Towards Accurate RISC-V Full System Simulation via Component-Level Calibration

Pathak, Karan; Klein, Joshua; Ansaloni, Giovanni; Hamdioui, Said; Gaydadjiev, Georgi; Zapater, Marina; Atienza, David

DOI
10.1145/3737876

Publication date
2025

Document Version
Final published version

Published in
ACM Transactions on Embedded Computing Systems

Citation (APA)
Pathak, K., Klein, J., Ansaloni, G., Hamdioui, S., Gaydadjiev, G., Zapater, M., & Atienza, D. (2025). Towards Accurate RISC-V Full System Simulation via Component-Level Calibration. *ACM Transactions on Embedded Computing Systems*, *24*(4), Article 57. https://doi.org/10.1145/3737876

Towards Accurate RISC-V Full System Simulation via Component-Level Calibration

KARAN PATHAK, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Netherlands, and Haute ecole d'ingenierie et de gestion du canton de Vaud Institut Reconfigurable & Embedded Digital Systems, Yverdon-les-Bains, Switzerland
JOSHUA KLEIN, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland
GIOVANNI ANSALONI, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland
SAID HAMDIOUI, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Netherlands
GEORGI GAYDADJIEV, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Netherlands and Computer Science and Engineering, Chalmers University of Technology, Goteborg, Sweden
MARINA ZAPATER, Reconfigurable and Embedded Digital Systems, Haute ecole d'ingenierie et de gestion du canton de Vaud, Yverdon-les-Bains, Switzerland and Electrical Engineering, Ecole polytechnique federale de Lausanne, Lausanne, Switzerland
DAVID ATIENZA, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland

This work is partially supported by Intel as part of the Intel Center for Heterogeneous Integrated Platforms (HIP); in part by the the Swiss NSF Edge-Companions project (GA No. 10002812); in part by the EC H2020 FVLLMONTI Project under Grant 101016776. This research was partially conducted by ACCESS – AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.
Authors' Contact Information: Karan Pathak, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, VD, Switzerland, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Zuid-Holland, Netherlands, and Haute ecole d'ingenierie et de gestion du canton de Vaud Institut Reconfigurable & Embedded Digital Systems, Yverdon-les-Bains, VD, Switzerland; e-mail: karan.pathak@epfl.ch; Joshua Klein, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, VD, Switzerland; e-mail: Joshua.Klein@imec.be; Giovanni Ansaloni, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, VD, Switzerland; e-mail: giovanni.ansaloni@epfl.ch; Said Hamdioui, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Zuid-Holland, Netherlands; e-mail: S.Hamdioui@tudelft.nl; Georgi Gaydadjiev, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Zuid-Holland, Netherlands and Computer Science and Engineering, Chalmers University of Technology, Goteborg, Sweden; e-mail: g.n.gaydadjiev@tudelft.nl; Marina Zapater, Reconfigurable and Embedded Digital Systems, Haute ecole d'ingenierie et de gestion du canton de Vaud, Yverdon-les-Bains, VD, Switzerland and Electrical Engineering, Ecole polytechnique federale de Lausanne, Lausanne, VD, Switzerland; e-mail: marina.zapater@heig-vd.ch; David Atienza, Electrical Engineering, Ecole Polytechnique Federale de Lausanne, Lausanne, VD, Switzerland; e-mail: david.atienza@epfl.ch.

ACM 1539-9087/2025/07-ART57
https://doi.org/10.1145/3737876

Full-System (FS) simulation is essential for performance evaluation of complete systems that execute complex applications on a complete software stack consisting of an operating system and user applications. Nevertheless, they require careful fine-tuning against real hardware to obtain reliable performance statistics, which can become tedious, error-prone, and time-consuming with typical trial-and-error approaches. We propose a novel, streamlined, component-level calibration methodology to address these shortcomings to validate FS simulation models. Our methodology greatly accelerates the validation process without sacrificing accuracy. It is Instruction Set Architecture (ISA)-agnostic, and can tackle hardware specifications at different levels of detail. We demonstrate its effectiveness by validating FS models against both open-hardware and IP-protected (closed hardware) RISC-V silicon, achieving a mean error of 19%–23% for the SPEC CPU2017 suite in the two cases. We introduce the first open-source RISC-V-based FS-validated simulation models with a complete and replicable methodology.

CCS Concepts: • **Hardware** → *Emerging tools and methodologies*; **Emerging simulation**; • **Computing methodologies** → *Simulation tools*; • **Computer systems organization** → Real-time system architecture;

Additional Key Words and Phrases: Architectural simulator, performance validation, full-system (FS) simulation

ACM Reference Format:
Karan Pathak, Joshua Klein, Giovanni Ansaloni, Said Hamdioui, Georgi Gaydadjiev, Marina Zapater, and David Atienza. 2025. Towards Accurate RISC-V Full System Simulation via Component-Level Calibration. *ACM Trans. Embedd. Comput. Syst.* 24, 4, Article 57 (July 2025), 19 pages. https://doi.org/10.1145/3737876

1 Introduction

Computer architects are often tasked with balancing the performance, power, and area requirements of the chip while maintaining stringent time-to-market constraints. Because hardware prototyping is expensive and time-consuming, simulators are vital for architectural exploration and provide a test bed for new solutions. A **Full System** (**FS**) simulator provides a means to gauge the impact of architectural innovations quickly by running user space applications (that require libraries and a kernel) on the simulated hardware. It facilitates early performance and power [21] estimation of the compute system by exposing the architectural parameters of chip design at the level of systems, thereby avoiding ill taped-out architectures. Contrary to RTL/HDL simulations that have poor simulation throughput on the order of a few **Kilo Instruction per Second** (**KIPS**) [19], the event-driven FS simulator provides high simulation throughput (typically hundreds of KIPS) [30] and much greater reconfigurability, at the cost of the accuracy with which the performance of actual hardware is captured. FS-simulators require validation against the target hardware to regain statistical accuracy with respect to the hardware. Until now, a validated open-source FS-simulator with CPU model calibrated and benchmarked with SPEC suite does not exist for the RISC-V ISA. We address this gap by proposing a validation methodology, fully integrated into the open-source **gem5 extensions for RISC-V** (**gXR5**) [12] and gem5v24 (built on top of gem5 "develop" branch). Our choice of supporting the (also open-source) RISC-V ISA paves the way for hardware-validated system explorations completely unimpeded by proprietary licenses.

gXR5 is a Linux-capable FS-simulator built on top of the open-source gem-5 architectural simulator. gem5 [4] is an event-driven architectural simulator built by merging the simulation frameworks m5 and GEMS. gem5 itself provides ISA-independent and tunable (micro) architectural models that allow the simulation of a variety of ISAs (such as ARM and x86). It is highly reconfigurable by the inclusion of Python configuration scripts that tune specific hardware

attributes (e.g., latencies, bus widths, buffer sizes), while maintaining host performance through the use of compiled C++ code for all tuned modules.

In the past, to accelerate the process, simulator models have been validated using synthetic micro-benchmarks [2, 9, 17]. However, the design of these micro-benchmarks are ad-hoc and specific to the architecture of the target hardware. Huppert [17] et al. have designed micro-benchmarks to calibrate the memory hierarchy in gem5 against the MediaTek Helio X20 SoC. Despite this, the CPU calibration has been left as being too complicated, given the huge set of parameters to be tuned. In this work, we tackle this challenge by introducing a **novel "component-level" calibration methodology that hammers CPU** to expose micro-architectural details required to calibrate the simulated models. This is done by selecting a suite of micro-benchmarks that are representative of the actual workloads, thereby reducing the time to validate architectural simulator while not giving up on the accuracy of the validation results. The validation accuracy is at par with other published work. Also, the proposed methodology can facilitate varying degrees of (target) hardware specifications, thereby providing a generalized methodology that is missing in the state-of-the-art works. We relaxed the need for extensive hardware characterisation using **Hardware Performance Counters** (**HPCs**) as only Instruction and Cycle count are being used. This is essential as commercial hardware may not have extensive HPCs built into them. With this, we streamline the validation methodology and micro-benchmarks' design. We explore the soundness of our approach against two simulation targets: an IP-protected RISC-V board and an open-source system-on-chip. Our contributions are as follows:

— We propose a novel "component-level" calibration methodology for fine-tuning and validating full system simulators and demonstrate its advantages using the open-source gXR5 and two real RISC-V implementations.
— We validate our simulator using selected SPEC CPU2017 benchmarks in a real HiFive Unleashed SoC [27] and a Rocket Chip [13] emulated on FPGA, achieving an average error in performance estimation within 19%–23% for the two hardware targets above.
— We provide the first validated Linux-capable Full System simulator (gXR5) for RISC-V 64-bit architecture in gem5. The simulator also includes the first Gshare branch predictor model compatible with current multi-threaded gem5 CPU models.
— We provide two validated boards, namely the *RISCVUnleashedBoard* and *RISCVRocketBoard* boards, as part of standard pre-built library in gem5v24, tested with standard gem5-resources (comprising Linux kernel and Ubuntu v22.04 disk image).

We intend to release and open source our validated simulator and all supporting materials, including bootloader, kernel, filesystem, and technical manual, for quick and easy adoption by the RISC-V community.

The article is structured as follows: Section 2 compares the past validation efforts with our work, highlighting the shortcomings in the existing validation approaches. Section 3 describes our proposed methodology. In Section 4, the simulator models and the targets are described in detail. Finally, in Section 5, the "component-level" calibration methodology is employed to validate gXR5 against the two target hardware.

2 Related Works

One of the earliest works to accelerate performance validation by Desikan et al. [9] used synthetic micro-benchmarks to calibrate the SimAlpha simulator against the Alpha 21264 processor of the DS-10L Workstation. [2] performed similar work by designing "SiNUCA" micro-benchmarks for validating the SiNUCA simulator. These works achieve a similar IPC error of 18%–19% for SPEC

suites. However, these micro-benchmarks are specific for calibration against a particular target micro-architecture and do not provide a generic methodology. Moreover, micro-benchmarks are in general not representative of realistic workloads, resulting in validated simulators having large performance error for large applications such as the ones in the SPEC suite, even if they exhibit very low errors for micro-benchmarks [1, 2, 9]. The underlying issue is that parameters tuned for calibration of the simulated model are over-fitted for the tested micro-benchmarks, thus failing to propagate low error in performance statistics for macro-benchmarks. Hence, there is no way to further reduce the error using these benchmarks as they become un-representative of the performance beyond a point (i.e., they are overfitted for certain microbenchmarks). The stress-ng benchmarks, though, achieve similar accuracy (in our work), but always give a fair idea to the architects of the disparity in performance if and when macro-benchmarks are executed. In general, the error in IPC for stress-ng benchmarks is around 14%–16% while resulting in error of 20%–22% in execution time for SPEC suite. In terms of computational and time complexity, stress-ng stressors are more representative of the SPEC suite than the above-mentioned microbenchmarks. This issue has been bypassed in [10, 15, 24] by using macro-benchmarks to directly calibrate models in the simulator. However, this approach is impractical due to the very long real run times of macro-benchmarks in simulation (usually on the order of days or weeks per macro-benchmark run). A further complexity is the dimensional explosion of the design space for fine-tuning models in the simulator when marco-benchmarks are executed. At best, this approach can be described as "validation by inspection" or "trial-and-error" approach. Other works [5, 6] use SPLASH-2, ALP-Bench, STREAM benchmarks, Rodinia, lmbench, and so on. to validate the gem5 simulator for the heterogeneous architecture ARM A8 (dual core), A9 cores and ARM big.LITTLE (A15/A7), respectively. However, these works do not propose any validation methodology. An attempt has been made to use Pearson's correlation between micro-architectural events and error in IPC [1] to streamline the calibration methodology. But the validated simulator has not been tested on macro-benchmarks (e.g., SPEC) representative of the user's workload. Thus, there exists no validation methodology that has been employed to tackle varying architectural specifications. [16] realize the importance of having RISC-V full system simulation and functionally verify against QEMU [3] based full-system emulation by performing trace analysis. However, the detailed implementation of RISC-V in gem5 has not been validated for timing. [7] validates the gem5 models against RTL simulations (simulating an open-source RISC-V core) but fails to provide a validation methodology.

Table 1 summarizes the validation efforts with the training set (micro-benchmarks) and test set (macro-benchmarks) used. Huppert et al. [17] provide a methodology for memory hierarchy calibration that achieves a mean error of 20% in IPC for the SPEC CPU2006 benchmark suite. This simplistic methodology (of "data pinning") has not been extended to other components (such as the CPU, Branch Predictors, TLBs, Page Walkers, and others), and lists this as a limitation to their proposed work.

Our proposed methodology uses stress-ng benchmarks to calibrate the CPU model to overcome all the shortcomings listed above [26]. It adopts stressors to calibrate specific architectural components in a system, as opposed to ad-hoc synthetic micro-benchmarks designed for particular target hardware [2, 9, 17]. It is also much faster than adopting macrobenchmarks such as SPEC for calibration, with up to 3000x in simulation time. Moreover, it only employs **Instruction per Cycle** (**IPC**) as a metric to match the simulator and hardware performance, reducing the need for extensive hardware characterization. Lastly, the methodology matches the accuracy of other state-of-the-art works and is representative of hardware performance in terms of the execution time of SPEC2017 applications.

Table 1. State-of-the-art System Validation Strategies

Validated Simulator	Training Set		Test Set		Proposed Methodology
	Suite	Mean Error	Suite	Mean Error	Generic/Specific
gem5 Vs. MediaTek Helio X20 [17]	Synthetic Micro-benchmarks	–	SPEC 2006	20% (IPC)	Memory Specific
SimAlpha Vs. DS-10L workstation [9]	Synthetic Micro-benchmarks	2% (IPC)	SPEC 2000	18% (IPC)	CPU specific
SiNUCA Vs. Sandy bridge processor and Intel Core 2 Duo [2]	Synthetic Micro-benchmarks	6%–10% (IPC)	SPEC 2006	19% (IPC)	CPU specific
gem5 Vs. Intel Core-i7 Haswell [1]	Synthetic Micro-benchmarks	6% (IPC)	None	None	None
gem5 Vs. ARM Versatile Express TC2 dev. board [15]	SPEC, PARSEC	13%, 16%–17% (Run-Time)	None		None
gem5 Vs. ARM Cortex-A8, A9 [10]	PARSEC	8% (Run-Time)	None		None
gem5 Vs. Arm R8 CPU [28]	Embench workload	13 % (CPI)	None		None
GEMS Vs. ARM Cortex A9 [6]	SPLASH-2, ALPBench, STREAM	17.94% maximum (Run-Time)	None		None
gem5 Vs. Samsung Exynos 5 Octa (ARM A15/7) [5]	Rodinia and lmbench	20% Run-TIme	None		None
Our Work					
gXR5 Vs. Sifive Unleashed [22] , Rocket Chip	**Stress-ng**	**14%–16 % (IPC)**	**SPEC 2017**	**19%–23 % (Run-Time)**	Generic - tested with CPU stressors

3 Methodology

3.1 Overview

The calibration of simulated models against actual silicon or emulated systems require identifying the parameters of interest. The detailed CPU models used for performance modeling of systems provide elaborate parameters such as delays between different pipeline stage, queue (e.g., load and store) sizes, number of micro-operations in flight, and so on. These parameters can take integer values, thereby causing dimensional explosion in the design space for calibration. More recent work [23] has tackled the Design space using statistical tools of sensitivity analysis followed by (single-objective) optimization for calibration. However, such statistical techniques can lead to over-fitting. The calibrated models in [23] have not been tested on untrained benchmark-set.

In our component-level calibration methodology (depicted in Figure 2), the stress-ng [26] benchmarks are used as a "training set" to fine-tune the attributes of gXR5 by targeting IPC to reduce performance disparity. The stress-ng benchmarks were originally designed to perform accelerated

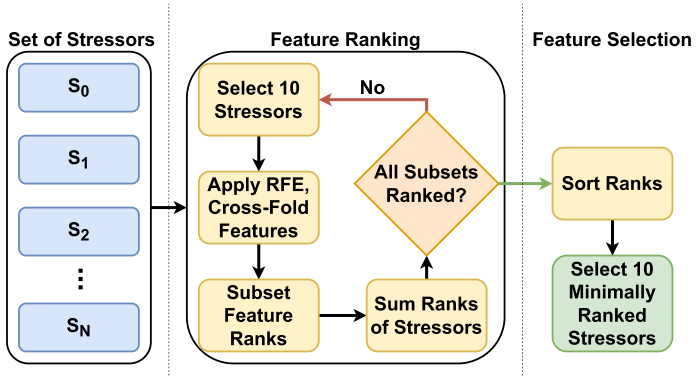


Fig. 1. Stressor selection via recursive feature elimination.

stress-tests of a particular system component or routine in order to provoke thermal overruns. Contrary to existing synthetic micro-benchmarks [2, 9] whose application for RISC-V ISA is not proven, the stress-ng benchmarks have been tested for RISC-V ISAs [26] in our work.

The stress-ng benchmark suite is divided into classes, each of which hammers a particular component or routine and often contains numerous stressors, with CPU-class stressors numbering over 100. To reduce the exploration design space and overhead of experiment iterations, as well as to better target our validation, we initially narrow the number of stressors used for validation iterations by selecting 10 CPU-classed stressors. We select the most representative subset of Stressors (with respect to the whole set) through the statistical technique of **recursive feature elimination (RFE)** via decision tree regressor. This is a common technique for reducing the number of features that represent a whole set, with applications in a wide range of data-intensive fields from remote sensing and cybersecurity, to genomics and bioinformatics [8, 14, 18, 29].

The process of selecting 10 statistically representative stressors is shown in Figure 1. In summary, this was done by first running all stressors in the uncalibrated gXR5 to obtain run-time statistics, including the *Number of Instructions*, *L1/L2 hits/misses*, *active* and *idle cycles* and *op mix*. We then apply RFE to all permutations of subsets of 10 stressors to obtain a rank for each individual stressor's ability to represent the whole set, where a lower rank is better. We then sum all the ranks achieved for each stressor and take the 10 most minimally-ranked stressors to be our representative set for quick validation iterations. We prove experimentally that the chosen 10 stressors are sufficient to achieve state-of-the art accuracy, although the proposed methodology is independent of the number of chosen stressors. The minimum subset, which was representative of the entire CPU class stressors, was 10. Once the stressors are selected, the stressors are profiled and classified to perform targeted parameter tuning. This is represented in Figure 2, wherein the selected stressors (post Principal Component Analysis) are profiled, and subsequently classified. Once classified, the component and parameters to be tuned are identified based on the profiling results and the hardware specifications. The tuned parameters are incorporated in the simulated models and the stressors are executed. The iteration of tuning simulations, obtaining gem5 statistics, and selecting parameters/components is captured in the Algorithm 1. With each iteration, the IPC of the stressors executed on top of the tuned simulator converges to the IPC of the actual hardware. The δ refers to the marginal increase in IPC after each tuning simulation. At no point of experimentation, the IPC of the modeled system (for each benchmark) is allowed to increase and exceed that of the actual hardware. This ensures that the performance of the selected microbenchmark suite converges (that is, the IPC increases after every tuning simulation)

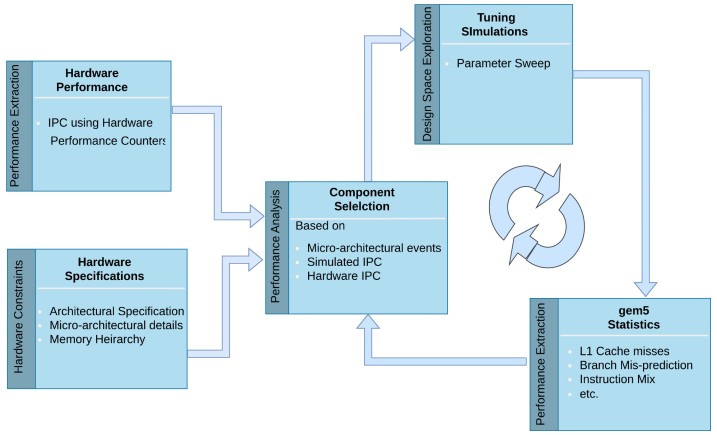


Fig. 2. Methodology: Component level calibration.

ALGORITHM 1: Component-level validation methodology.

Require: $IPC_{gXR5} = IPC_{hardware}$
Ensure: $\delta \leq IPC_{hardware} - IPC_{gXR5}$
  while $IPC_{gXR5} < IPC_{hardware}$ **do**
    $IPC_{gXR5} \leftarrow IPC_{gXR5} + \delta$
  end while

In the following section, we classify the profiling results of the selected CPU class stressors. This is in line with existing validation efforts [9] that divide micro-benchmarks into broadly three categories: *control* (or front-end intensive), *arithmetic*, and *memory* intensive.

3.2 Profiling and Classification

The stressors are run on the Linux distribution (Ubuntu) running on the simulated system (gXR5). Figures 3 and 4 depict the instruction mix and control intensity in the selected stressors. It is ensured that the number of instruction executed is on the order of billions (the same as macro-benchmarks) by executing the stressors for 5 seconds each. The selected CPU class stressors are placed into three categories according to their instruction mix:

— **Control Intensive:** The stressor has complex control structures (e.g., nested for-if-else) with at least 100 BPKI. These are “Int64LongDouble”, “Matrixprod”, “Trig”, “Prime”, “Sqrt”, and “LongDouble”.
— **Memory Intensive:** The stressor has 20% or higher load/store instructions in the total op-code mix of the program. Up to 30% of instructions are load/store (for stressors “queens” and “rand48”).
— **Arithmetic Intensive:** The stressor has 20% or more Instructions using Integer/Float functional units. All the stressors are arithmetic/compute intensive.

Alternate metrics such as **Branch Misses per Kilo Instruction** (**BPKI**) and Last-level cache misses can be used to measure the intensity of control and memory **Dynamic Random Access Memory (DRAM)** access incidence. However, choosing metrics that make classification dependent on the type of branch predictor or cache associativity is not ideal as the classification of stressors is intended to be static, that is, independent of the micro-architectural changes in the simulated models. This ensures that minimum number of tuning simulations are run to achieve the desired

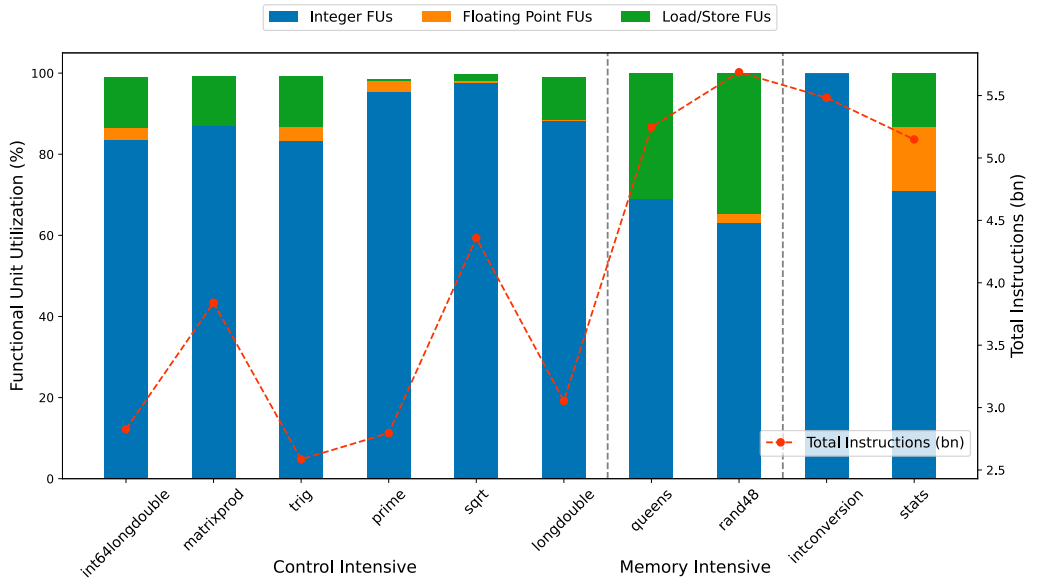


Fig. 3. Profiling selected Stress-ng benchmarks.

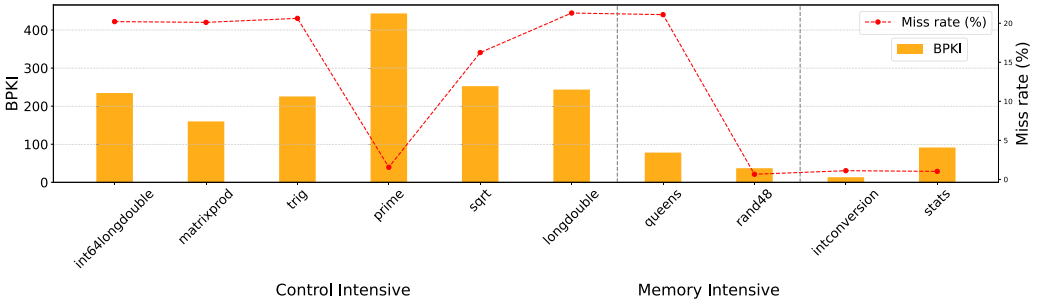


Fig. 4. Branch intensity and mis-prediction rates (in baseline gXR5).

(state-of-the-art) accuracy. The Figures 3 and 4 depict the instruction mix and the branch intensity for the selected stress-ng stressors, that was used to classify the stressors as discussed above.

3.3 Implementation

The CPU model is tuned to reduce the **Mean Absolute Percentage Error** (**MAPE**) in IPC for the stressors. We choose IPC as the metric of merit since it captures even small changes in performance of the simulated system. A greedy approach is followed to match real hardware performance, which consists on searching for the most prominent sources of error first, performing component tuning and re-assessing the source of IPC error via profiling analysis. Several tuning simulation runs are required for various sub-components of CPU model such as Functional Units and the Branch Predictor. The Design Space to be explored for validation is restricted by choosing stress-ng stressors of CPU class (as against macro-benchmarks like SPEC, PARSA [10, 15]). Each stressor is executed in a separate instance of gXR5, **taking a maximum of 3 hours to complete, thus significantly shortening the tuning process.** For comparison, executing the SPEC suite may take from days to over a week of real time on gem5/gXR5. Once the stressors are executed on

un-calibrated gXR5 (initial profiling) for classification, the proposed component level calibration methodology is implemented. This section provides the details into the methodology.

3.3.1 Tuning Simulations. The "Tuning Simulations" box runs the selected CPU class stressors for 5 (simulated) seconds on FS-simulator (usually 3 hours of real time). The component to be tuned is selected based on the hardware specifications, hardware performance and the simulated performance. Once the component (say branch predictor) is identified as the source of discrepancy, the attributes of the component (say, the type and size of branch predictor) are set as part of the "Design Space Exploration" in "Tuning Simulations". The values that attributes can take is decided as part of the "Performance Analysis" done during "Component Selection". Initial simulations provide an estimate of the range of (integer) values the attributes of the component can take.

3.3.2 Component Selection. The component to be calibrated is selected using the "Hardware Specifications", "Hardware Performance" (extracted using hardware performance counters when stressors are executed on hardware) and the "Tuning Simulation" statistics (generated in gem5 simulator). The performance (Instruction per cycle) of each stressor executed on hardware and simulator (gXR5) is compared. The methodology targets execution related sources of discrepancy foremost. These include the number, latency and type of the functional units in the simulated model. Subsequently, the control hazard related performance discrepancy is targeted. The initial design space for tuning simulations includes sweeping across the branch predictor (type and size, branch delay slot execution, etc.). The branch prediction calibration has an effect on both data and instruction cache access patterns, and hence, the memory hierarchy-related performance discrepancy of the simulated system is taken up at the last. The micro-architectural events generated in gXR5 simulation statistics (e.g., L1 cache misses, branch miss-prediction rates) gives details about the potential sources of error for individual stressors. The configuration (i.e., calibrated attributes of the selected component) that gives least absolute mean error in IPC is fixed before selecting the next component to be calibrated.

3.3.3 Hardware Performance. The stressors are run for 5 seconds on real hardware and the performance (IPC) is extracted using the *perf* tool that is supported by Ubuntu stress-ng stressors [26]. The *perf* tool captures the micro-architectural events (stored in hardware as part of performance counters) when user-space applications are run. The hardware performance extraction is a once-only implementation step. The instruction count and the cycles elapsed while executing stressors (on top of the OS) are recorded. The IPC gives a high-level estimate of the hardware performance without the extensive use of performance counters (and the micro-architectural analysis of the hardware). Many commercial systems may not have extensive hardware performance counters to protect against IP attacks. Also, IPC is directly related to the performance of the system, unlike the micro-architectural events. In this way, our proposed methodology can address the varying degree of hardware architectural specifications available for use.

3.3.4 Hardware Specifications. ISA-level architectural specifications as well as micro-architectural details of the hardware are considered as constraints on the design space to be explored during "Tuning simulations". For example, the pipleline depth, number of functional units, cache hierarchy, and so on, of the target hardware are matched in the simulated system. Hence, the selection of components and the calibrated attributes do not violate the hardware specifications.

3.3.5 gem5 Statistics. gXR5/gem5 simulations executing the stressors generate statistics that are used to perform analysis of the simulated system. Apart from Instruction per cycle count, relevant statistics are selected as part of "Performance Extraction" to be fed for the "Performance

Analysis". These include L1, L2/Last-level cache misses, branch misprediction rates, average memory access time, and so on. These micro-architectural events provide the source of discrepancy in the IPC of the simulated system.

The MinorCPU of gem5 is used in our work. It is a (configurable) four-stage pipelined CPU comprising of Fetch1, Fetch2, Decode, and Execute stages. These stages are connected through size-tunable buffers that hold the instructions in case of a stall. The Fetch1 stage fetches the instructions from the L1 instruction cache, while the Fetch2 stage has a branch predictor unit with **Branch History Table** (**BHT**) and **Branch Target Buffer** (**BTB**). The Decode stage converts the instruction into micro-operations before passing them to the Execute stage, which hosts the functional units and the **Load/Store unit** (**LSU**). The functional units are modeled as black boxes with finite ***Operation (Op)*** *Latency*. The MinorCPU model is designed with extensive details so that it can model a vast majority of in-order CPUs. However, the tradeoff is the huge design space that needs to be explored for validation of the model.

Following the methodology outlined in Figure 2, the arithmetic functional units of the MinorCPU model are fine-tuned first by using the stressors (as all stressors are compute/arithmetic intensive). Additional functional units are created to match the hardware specifications. The control related micro-architectural parameters such as type of branch predictor, branch prediction delay, and so on. are taken up in the next set of iterations as the affect of these parameters go beyond the cpu core. A branch mis-prediction may cause thrashing in Instruction and Data caches because of un-intended pre-fetching of instructions and data, respectively. Memory-intensive stressors can make the caches or main memory work incorrectly, subject to the size of the working set relative to the cache capacity. The working set size of all the stressors (except "Matrixprod" and "Stats") is only a few bytes, making these L1 (data) cache intensive. The order of selecting components and their parameters become even more important when the target hardware is an IP-protected black box (as in Section 4). We restrict our component tuning to the models of the CPUs and L1 caches, as these are the most challenging [17], due to their huge architectural design space. The pseudocode describing the calibration process is given in Algorithm 1. The increase in simulated IPC (δ) at each iteration of fine-tuning of the CPU model/caches (or core) is upper bounded to ensure that the simulator does not overestimate the performance of stressors and converges to the actual hardware performance. In practice, only a handful of execution runs in gXR5 (each taking 3 hours of simulation time) are required to calibrate the MinorCPU model with the proposed Algorithm 1. Once the models are fine-tuned, the SPEC2017 suite applications are run on the validated simulator (purple box in Figure 2). We test this methodology by fine-tuning the MinorCPU model and Classic caches of gXR5 against the Sifive HighFive Freedom Unleashed and Rocket Chip emulated on the FPGA. We then create two new boards - *RISCVUnleashedBoard* and *RISCVRocketBoard* on top of gem5 "develop" branch and validate them by replicate the proposed methodology.

4 Experimental Setup

4.1 Simulation Model

The diagram of the adopted gXR5 and gem5v24 full system model, which simulates execution on the single in-order core HiFive Unleashed SoC and Rocket system emulated on the VC707 FPGA, can be seen in Figure 5. The difference between gXR5 ad gem5v24 are the PLIC, CLINT, and MMU models. Rest, both the simulators use existing models of DRAM memory controller, UART, IDE disk controller, classic caches, and so on, in gem5 to build the simulated system. Also the two new boards, the *RISCVUnleashedBoard* and *RISCVRocketBoard*, are built using the existing HiFive platform components including the CLINT, PLIC, and UART models. The MMU and Classic Cache models are taken from gem5 v24. The model takes advantage of the modularity

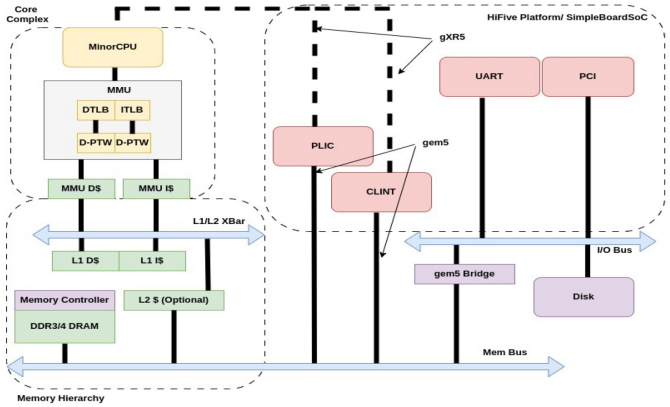


Fig. 5. Simulated full system model in gXR5 (SimpleBoardSoC) and gem5v24 (Hifive).

Table 2. Sifive Highfive Freedom Unleashed U54 Core Micro-architectural Specifications [27]

Instructions	Latency
LW	Two-cycle
LH, LHU, LB, LBU	Three-cycle
CSR reads	Three-cycle
MUL, MULH, MULHU, MULHSU	Five-cycle
DIV, DIVU, REM, REMU	Two-cycle to 65-cycle

of gem5 by integrating standard simulation components, such as the in-order MinorCPU, which simulates the in-order single core of the two aforementioned hardware targets. In the case of gXR5, the **core-local interrupter** (**CLINT**) is modeled as a single simulation object with a separate timer for each CPU core. The rest of the SoC is derived from the SimpleBoard model [25]. The **platform-level interrupt controller** (**PLIC**) model is based on the real FU540-C000 core specifications and is responsible for interrupts from external devices. The software stack of the gXR5 full-system simulator includes an OpenSBI bootloader, the Linux v5.8 kernel, and a buildroot file system, whereas for gem5v24 it uses standard gem5 resources for bootloader, Linux kernel (v5.10), and the disk image (Ubuntu v22.04LTS).

4.2 Target Hardware

4.2.1 Sifive Hifive Freedom Unleashed. The Sifive Unleashed was the first commercially available Linux-capable RISC-V system. Figure 6 shows the SoC, with one small CPU (S51) that can host RTOS, while the other four U54 cores are larger 5-stage pipelined in-order processors. We consider a U54 core as the first target for our validation methodology. It is an IP protected design, and so the details of its micro-architectural implementation are not readily available. The **U54 core** is an in-order 5-stage pipelined CPU with 32 KB of 16-way L1 Instruction and Data caches. It has a branch predictor unit with a 30-entry BTB that caches the target of taken branches, a 256-entry BHT that stores the direction of conditional branches, and a 6-entry **return-address stack** (**RAS**). The latency of the integer multiplier is 5 clock cycles. The integer division unit has a latency between 2 and 65 clock cycles [27]. The details of the rest of the micro-architecture is closed-source and, therefore, it poses a challenge as it is a source of specification errors. The Table 2 summarizes the micro-architectural specifications of U54 core that are available in the open-source.

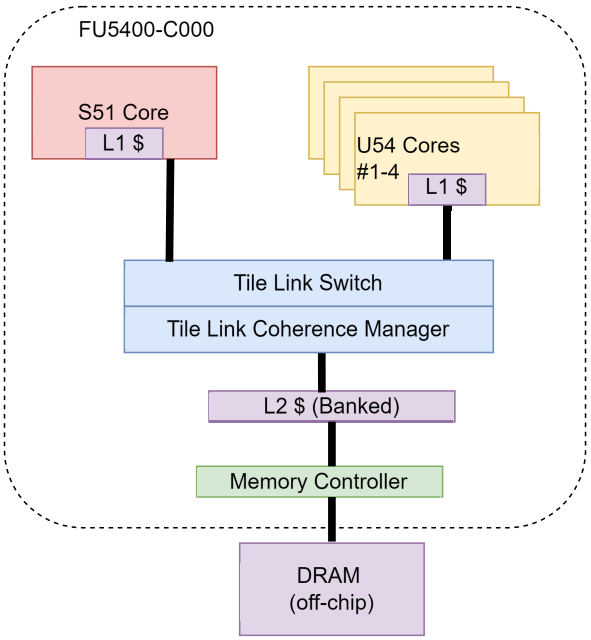


Fig. 6. Sifive highfive freedom unleashed diagram.

4.2.2 Rocket Chip. The **Rocket Core** is a 5-stage in-order scalar processor with L1 Instruction and Data caches. The default configuration of the core includes floating point units (Single Precision and Double Precision Fused Multiply Accumulate and Division). The integer functional units include an 8-cycle iterative integer multiplier (with one cycle each to load operands and place the result on the output bus). It comes with an integer ripple carry adder and an integer division unit. It has a Gshare branch predictor that uses the hash of the branch address XORed with the hash of **global history** (***gh***) to index into **pattern history tables** (**PHT**) containing a 2-bit counter. The hash functions are as follows:

$$Hash\_1(gh) = \sqrt{\frac{3}{2}} * 2^{History\_length} * gh, \tag{1}$$

$$Hash\_2(pc) = pc >> log_2(fetch\ bytes). \tag{2}$$

The L1 caches are pipelined and have a data access latency of 2 cycles in case of a hit. The Data cache has an additional stage for hosting the **Miss Status Holding Register** (**MSHR**) that keeps track of hits under a miss. As an example of open-source hardware design, we consider the Rocket core, one of the most popular open-source RISC-V hardware systems [11]. A high-level view of the Rocket chip emulated on VC707 FPGA is given in Figure 7.

The rocket core comes with Tilelink interface for the system bus (with coherence manager for multi-core systems). We wrapped around the core with Tilelink interface with an AXI4 compliant interface for both memory and IO bus. The AXI4 interconnect fabric is then connected to board DDR3-DRAM that is controlled by **Memory Interface Generator** (**MIG3**) or the memory controller. Similarly, the SD card controller and the UART are interfaced with the AXI4 interconnect. The SD card stores the First-stage bootloader (comprising of OpenSBI v0.8 and U-boot) and the filesystem containing the user-space programs. The Zero-stage bootloader comprises a device tree binary stored in the (Boot) ROM that is implemented inside the programming logic of the FPGA

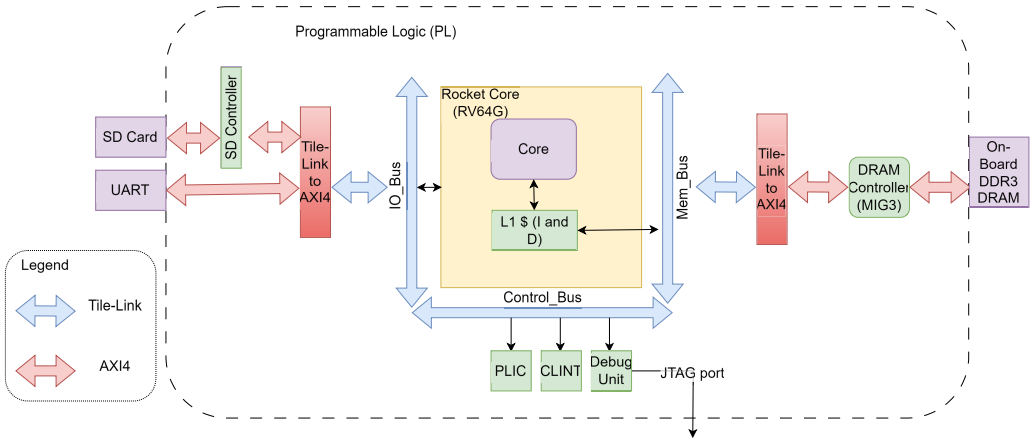


Fig. 7. Rocket chip diagram.

Table 3. Specifications of Simulated Models and of the Corresponding Hardware Platforms

Component	HiFive Unleashed		Rocket	
	Hardware	gXR5	Hardware	gXR5
CPU Core	U54	MinorCPU	Big Core	MinorCPU
CPU	RV64GC		RV64G	RV64GC
CPU Freq	1		0.1 GHz	0.1 GHz
L1 I and D $	32 KB 8-Way		16 KB 4-Way	
L2 $	2 MB 16-Way		None	
MMU	Sv39			
Modes	Machine, Supervisor, User			
RAM	DDR4_64	DDR4_4x16	DDR3_64	DDR3_8x8
RAM Freq	2,400 MHz		800 MHz	2,133 MHz
RAM Size	8 GB		1 GB	
System Bus	TileLink	XBar	TileLink	XBar

(using Block-RAMs). We use **Xilinx System Debugger** (**XSDB**) which gives access to the registers (CSRs, etc.) inside the core and facilitates downloading binaries into the DDR3-DRAM via the boundary scan chain for executing bare-metal programs for testing the hardware. The stress-ng stressors and the SPEC2017 suite are cross-compiled and stored in the file-system put on the SD-card. The UART on the IO bus prints the performance statistics (extracted using *perf* tool) post execution of the benchmarks.

By validating both a black-box and white-box RISC-V CPU core, we demonstrate the versatility of the proposed methodology that adapt to the varying degree of available hardware specifications. Table 3 summarizes the simulator set-up and the corresponding hardware specifications. The attributes of the simulator such as latency of modeled functional units, cache access latency, and so on, are matched with those of the hardware (subject to open-sourced technical specifications). The Fetch1 to Fetch2 stage delay is set to 2 clock cycles, making the MinorCPU model 5 stage pipelined.

Table 4. Selected Attributes of Baseline and Validated gXR5 Models

Component	Attribute	Baseline gXR5	Validated gXR5
Sifive Unleashed			
MinorCPU Model			
IntALU	OpLAT	3 (cycles)	2 (cycles)
IntDIV	OpLAT	33 (cycles)	19 (cycles)
ReadMem	OpLAT	4 (cycles)	3 (cycles)
WriteMem	OpLAT	2 (cycles)	1 (cycle)
Decode	I/P Buffer Size	1	4
Execute Unit	I/P Buffer Size	1	4
Branch Predictor	Type	Tournament	Multiperspective Perceptron
Classic Cache Models			
L1 D	Data Latency	4 (cycles)	1 (cycle)
L1 I and D	Clusivity	Exclusive	Inclusive
Rocket Chip			
MinorCPU Model			
IntDIV	OpLAT	33 (cycles)	16 (cycles)
Branch Predictor	Type	Tournament	Gshare

5 Results and Discussion

5.1 Validation against Sifive Unleashed

We name the simulated systems, configured based on the available specifications alone and prior to fine-tuning, as "baseline" implementations in the following section. We observed that the baseline configuration underestimates performance (IPC) by 25%–40% compared to the actual hardware for almost all stressors due to an excessive number of CPU stalls. We identify the source of bottlenecks by profiling the baseline simulator and proceed with the calibration methodology by calibrating functional units. The latency of the functional units, namely, IntALU, IntDIV, ReadMem and WriteMem functional units were tuned as part of the tuning simulation, resulting in error reduction by 3% (δ_1). Table 4 defines the final *OpLatvalues* assigned to these functional units. The control related attributes of the simulated architecture were selected for the next tuning iteration. A design space exploration was performed for the existing branch predictors available in gem5 (Tournament, TAGE, Bi-mode, etc.). We measured that the MPBP model gives the least MAPE in IPC (giving a δ_2 of 6%). The branch mis-predictions cause thrashing in the L1 instruction and data caches. Hence, the L1 cache and load / store units were tuned after switching to MPBP. The memory-related stalls were removed by decreasing the data access latency of L1 cache and making L2 cache inclusive of L1 cache. This significantly improves performance of stressors having a working set size larger than L1 cache capacity, as the L1 misses end up accessing the main memory in case of L1 being exclusive of L2 cache. The calibration of L1-level caches reduces the error in IPC by 5% (δ_3). Finally, the input buffers for the Execute stage were increased to ensure higher functional unit utilization even in the event of control-related stalls. The MAPE in IPC of gXR5 at various stages of calibration is depicted in Figure 10. The fine-tuning of the MinorCPU model reduces the MAPE in IPC for stress-ng benchmarks to 14.1% post-application of Algorithm 1. Table 4 summarizes the attributes of validated gXR5 against Sifive Unleashed.

Selected SPEC2017 int rate applications (with test inputs) are run on the validated simulator and the target hardware. The execution time of the application gives an estimate of the performance of

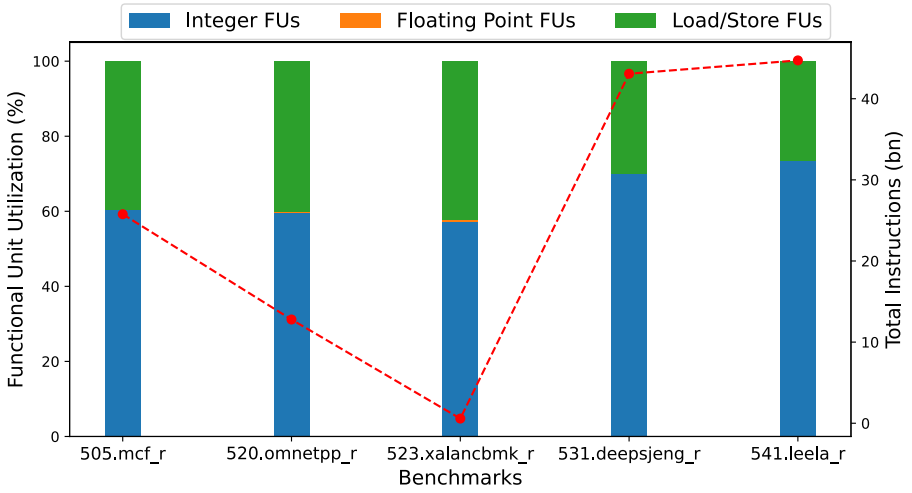


Fig. 8. Functional unit utilization in SPEC 2017 suite.

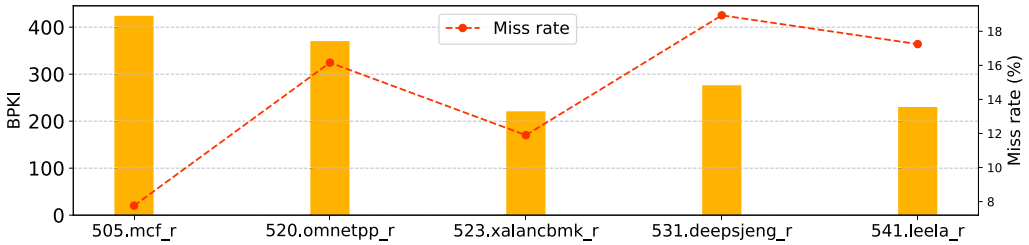


Fig. 9. Branch intensity of SPEC 2017 suite.

the simulated system. Figure 11(b) depicts the final error in the validated simulator (with negative sign indicating that the simulator under-estimates the IPC). The error is brought down from 44% to 23% (for cycles) by calibrating the MinorCPU model in gXR5 using stress-ng benchmarks.

As expected, the two applications that have higher load/store instructions (mcf and omnetpp) have a higher run-time error. The relatively high Load/Store instruction count (compared to stress-ng benchmarks) can be seen in Figure 8, which depicts the profiled SPEC suite applications in gXR5. Separate "Memory" class stressors have up to 75% MAPE in IPC, confirming that the memory hierarchy is the next largest source of errors. The branch intensity of the SPEC suite, depicted in Figure 9 is similar to that of the stress-ng stressors. For the two applications, namely "leela" and "deepsjeng" that have a similar instruction mix as the stress-ng benchmarks, the run-time error is a mere 3.4% and 1.7%, highlighting the benefit deriving from the adoption of our proposed methodology. The selected SPEC suite application were the only ones that were verified to complete correct execution inside the simulator. The other applications suffered from technical and dependency issues owing to lack of SPEC support for RISC-V. Moreover, latest version (1.1.9) of the SPEC CPU suite, only one particular version of Linux is stated to have been tested in a RISC-V environment.[1] The primary reason for being unable to run the entire SPEC suite is a consequence of software portability and the simulated environment. More complex libraries often have numerous dependencies which can be verified only for select Linux versions and configurations. We also

[1] https://www.spec.org/cpu2017/Docs/changes.html#v119

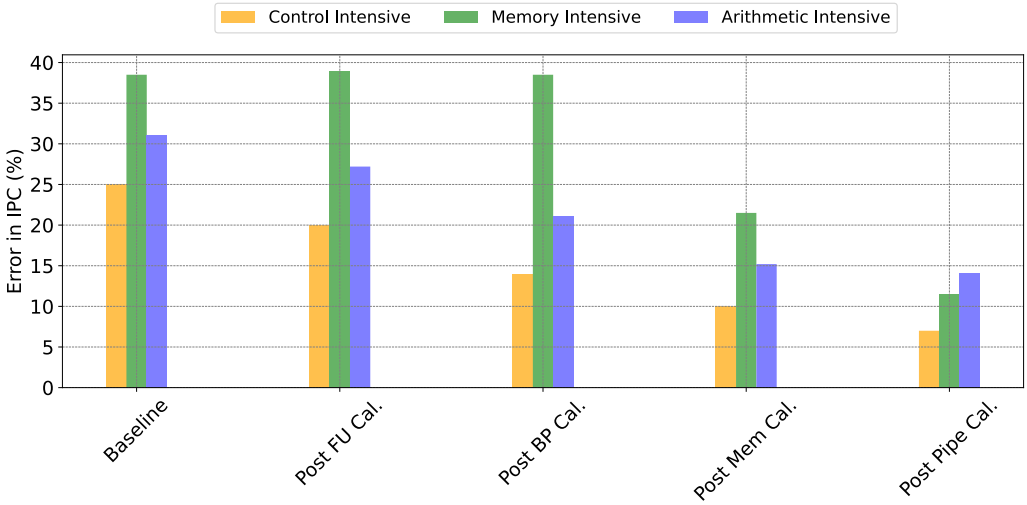
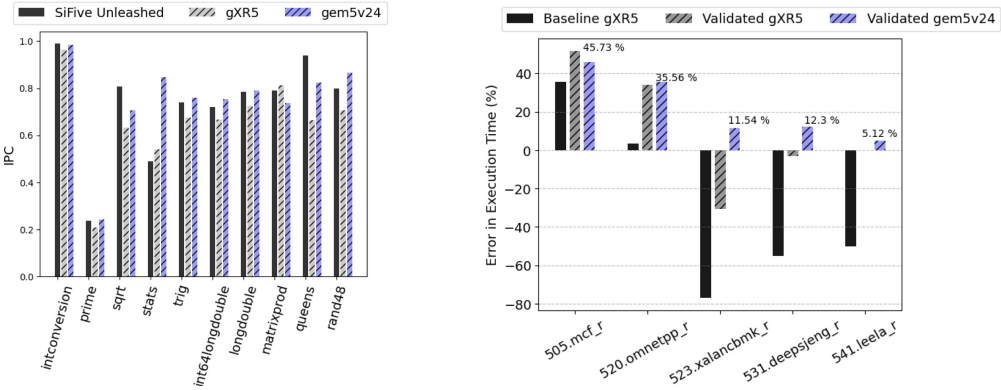


Fig. 10. MAPE in IPC for stressors run on gXR5 and Sifive Unleashed at various calibration stages.



(a) Stress-ng benchmarks executed in simulator and actual silicon

(b) Percentage Error for the validated gem5v24 simulator

Fig. 11. Performance of *RISCVUnleashedBoard* in gXR5 and gem5v24 against Sifive Unleashed Board (silicon).

provide the calibrated models built on top of current gem5. Figure 11(b) depicts the performance of *RISCVUnleashedBoard* (provided as a pre-built gem5 library board). The mean error in IPC for this board is 12.2% for sress-ng stressors (v.0.14.03). We used the same methodology to calibrate the models. The mean error in execution time for the selected SPEC suite is 23% for *RISCVUnleashedBoard*. The mean error in execution time for the selected SPEC suite is 23%, proving that the simulated boards are still accurate.

5.2 Validation against Rocket Chip

The (baseline) gXR5 attributes are matched to the hardware specification as discussed in Section 4. We implemented a Ghsare branch predictor model in gXR5 that uses a program counter hash and a global history hash to access the 2-bit prediction counters as proposed in the original branch predictor [20]. This model is compatible with the gem5 CPU (multi-threaded) models and can

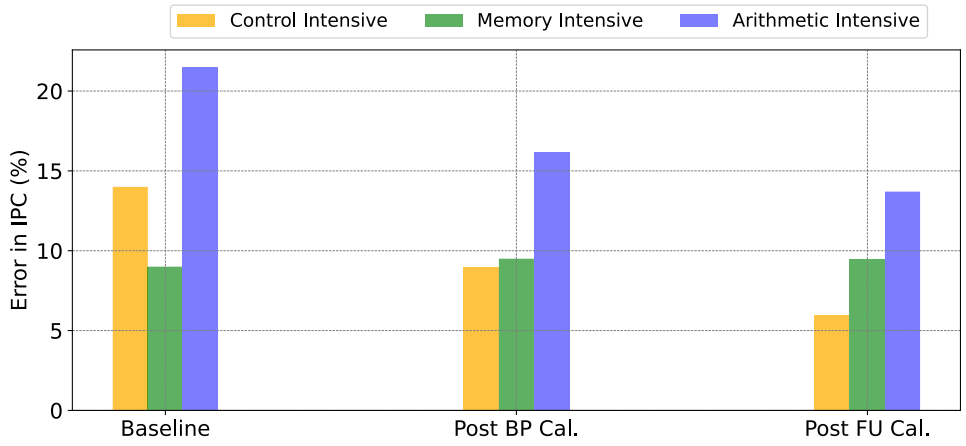
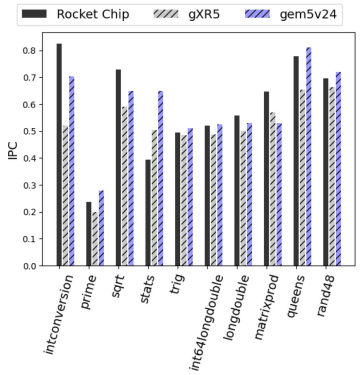
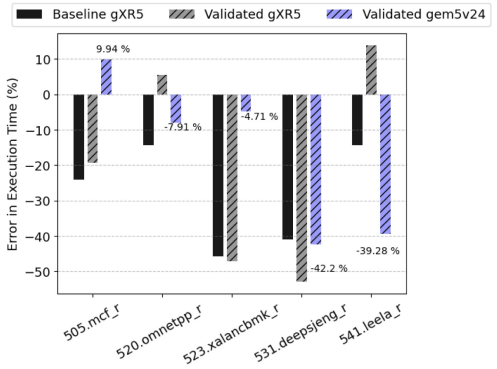


Fig. 12. MAPE in IPC for stressors run on gXR5 and Rocket Chip at various calibration stages.



(a) IPC of stress-ng benchmarks executed in the simulator and FPGA emulation



(b) Execution time of selected SPEC2017 suite applications executed in the simulator and FPGA emulation

Fig. 13. Performance of *RISCVRocketBoard* in gXR5 and gem5 against FPGA emulated system.

be configured via the corresponding python scripts in gem5. The size of the branch predictor is matched to the hardware branch predictor. The data access latency of the L1 cache and the latency of integer functional units in gXR5 are matched to the actual hardware. The implementation of the Gshare branch predictor model and fine-tuning of integer division functional unit (with 16 clock cycle latency) reduces the MAPE in IPC to 6% for control intensive stressors and to 16.2% (overall). The reduction in MAPE in IPC for stressors in various tuning stages is depicted in Figure 12.

The Rocket system emulated on the VC707 FPGA runs at a frequency different from the system modeled on gXR5. Moreover, the ratio of DRAM Controller frequency and CPU operating frequency is kept at 2.67 for both hardware and gXR5. The number of clock cycles elapsed while executing SPEC suite applications is used to compare the performance statistics of the hardware and validated gXR5. The MAPE in the execution cycles is reduced from 31% to 18.9 % for gXR5. We also provide the validated *RISCVRocketBoard* built on top of gem5 "develop" branch by matching the configuration of the calibrated MinorCPU, caches, MMU cache, and so on. For the RISCVRocketBoard, the core of both the emulated and simulated system is clocked at 100 MHz. Figure 13

shows the accuracy of the *RISCVRocketBoard*, with mean error in execution time of 20% for the selected SPEC CPU 2017 suite applications.

Using SPEC suite to calibrate FS-simulators takes 3-7 days per simulation. This compared to 3 hours per simulation for stress-ng benchmarks, greatly speeding up the validation process.

6 Conclusion

In this work, we have presented a new approach streamlining CPU fine-tuning methodology for FS-simulators. The proposed methodology is generic (i.e., extensible to both other ISAs and to other modeled components, including Out-of-Order CPU, memory hierarchy) as it fine-tunes ISA-agnostic models of FS-simulator. We applied this methodology to fine-tune the in-order MinorCPU model of gXR5 against two hardware targets having varying degrees of available specifications. To this end, we employed component-level stressors, which are much more representative of realistic workloads than synthetic micro-benchmarks [26], while requiring much less simulation time than full benchmark suites such as SPEC. As a result, our validated simulated systems exhibit similar errors in performance metrics for both training (stress-ng) and large (SPEC) applications, unlike existing methodologies. With component-level calibration, we reduced the MAPE in execution time from 44% to 22.9% and from 30% to 18.9% for the two RISC-V hardware targets. We make use of the modularity offered in gem5, to create two validated boards as part of the standard gem5 library. The boards have a mean error of 20%–23% in execution time, re-affirming the reproducibility of our methodology.

References

[1] Ayaz Akram and Lina Sawalha. 2019. Validation of the gem5 simulator for x86 architectures. In *Proceedings of the 2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. 53–58. DOI: https://doi.org/10.1109/PMBS49563.2019.00012

[2] Marco Antonio Zanata Alves, Carlos Villavieja, Matthias Diener, Francis Birck Moreira, and Philippe Olivier Alexandre Navaux. 2015. SiNUCA: A validated micro-architecture simulator. In *Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. 605–610. DOI: https://doi.org/10.1109/HPCC-CSS-ICESS.2015.166

[3] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC'05)*, USENIX Association, Anaheim, CA, 41.

[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7. https://doi.org/10.1145/2024716.2024718

[5] Anastasiia Butko, Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli, David Novo, Lionel Torres, and Michel Robert. 2016. Full-system simulation of big.LITTLE multicore architecture for performance and energy exploration. In *Proceedings of the 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. 201–208. DOI: https://doi.org/10.1109/MCSoC.2016.20

[6] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. 2012. Accuracy evaluation of GEM5 simulator system. In *Proceedings of the 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip*. 1–7. DOI: https://doi.org/10.1109/ReCoSoC.2012.6322869

[7] Odysseas Chatzopoulos, George-Marios Fragkoulis, George Papadimitriou, and Dimitris Gizopoulos. 2021. Towards accurate performance modeling of RISC-V designs. In *Proceedings of the Workshop on RISC-V for Computer Architecture Research*. 1–8.

[8] Qi Chen, Zhaopeng Meng, and Ran Su. 2020. WERFE: A gene selection algorithm based on recursive feature elimination and ensemble strategy. *Frontiers in Bioengineering and Biotechnology* 8, 496 (2020), 496.

[9] R. Desikan, D. Burger, and S. W. Keckler. 2001. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*. 266–277. DOI: https://doi.org/10.1109/ISCA.2001.937455

[10] 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 *Proceedings of the 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. 266–273. DOI: https://doi.org/10.1109/SAMOS.2014.6893220

[11] Alexander Dörflinger, Mark Albers, Benedikt Kleinbeck, Yejun Guan, Harald Michalik, Raphael Klink, Christopher Blochwitz, Anouar Nechi, and Mladen Berekovic. 2021. A comparative survey of open-source application-class RISC-V processor implementations. *In Proceedings of the 18th ACM International Conference on Computing Frontiers.*

[12] Joshu Klein, Yasir Qureshi, Marina Zapater, and David Atienza. 2020. *gXR5: A gem5-based Full-system RISC-V Simulator.* Retrieved January 10, 2023 from https://www.epfl.ch/labs/esl/research/2d-3d-system-on-chip/gXR5

[13] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt Stephen Twigg, Huy Vo, and Andrew Waterman. 2021. *The Rocket Chip Generator.* Retrieved March 01, 2023 from http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[14] Baptiste Gregorutti, Bertrand Michel, and Philippe Saint-Pierre. 2017. Correlation and variable importance in random forests. *Statistics and Computing* 27 (2017). DOI: https://doi.org/10.1007/s11222-016-9646-1

[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 *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software.* 13–22. DOI: https://doi.org/10.1109/ISPASS.2014.6844457

[16] Peter Yuen Ho Hin, Xiongfei Liao, Jin Cui, Andrea Mondelli, Thannirmalai Muthukaruppan Somu, and Naxin Zhang. 2021. Supporting RISC-V full system simulation in gem5. In *Proceedings of the 5th Workshop on Computer Architecture Research with RISC-V.* 14–19.

[17] Quentin Huppert, Timon Evenblij, Manu Perumkunnil, Francky Catthoor, Lionel Torres, and David Novo. 2021. Memory hierarchy calibration based on real hardware in-order cores for accurate simulation. In *Proceedings of the 2021 Design, Automation and Test in Europe Conference and Exhibition.* 707–710. DOI: https://doi.org/10.23919/DATE51398.2021.9474108

[18] Ilhan Firat Kilincer, Fatih Ertam, Abdulkadir Sengur, Ru-San Tan, and U. Rajendra Acharya. 2023. Automated detection of cybersecurity attacks in healthcare systems with recursive feature elimination and multilayer perceptron optimization. *Biocybernetics and Biomedical Engineering* 43, 1 (2023), 30–41.

[19] Guillem López-Paradís, Brian Li, Adriá Armejach, Stefan Wallentowitz, Miquel Moretó, and Jonathan Balkind. 2023. Fast behavioural RTL simulation of 10B transistor SoC designs with metro-Mpi. In *Proceedings of the 2023 Design, Automation and Test in Europe Conference and Exhibition.* 1–6. DOI: https://doi.org/10.23919/DATE56975.2023.10137080

[20] S. McFarling. 1993. *Combining Branch Predictors.* Technical Report. 250 University Avenue Palo Alto, California 94301 USA.

[21] Rafael Medina, Darong Huang, Giovanni Ansaloni, Marina Zapater, and David Atienza. 2023. REMOTE: Re-thinking task mapping on wireless 2.5D systems-on-package for hotspot removal. In *Proceedings of the 2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration.* 1–6. DOI: https://doi.org/10.1109/VLSI-SoC57769.2023.10321912

[22] Karan Pathak, Johsua Klein, Giovanni Ansaloni, Marina Zapater, and David Atienza. 2023. Validating full-system RISC-V simulator: A systematic approach. In *Proceedings of the RISC-V Summit Europe, Barcelona* (Barcelona, Spain). RISC-V Summit Europe, Barcelona, Spain, 2 pages.

[23] Yudi Qiu, Tao Huang, Yuxin Tang, Yanwei Liu, Yang Kong, Xulin Yu, Xiaoyang Zeng, and Yibo Fan. 2024. Gem5Tune: A parameter auto-tuning framework for Gem5 simulator to reduce errors. *IEEE Transactions on Computers* 73, 3 (2024), 902–914. DOI: https://doi.org/10.1109/TC.2023.3347675

[24] Yasir Mahmood Qureshi, William Andrew Simon, Marina Zapater, David Atienza, and Katzalin Olcoz. 2019. Gem5-X: A Gem5-based system level simulation framework to optimize many-core platforms. In *Proceedings of the 2019 Spring Simulation Conference.* 1–12. DOI: https://doi.org/10.23919/SpringSim.2019.8732862

[25] Robert Scheffel. 2018. *Simulation of RISC-V based Systems in gem5.* Master's Thesis. TU Dresden.

[26] ubuntu wiki. 2020. *Stress-ng.* Retrieved January 01, 2023 from https://wiki.ubuntu.com/Kernel/Reference/stress-ng

[27] SiFive FU540-C000 Manual v1p4. 2021. Retrieved January 01, 2023 from https://sifive.cdn.prismic.io/sifive/d3ed5cd0-6e74-46b2-a12d-72b06706513e_fu540-c000-manual-v1p4.pdf

[28] Irene Wang, Prasenjit Chakraborty, Zi Yu Xue, and Yen Fu Lin. 2022. Evaluation of gem5 for performance modeling of ARM Cortex-R based embedded SoCs. *Microprocessors and Microsystems* 93, 0141-9331 (2022), 104599. DOI: https://doi.org/10.1016/j.micpro.2022.104599

[29] Yuzhen Zhang, Jingjing Liu, and Wenjuan Shen. 2022. A review of ensemble learning algorithms used in remote sensing applications. *Applied Sciences* 12, 17 (2022), 8654.

[30] Niko Zurstraßen, José Cubero-Cascante, Jan Moritz Joseph, Li Yichao, Xie Xinghua, and Rainer Leupers. 2023. par-gem5: Parallelizing gem5's Atomic Mode. In *Proceedings of the 2023 Design, Automation and Test in Europe Conference and Exhibition.* 1–6. DOI: https://doi.org/10.23919/DATE56975.2023.10137178

Received 5 August 2024; revised 2 May 2025; accepted 9 May 2025