autonomous random data-race-free testing framework to validate GPU memory systems. The tester works by randomly generating and injecting sequences of data-race-free reads and writes that are well synchronized by proper atomic operations and memory fences to a target memory system. By maintaining the data-race freedom of all generated sequences, the tester is able to validate responses from the system under test. The tester is also able to periodically check for forward progress of the system and report possible deadlock and livelock issues. Once encountering a failure, the tester generates an event log that captures only related memory transactions related to the failure, which significantly eases the debugging process. Tuan Ta et al. showed how the tester effectively detected bugs in the implementation of VIPER protocol in gem5 [10].

2.10 Syscall Emulation Improvements[21]

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

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

2.11 ARM Improvements

Heard back, waiting for text.

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

2.11.1 ARMv8 Support[22]. 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) 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.

2.11.2 Support for the Arm Scalable Vector Extension (SVE)[23]. 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 perdicate 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.

2.11.3 Trusted Firmware Support[24]. 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 Call5 (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

[22] by Giacomo Gabrielli, Javier Setoain, and Giacomo Travaglini
[23] by Giacomo Gabrielli, Javier Setoain, and Giacomo Travaglini
[24] by Adrian Herrera

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`.

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.

2.12 Vector Instructions Extensions[25]

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[26]. 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.

[25] by Javier Setoain
[26] by Andreas Sandberg

2.13.2 Python 3[27]. Even before m5 became gem5, the architecture of the built on a core written in C++ and configuration scripts using that core as a Python library. While the fundamental design has not changed, there have been many changes to the underlying implementation over the past years. The original implementation frequently suffered from bugs in the code generated by SWIG and usability was hampered by poor adherence to modern standards in SWIG's C++ parser. The move to PyBind11 [] greatly improved the reliability of the bindings by removing the need for a separate C++ parser, and made it easier to expose new functionality to Python in a reliable and type-safe manner.

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

2.13.3 Asynchronous Modeling in gem5[28]. 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 Flexible DRAM Controller[29]

Heard back, waiting for text.

2.14.1 [30]. Across applications, DRAM is a significant contributor to the overall system power. For example, the DRAM access energy per bit is up to three orders of magnitude higher compared to an on-chip memory access. Therefore, an accurate and fast power estimation is crucial for an efficient design space exploration. DRAMPower (cite: DRAMPower: Open-source DRAM Power & Energy Estimation Tool Karthik Chandrasekar, Christian Weis,

[27] by Andreas Sandberg and Giacomo Travaglini
[28] by Giacomo Travaglini
[29] by Wendy Elsasser, Matthais Jung, and others
[30] by Matthias Jung

Yonghui Li, Sven Goossens, Matthias Jung, Omar Naji, Benny Akesson, Norbert Wehn, and Kees Goossens URL: http://www.drampower.info) is an open source tool for fast and accurate power and energy estimation for several DRAM memories based on JEDEC standards. It supports unique features like power-down, bank-wise power estimation, per bank refresh, partial array self-refresh, and many more.

In contrast to Micron's DRAM Power estimation spread sheet (cite Micron. DDR3 SDRAM System Power Calculator), which estimates the power from device manufacturer's data sheet and workload specifications (e.g. Rowbuffer-Hit-Rate or Read-Write-Ratio), DRAMPower uses the actual timings from the memory transactions, which leads to a much higher accuracy in power estimation. Furthermore, the DRAMPower tool performs DRAM command trace analysis based on memory state transitions and hence, avoids cycle-by-cycle evaluation, thus speeding up simulations.

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

2.14.2 Quality of Service Extensions[31].

Heard back, waiting for text. It may be a bit late.

2.15 Virtualized Fast Forwarding[32]

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

Support for hardware virtualization in gem5 enabled research into novel ways of accelerating simulation. The original intention was to use KVM to generate checkpoints and later simulate those checkpoints in parallel. However, we quickly realized that the checkpointing step could be eliminated by cloning the simulator state at runtime. This led to the introduction of the fork call in gem5's Python API. Under the hood, this call drains the simulator to make sure everything is in a consistent state, it then uses the UNIX fork call to create a copy of the simulator. A typical

[31] by Matteo Andreozzi
[32] by Andreas Sandberg

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[33]

Haven't head back, yet.

2.17 Memory Traces and Traffic Generator[34]

Haven't heard back, yet.

2.18 Classic Caches Improvements[35]

The classic memory system implements a snooping MOESI-like coherence protocol that allows for flexible, configurable cache hierarchies that do not need a change of the coherence protocol for different topologies. The coherence protocol is primarily implemented in the Cache with some support also implemented in CoherentXBar and crucial optimizations in the SnoopFilter.

Over the years, the components of the classic memory systems have received a lot of contributions that enhanced the accuracy and allowed for more flexibility in gem5 to model the on-chip memory system.

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 model and can only be placed 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 performs only fetches and writebacks to memory below. 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 be used to model system-level caches. System-level Caches are often larger in size and can be used by CPUs and other devices in the system.

2.18.2 Write Streaming Optimizations. Write streaming is a common access pattern that is typically encountered when software initializes or copies large memory buffers (e.g., memset, memcpy). Write streaming can create a lot of pressure to the memory system both for off-chip memory bandwidth and on-chip cache capacity. These streaming writes will result in invalidation to other caches and fetches from off-chip memory to fill-in cache lines that are immediately overwritten with new data. In addition, these fills can thrash the L1 or L2 cache.

Common optimizations [] ([1] Conway, Pat, Nathan Kalyanasundharam, Gregg Donley, Kevin Lepak, and Bill Hughes. Cache hierarchy and memory subsystem of the AMD Opteron processor. IEEE micro 30, no. 2 (2010): 16-29) coalesce writes to form full cache line writes and avoid unnecessary data fetches. As a result, we can achieve significant reduction in on-chip memory traffic and off-chip memory bandwidth. In addition, when the buffer we write to is larger than the size of the L1 cache we need to avoid thrashing the L1.

[33] by Cutis Dunham
[34] by Andreas Hanson
[35] by Nikos Nikoleris

We have implemented a simple mechanism to detect write streaming memory access patterns. The mechanism can attach to any cache in the system but has been primarily designed for use in L1 data cache. It can detect a single write-streaming access pattern with configurable thresholds for switching to write coalescing and switching to bypassing allocation in the current cache level and avoid thrashing.

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 all copies of the block above the point-of-coherence, invalidate them, and if any of them is dirty 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[36]

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

[36] by Mohammad Alian

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[37]

Haven't heard back, yet.

2.21 Runtime Power Modeling and DVFS Support[38]

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 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.

[37] by Andrew Bardsley
[38] by Stephan Diestelhorst

2.22 Elastic Traces[39]

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

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

3 OTHER WORK BUILDING OFF OF GEM5

Note: Come up with a better name.

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

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

gem5-gpu Power et al.

gem5-aladdin Shao et al.

Many others.

4 ACKNOWLEDGEMENTS

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

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.

NSF CCRI. Brookhaven.

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

[39] by Radhika Jagtap

REFERENCES

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

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

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718

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

[5] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, and Joel Emer. 2010. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (Saint-Malo, France) *(ISCA '10)*. Association for Computing Machinery, New York, NY, USA, 60–71. https://doi.org/10.1145/1815961.1815971

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

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

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

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

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

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