Towards Developing High Performance RISC-V Processors Using Agile Methodology

Yinan Xu*†, Zihao Yu*, Dan Tang*‡, Guokai Chen*†, Lu Chen*†, Lingrui Gou*†, Yue Jin*†, Qianruo Li*†, Xin Li*†, Zuojun Li*†, Jiawei Lin*†, Tong Liu*, Zhigang Liu*, Jiazhan Tan*, Huaqiang Wang*†, Huizhe Wang*†, Kaifan Wang*†, Chuanqi Zhang*†, Fawang Zhang‖, Linjuan Zhang*†, Zifei Zhang*†, Yangyang Zhao*, Yaoyang Zhou*†, Yike Zhou*, Jiangrui Zou‖, Ye Cai‖, Dandan Huan¶, Zusong Li¶, Jiye Zhao¶, Zihao Chen§, Wei He§, Qiyuan Quan§, Xingwu Liu**, Sa Wang*†, Kan Shi*, Ninghui Sun*† and Yungang Bao*†

*State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, China

†University of Chinese Academy of Sciences, China

‡Beijing Institute of Open Source Chip, China

§Peng Cheng Laboratory, China

¶Beijing VCore Technology Co., Ltd., China

‖Shenzhen University, China

**Dalian University of Technology, China

***Abstract*—While research has shown that the agile chip design methodology is promising to sustain the scaling of computing performance in a more efficient way, it is still of limited usage in actual applications due to two major obstacles: 1) Lack of tool-chain and developing framework supporting agile chip design, especially for large-scale modern processors. 2) The conventional verification methods are less agile and become a major bottleneck of the entire process. To tackle both issues, we propose MINJIE, an open-source platform supporting agile processor development flow. MINJIE integrates a broad set of tools for logic design, functional verification, performance modelling, pre-silicon validation and debugging for better development efficiency of state-of-the-art processor designs. We demonstrate the usage and effectiveness of MINJIE by building two generations of an open-source superscalar out-of-order RISC-V processor code-named XIANGSHAN using agile methodologies. We quantify the performance of XIANGSHAN using SPEC CPU2006 benchmarks and demonstrate that XIANGSHAN achieves industry-competitive performance.**

***Keywords*-agile development; open-source hardware; micro-architecture;**

I. INTRODUCTION

In the data-driven world, chip architects need to carefully balance the conventional design metrics such as performance, power and area (PPA), against various metrics such as time-to-market and limited budgets when designing high performance processors. This requires novel methods and tool flows that oversee and facilitate the entire process.

Over the past few years, agile chip design methodologies have attracted attention from both academia and industry, because these approaches could potentially limit the large engineering costs and reduce the long design cycles for chip development while maintaining reasonable PPA specifications [1] [2] [3]. There is an increasing number of researches demonstrating the usage of agile methodologies in designing CPU cores [4] [5] [6], System-on-Chips [7] [8] and hardware accelerators [9] [10]. In addition, several agile development frameworks [7] [11] [12] [13] are proposed to support rapid chip design, simulation and testing.

Despite the benefits, agile design methodology is still of limited usage, especially for practical modern designs. To further investigate the issue, we talk to chip architects, research scientists and lead engineers from twelve hyperscalers and major chip companies across multiple sectors, including data center, smartphone, communication and automobile. According to this survey, we summarize two common concerns regarding the agile methodology:

1) Agile approaches are not quite ready for *complicated* processors. Although a number of chips have been built using agile approaches, most of them are research prototypes and are relatively small or less complicated designs [1] [14] [15]. It is not clear if similar approaches can be directly applied to large-scale designs such as modern processors.

2) The verification process is still *less agile*. While modern verification methodologies such as UVM [16], constrained random [17] [18] and formal methods [19] [20] are widely adopted to accelerate the process, verification is still a major bottleneck of the entire chip development process, especially for large designs [21] [22] [23]. Existing frameworks for agile development focus more on rapid prototyping instead of verification. Therefore more tools are required to support the entire agile workflow.

In this paper, we aim to address both concerns. Towards the first concern, we propose MINJIE, a platform that integrates a rich set of agile development tools for logic design, functional verification, performance modeling, pre-silicon validation and debugging, in order to further improve development efficiency of state-of-the-art processor designs. To demonstrate the usage and capability of this platform, we develop XIANGSHAN, a high performance superscalar out-of-order RISC-V processor. With the support of MINJIE, we manage to finish the first generation of XIANGSHAN in ten months. XIANGSHAN was taped out at a frequency of

1.3GHz on a 28nm CMOS process in July 2021, achieving 7.01 on SPEC CPU2006 tested using real chips at 1GHz. We then optimized the second generation in an agile way, which is scheduled to be taped out at 2GHz on a 14nm CMOS process in Q4 2022. Experimental results show that the second generation can achieve a normalized score of 10.06/GHz on SPEC CPU2006 using representative program fragments on the RTL-simulation platform.

Towards the second concern, we further investigate the bottlenecks of the verification workflow and enhance the conventional verification methodologies by proposing agile tools and a novel *Diff-Rule based Agile Verification (DRAV)* mechanism which together form the development platform MINJIE.

1) To reduce the verification overhead, DRAV relaxes the strict equivalence of testing outcomes to checking specific *diff-rules*, forming an *N-to-1 correspondence* between the designs-under-test (DUTs) and reference model (REF). The key insight is that different outcomes under diverse micro-architectures may be legal under the design specifications. DRAV abstracts architectural behaviors as diff-rules. They are defined according to specifications that are deterministic and persistent across different processor designs. Therefore, different designs under the same design specification can be verified by one REF with the same set of diff-rules.

2) To identify the sources of behavioral non-determinism and specify the diff-rules for RISC-V processors, we propose a new verification framework DiffTest composed of diff-rule checkers and information probes. The key insight is to embed information probes into high-level HDL (e.g., Chisel) designs and use them to convey information to the diff-rule checkers.

3) To address the critical performance overhead introduced by debugging information in RTL-simulation, we propose a lightweight simulation snapshot technique (LightSSS) for on-demand transitions between normal-mode and debug-mode. LightSSS efficiently creates circuit-agnostic snapshots by storing only diffs with the copy-on-write mechanism.

This paper makes the following contributions.

- A platform MINJIE supporting agile processor development flow, with newly developed tools for logic design, verification, validation, performance modeling and debugging for large-scale digital circuit design such as high performance processors.
- We build two generations of a superscalar out-of-order RISC-V processor code-named XIANGSHAN with an industry-competitive performance by adopting agile methodologies provided by MINJIE.
- Both MINJIE and XIANGSHAN are open-sourced to facilitate future research on micro-architecture and system-level designs using agile methodologies.

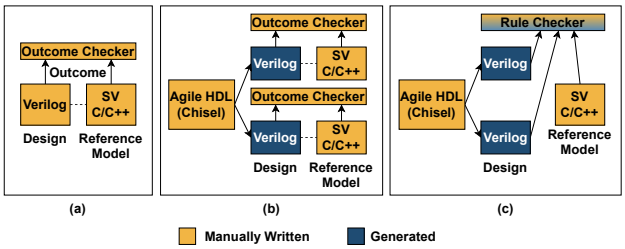


Figure 1. Different ways of building high performance processors. (a) Traditional design and verification. (b) Agile design with conventional case-by-case verification. (c) Agile design and verification as generators.

II. BACKGROUND

A. Agile and Open-Source Hardware

Agile chip development and open-source hardware have gained ever-growing attention over the past years. The goal of agile and open approaches is to reduce significant engineering costs and long design cycles for chip development. For example, Lee et al. [1] adopted an agile development approach for building 11 RISC-V microprocessors, which were taped out on 28-nm and 45-nm CMOS processes in five years. With open-source hardware design tools, five undergraduate students [14] [24] are able to design a Linux-compatible RISC-V processor on 110-nm in four months.

However, the agile and open methodology is still unrecognized for high performance and complicated designs due to the lack of high-quality hardware projects with agile development tools. BOOM [6] and XT910 [25] are two of the highest performance open-source RISC-V processors, but their performance still lags behind commercial x86 and ARM processors. Besides, the development and verification efforts on these processors are not fully revealed to the public. Therefore, the capability of agile chip development is still not fully demonstrated to convince the industry community, which prevents the agile methodology from being widely adopted.

B. Processor Functional Verification

The common practice of processor verification is to build a co-simulation framework to compare the equivalence between the outcomes of the design-under-test (DUT) and a reference model (REF), as shown in Figure 1. The outcome can be configured with various contents depending on the verification goals. For example, to ensure ISA-level compliance, the outcome is defined as the architectural states of a processor, such as the general-purpose registers and the program counter. The underlying framework compares the outcomes of a DUT with those from an instruction set simulator (ISS), which serves as the golden model of the target ISA.

However, this strict equivalence testing strategy must address the issue of non-deterministic behaviors affected by the micro-architectural implementation details of the design.

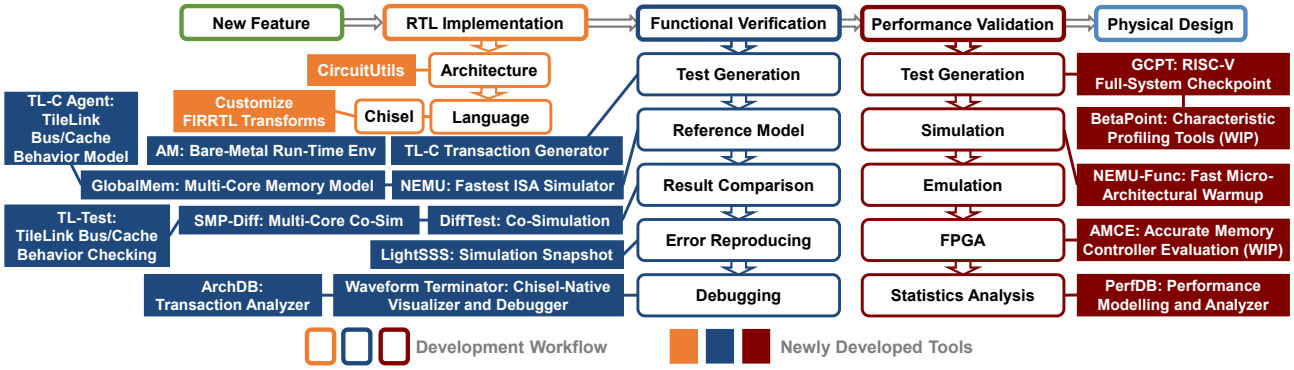


Figure 2. Agile workflow of MINJIE and the related tools.

For example, imprecise exceptions [26] and asynchronous interrupts [27] may unpredictably break the normal instruction flow of the processor. The state-of-the-art co-simulation framework Dromajo targets deterministic architectural states by forcing the REF to take the same interrupts as DUT [27]. However, for reference models without micro-architectural information, there exist more scenarios of non-determinism, and these events usually cause a divergence between the outcomes of DUT and REF. It is also a widely accepted challenge in formal verification and software testing [28] [29] [30], requiring a large state space to be maintained and explored. We demonstrate addressing this issue of the non-determinism in co-simulation environments in Section III-A.

C. Verification Framework

Figure 1(a) illustrates the traditional design and verification interactions. Given the same design specifications, RTL engineers build a processor in Verilog, while verification engineers develop corresponding reference models for various parts and levels of the processor in other languages (e.g., SystemVerilog, C/C++). The Verilog design (DUT) and the REF are put into a verification framework such as UVM to check the equivalence between their outcomes.

While traditional verification focuses on a specific design instance, agile development encourages maintaining generators. As shown in Figure 1(b), the advent of agile design involves high-level HDLs that frequently generate updated design instances. This new design paradigm brings two challenges on the setup of the reference models and their interactions with the DUT. First, the REF needs to cover the different micro-architecture-dependent details across different iterations of REF and avoid mismatches between their outcomes. Second, minor modifications in high-level HDLs may lead to significant changes in Verilog and substantially break the original verification code. Therefore, verification engineers suffer from agile development. We address these two issues in Section III-B.

D. RTL-Simulation and Debugging

Hardware designs require a long time of RTL-simulation through the verification process. Common approaches include software-, FPGA-, and emulator-based RTL-simulation.

FPGA runs the fastest (tens of MHz) and is important for hardware prototyping. While a lot of works have addressed the issues of scalability [31] [32] [33], debuggability [34] [35] [36] [37] [38], and long synthesis time [39], FPGAs still need to be improved for large-scale designs [40] [41] [42].

Emulators such as Cadence Palladium [43], Mentor Veloce [44], and Synopsys Zebu [45] can simulate large circuits at a speed of MHz with full visibility and good debuggability. However, they are pretty expensive and thus difficult to deploy at scale by both academia and industry.

Software-based RTL-simulators [46] [47] are the most commonly used tools and provide full visibility of the simulated circuits. However, the software approach runs at a speed of KHz for large-scale designs and becomes even slower when necessary debugging information is enabled such as the waveform and logs (12× performance reduction in our case on XIANGSHAN). It is of great significance to address the debugging efficiency for software-based RTL-simulators as they are being used for almost all hardware projects. Section III-C demonstrate our approach to speed up the debugging process.

E. Performance Evaluation

Accurate pre-silicon performance evaluation and validation are important for high performance processors. It is also of critical importance to evaluate a processor design with an agile approach because it affects the efficiency of feature exploration. There are various ways of feature exploration and performance evaluation.

Architectural simulators [48] [49] are widely used for design space exploration and performance evaluation due to their high running speed and programming simplicity. However, there is a huge gap between the performance

and architecture of the open-source version of GEM5 and state-of-the-art high-performance processors. According to our experiments, a roughly XIANGSHAN-parameter-aligned GEM5 model achieves ∼7/GHz on SPEC CPU2006 at 2GHz, which is ∼30% less than XIANGSHAN and ∼70% less than the Apple M1 [50]. For better accuracy, companies have to maintain independent teams for architectural simulators to align with the micro-architecture of the real RTL model, which requires a huge investment of human resources.

Performance evaluation on RTL models relies on cycle-accurate RTL-simulation, with a typical simulation speed of KHz on software and MHz on FPGAs. However, it still takes more than 150 hours for XIANGSHAN to finish SPEC CPU2006 at 50MHz on a single FPGA, resulting in a one-week iteration cycle that does match the agile development requirements. Besides, unlike the real chips, FPGAs have the issue of inconsistent frequency ratio between the CPU (50MHz) and DDR4 (up to 2400MHz) [51], making the performance results on FPGAs unrealistic. Thus, it's of significant importance to investigate how to accurately and agilely evaluate the pre-silicon performance of a processor. We demonstrate a software RTL-simulation based performance evaluation workflow in Section III-D.

III. TOWARDS AN AGILE WORKFLOW USING PLATFORM: MINJIE

MINJIE is an open-source platform designed to support the agile chip development workflow, integrating a rich set of existing and newly developed tools and tool-chain for logic design, functional verification, performance modeling, validation and debugging, as shown in Figure 2. In this section, we detail the functionalities and design methods of the representative tools for agile verification, debugging, and performance evaluation, together with the description of the entire workflow for agile chip development.

A. Diff-Rule based Agile Verification (DRAV)

Functional verification is key to ensuring successful chip development. The key to function verification though, is to build reference models (REFs) that can properly reflect the design specifications.

An intricate example is shown in the top part of Figure 3. According to the RISC-V instruction set manuals, any store (PTE Write, Store Retire) before an explicit `sfence.vma` instruction may or may not take effect (Store Complete) in the virtual address translation process (TLB Page Walk). This flexibility makes the behavior of translation look-aside buffers (TLBs) and the instruction execution (Normal Commit or Page Fault) look non-deterministic from the outside, whereas both behaviors are legal. As for RISC-V implementations with a store queue, whether the TLB receives the PTE updates depends on whether the PTE store leaves the store queue to the memory subsystem. However, since the REF does not have such micro-architectural states, there will be a possible behavioral divergence between the DUT and REF.

Therefore, the verification strategy must address *the micro-architecture dependency problem*: when the REF is lack of enough micro-architectural states and behaves differently with the DUT, the verification strategy should determine whether this mismatch is due to true bugs or false positives.

The traditional model as shown in Figure 1(a) tackles this issue by setting up various simplified REFs that are specifically targeted on certain parts of the design. Given a processor P_i, behaviors of both DUT and REF can be described as how the micro-architectural state $s_{P_i} \in S_{P_i}$ changes upon an event $e \in E$, where S_{P_i} is the micro-architectural state space of processor P_i, and E is an event space, including instruction commit, exception, interrupt, etc. That is, both DUT and REF define a mapping R_{P_i} from the state space and event space to the state space:

$$R_{P_i} : S_{P_i} \times E \longrightarrow S_{P_i}.$$

Verification is to check the equivalence of mappings defined by DUT and REF, which implies an *1-to-1 correspondence* between the DUT and the REF. This model is useful for designs such as commercial processors, with a long production life-cycle where the verification IPs and frameworks can be reused throughout the entire project.

Nevertheless, maintaining the 1-to-1 correspondence between REF and DUT can be difficult with a short development cycle, rapid feature changes and limited resources, commonly seen in the agile development model as shown in Figure 1(b).

Rethinking the principle of verification, we argue that a given design specification can lead to diverse implementations. Therefore, once the behavior of a DUT satisfies the definition of specifications, the implementation details are allowed to be diverse and the requirement of strict equivalence can be relaxed.

Following this principle, it is feasible that DUTs with different implementation details can be verified by the same REF if their behaviors are conforming to the same specification. Given a specification $\mathcal{P}$ such as the instruction set, one can build a REF according to $\mathcal{P}$ that defines the legal architectural state transitions starting from $s_{\mathcal{P}} \in S_{\mathcal{P}}$ with an event $e \in E$, where $S_{\mathcal{P}}$ is the architectural states defined by $\mathcal{P}$. Due to the existence of diverse micro-architecture dependent behaviors in $\mathcal{P}$, for given current state $s_{\mathcal{P}}$ and e, the REF $\mathcal{R}$ outputs a set of next states. That is,

$$\mathcal{R} : S_{\mathcal{P}} \times E \longrightarrow 2^{S_{\mathcal{P}}},$$

where $2^{S_{\mathcal{P}}}$ is the power set of $S_{\mathcal{P}}$. Given a set of processors $\{P_1, ..., P_n\}$ following $\mathcal{P}$, it is straightforward to find the mapping f_{P_i} from processor micro-architectural state to architectural state: $f_{P_i} : S_{P_i} \longrightarrow S_{\mathcal{P}}$. Thus, for any $P_i, i \in \{1, ..., N\}$, verification of processor P_i is to check whether $f_{P_i}(R_{P_i}(s_{P_i}, e)) \in \mathcal{R}(f_{P_i}(s_{P_i}), e)$. This forms an

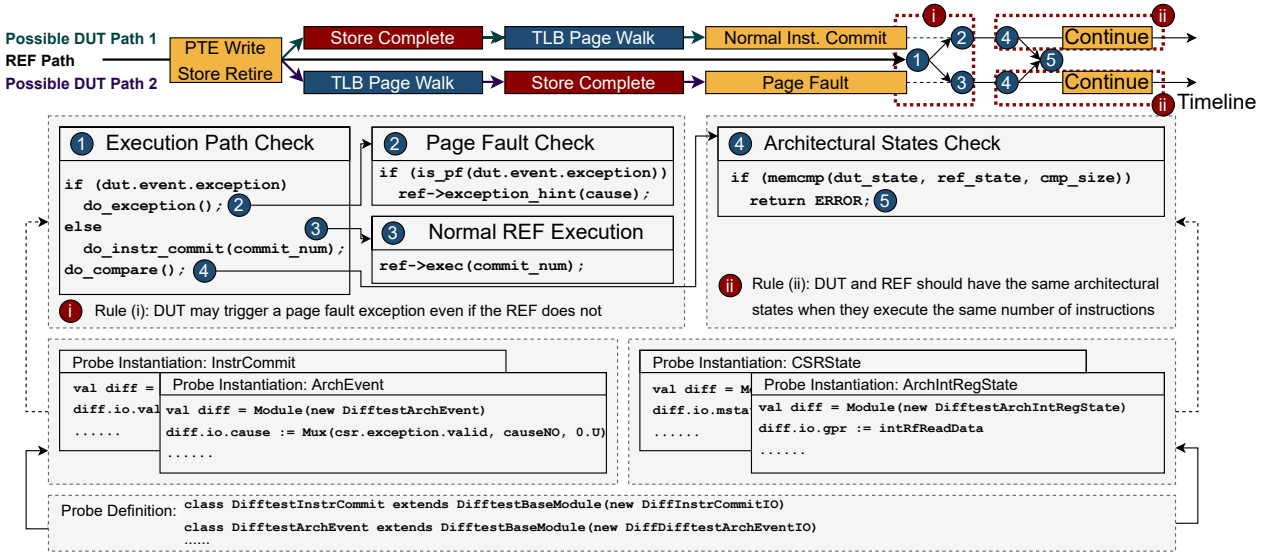


Figure 3. An illustrative example of the micro-architecture dependent behavior in TLBs with the MINJIE solutions. Invalid PTEs are allowed to be cached in TLBs, resulting in a page fault when TLBs speculatively access the page table (TLB Page Walk) before the Linux kernel allocates with a valid page (Store Complete).

N-to-1 mapping between DUTs and REF. Since only one REF $\mathcal{R}$ is maintained and $\mathcal{R}$ is simpler than $\{R_{P_i} | i \in \{1, ..., N\}\}$, the verification overhead is reduced, as shown in Figure 1(c).

Based on the observations, we leverage *diff-rules* to abstract legal behaviors defined in specifications and deal with non-determinism in verification. A diff-rule $r \in \mathcal{R}$ is deterministic and stable across different hardware implementations with the same design objectives. We propose a novel ***Diff-Rule based Agile Verification (DRAV)*** mechanism that enhances conventional verification methodology by providing both the verification infrastructures and a real implementation for RISC-V processors, as shown in Figure 3. The DRAV mechanism consists of diff-rules $\mathcal{R}$, information probes f_{R_i} and rule checkers that can be freely used by verification engineers to express their expectations of the outcomes of the DUT and REF. These components are carefully designed to achieve higher efficiency in the context of open-source and agile hardware development.

B. DiffTest: DRAV for RISC-V Processors

1) Overview: DiffTest is a co-simulation based verification framework as shown in Figure 4, it accelerates the verification convergence by adopting the DRAV methodology and relaxing the non-critical equivalence checks from design specifications. Specifically, DiffTest provides flexibility to add diff-rules and reconfigure the reference model on-the-fly, thus being scalable to support multiple designs simultaneously.

DiffTest adopts a co-simulation mode where a DUT and a REF can run simultaneously cycle by cycle and are synchronized by diff-rules in the form of probes and checkers. In every cycle, the DUT outcome (e.g. instruction commit

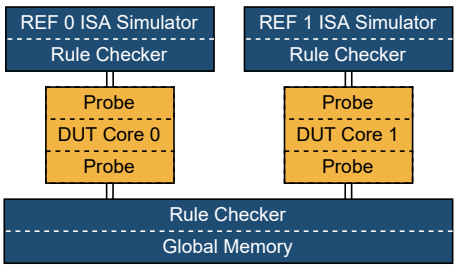


Figure 4. Components of DiffTest.

information) is obtained by probes and conveyed to checkers. If there is only one rule for this outcome, the REF will not receive any hints, and hence the outcome is directly used for the equivalence checking. If there are multiple legal rules and the DUT outcome matches one of them, the REF will be triggered to perform the same operations as the DUT. Once a mismatch occurs, the checking mechanism identifies potential issues of the DUT and aborts the RTL-simulation. DiffTest enables rapid refining of the micro-architecture-dependent behavior of a REF on-the-fly, thus significantly reducing the REF implementation cost.

2) Non-deterministic Behaviors and Diff-Rules: The key challenge of devising diff-rules is identifying sources of non-deterministic micro-architecture dependent behaviors. Fortunately, it affects only verification efficiency rather than correctness. The more sources are identified for devising rules, the fewer details in REFs are required for verification. Therefore, the efficiency can be continuously improved by

adding more diff-rules. However, one needs to balance the tradeoff between the number of diff-rules and the setup cost of REFs and the framework.

To identify the non-determinism sources on machine-mode CSRs in RISC-V instruction set, we investigate the RISC-V privilege specification and devise at least 120 rules, which are relatively simple to distinguish and set up as they mostly affect the CSR read/written values only.

Next, we present several representative sources of non-determinism in the ISA- and bus-level verification of multi-core high-performance processors. It is worth noting that the attempt to cover more diff-rules of processors is never enough, since the increase of rule checking coverage only increases the confidence of correctness but never ensures the correctness. Instead, agility addresses the verification issue by increasing the workflow efficiency and thus allows more verification trials within a fixed time interval.

a) Speculative virtual address translations: Figure 3 shows the non-deterministic behavior of the TLB. Specifically, Linux chooses not to execute a memory-barrier instruction after allocating a new physical page to avoid flushing instructions until a page fault exception [52]. In most cases, the in-memory page table entries (PTEs) are updated quickly after the retirement of the store instructions. However, if the store operation that updates a PTE does not take effect when the TLB accesses the memory, a memory instruction accessing the page will trigger a page fault exception, which brings the processor out of the normal execution to the exception handler. Two diff-rules are used to address this issue: (1) The DUT may trigger a page fault exception even if the REF does not trigger; (2) DUT and REF should have the same architectural states after executing the same instruction.

b) Cache hierarchy and multi-core scenarios: Load and store instructions may have different outcomes due to the exponential interleaving space of concurrent memory accesses, making it infeasible to build a sound and complete multi-core REF for co-simulation with a multi-core processor. While the memory consistency model defines a large set of legal software-visible behaviors, micro-architectural details determine how certain memory operations are ordered. We leverage diff-rules to prune the astronomically large interleaving space, and build a co-simulation framework for multi-core processors with simple single-core REFs.

As an example, the store buffer is a source of divergence between the DUT and REF or non-determinism of the DUT. Under the RISC-V weak memory order (RVWMO), load instructions are allowed to first bypass the value from the private store buffer, and access the global memory if the bypass fails [53]. A naive multi-core co-simulation requires maintaining a correct store buffer in the REFs, unless disagreements between the DUT and REF will frequently break the co-simulation flow. In DiffTest, we devise a diff-rule from the RVWMO specification that allows DUTs to maintain the global memory and updates the REF memory when they disagree. It introduces the Global Memory that records the store requests that enter the cache hierarchy in DUT. Data correctness is checked by both Global Memory and local memory of single-core REFs. When the single-core REF executes a load with a different value from the DUT, DiffTest accesses the same load address in Global Memory to check whether this load value is possibly written by other hardware threads. If so, the value will be updated to both the local memory and the destination register of the load instruction in the single-core REF.

Besides the RISC-V ISA-level diff-rules and co-simulation, DiffTest is further enhanced with diff-rules for the consistent cache hierarchy. We observe that caches can be regarded as black boxes, ignoring internal implementation details. As long as we monitor the transactions between caches and cores with some additional historical information, the correctness of the caches can be checked. We devise two major categories of rules for cache consistency. (1) To define legal transactions received in different situations, a series of rules are set up according to the bus protocol specification. (2) A permission scoreboard is maintained to track the permission of each data block according to the cache coherence protocol.

c) More sources of non-determinism: We selectively choose some micro-architectural components that are complicated to implement on the system level REFs but relatively unimportant and devise diff-rules to abstract their behaviors. For example, we avoid in REFs the details of most memory-mapped IO devices, sources of external interrupts, reservation sets for the load-reserved/store-conditional (LR/SC) instruction pair [53], macro-instruction fusion [54], performance counter reads, and so forth. In these cases, we devise relaxed diff-rules to achieve agile verification, where the DUT is trusted to trigger the corresponding events. The architectural states afterward are checked by DiffTest assuming the REF is being forced to trigger the same events. For example, SC instructions are allowed to fail on a timeout between the LR and SC. Due to its dependence on micro-architectural states, a timeout failure of the SC instruction on DUT will be trusted, while the REF is notified by this event to refine its behavior with a failure as well.

To leverage diff-rules in verifying the processors, one should be careful to ensure the soundness of diff-rules and the applicability to various processor implementations. In both the page-fault and SC-failure examples, DiffTest implements more checkers to ensure verification quality. The number of forced page-faults and SC failures with their positions in the program are tracked and asserted not to repeatedly occur to avoid false negatives. Furthermore, to efficiently verify these components, module-level DiffTest with fine-grained diff-rules is used instead. For example, diff-rules based on the design specification of TLBs are used to verify the TLB module, forming a multi-level verification plan.

3) Information probes: Towards the second challenge of broken verification code caused by high-level HDLs, DiffTest

is decomposed into diff-rule checkers and the probes to automate the extraction of the required information. As shown in Figure 3 and Figure 4, a probe is a piece of logic put inside a processor design.

Traditionally, necessary verification information is extracted by the monitor or adapter under the widely accepted universal verification methodology (UVM) [16]. Conventionally, they are written manually by verification engineers who know the verification requirements and look into the design to find the signals of interest. However, under the agile development methodology, hardware designs may frequently change along with a large number of internal signal definitions, causing repetitive and meaningless porting work by the verification engineers. Instead, probes are intentionally written by designers who know the implementation details and undertake the responsibility of advancing the design.

High-level HDLs like Chisel have been widely adopted to accelerate the hardware design process. As shown in the Probe Definition part of Figure 3, probes are injected into a processor design and implemented as native interfaces in the same high-level HDL as the design. With the support of the code auto-generation feature of Chisel, a probe is defined as a bundle of signals with their directions and types, and the instantiable module is automatically generated according to the definition of the bundle. When the verification process requires some more information, the only thing to do is defining the format of the required bundle of signals. Then the interface can be generated and used for the communication between design and verification.

To maintain the compatibility of probes between various designs and keep a low maintenance cost as the design evolves, probes are designed to be the basic building blocks of the verification information. For example, superscalar processors may execute and commit more than one instruction per clock cycle. To verify these processors, a verification framework requires the information for multiple instructions that commit at the same clock cycle. In our design, the information probe for instructions is defined as the information extractor for only one instruction and is expected to be instantiated more than once in a superscalar processor. In this way, different processors share the same basic probes but have different usages of the probes. Furthermore, some static design information is implicitly conveyed to the verification. The number of function calls of the instruction probe indicates the commit width of the superscalar processor under test.

Information probes are inserted into a processor design at the design phase and extract detailed information during the RTL-simulation. The information is used for not only co-simulation and verification in DiffTest but also more debugging tools. For example, *ArchDB* is an SQLite-based database storing and parsing information obtained by probes, where tables are automatically generated by the probe definitions. ArchDB can be further used to filter and visualize the events and transactions for agile debugging. Thus, with the introduction of probes, the information gap between design and verification is mitigated.

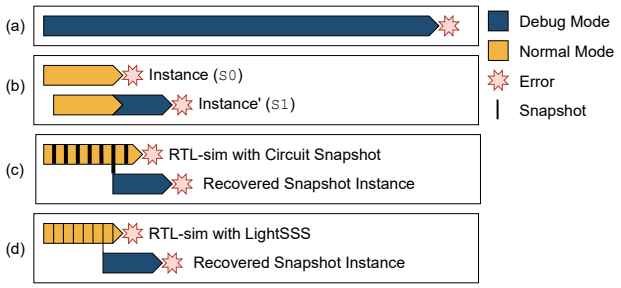


Figure 5. Various ways to obtain debugging information. (a) Always running RTL-simulation in debug mode. (b) The state-of-the-art method with two RTL-simulation instances running successively. (c) RTL-simulation with circuit snapshots. (d) RTL-simulation with LightSSS.

C. LightSSS: On-Demand Debugging

1) Overview: Debugging requires information such as the waveform and logs that slow down the simulation speed, as stated in Section II-D, To address this issue, we propose LightSSS, a lightweight simulation snapshot technique for on-demand transitions between fast-mode and debug-mode (debugging information enabled).

Figure 5 demonstrates various ways to obtain debugging information. As shown in Figure 5(a), due to the extra debugging information acquired, RTL-simulation in debug mode is usually much slower than in normal mode, regardless of the format of the information. An example of XIANGSHAN shows that, when waveform is enabled, the RTL-simulation speed drops to ∼8.5%, whereas the export of text logs causes a ∼55% decrease of simulation speed as well.

As shown in Figure 5(b), the state-of-the-art debugging method [35] runs two identical RTL-simulation instances in parallel with one instance (`S0`) launched N seconds later than the other (`S1`). When `S0` reports a bug, it notifies `S1` to turn on the debugging information like waveform and continue the simulation until the bug occurs. Since `S1` is identical with `S0` except for the delay of N seconds, when `S1` reports the same bug, we can obtain the debugging information for the last N seconds (region of interest, ROI) of simulation. This method avoids simulating in debug mode most time but requires doubled computation resources.

2) SSS by MINJIE*:* Another existing method to focus on the ROI is taking snapshots for the RTL-simulation and restoring a recent snapshot on-demand, as shown in Figure 5(c). A snapshot is generally created by concatenating the circuit states and other necessary simulation information in the form of a file image containing all the required information. However, frequently dumping snapshots cause serious performance degradation, 10% to 20% reported by the state-of-the-art RTL-simulation snapshot tool LiveSim [55]. MINJIE also designs a circuit-dependent snapshot format SSS

with a performance overhead of 17%. Each snapshot contains the entire circuit state of DUT generated by the `savable` option of Verilator [56] and other verification states required by DiffTest, such as the simulated memory in the REF. The snapshot overhead will even grow as size of the simulated memory ranges from 256MB to 16GB, resulting in serious pressure on either the I/O or memory depending on whether snapshots are stored on disks.

3) Lightweight simulation snapshot: To address the issue of snapshot efficiency, we make a key observation that most contents of the DUT memory remain unchanged between two snapshots. So instead of storing a whole snapshot, we can record only the differential states between two RTL-simulation snapshots. This observation motivates us to use a diff-based in-memory snapshot.

In this paper, we propose **a Lightweight Simulation Snapshot (LightSSS)** technique. As shown in Figure 5(d), during the RTL-simulation, snapshots are created periodically every N cycles. Each snapshot only stores its unique states (differential) and shares identical states with other snapshots. Because we only care about the region around a bug, we only reserve the most recent two snapshots and drop the earlier ones to reduce the overhead. When an error is reported, the RTL-simulation process notifies the older snapshot to replay the simulation in debug mode until the error is reproduced. In this way, the snapshot instance only needs to replay as most $2N$ cycles to acquire the debugging information.

Portability is another issue of SSS. External models in C/C++ that are linked to the simulation require manual saving and restoring of the details. It is difficult to port them because of the high complexity of internal states and the requirements of case-by-case implementations. We make a key insight of LightSSS that the software-based RTL-simulation is a process running on the Linux kernel, and utilities from the kernel can be used for circuit-transparent snapshots.

Instead of taking snapshots of simulated circuits, we take snapshots of the RTL-simulation process with a system call of the Linux kernel. LightSSS periodically calls `fork()` from the RTL-simulation process and treats the forked process as a snapshot. When the RTL-simulation process continues and updates memory contents, the Copy-on-Write (COW) technique [57] inherently serves as diff-based storage: the operating system allocates physical pages for only the modified pages, leaving unmodified pages shared between processes. By using the system call to create snapshots, we avoid the details of the simulated circuits and generalize the snapshots method to be transparent over any DUTs and other external functional models, such as the DRAMsim3.

Table I shows the comparison between different snapshot techniques for software RTL-simulation. Compared with existing general snapshots like CRIU [58], LightSSS is designed and tuned for agilely debugging the RTL-simulated hardware with less overhead due to the in-memory nature. Compared with snapshot tools for RTL-simulation, LightSSS causes far less overhead because of the incremental and circuit-agnostic implementation using `fork()`.

Table I
A COMPARISON BETWEEN DIFFERENT SNAPSHOT SCHEMES FOR SOFTWARE RTL-SIMULATION.

	In-Memory	Incremental	Circuit-Agnostic
CRIU [58]	×	✓	✓
Verilator [56]	×	×	×
LiveSim [55]	✓	×	×
LightSSS	✓	✓	✓

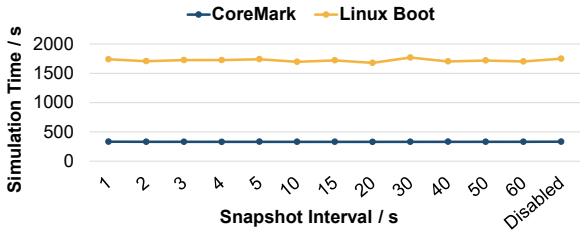


Figure 6. Simulation time when LightSSS is enabled with different snapshot intervals or disabled.

4) Evaluation: There are two major concerns about the overhead of LightSSS: (1) How much extra run-time is introduced by the `fork()` system call? (2) How often does the simulation process diverge, and how much time is it to lazily copy the modified pages? To address these two concerns, we perform an evaluation on both single-core XIANGSHAN and the larger dual-core XIANGSHAN.

We simulate a single-core XIANGSHAN to run ten iterations of CoreMark (∼5.5 minutes), and a dual-core XIANGSHAN to boot an SMP Linux (∼ 30 minutes). The RTL-simulation speed on the AMD Ryzen 5950X processor is 5.1KHz for single-core (8.09M lines of C++ generated by Verilator) and 2.4KHz for dual-core (15.47M lines of C++) XIANGSHAN using eight threads.

We first measure the overhead of the `fork()` system call. Because of the COW, the duplication of the RTL-simulation process only requires copying the process control block, which stores the basic information of a process. As expected, a `fork()` system call takes only 535 us. By contrast, creating a snapshot with SSS in Section III-C2 takes 3.671 seconds.

The overhead of the COW mechanism can be evaluated with different snapshot intervals, typically ranging from 1 second to 60 seconds. We run more than ten trials for each interval size but still observe a minor performance variation due to the 8-thread scheduling [59]. As demonstrated in Figure 6, the simulation time is barely affected by either the existence or the interval size of snapshots. The results show that LightSSS introduces an order of magnitude lower overhead than the-state-of-the-art state-of-the-art LiveSim [55] that reports a 10% to 20% performance overhead.

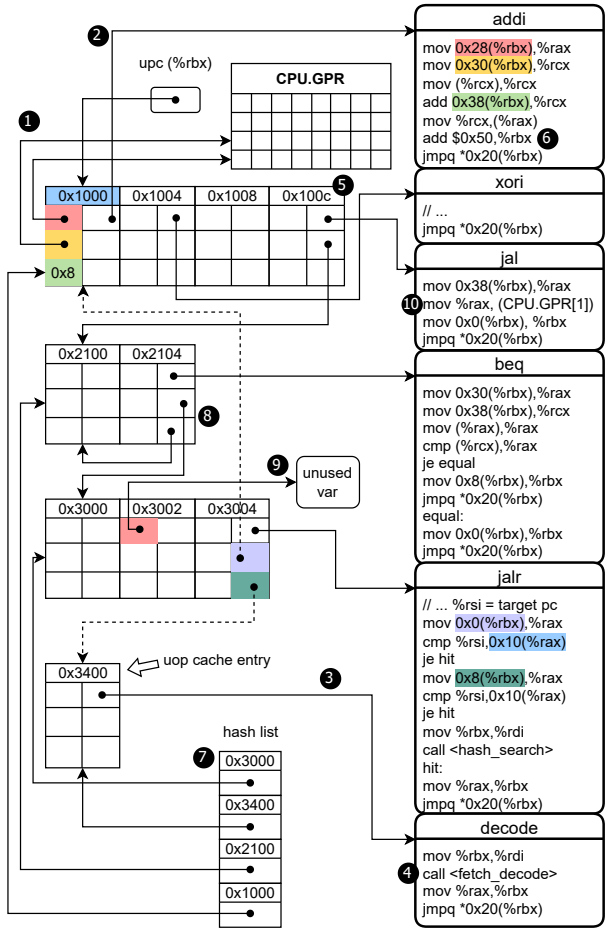


Figure 7. Overview of NEMU, which mainly shows the uop cache (left) and the threaded code model (right).

D. NEMU: Fast Interpreter for Performance Evaluation

Pre-silicon performance evaluation is important for high performance processors. Traces and checkpoints have long been used in architectural simulators [60] [61] to increase the simulation parallelism and speed up the evaluation process. Towards an accurate and agile performance evaluation approach, first MINJIE provides a high-performance instruction set interpreter call NEMU (New EMUlator), which demonstrates high flexibility to different instrumenting and profiling tasks. Note that as an interpreter instead of a binary translator, NEMU can also be used as an easy-to-develop REF for DiffTest to reduce the verification overhead. Second, MINJIE proposes a new architectural checkpoint format. MINJIE can generate checkpoints efficiently with NEMU and restore them with parallel simulation instances.

1) NEMU Design: Figure 7 shows the architecture of NEMU. It is a threaded code [62] interpreter which inlines all execution routines (the right part of Figure 7) into the execution dispatcher. To achieve high performance, NEMU adopts a lot of optimization techniques.

a) Optimizing Fetch and Decode by uop cache: Different from an instruction cache which is used to cache the instruction itself, a uop cache is used to cache all decoded results of an instruction. These results include operands and the execution routine (❶ and ❷ in Figure 7). By using a uop cache, NEMU will fetch and decode an instruction only when there is a uop cache miss (❸ and ❹ in Figure 7).

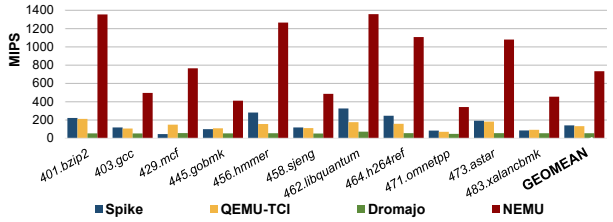
To reduce the number of uop cache miss, NEMU further organizes uop cache entries by trace, the dynamic instruction sequence. For example, in Figure 7, the uop cache entries with PC from 0x1000 to 0x100c are allocated sequentially (❺ in Figure 7). The key idea is to completely eliminate conflict misses by assigning an entry with a specific address. Note that for a traditional instruction cache, different addresses may be mapped to the same entry. Therefore, conflict misses may occur and replacement is necessary. As a result, NEMU has higher probability to stay in fast path, which in turn reduces the number of fetch and decode. Note that the uop cache will be flushed only when it is full, or there happens a system event, such as context switch. These events are infrequent during program execution.

Due to the trace organization, NEMU can fetch the next uop cache entry inside a basic block by only adding 1 to `upc` (❻ in Figure 7), which yields good locality. For unconditional indirect jumps, a hash list is used to query the uop cache entry by the target address (❼ in Figure 7). For unconditional direct jumps and conditional branches, the block chaining technique is employed. An example is the uop cache entry with PC 0x2104 (a `beq` instruction), ❽ in Figure 7. The query of hash list is performed in decode stage, which is in the slow path.

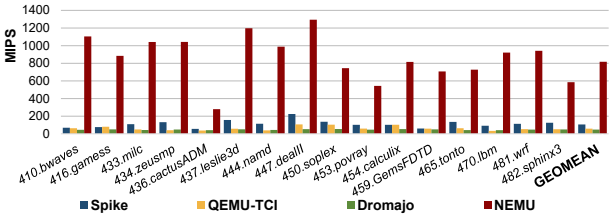
b) Writing to Zero Register: RISC architecture usually defines a zero register, whose value is always 0. To implement this feature, a traditional interpreter may check the destination register before writing to GPR, or reset the value of the zero register to 0 for every instruction. Instead, NEMU checks whether an instruction is going to write the zero register in the decode stage. If it is the case, the pointer of the destination register (part of the decode results) stored in the uop cache will be redirected to an unused variable (❾ in Figure 7). After that, the execution routine will write to the unused variable, protecting the zero register from being overwritten.

c) Pseudo-Instructions and Compressed Instructions: For a pseudo-instruction, there is usually at least one operand which is constant or fixed. For example, `ret` is a special case of `jalr rd, imm(rs)`, where `rd = zero` (constant), `rs = ra` (fixed), and `imm = 0` (constant). For instructions with such property, NEMU will define dedicated execution routines to inline these operands (❿ in Figure 7). This technique can be also applied to compressed instructions.

d) Floating-point Instructions: NEMU leverages host floating-point instructions to interpret guest floating-point instructions. For example, the execution routine of the



(a) SPECint 2006



(b) SPECfp 2006

Figure 8. The performance of Spike, QEMU-TCI, Dromajo and NEMU.

`fadd.s` instruction can be implemented by a "+" with `float` data type in C language, which will be compiled to several host (x86) floating-point instructions. For some complicated floating-point operations, NEMU will call the math library. An example is to implement the fused multiply-add instruction by calling the library function `fma()`.

2) Evaluation: We compare NEMU with Spike [63] (git commit 6c18ef56), QEMU-TCI (version 6.2.0) and Dromajo [27] (git commit fae7b3a1). Among them, Spike is the state-of-the-art RISC-V interpreter. QEMU-TCI is the interpreter mode of QEMU [64]. Dromajo is the reference model of BOOM. For the configuration of software cache size, we run different size from 1024 to 32768 for Spike, and select 16384 due to the best result. NEMU also adopts such size for the uop cache. This size does not apply to Dromajo, since there is no cache in Dromajo. We use the default cache size for QEMU-TCI. We run these interpreters on a server with Intel Core i9-9900K CPU.

SPEC CPU2006 benchmark suite is run with the *test* input sets in user mode, where system calls are handled by emulation. NEMU and QEMU-TCI have native support for user mode by managing guest memory and forwarding most of the system calls to the host operating system. Spike supports user mode with the help of `riscv-pk` [65], a proxy kernel module. Dromajo does not support user mode by default. We enhance it in a way similar to NEMU. We exclude *400.perlbench* and *435.gromacs* due to unimplemented system calls in `riscv-pk` [66].

As shown in Figure 8, among Spike, QEMU-TCI and Dromajo, Spike performs the best. For SPECint 2006, Spike can achieve 142 MIPS (million instructions per second) on average. Meanwhile, NEMU can finally achieve 733 MIPS on average, which is about 5.16× of Spike. For SPECfp 2006,

Figure 9. RISC-V Architectural Checkpoint Format.

Spike runs even slower, which only achieves 106 MIPS. This is because Spike interprets a floating-point instruction by calling SoftFloat [67]. By adopting the host floating-point instruction, NEMU can achieve 817 MIPS for floating-point benchmarks, which is about 7.71× of Spike, even 16× for some benchmarks (*410.bwaves*).

3) Use Case of Checkpoint: MINJIE defines a RISC-V ISA-level architectural checkpoint format with only basic RV64 privilege instructions, as shown in Figure 9. When compared with the existing publicly available checkpoint format [27], our checkpoint format is independent of RISC-V debug mode [68] and capable of enabling early-stage processors without external debug features. As an example, checkpoints of CoreMark-PRO [69] can be efficiently generated using NEMU at a speed of >300 MIPS.

We further adopt SimPoint [70] to sample the instruction fragments. Note that it is easy to compute the Basic Block Vector in NEMU, since it is straightforward to collect information about instructions in an interpreter. We simulate the processor design with selected checkpoints and calculate a weighted cycles per instruction (CPI) for performance validation. This checkpoint-based evaluation flow must address the issues of micro-architectural warming and sampling representativeness [71] [72] [73] [74] [75]. During the warmup period, micro-architectural states in components like branch predictors and caches are updated by executing the instructions on the hardware. The future work is to achieve better performance estimation accuracy using less time and less computation resources.

By paralleling the software-based RTL-simulation instances, a large number of checkpoints can be simulated simultaneously. With five 128-core servers and ∼1K checkpoints for 100% clustering coverage, we can estimate the SPEC CPU2006 scores within three days. By contrast, XIANGSHAN requires more than 150 hours to finish SPEC CPU2006 programs on a single FPGA at 50MHz, with roughly the same price as the five x86 servers. If we reduce the clustering coverage to 80%, preliminary evaluation results can be obtained within 24 hours. The accuracy of the evaluation result will be demonstrated in Section IV-B.

E. Put It All Together

As shown in Figure 2, MINJIE integrates a set of open-source tools to enhance the conventional verification methodologies by proposing agile tools supporting functional simulation of designs implemented by high-level languages, efficient performance modeling and validation for both software simulation and FPGA-based prototyping, and debugging metrics across each step of the development flow.

Table II
TAPE-OUT MICRO-ARCHITECTURE PARAMETERS OF TWO GENERATIONS OF XIANGSHAN: YQH AND NH.

Feature	YQH	NH
ISA	RV64GC	RV64GCBK
Process Node	28nm	14nm
Frequency	1.3GHz	2GHz
Core Number	1	2
microBTB	32 entries	256 entries
BTB	2K entries	4K entries
TAGE-SC	16K entries	16K entries
Others	RAS	RAS, ITTAGE
L1 ICache	16KB, 4-way	128KB, 8-way
L1+ Cache	128KB, 8-way	-
L1 DCache	32KB, 8-way	128KB, 8-way
L2 Cache	1MB 8-way, inclusive	1MB 8-way, non-inclusive
L3 Cache	-	6MB 6-way, non-inclusive
L1 ITLB	40 entries	40 entries
L1 DTLB	40 entries	136 entries
STLB	4K entries	2K entries
Fetch Width	8*4B instr./cycle	8*4B instr./cycle
Dec./Ren. Width	6 instr./cycle	6 instr./cycle
ROB/LQ/SQ	192/64/48	256/80/64
Phy. Int/FP RF	160/160	192/192
Execution Units	ALU, MUL/DIV, JUMP/CSR/I2F, LD, ST, FMAC, FMISC	ALU, MUL/DIV, JUMP/CSR/I2F, LD, STA, STD, FMAC, FMISC
Instruction Fusion	-	Yes
Move Elimination	-	Yes

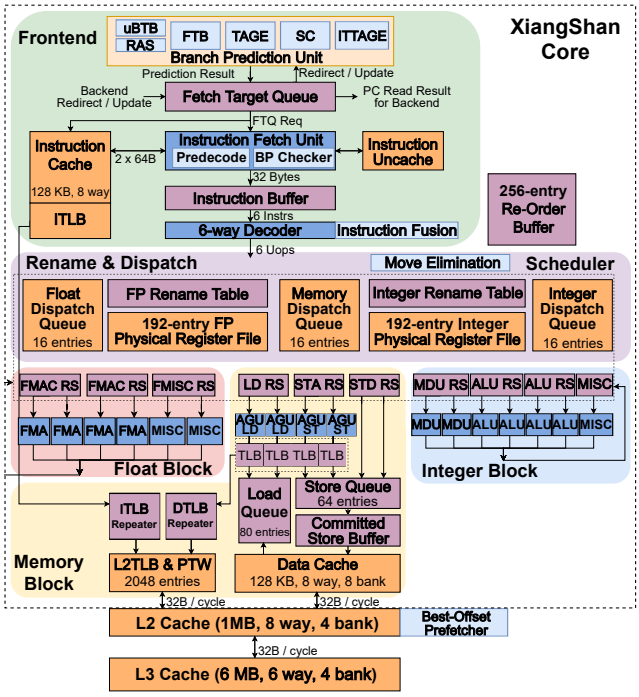


Figure 10. Micro-architecture of XIANGSHAN (NH).

We have demonstrated in detail the functional verification workflow. When new features are implemented, developers only need to launch the RTL-simulation, and these tools will be automatically invoked. If no errors are reported from DiffTest, it is of high possibility that the design is aligned with the design specification upon the corresponding test cases, and performance results can be further analyzed. Otherwise, LightSSS will extract the debugging information including the waveform and logs. We develop and open-source more debugging tools with better support for Chisel (ArchDB, Waveform Terminator, etc.). They can be employed by developers to further investigate potential bugs.

MINJIE also integrates the performance evaluation toolkit for nightly performance regression. By selecting representative checkpoints, the time-to-results for accurate performance verification is significantly reduced, further accelerating design space exploration for high performance processors. For now, MINJIE mainly adopts the RTL simulation-based verification flow, which we believe is essentially complementary to workflows on emulators and FPGAs.

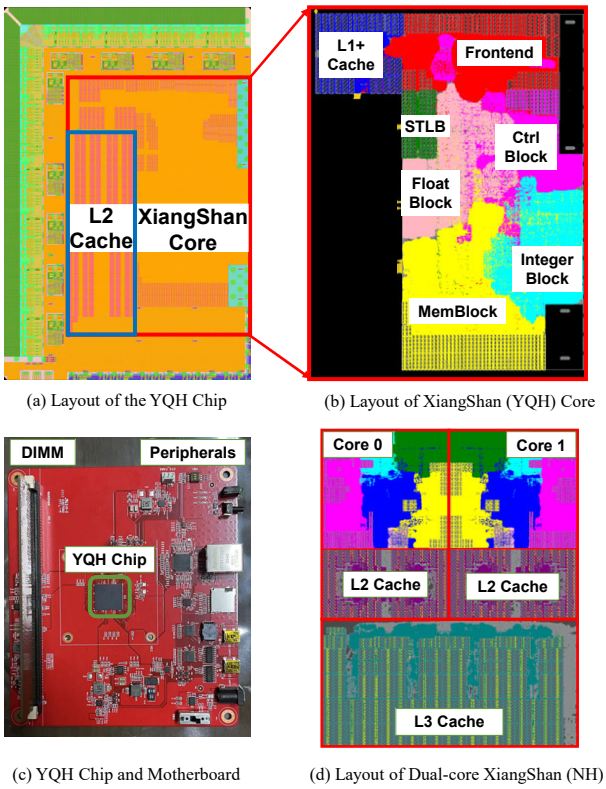
IV. BUILDING AN OPEN-SOURCE HIGH PERFORMANCE PROCESSOR

To demonstrate the effectiveness of MINJIE, we use it to build a high performance RISC-V processor named XIANGSHAN. We have developed two major generations of codenamed YQH and NH respectively since June 2020. This section presents the micro-architecture details of XIANGSHAN and our experience with MINJIE and XIANGSHAN.

A. Micro-architecture

XIANGSHAN is a superscalar out-of-order RISC-V processor with RV64GCBK ISA support. As the first step of agile development, XIANGSHAN is implemented using Chisel HDL [76] with a total line of code of ∼63K. The micro-architecture parameters of both generations of XIANGSHAN are shown in Table II. It is worth noting that most of the design parameters are configurable based on given constraints on timing, area and budget such as cache sizes, while Table II illustrates the parameters that we use for tape-out with a target frequency of 1.3GHz and 2GHz, for YQH and NH micro-architecture, respectively.

The micro-architecture diagram of NH, the second generation of XIANGSHAN, is shown in Figure 10. It can be seen that in the frontend, it features a decoupled style of branch prediction unit (BPU) and instruction fetch unit (IFU), where BPU runs ahead and provides sufficient instruction fetch requests. XIANGSHAN uses a 4-table 16K-entry TAGE-SC branch predictor with an indirect-jump predictor. In the decode stage, macro-op fusion is utilized such that certain consecutive arithmetic instructions can be fused into a single micro-operation, in order to reduce the execution latency and increase the effective size of various buffers such as re-order buffer and issue queue.



(a) Layout of the YQH Chip

(b) Layout of XiangShan (YQH) Core

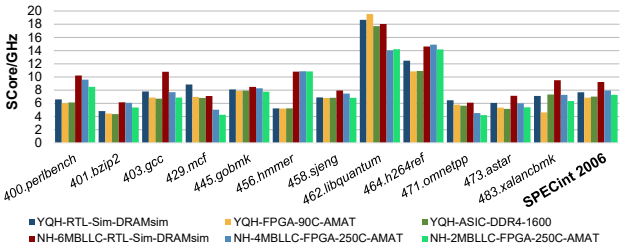
(c) YQH Chip and Motherboard

(d) Layout of Dual-core XiangShan (NH)

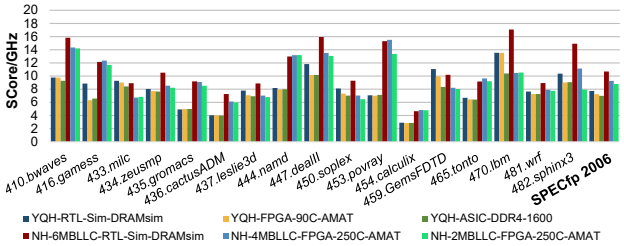
Figure 11. Layout of XIANGSHAN and the motherboard for YQH.

Table III
PHYSICAL IMPLEMENTATION DETAILS OF YQH

Die Size	8.6 mm^2
Std Cell Num/Area	5053679, 4.27 mm^2
Mem Num/Area	261, 1.7 mm^2
Density	66%
Cell	ULVT 1.04%, LVT 19.32%, SVT 25.19%, HVT 53.67%
Power	5W
Frequency	1.3 GHz, TT85°



(a) SPECint 2006 Scores



(b) SPECfp 2006 Scores

Figure 12. Measured and estimated SPECint 2006 and SPECfp 2006 scores of the XIANGSHAN processor. It is worth noting that NH-FPGAs have smaller LLCs than those on the simulation due to limited BRAM resources.

XIANGSHAN implements separated general-purpose register file and floating-point register file. Move elimination is enabled by a reference counting mechanism for the integer physical registers. The reservation stations are distributed and grouped as 32-entry or 16-entry smaller ones which issue two or one instruction to the execution units in every clock cycle. Floating-point multiply-add (FMA) instructions are executed in the cascade FMA units [77] with a five-cycle latency. FMA instructions are allowed for early execution when the multiplication operands are ready, and being issued again when the third operand is ready.

In the memory subsystem, the two load pipelines are bank-interleaved. Stores are decoupled into data and address micro-operations. The L1 DTLB has 128 direct-mapped entries and 8 fully associative entries to ensure a larger capacity without negatively impacting timing. XIANGSHAN has a 128KB virtually-indexed physically-tagged (VIPT) L1 instruction cache and a VIPT 128KB L1 data cache. We devise a hardware-based solution to address the aliasing problem in large VIPT caches. Both the L2 cache and L3 cache are non-inclusive.

As the first version of XIANGSHAN, YQH was taped-out on 28-nm CMOS technology in July 2021 with an operating frequency at 1.3GHz. Table III lists the physical implementation statistics for the CPU core. The layout of YQH chip and core are depicted in Figure 11.

As the second generation of XIANGSHAN, NH is a dual-core implementation that will be taped-out at a target frequency of 2GHz on 14-nm CMOS technology in Q4 2022. The layout of NH is shown in Figure 11(d).

B. Performance

Figure 12 shows the performance evaluation results of the XIANGSHAN processor with the SPEC CPU2006 benchmark suite. It is compiled with an `-O2` optimization flag with an ISA of RV64GC for YQH and RV64GCB for NH. We use the widely accepted SPEC/GHz metric [25] [78] [79], which is proportional to IPC [6] [80], for the quantitative evaluation.

The YQH chip, as shown in Figure 11(c), was successfully brought up in February 2022. Results (YQH-ASIC-DDR4-1600 in Figure 12) demonstrate that XIANGSHAN achieves 7.03 on SPECint 2006 and 7.00 on SPECfp 2006 at 1GHz.

Besides, we evaluate the performance of XIANGSHAN on FPGA. With a fixed and average memory access time (AMAT) of 90 clock cycles, YQH achieves normalized 6.87/GHz on SPECint 2006 and 7.23/GHz on SPECfp 2006.

With a 4MB last-level cache (LLC) and an AMAT of 250 cycles, NH achieves normalized 7.94/GHz on SPECint 2006 and 9.27/GHz on SPECfp 2006. To demonstrate how the LLC size affects the performance and validate the 6MB LLC statistics from simulation, we further evaluate NH with a 2MB LLC. As shown in Figure 12, compared with NH-2MBLLC-FPGA-250C-AMAT, NH-4MB-FPGA-250C-AMAT achieves 8.9% and 5.4% of performance increase on SPECint and SPECfp respectively.

As discussed in Section II-E, DDR on FPGA runs at 1.6GHz while the CPU runs at 50MHz, resulting in an one-CPU-cycle DDR access latency. Though we manually add 250 padding cycles to memory requests, the access latency and bandwidth still do not match the real chips. On the FPGA prototyping platform, most read and write requests have a longer latency, while the memory bandwidth is unlimited. Due to program characteristics and preferences of memory latency, SPECint 2006 evaluation on FPGA is conservative.

In addition to hardware testing, we can also evaluate the performance using software with the support of MINJIE. Using the workflow as introduced in Section III-D3, we evaluate the performance of XIANGSHAN under RTL-simulation environment with DDR4-2400. YQH reaches 7.67/GHz SPEC score at 1.3GHz, while NH achieves 10.06/GHz SPEC score at 2GHz. Compared with results from FPGAs and the real chips, the proposed evaluation methodology using MINJIE comes with a deviation of 5%∼10%. This is acceptable since the memory configurations are inconsistent.

To the best of our knowledge, NH achieves the highest performance of open-source RISC-V processors.

C. Debug with MINJIE

In this section, we demonstrate the effectiveness of MINJIE using a bug during the regression testing of XIANGSHAN. Though we have performed comprehensive unit testing for L2 cache, this bug still escapes to the system-level testing.

This complicated bug is exposed when running the Redis benchmark [81] on dual-core XIANGSHAN for over 168 hours after 3B simulated cycles. DiffTest reports a data mismatch between DUT and the Global Memory using the diff-rule described in Section III-B2b. After that, the second to last snapshot produced by LightSSS (Section III-C3) is activated to re-run the RTL-simulation. It takes only 3 minutes to simulate the last 30.8K cycles with waveform enabled.

The debugging stage starts with the ArchDB mentioned in Section III-B3 that records transaction information in the multi-level caches. We find that an Acquire [82] request from L2 cache to L3 is overlapped with a Probe transaction from L3 to L2 in the same cache block. Though L2 acquires the correct data from L3, later it grants the wrong data upward to L1, indicating a bug in the arbitration logic. Further investigation into the waveform confirms that L2 MSHR does not handle the overlapping correctly when Probe and GrantData from L3 cache arrive at a specific time interval.

This case demonstrates how MINJIE benefits the verification and debugging of a complicated processor. With DiffTest for both single-core and multi-core scenarios, bugs can be exposed when the processor runs real programs. LightSSS greatly speeds up the bug reproducing, without which it may take extra 16∼32 hours to dump the snapshots (with either LiveSim or SSS). ArchDB and other tools further increase the efficiency of hardware debugging. It is worth noting DiffTest on FPGA may drastically improve the testing throughput, which is an important future work of MINJIE.

D. Feature Exploration

In this section, we demonstrate the capabilities of MINJIE and XIANGSHAN with a case study, and show that they can be utilized as architecture innovation platforms for agile hardware development and high-performance processor research. On this platform, we implement and evaluate a micro-architecture technique that is proposed in 2018 and aims to improve the performance of out-of-order processors. This case study aims to demonstrate that innovative ideas can be easily realized and evaluated on XIANGSHAN, with MINJIE significantly speeding up the development process.

1) Experimental setup: We use the Chisel language and reuse basic utilities in the XIANGSHAN codebase to implement the ideas. Our implementation may differ from what the original paper presents for two reasons. One reason is that the initial implementation is based on an architectural simulator, which contains far fewer details than a real RTL-level processor implementation. The other reason is that we implement the idea mainly for a proof-of-concept and do not intend to reproduce the same results as the original work. To evaluate the idea, we select representative SPEC CPU2006 program checkpoints and simulate the RTL-code generated by the implementations on XIANGSHAN using Verilator. The evaluation is done on a 128-core x86 server.

2) Prioritizing unconfident branch slices: Modern processors use branch prediction techniques to improve performance, by either improving the prediction accuracy or reducing the misprediction penalty. Meanwhile, OoO processors have a large instruction window where ready instructions can be scheduled regardless of the program order. Different instructions have different impacts on the overall performance, i.e., the criticality.

Various issue strategies have been proposed to issue critical instructions first. Hideki Ando proposes the Prioritizing Unconfident Branch Slices (PUBS) [83], targeting reducing the branch misprediction penalty via a prioritized issue strategy. The insight is to prioritize the issue of unconfident branch instructions and the producer instructions for their operands. Unconfident branch instructions are the instructions with lower branch prediction accuracy. The producer instructions affect when the branch instructions' operands are ready. If producer instructions can be issued as soon as possible, the unconfident branches can be resolved earlier.

Figure 13. The implementation process of PUBS on XIANGSHAN.

To track the unconfident branch instructions and issue them with higher priority, PUBS contains four basic components, including a confidence estimation table `ConfTable`, a branch slice table `BrSliceTable`, a define table `DefTable` and a prioritized issue policy `PriorityIssue`. Every instruction looks up the `BrSliceTable` for its correlated branch instruction and checks the prediction confidence. If the confidence is low, this instruction is assigned with a high priority. PUBS is originally implemented and evaluated on the SimpleScalar simulator. Reported by the paper, PUBS improves the performance by 7.8% for SPEC CPU2006 programs whose branch mispredictions per kilo instructions (MPKI) exceeds 3.0.

Using MINJIE platform, we implement PUBS on XIANGSHAN by adopting the agile development methodology. We decompose the task into four features, which can be implemented iteratively, as shown in Figure 13. It takes less than 200 minutes to implement the PUBS on XIANGSHAN with approximately 300 lines of modified Chisel code.

To evaluate the performance of PUBS, we select representative checkpoints from SPEC CPU2006 and run them on XIANGSHAN with PUBS. Our first attempt is on ten checkpoints of *sjeng*, which is identified as the program with the highest speedup in the PUBS paper. We simulate 20M instructions for warmup and the following 20M instructions for performance profiling. AGE is used as the baseline in this experiment; that is, the oldest instruction is scheduled for issue with the highest priority.

As shown in Figure 14, we do not observe any visible performance deviation for PUBS on *sjeng*, though the PUBS paper reports a 6.5% IPC increase over baseline[1]. However, this result is to some extent acceptable because we are using the default configurations of XIANGSHAN, which are quite different from the processor configurations used in the PUBS paper. For example, XIANGSHAN adopts a larger instruction window and a wider issue width, reducing the performance impacts of the issue policy.

To confirm our conjecture that the larger issue width on XIANGSHAN reduces the speedup, we look into the detailed performance counters obtained from simulation. XIANGSHAN implements a distributed issue queue, each of which allows two ready instructions to issue at one clock cycle. Figure 15 shows the time distribution of the number of instructions that can be scheduled for issue. For

[1]The PUBS paper does not explicitly report the exact percentage of increase. We use the average increase from the PUBS paper considering that *sjeng* has the max increase of IPC.

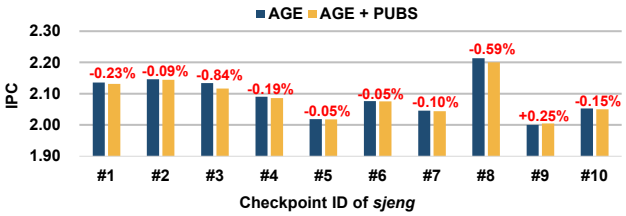


Figure 14. IPC difference when PUBS is enabled.

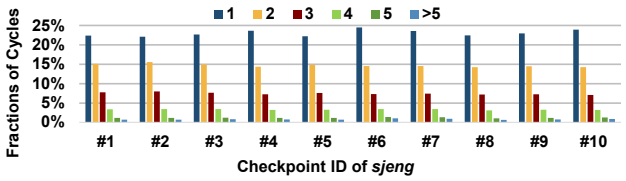


Figure 15. Fraction of cycles with the number of ready instructions when PUBS is disabled.

ten checkpoints of *sjeng*, the case when more than two instructions can be issued occurs in 12.8% of the total cycles, with an average number of blocking instructions at 0.215. In the meantime, an average of 5.9% of total instructions are labeled of high priority. Therefore, the average number of blocking high-priority instructions is $0.215 \times 5.9\%$. Besides, the expected blocking cycles of an instruction is at most 0.168 assuming it is blocked at every cycle when there are more than two ready instructions. Thus, approximately less than 1.3% instructions can be scheduled earlier for one clock cycle with PUBS. This small fraction of instructions and fewer branch instructions would not bring significant IPC improvement, matching the evaluation results in Figure 14.

We also evaluate PUBS with other checkpoints whose MPKI exceeds 3 on more configurations of XIANGSHAN, but does not observe visible performance improvements.

MINJIE enable developers to use their familiar workflow as the architectural simulators, while being more accurate by using cycle-exact RTL-simulators for a high performance micro-architecture of XIANGSHAN. First, Chisel significantly improves code density and programming efficiency. PUBS takes only 300 lines of Chisel code, including the license header, empty lines, and definitions for performance counters. Second, DiffTest and LightSSS enable rapid functional verification of the new features without breaking other parts. For example, a potential issue in the modified issue policy triggers an assertion caught by the DiffTest framework. The simulation is then aborted but LightSSS restores the recent snapshot and outputs debugging information. The automated acquisition of information significantly speeds up the debugging process. Third, representative checkpoints from SPEC CPU2006 allow us to evaluate PUBS within hours, and detailed performance counters can be acquired from the simulation.

V. Related Work

A. Hardware Description Languages

Most recently proposed hardware description languages provide high-level design paradigms [84] [85]. Bluespec SystemVerilog [86] proposes the guarded atomic actions that allow the compiler to pick a scheduler for the hardware manner and emit the optimized hardware design. Python-based HDLs [87] [88] [89] provide parameterized hardware structure descriptions in Python. Scala-based hardware construction languages [76] [90] provide flexible object-oriented and functional programming features. Chisel adopts the FIRRTL [91] [92] intermediate representation to optimize the circuits with customized circuit transforms. Both traditional and emerging languages can be adopted on MinJie.

B. Simulation and Hardware Verification

Hardware verification adopts both static and dynamic approaches. Static and formal methods are able to prove the theoretical correctness but have the limitation of state space explosion [93], thus not being practical for large-scale circuits. Dynamic RTL-simulation based verification is the most commonly used tool, which can be further categorized into correctness checking and generation of test cases.

Various frameworks have been proposed to increase the efficiency of RTL-simulation and enhance the ability of correctness comparison. FireSim [31] is an FPGA-accelerated simulation technique running on public cloud to improve usability and elasticity at a lower cost. On top of FireSim, FirePerf [94] provides a set of system-level performance profiling tools. Synthesizable assertion and print statements [35] [38] have been implemented on FPGAs to address the debuggability issues. Besides the FPGA approaches, software based simulation is important for hardware verification and some works have addressed its efficiency issue [95] [96].

Test generation schemes have been proposed for RISC-V [97] [98] [99] [100] [101] and other processors, and they work in parallel with the previously introduced frameworks. Besides, fuzzing [102] [103] [104] and machine learning [105] based techniques have been adopted to effectively generate higher-quality test cases to achieve better coverage. For now, MinJie and XiangShan use the existing open-source test generation frameworks.

C. Verification with New HDLs and RISC-V

Native verification frameworks for high-level HDLs like Chisel [106] [107] [108] have been proposed to improve the development agility. Mamba [109] proposes a just-in-time compiler to directly simulate the high-level PyMTL code that reaches a comparable performance of commercial HDL simulators. These frameworks aim at providing high-level interfaces for general-purpose verification without targeting any specific design [110], which is orthogonal to MinJie.

RiVer [111] provides a python based verification environment and supports running tests and comparing results with a valid reference model. However, RiVer does not provide any implementation of the test generator and the reference model. Dromajo [27] is the state-of-the-art co-simulation framework for RISC-V processors and provides Logic Fuzzer to achieve higher coverage. However, it relies on deterministic architectural states during co-simulation and manually sends the external interrupt to REFs, which is the only identified source of divergence. By contrast, we make a key observation that divergence is the norm of co-simulation and propose the diff-rules to address the general non-deterministic issues. Compared with Dromajo, MinJie identifies more sources of non-determinism and supports more complicated co-simulation scenarios such as multi-core and cache hierarchies.

D. Agile Development Platforms

Integrated and agile development platforms [8] [11] [12] have been proposed for various purposes. Chipyard [7] is an integrated SoC design, simulation framework including many RTL generators such as BOOM [6] [112] and Hwacha [113] and verification tools [114] [115]. Existing platforms and tools have always been valuable for developers and MinJie adopts some of the existing development tools. This paper promotes the research on agile development further on the verification of complicated processors. The methodology and tools proposed by MinJie are complementary to the existing works in the community.

VI. Conclusion

In this paper, we propose MinJie, an agile development platform for high performance chip designs, and XiangShan, a high performance RISC-V processor. MinJie incorporates a broad set of tools for agile development workflow. By adopting the agile approaches, XiangShan achieves industry-competitive performance with reduced development cycles and limited efforts. We are working towards the third generation of XiangShan, code-named KMH, which targets SPEC CPU2006 15/GHz at 3GHz. Both MinJie and XiangShan are open-sourced.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable feedback and comments. We also give special thanks to our group members, Yifei He, Xuan Hu, Ziyue Zhang, and Hao Zhen, for their help on the work. This work was supported in part by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC05030200), the National Natural Science Foundation of China (Grant No. 62090020, 62172388, 62072433), Youth Innovation Promotion Association of Chinese Academy of Sciences (2020105), ICT Innovation Grant E261100, and the Fundamental Research Funds for the Central Universities (Grant No. DUT21RC(3)102, DUT21LAB302).

APPENDIX

A. Abstract

MINJIE is an open-source platform supporting agile processor development. XIANGSHAN is a high performance RISC-V processor developed with MINJIE. We demonstrate the workflow for functional validation and performance evaluation of XIANGSHAN with MINJIE, including tools like DiffTest, LightSSS, NEMU, and RISC-V checkpoints.

B. Artifact check-list (meta-information)

- **Hardware:** x86-64 Ubuntu servers
- **Experiments:** (1) DiffTest on dual-core XIANGSHAN; (2) debugging with LightSSS and evaluation of performance overhead; (3) XIANGSHAN performance estimation with SPEC CPU2006 checkpoints by RTL simulation; (4) evaluation of PUBS on XIANGSHAN
- **Code license:** Mulan Permissive Software License, Version 2
- **Archived:** 10.5281/zenodo.7030506
- **Online:** Latest version maintained on GitHub

C. Description

XIANGSHAN and MINJIE are open-sourced at GitHub. An archived version of the artifacts is provided at Zenode, which is used as the example in this appendix.

Due to the proprietary issues, we cannot publicly provide the full data set, such as the executable RISC-V checkpoints generated with SPEC CPU2006 and source code for the FPGA prototype.

D. Installation

1) Setup toolchain: Some packages are requried to build XIANGSHAN and use MINJIE.

```
$ sudo -s ./A.5.1-difftest/setup-tools.sh
$ git clone \
  https://github.com/OpenXiangShan/riscv-gnu-toolchain
$ # follow the README to install dependencies
$ cd riscv-gnu-toolchain && mkdir build && cd build
$ ../configure --prefix=PATH_TO_INSTALL \
    --with-arch=rv64gc_zba_zbb_zbc_zbs
$ make linux -j16 && export PATH=$RISCV/bin:$PATH
```

E. Experiment workflow

1) DiffTest (Section III-B): We simulate dual-core XIANGSHAN with DiffTest, which checks the correctness on-the-fly and reports any mismatch between the REF and XIANGSHAN. We intentionally inject a fault into XIANGSHAN, which will trigger a bug caught by the "cache hierarchy and multi-core scenarios" diff-rules (Section III-B2b).

```
$ cd A.5.1-difftest && source ./env.sh
$ bash build.sh && cd XiangShan
$ ./build/emu -i ../linux-4.18-smp-hello/bbl.bin     \
--diff ./ready-to-run/riscv64-nemu-interpreter-dual-so\
2> /dev/null
```

2) LightSSS (Section III-C): First, we verify the functional correctness of LightSSS, a simulation snapshot mechanism. We simulate the dual-core XIANGSHAN with the same workload as in the previous experiment. When the simulation aborts, LightSSS will restore a snapshot (the child process created by `fork`) and re-run the last several seconds of the simulation to generate waveform and debugging logs.

```
$ ./build/emu -i ../linux-4.18-smp-hello/bbl.bin     \
--diff ./ready-to-run/riscv64-nemu-interpreter-dual-so\
--enable-fork 2> simulator_stderr.txt
```

Second, we evaluate the performance overhead of LightSSS to reproduce the results in Figure 6. We will evaluate LightSSS with two configurations (single-core and dual-core).

```
$ cd A.5.2-lightsss && source ./env.sh
$ bash build_single_core.sh
$ bash test_single_core.sh | tee single_core.log
$ bash build_dual_core.sh
$ bash test_dual_core.sh | tee dual_core.log
$ python3 report_lightsss.py
```

3) NEMU (Section III-D): We evaluate the performance of NEMU using the *test* input of SPEC CPU2006. We compare its performance with Spike, QEMU-TCI, and Dromajo, whose versions are shown at the top of the scripts.

```
$ cd A.5.6-NEMU && source ./env.sh
$ make -C NEMU/tools/regression run-riscv64-user
$ bash run_spike.sh
$ bash run_qemu_tci.sh
$ bash run_dromajo.sh
```

Second, we evaluate the performance of RISC-V checkpoint generation by NEMUs.

```
$ cd A.5.3-performance && source ./env.sh
$ bash build_and_run_iss.sh
```

Third, we verify that NEMU is able to generate RISC-V checkpoints using CoreMark-PRO as an example.

```
$ bash generate_checkpoints.sh
```

We also confirm that XIANGSHAN is able to restore and run the generated RISC-V checkpoint.

```
$ bash build_xs_and_run_checkpoint.sh
```

4) XIANGSHAN (Section IV-B): Performance of the XIANGSHAN processor in Figure 12 is evaluated on three platforms, i.e., RTL-simulation, FPGAs, and chips.

Due to the specialized hardware and proprietary software requirements, we cannot publicly share the data set now, including the checkpoints used for performance evaluation. However, we encourage to build SPEC CPU2006 from source code and generate checkpoints with NEMU and SimPoint.

First, we use the pre-generated checkpoints for SPEC CPU2006 to estimate the performance of YQH and NH.

```
$ cd A.5.4-XS && source ./env.sh
$ bash build_xs.sh
$ # all SPEC CPU2006 checkpoints
$ bash run_xs_checkpoints.sh
$ # namd checkpoints only
$ bash run_xs_checkpoints_namd.sh
```

To estimate performance of NH, run the above steps but replace the directory with `A.5.4-XS-NH`.

Second, we use the FPGA platform to evaluate the performance of YQH and NH. Use `A.5.4-XS-FPGA-YQH`, `A.5.4-XS-FPGA-NH-2MB`, and `A.5.4-XS-FPGA-NH-4MB` directories to reproduce the performance data for YQH-FPGA-90C-AMAT, NH-2MBLLC-FPGA-250C-AMAT, and NH-4MBLLC-FPGA-250C-AMAT configurations.

```
$ cd DIR && source ./env.sh
$ bash build_xs.sh
$ bash run_fpga.sh
```

Third, we evaluate the performance of YQH chip using SPEC CPU2006 benchmarks.

```
$ ssh root@172.28.2.145 # Private network only
$ cd spec2006-lite # SPEC CPU2006 wrapper
$ make run-all-ref | tee ./logs/run-all-ref.log
$ python3 scripts/report.py
```

5) PUBS (Section IV-D): We use `sjeng` checkpoints from SPEC CPU2006 to evaluate PUBS, as shown in Figure 14 and 15. There are three configurations, i.e., (1) baseline, (2) AGE, and (3) AGE+PUBS on three directories (`A.5.5-PUBS-BASELINE`, `A.5.5-PUBS-AGE-ONLY`, `A.5.5-PUBS`). They are evaluated one by one.

```
$ cd DIR && source ./env.sh
$ bash build_xs.sh
$ bash run_xs_checkpoints.sh
$ bash report_xs_checkpoints.sh
```

F. Evaluation and expected results

1) DiffTest (Section III-B): DiffTest should abort with "ICache Refill test failed".

```
ICache Refill test failed!
addr: 8030c1c0
......
Seed=0 Guest cycle spent: 1,081,292
Host time spent: 664,997ms
```

2) LightSSS (Section III-C): DiffTest should abort and LightSSS will generate the waveform (`build/*.vcd`) and logs (`simulator_stderr.txt`).

```
ICache Refill test failed!
[FORK_INFO pid(x)] the oldest checkpoint start ...
[FORK_INFO pid(x)] dump wave to name.vcd...
......
Seed=0 Guest cycle spent: 1,081,292
Host time spent: 717,054ms
```

For the performance overhead evaluation, the simulation time should remain stable across different snapshot intervals.

3) NEMU (Section III-D): First, we verify that NEMU interprets the *test* input of `bzip2` in SPEC CPU2006 at around 1200 MIPS as in Figure 8. Performance of other instruction set simulators are reported by the wrapper scripts. Results demonstrate NEMU runs much faster than the others.

```
host time spent = 17,596,041 us
total guest instructions = 22,380,683,540
simulation frequency = 1,271,915,855 instr/s
```

Second, performance of NEMU used for checkpoints is reported after 10B instructions. Expected to be large than 300M instr/s. In total, 8 RISC-V checkpoints should be generated at `output_top/test/coremarkpro` and correctly run on XIANGSHAN.

```
host time spent = 31,238,591 us
total guest instructions = 10,000,000,005
simulation frequency = 320,116,870 instr/s
```

4) XIANGSHAN (Section IV-B): First, a script is available to print the estimated scores for the RTL-simulation with RISC-V checkpoints. Expected to be ∼7/GHz on YQH and ∼10/GHz on NH.

```
$ bash report_xs_checkpoints_30percent.sh
```

Second, run-time of the SPEC benchmarks are shown in the serial port. The total SPEC scores are computed as the GEOMEAN of the benchmarks. Expected to be ∼7/GHz on YQH and ∼10/GHz on NH.

Third, we provide a Python script to report SPEC scores on the YQH chip. Expected to be ∼7/GHz.

```
$ python3 scripts/report.py
```

5) PUBS (Section IV-D): Performance counters are generated at `sjeng_*.csv`. Figure 14 is generated by the `global.IPC` line. Figure 15 is generated by the `global.num_ready_frac_*` lines.

G. Notes

Both XIANGSHAN and MINJIE are developed in the open-source community. For now, due to the proprietary issues, we cannot publicly share the full data set. However, we are working on providing a publicly accessible version and will share it on GitHub once we make it.

References

[1] Y. Lee, A. Waterman, H. Cook, B. Zimmer, B. Keller, A. Puggelli, J. Kwak, R. Jevtic, S. Bailey, M. Blagojevic *et al.*, "An Agile Approach to Building RISC-V Microprocessors," *IEEE Micro*, vol. 36, no. 2, pp. 8–20, 2016.

[2] Y. Bao and T. E. Carlson, "Agile and Open-Source Hardware," *IEEE Micro*, vol. 40, no. 4, pp. 6–9, 2020.

[3] J. L. Hennessy and D. A. Patterson, "A New Golden Age for Computer Architecture," *Commun. ACM*, vol. 62, no. 2, p. 48–60, jan 2019.

[4] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The Rocket Chip Generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[5] C. Celio, P.-F. Chiu, K. Asanović, B. Nikolić, and D. Patterson, "BROOM: An Open-Source Out-of-Order Processor With Resilient Low-Voltage Operation in 28-nm CMOS," *IEEE Micro*, vol. 39, no. 2, pp. 52–60, 2019.

[6] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanović, "SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, 2020.

[7] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[8] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile SoC development with open ESP," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[9] D. Petrisko, F. Gilani, M. Wyse, D. C. Jung, S. Davidson, P. Gao, C. Zhao, Z. Azad, S. Canakci, B. Veluri *et al.*, "BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 93–102, 2020.

[10] D. Giri, K.-L. Chiu, G. Eichler, P. Mantovani, and L. P. Carloni, "Accelerator Integration for Open-Source SoC Design," *IEEE Micro*, 2021.

[11] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang *et al.*, "OpenPiton: An Open Source Manycore Research Framework," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 217–232, 2016.

[12] P. N. Whatmough, M. Donato, G. G. Ko, S. K. Lee, D. Brooks, and G.-Y. Wei, "CHIPKIT: An Agile, Reusable Open-Source Framework for Rapid Test Chip Development," *IEEE Micro*, vol. 40, no. 4, pp. 32–40, 2020.

[13] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 367–374.

[14] H. Wang, Z. Zhang, Y. Jin, L. Zhang, and K. Wang, "Nutshell: A Linux-Compatible RISC-V Processor Designed by Undergraduates," in *RISC-V Global Forum 2020*. RISC-V International, 2020.

[15] S. Zhang, A. Wright, T. Bourgeat, and A. Arvind, "Composable Building Blocks to Open up Processor Design," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 68–81.

[16] IEEE, "IEEE Standard for Universal Verification Methodology Language Reference Manual," *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017)*, pp. 1–458, 2020.

[17] W. Chen, L.-C. Wang, J. Bhadra, and M. Abadir, "Simulation Knowledge Extraction and Reuse in Constrained Random Processor Verification," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[18] V. Herdt, D. Große, and R. Drechsler, "Towards Specification and Testing of RISC-V ISA Compliance," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 995–998.

[19] K. Cheang, C. Rasmussen, D. Lee, D. W. Kohlbrenner, K. Asanović, and S. A. Seshia, "Verifying RISC-V Physical Memory Protection," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) Workshop on Secure RISC-V Architecture Design*, 2020.

[20] T. Bourgeat, I. Clester, A. Erbsen, S. Gruetter, A. Wright, and A. Chlipala, "A Multipurpose Formal RISC-V Specification," *CoRR*, vol. abs/2104.00762, 2021. [Online]. Available: https://arxiv.org/abs/2104.00762

[21] H. D. Foster, "Why the Design Productivity Gap Never Happened," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 581–584.

[22] H. Foster, "Trends in Functional Verification: A 2014 Industry Study," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[23] M. H. Safieddine, F. A. Zaraket, R. Kanj, A. El-Zein, and W. Roesner, "Verification at RTL Using Separation of Design Concerns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1529–1542, 2019.

[24] C. Yuan, X. Gao, Y. Chen, and Y. Bao, "Teaching Undergraduates to Build Real Computer Systems," *Communications of the ACM*, vol. 64, no. 11, p. 48–49, oct 2021. [Online]. Available: https://doi.org/10.1145/3481606

[25] C. Chen, X. Xiang, C. Liu, Y. Shang, R. Guo, D. Liu, Y. Lu, Z. Hao, J. Luo, Z. Chen, C. Li, Y. Pu, J. Meng, X. Yan, Y. Xie, and X. Qi, "Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline out-of-Order 64-Bit High Performance RISC-V Processor with Vector Extension," in *Proceedings*

of the ACM/IEEE 47th Annual International Symposium on Computer Architecture, ser. ISCA '20. IEEE Press, 2020, p. 52–64.

[26] M. Katrowitz and L. M. Noack, "I'm Done Simulating; Now What? Verification Coverage Analysis and Correctness Checking of the DEC Chip 21164 Alpha Microprocessor," in *Proceedings of the 33rd Annual Design Automation Conference*, ser. DAC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 325–330.

[27] N. Kabylkas, T. Thorn, S. Srinath, P. Xekalakis, and J. Renau, "Effective Processor Verification with Logic Fuzzer Enhanced Co-simulation," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 667–678.

[28] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle Problem in Software Testing: A Survey," *IEEE Trans. Software Eng.*, vol. 41, no. 5, pp. 507–525, 2015.

[29] C. Jard and G. von Bochmann, "An Approach to Testing Specifications," in *Proceedings of the symposium on High-level debugging, SIGSOFT 1983, Pacific Grove, California, USA, March 20-23, 1983*, R. E. Fairley and M. S. Johnson, Eds. ACM, 1983, pp. 53–59.

[30] A. Fedotov and J. Schmaltz, "Automatic Generation of Hardware Checkers from Formal Micro-architectural Specifications," in *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, J. Madsen and A. K. Coskun, Eds. IEEE, 2018, pp. 1568–1573.

[31] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra *et al.*, "FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 29–42.

[32] Z. Tan, A. Waterman, H. Cook, S. Bird, K. Asanović, and D. Patterson, "A Case for FAME: FPGA Architecture Model Execution," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 290–301.

[33] A. Magyar, D. Biancolin, J. Koenig, S. Seshia, J. Bachrach, and K. Asanović, "Golden Gate: Bridging The Resource-Efficiency Gap Between ASICs and FPGA Prototypes," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[34] Z. Tan, Z. Qian, X. Chen, K. Asanovic, and D. Patterson, "DIABLO: A Warehouse-Scale Computer Network Simulator using FPGAs," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 207–221, 2015.

[35] D. Kim, C. Celio, S. Karandikar, D. Biancolin, J. Bachrach, and K. Asanović, "DESSERT: Debugging RTL Effectively with State Snapshotting for Error Replays across Trillions of Cycles," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 76–764.

[36] K. Camera and R. W. Brodersen, "An Integrated Debugging Environment for FPGA Computing Platforms," in *2008 International Conference on Field Programmable Logic and Applications*. IEEE, 2008, pp. 311–316.

[37] Y. S. Iskander, C. D. Patterson, and S. D. Craven, "Improved Abstractions and Turnaround Time for FPGA Design Validation and Debug," in *2011 International Conference on Field Programmable Logic and Applications*. IEEE, 2011, pp. 518–523.

[38] J. Ma, G. Zuo, K. Loughlin, H. Zhang, A. Quinn, and B. Kasikci, "Debugging in the Brave New World of Reconfigurable Hardware," in *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, B. Falsafi, M. Ferdman, S. Lu, and T. F. Wenisch, Eds. ACM, 2022, pp. 946–962.

[39] E. Schkufza, M. Wei, and C. J. Rossbach, "Just-In-Time Compilation for Verilog: A New Technique for Improving the FPGA Programming Experience," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 271–286.

[40] W. N. N. Hung and R. Sun, "Challenges in Large FPGA-based Logic Emulation Systems," in *Proceedings of the 2018 International Symposium on Physical Design, ISPD 2018, Monterey, CA, USA, March 25-28, 2018*, C. Chu and I. Bustany, Eds. ACM, 2018, pp. 26–33.

[41] A. Agnesina, S. K. Lim, E. Lepercq, and J. E. D. Cid, "Improving FPGA-Based Logic Emulation Systems through Machine Learning," *ACM Trans. Design Autom. Electr. Syst.*, vol. 25, no. 5, pp. 46:1–46:20, 2020.

[42] D. Biancolin, A. Magyar, S. Karandikar, A. Amid, B. Nikolic, J. Bachrach, and K. Asanovic, "Accessible, FPGA Resource-Optimized Simulation of Multiclock Systems in FireSim," *IEEE Micro*, vol. 41, no. 4, pp. 58–66, 2021.

[43] Cadence, "Cadence® Palladium® Emulation Platforms." [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html

[44] Siemens Digital Industries Software, "Veloce Hardware-assisted Verification System." [Online]. Available: https://eda.sw.siemens.com/en-US/ic/veloce/

[45] Synopsys, "Synopsys ZeBu® EP1." [Online]. Available: https://www.synopsys.com/verification/emulation.html

[46] W. Snyder, "Verilator." [Online]. Available: https://veripool.org/verilator/

[47] Synopsys, "VCS Functional Verification Solution." [Online]. Available: https://www.synopsys.com/verification/simulation/vcs.html

[48] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 Simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[49] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[50] A. Frumusanu, "The 2020 Mac Mini Unleashed: Putting Apple Silicon M1 To The Test," Nov 2020. [Online]. Available: https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested/4

[51] D. Biancolin, S. Karandikar, D. Kim, J. Koenig, A. Waterman, J. Bachrach, and K. Asanović, "FASED: FPGA-Accelerated Simulation and Evaluation of DRAM," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 330–339.

[52] S. Hung, "[patch 5.1 096/121] riscv: mm: Synchronize MMU after PTE change." [Online]. Available: https://lore.kernel.org/all/20190624092325.659180675@linuxfoundation.org/

[53] A. Waterman and K. Asanović, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213," RISC-V Foundation, Tech. Rep., 2019.

[54] C. Celio, D. P. Dabbelt, D. A. Patterson, and K. Asanović, "The Renewed Case for the Reduced Instruction Set Computer: Avoiding ISA Bloat with Macro-Op Fusion for RISC-V," *CoRR*, vol. abs/1607.02318, 2016. [Online]. Available: http://arxiv.org/abs/1607.02318

[55] H. Skinner, R. Trapani Possignolo, S.-H. Wang, and J. Renau, "LiveSim: A Fast Hot Reload Simulator for HDLs," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 126–135.

[56] Verilator, "Simulating (verilated-model runtime): Save/Restore." [Online]. Available: https://verilator.org/guide/latest/simulating.html#save-restore

[57] D. P. Bovet, M. Cassetti, and A. Oram, *Understanding the Linux Kernel*. USA: O'Reilly & Associates, Inc., 2000.

[58] criu.org, "CRIU – A project to implement checkpoint/restore functionality for Linux." [Online]. Available: https://github.com/checkpoint-restore/criu

[59] T. Hoefler and R. Belli, "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses When Reporting Performance Results," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, J. Kern and J. S. Vetter, Eds. ACM, 2015, pp. 73:1–73:12.

[60] GEM5, "Checkpoints." [Online]. Available: https://www.gem5.org/documentation/general_docs/checkpoints/

[61] GEM5, "gem5: Trace CPU Model." [Online]. Available: https://www.gem5.org/documentation/general_docs/cpu_models/TraceCPU

[62] J. R. Bell, "Threaded code," *Communications of the ACM*, vol. 16, no. 6, pp. 370–372, 1973.

[63] RISC-V International, "Spike, a RISC-V ISA Simulator." [Online]. Available: https://github.com/riscv-software-src/riscv-isa-sim

[64] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. USA: USENIX Association, 2005, p. 41.

[65] RISC-V International, "RISC-V Proxy Kernel and Boot Loader." [Online]. Available: https://github.com/riscv-software-src/riscv-pk

[66] C. Celio, "A Wrapper for the SPEC CPU2006 Benchmark Suite." [Online]. Available: https://github.com/ccelio/Speckle

[67] J. Hauser, "Berkeley SoftFloat." [Online]. Available: http://www.jhauser.us/arithmetic/SoftFloat.html

[68] E. Edgar and T. Newsome, "RISC-V Debug Support Version 1.0.0-STABLE," RISC-V Foundation, Tech. Rep., 2019.

[69] Embedded Microprocessor Benchmark Consortium, "Containing dozens of real-world and synthetic tests, CoreMark®-PRO (2015) is an industry-standard benchmark that measures the multi-processor performance of central processing units (CPU) and embedded microcrontrollers (MCU)." [Online]. Available: https://github.com/eembc/coremark-pro

[70] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 45–57, 2002.

[71] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ser. ISCA '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 84–97.

[72] T. Wenisch, R. Wunderlich, B. Falsafi, and J. Hoe, "Statistical Sampling of Microarchitecture Simulation," in *2006 IEEE International Symposium on Performance Analysis of Systems and Software*, 2006, pp. 2–12.

[73] T. Wenisch, R. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. Hoe, "SimFlex: Statistical Sampling of Computer System Simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, 2006.

[74] T. E. Carlson, W. Heirman, K. Van Craeynest, and L. Eeckhout, "BarrierPoint: Sampled simulation of multi-threaded applications," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 2–12.

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

[76] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing Hardware in a Scala Embedded Language," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 1212–1221.

[77] S. Galal and M. Horowitz, "Latency Sensitive FMA Design," in *2011 IEEE 20th Symposium on Computer Arithmetic*, 2011, pp. 129–138.

[78] SiFive, "SiFive Performance P550." [Online]. Available: https://www.sifive.com/cores/performance-p550

[79] SiFive, "SiFive Performance P650." [Online]. Available: https://www.sifive.com/cores/performance-p650

[80] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures," in *27th International Symposium on Computer Architecture (ISCA 2000), June 10-14, 2000, Vancouver, BC, Canada*, A. D. Berenbaum and J. S. Emer, Eds. IEEE Computer Society, 2000, pp. 248–259.

[81] Redis Ltd, "Redis Benchmark." [Online]. Available: https://redis.io/docs/reference/optimization/benchmarks/

[82] W. Terpstra, "TileLink: A Free And Open Source, High Performance Scalable Cache Coherent Fabric Designed for RISC-V," in *7th RISC-V Workshop*. RISC-V Foundation, 2017.

[83] H. Ando, "Performance Improvement by Prioritizing the Issue of the Instructions in Unconfident Branch Slices," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 82–94.

[84] Intel Corporation, "Rapid Open Hardware Development (ROHD) Framework." [Online]. Available: https://github.com/intel/rohd

[85] S. F. Hoover and A. Salman, "Top-Down Transaction-Level Design with TL-Verilog," *CoRR*, vol. abs/1811.01780, 2018. [Online]. Available: http://arxiv.org/abs/1811.01780

[86] R. Nikhil, "Bluespec System Verilog: Efficient, Correct RTL from High Level Specifications," in *Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE'04.* IEEE, 2004, pp. 69–70.

[87] P. Haglund, O. Mencer, W. Luk, and B. Tai, "Hardware Design with a Scripting Language," in *International Conference on Field Programmable Logic and Applications*. Springer, 2003, pp. 1040–1043.

[88] D. Lockhart, G. Zibrat, and C. Batten, "PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 280–292.

[89] D. Dangwal, G. Tzimpragos, and T. Sherwood, "Agile Hardware Development and Instrumentation With PyRTL," *IEEE Micro*, vol. 40, no. 4, pp. 76–84, 2020.

[90] SpinalHDL, "Spike, a RISC-V ISA Simulator." [Online]. Available: https://github.com/SpinalHDL/SpinalHDL

[91] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach, "Reusability is FIRRTL Ground: Hardware Construction Languages, Compiler Frameworks, and Transformations," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2017, pp. 209–216.

[92] P. S. Li, A. M. Izraelevitz, and J. Bachrach, "Specification for the FIRRTL Language," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-9, Feb 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html

[93] C. Kern and M. R. Greenstreet, "Formal Verification in Hardware Design: A Survey," *ACM Transactions on Design Automation of Electronic Systems*, vol. 4, no. 2, p. 123–193, apr 1999.

[94] S. Karandikar, A. Ou, A. Amid, H. Mao, R. Katz, B. Nikolić, and K. Asanović, "FirePerf: FPGA-Accelerated Full-System Hardware/Software Performance Profiling and Co-Design," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 715–731.

[95] S. Beamer, "A Case for Accelerating Software RTL Simulation," *IEEE Micro*, vol. 40, no. 4, pp. 112–119, 2020.

[96] S. Beamer and D. Donofrio, "Efficiently Exploiting Low Activity Factors to Accelerate RTL Simulation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[97] Google, "Random Instruction Generator for RISC-V Processor Verification." [Online]. Available: https://github.com/google/riscv-dv

[98] UC Berkeley Architecture Research, "RISC-V Torture Test." [Online]. Available: https://github.com/ucb-bar/riscv-torture

[99] OpenHW Group, "Instruction Set Generator Initially Contributed by Futurewei." [Online]. Available: https://github.com/openhwgroup/force-riscv

[100] RISC-V International, "RISC-V Architecture Test." [Online]. Available: https://github.com/riscv-non-isa/riscv-arch-test

[101] S. Flur and P. Sewell, "RISC-V Architecture Concurrency Model Litmus Tests." [Online]. Available: https://github.com/litmus-tests/litmus-tests-riscv

[102] T. Trippel, K. G. Shin, A. Chernyakhovsky, G. Kelly, D. Rizzo, and M. Hicks, "Fuzzing Hardware Like Software," *CoRR*, vol. abs/2102.02308, 2021. [Online]. Available: https://arxiv.org/abs/2102.02308

[103] J. Hur, S. Song, D. Kwon, E. Baek, J. Kim, and B. Lee, "DifuzzRTL: Differential Fuzz Testing to Find CPU Bugs," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1286–1303.

[104] K. Laeufer, J. Koenig, D. Kim, J. Bachrach, and K. Sen, "RFUZZ: coverage-directed fuzz testing of RTL on FPGAs," in *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2018, San Diego, CA, USA, November 05-08, 2018*, I. Bahar, Ed. ACM, 2018, p. 28.

[105] S. Vasudevan, W. Jiang, D. Bieber, R. Singh, H. Shojaei, R. Ho, and C. Sutton, "Learning Semantic Representations to Verify Hardware Designs," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 23 491–23 504.

[106] UC Berkeley Architecture Research, "The Official Testing Library for Chisel Circuits." [Online]. Available: https://github.com/ucb-bar/chisel-testers2

[107] Y.-C. Tsai, "Dynamic Verification Library for Chisel," Master's thesis, EECS Department, University of California, Berkeley, May 2021. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-132.html

[108] A. Dobis, T. Petersen, H. J. Damsgaard, K. J. H. Rasmussen, E. Tolotto, S. T. Andersen, R. Lin, and M. Schoeberl, "ChiselVerify: An Open-Source Hardware Verification Library for Chisel and Scala," in *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*. IEEE, 2021, pp. 1–7.

[109] S. Jiang, B. Ilbeyi, and C. Batten, "Mamba: Closing the Performance Gap in Productive Hardware Development Frameworks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[110] cocotb, "cocotb, a coroutine based cosimulation library for writing VHDL and Verilog testbenches in Python." [Online]. Available: https://github.com/cocotb/cocotb

[111] InCore Semiconductors and Tessolve Semiconductors, "RiVer Core." [Online]. Available: https://github.com/incoresemi/river_core

[112] C. Celio, P.-F. Chiu, B. Nikolic, D. A. Patterson, and K. Asanovic, "BOOMv2: An Open-source Out-of-order RISC-V Core," in *First Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2017.

[113] Y. Lee, C. Schmidt, A. Ou, A. Waterman, and K. Asanović, "The Hwacha Vector-Fetch Architecture Manual, Version 3.8.1," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-262, Dec 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-262.html

[114] N. Pemberton and A. Amid, "FireMarshal: Making HW/SW Co-Design Reproducible and Reliable," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 299–309.

[115] E. Wang, C. Schmidt, A. Izraelevitz, J. Wright, B. Nikolić, E. Alon, and J. Bachrach, "A Methodology for Reusable Physical Design," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 243–249.