Enabling Realistic Virtualized Cloud Workload Evaluation in RISC-V

Nikos Karystinos
n.karystinos@di.uoa.gr
University of Athens
Greece

George-Marios Fragkoulis
gm.fragkoulis@di.uoa.gr
University of Athens
Greece

Dimitris Gizopoulos
dgizop@di.uoa.gr
University of Athens
Greece

Abstract

Cloud computing relies on hypervisors to multiplex hardware across virtual machines, making memory translation a critical performance path. Two-stage address translation is a standard requirement across modern ISAs that support virtualization, but the recent ratification of RISC-V's hypervisor/privileged extensions has only begun to propagate through implementations and software stacks—so the precise interactions among Translation Lookaside Buffers (TLBs), page-table walkers (PTWs), and the memory hierarchy on RISC-V remain an active and important area for study. Given RISC-V's growing traction in datacenter initiatives, a detailed understanding of virtualization performance is critical. Previous work added the H extension with functional support for two-stage translations in gem5, enabling correctness studies of virtualized RISC-V systems, but lacked the timing fidelity required to study full cloud stacks under realistic processor models. In this paper we introduce a cycle-accurate timing model of two-stage page walks integrated with gem5's out-of-order (O3) CPU, enabling—for the first time—the evaluation of a complete RISC-V cloud stack: a Linux guest VM running real workloads, managed by the type-1 hypervisor Xvisor. Our model incorporates instruction and data page-walk caches (PWCs), L1 ITLB/DTLBs, and PTWs that interact with the core and memory hierarchy. Using eleven workloads from the MiBench suite, we systematically vary L1 TLB and PWC sizes, quantifying the performance trade-offs of different MMU configurations and also identifying a cost-efficient, near-optimal design point. Results show that excessively enlarging both first-level TLBs and pagewalk caches yields diminishing returns. Our implementation will be open-sourced to foster further research in RISC-V cloud system design and evaluation.

Keywords: RISC-V, gem5, virtualization, hypervisor, Xvisor, page walk, TLB, page-walk cache, cloud computing

1 Introduction

The RISC-V ISA has emerged as a foundation for diverse computing domains, ranging from embedded devices to high-performance computing and cloud infrastructure. Although the RISC-V cloud ecosystem is still in its early stages, significant efforts [10] are underway to advance it to the level of its x86 counterpart. Its modular and extensible design enables features such as the recently ratified Hypervisor (H) extension, which provides hardware support for virtualization. This extension introduces a new privilege level for the hypervisor (HS) and supports two-stage address translation, thereby allowing type-1 hypervisors like Xvisor [2, 15] to efficiently manage multiple guest operating systems on RISC-V hardware. Virtualization is indispensable for cloud computing, as it enables flexible workload management, security isolation, and efficient resource utilization. However, it also amplifies the importance of the memory management unit (MMU), particularly under two-stage translation.

In our prior work [11], the H extension was integrated into gem5 [7], demonstrating functional correctness and enabling the first boot of Xvisor and Linux VMs inside the simulator. That work validated gem5 as a platform for studying virtualized RISC-V systems, but relied on simplified timing assumptions—memory accesses were simulated only at a functional level. As a result, the performance-critical behavior of two-stage page walks—latency amplification, cache pressure, and contention—was not accurately captured by the simulations.

Two-stage address translation poses fundamental challenges for performance evaluation. In the Sv39x4 setting with VS-stage and G-stage walks, a single TLB miss can require **up to 16 memory accesses**: each of the three VS-stage levels may trigger a full four-level G-stage walk, and a final four-level G-stage walk is still needed for the translated guest-physical address. Without caching mechanisms such as translation lookaside buffers (TLBs) as well as instruction and data page-walk caches (PWCs), this latency quickly becomes prohibitive. Furthermore, in out-of-order (OoO) processors, PTW stalls can propagate deeply into the front-end (through ITLB and I-PWC misses) and back-end (through DTLB and D-PWC misses), degrading pipeline utilization. Capturing these dynamics requires a timing-faithful integration of the page walker into gem5's detailed OoO CPU model.

This paper fills that gap by providing a cycle-accurate timing implementation of RISC-V two-stage translation in gem5, enabling—for the first time—the study of a realistic cloud software stack: a Linux VM executing real workloads under the management of the Xvisor type-1 hypervisor. With this capability, researchers can evaluate how MMU design decisions (e.g., L1 TLB sizes, PWC capacities, walker parallelism) affect cloud application performance. This bridges the gap between ISA-level functional correctness—which our prior work addressed— and accurate microarchitectural performance analysis.

The significance of this work lies in transforming gem5 from a correctness-focused prototype of virtualization support into a tool that can model cloud-relevant timing effects. Just as the prior integration of the H extension allowed us to boot virtual machines, our new contribution allows us to study them in detail: we can now quantify the trade-offs of MMU design under realistic cloud workloads.

In summary this paper makes the following contributions:

1. *Full cloud stack evaluation.* We demonstrate, for the first time, the simulation of Linux VMs under the Xvisor type-1 hypervisor in gem5 with realistic timing for two-stage translation.
2. *Timing-accurate page-walk model.* We implement cycle-accurate VS- and G-stage translation in gem5, modeling timing-accurate PTE fetches and interaction with the OoO CPU pipeline.
3. *Sensitivity analysis.* We run representative workloads while varying L1 ITLB/DTLB and I/D PWC sizes, quantifying the trade-offs between performance and cost of resource provisioning.
4. *Open-source release.* The H extension implementation has been integrated in the latest gem5 version [1]. A follow-up pull request (PR) will be made to update the two-stage translation walker enabling timing-accurate walks, as presented in this paper.

2 Background

2.1 The gem5 simulator

The gem5 simulator [1, 9, 14] is a state-of-the-art, open-source, cycle-accurate, full-system simulator widely used for performance evaluation. It supports a variety of CPU ISAs (RISC-V, ARM, x86) and offers extensive configurability for microarchitectural parameters, including the number and type of cores, pipeline structure, cache hierarchy, buffers, queues, and speculation mechanisms. gem5 can operate in two modes of simulation: functional (atomic) mode for fast emulation, and detailed (timing) mode for cycle-level accuracy. It also supports both syscall emulation (SE) and full-system (FS) simulation. In the former, the simulator substitutes the operating system's provisions artificially whereas in the latter the full operating system software is running and providing its services to user programs.

2.2 RISC-V H extension

The RISC-V H extension [4] was introduced to provide architectural support for virtualization, a capability that has become essential across modern computing platforms. Virtualization enables multiple operating systems and workloads to share hardware resources securely and efficiently, but implementing it purely in software often incurs significant overheads in address translation, context switching, and memory management.

To address these challenges, the H extension adds hardware mechanisms for managing virtual machines, including support for nested page tables, guest/host privilege modes, and hardware-assisted address translation. These features reduce the performance penalties of virtualization while strengthening isolation between guest environments.

The availability of hardware-assisted virtualization is particularly important in domains such as cloud infrastructure and data centers, where workload consolidation, multi-tenancy, and resource efficiency are critical. At the same time, the extension is designed with flexibility, making it applicable to embedded and edge systems that increasingly rely on lightweight virtualization for isolation and security.

2.3 Xvisor bare-metal hypervisor

Xvisor [2] is a type-1 (bare-metal) open-source hypervisor that provides a lightweight, flexible, and portable virtualization platform. It is optimized for high performance and minimal memory usage, and supports a variety of CPU architectures, including RISC-V. Its portability enables adaptation across a wide range of general-purpose hardware platforms. Xvisor offers full virtualization, allowing it to run unmodified guest operating systems such as Linux [6]. It also includes several modern hypervisor features, such as device tree-based configuration, a built-in threading framework, support for runtime loadable modules, dynamic creation and destruction of guest systems, network virtualization, and input device virtualization.

2.4 MiBench Suite

The MiBench benchmark suite, developed by the University of Michigan, comprises a collection of 35 applications categorized into six domains: consumer, networking, office, security, automotive, and telecommunications. These benchmarks are designed to emulate the performance characteristics of real-world applications, encompassing a variety of workloads such as multimedia processing, data compression, encryption, and network communication. By providing a diverse set of applications, MiBench serves as a valuable tool for evaluating and comparing the performance of different processor architectures and configurations.

Utilizing MiBench as a representative for cloud workloads is a reasonable approach, as it offers a broad spectrum of applications that mirror the diverse nature of tasks encountered in cloud environments. The suite's inclusion of applications with varying computational demands, memory access patterns, and input/output characteristics allows for a comprehensive assessment of system performance across different workload types. This diversity enables researchers and practitioners to gain insights into how cloud infrastructures handle a wide range of tasks, from data-intensive operations to latency-sensitive processes. Therefore, leveraging

MiBench in cloud performance evaluations can provide valuable benchmarks for understanding system behavior and optimizing cloud resource management.

3 Implementation

3.1 Two-Stage Translation

In our previous work [11], the functional two-stage address translation is implemented as illustrated in Figure 1. Guest address translation in RISC-V consists of two stages, each involving a complete page table walk. Both stages rely on hypervisor registers that store the base address of the corresponding page tables—namely, *vsatp* for the first stage and *hgatp* for the second.

The first stage, known as the *VS-stage*, is managed by the *vsatp* register and is responsible for translating a guest virtual address (GVA) to a guest physical address (GPA). However, the resulting GPA is still treated as a virtual address from the perspective of the host system. Therefore, this GPA must undergo a second translation stage—known as the *G-stage*—which translates it into a host physical address (HPA) using the *hgatp* register.

A key aspect of this process is that every memory access during the VS-stage walk (including intermediate page table accesses) produces a GPA that must also be translated by the G-stage before the host can access the corresponding physical memory. Thus, resolving a single GVA may require multiple nested translations, where every page table pointer generated by the VS-stage must itself be resolved through a G-stage page table walk. This recursion occurs until the final host physical address is determined.

Figure 1 visualizes the entire two-stage translation process. In the diagram, blue represents the guest virtual address (GVA), pink represents the guest physical address (GPA, still treated as virtual), and orange represents the final host physical address (HPA).

3.2 Two-Stage Timing Walks

gem5's timing CPU models support page walking with timing information to enable more realistic simulation. Specifically, every translation request issued either by the fetch stage or by the Load-Store Queue (LSQ) in the O3 model results in a TLB lookup. In the case of a TLB hit, the corresponding physical address is returned with no cycle penalty. Otherwise, a timed page table walk is initiated.

For each memory access required during the walk (e.g., to retrieve each page table entry and resolve the physical address), a packet is created and sent via ports to the appropriate pagewalk cache—instruction or data—to fetch the corresponding data, possibly traversing the upper levels of the memory hierarchy in case of misses. When the response is received, the *stepWalk()* process is called, as mentioned in [11], to advance the page walk process. This page walk process proceeds asynchronously. Any new translation requests

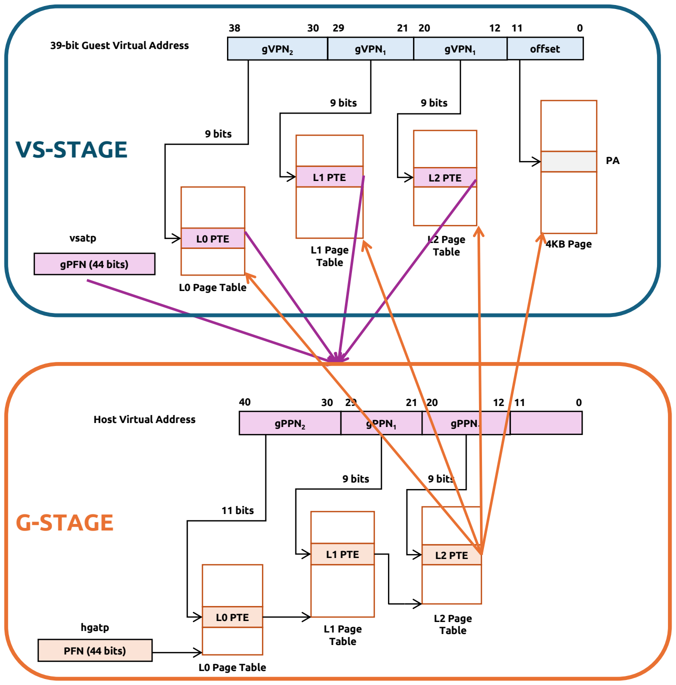

Figure 1. Two-Stage Translation in the RISC-V ISA.

that occur during an ongoing walk are stored in a queue and initiated only after the previous walks have completed.

In this work, we extend the existing approach to support two-stage timing walks. First, we split the *stepWalk()* process into two sub-processes in order to separate the first stage (single-stage or VS-stage) from the second stage (G-stage) accesses. This separation is necessary to preserve the correct sequencing of the two-stage walking process, since the timing behavior of two-stage translation cannot be encapsulated within a single function as before, but must instead be driven by event-based handling.

Building on this, when a packet is received, the appropriate stage of translation (VS or G) is invoked. After the VS-stage completes, an additional G-stage translation is triggered to resolve the resulting guest physical address. Finally, the TLB is updated only after the G-stage translation has completed.

4 Experimental Results

After validating our implementation using the H-extension targeted tests [3] and successfully booting Xvisor—both performed with the O3 CPU model in gem5—we conduct experiments using 11 benchmarks from the MiBench suite [13] and 18 configurations based on parameter sets from the values in Table 1 and detailed below. All experiments were performed on an AMD EPYC 9654 96-core processor (192 hardware threads) and 384GB of DDR5 RAM.

Figures 2 and 3 present the percentage change in simulated cycles across various configurations, as shown in Table 1, using the smallest size as the baseline configuration in each

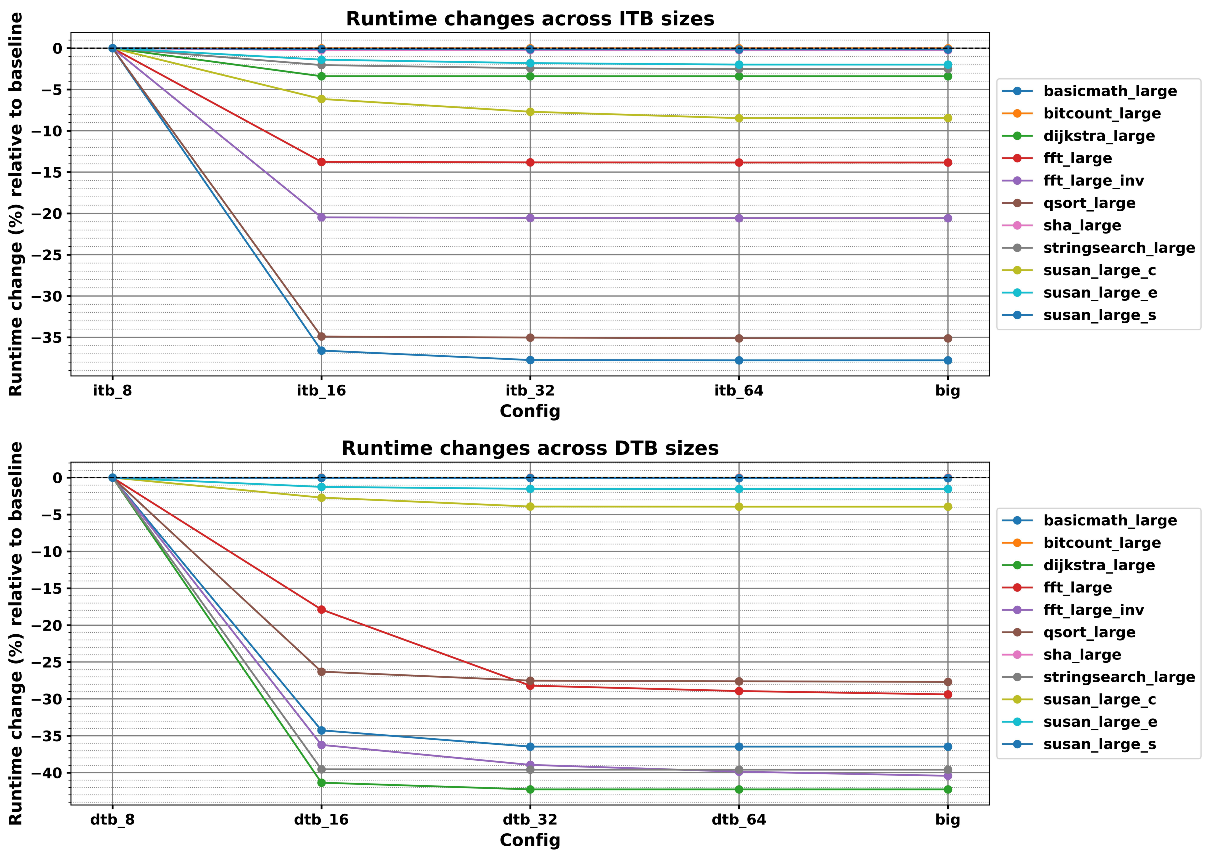

Figure 2. ITB (top) & DTB (bottom) performance comparison.

Parameter	Value
CPU	64-bit RISC-V OoO pipeline
ITLB	8, 16, 32, 64, 128 entries
DTLB	8, 16, 32, 64, 128 entries
IWC	128, 256, 512, 1024, 2048 bytes
DWC	128, 256, 512, 1024, 2048 bytes
I-Cache	32KB, 2-way, 64B block, 2-cycle latency
D-Cache	64KB, 2-way, 64B block, 2-cycle latency
L2-Cache	2MB, 8-way, 64B block, 20-cycle latency

Table 1. Summary of the microarchitectural configuration parameters and their supported values.

graph (i.e., `dtb_8` and `itb_8` in Figure 2, and `iwc_128` and `dwc_128` in Figure 3). The different configurations are laid out on the x-axis (from smaller to larger), while the y-axis shows the runtime change % relative to the baseline. The `big` configuration uses maximally sized components with respect to the parameters of Table 1. Each of the other configurations varies the size of a specific hardware component while keeping all other components constant and set to their maximum values (i.e., their values in the `big` configuration), in order to isolate and highlight the impact of that component on performance. For example, in `dtb_8` the DTB has 8 entries while the ITB was configured with 128 entries, the IWC with 2KB, and the DWC with 2KB. Note that the negative values on the Y-axis indicate performance improvement—relative to the baseline configuration—in percentage units.

The maximum change in cycle count observed across all configurations and benchmarks was 38% for the ITLB, 43% for the DTLB, 17% for the IWC, and 33% for the DWC, demonstrating the significant influence of TLBs on performance. The walker caches, which are accessed upon a TLB miss (during page-walking), also play an important role. Their complementary role—activated only when a TLB miss occurs—explains their measurable but secondary impact on performance. As shown in Figures 2 and 3, in most benchmarks, increasing the size of the TLBs provides a more substantial performance improvement than increasing the size of the page walk caches. For instance, the *dijkstra* benchmark, which heavily accesses data, saw a 43% performance improvement from enlarging the DTLB, compared to only a 9% improvement from increasing the DWC.

However, increasing the size of hardware components affects power consumption, chip area, and overall design cost.

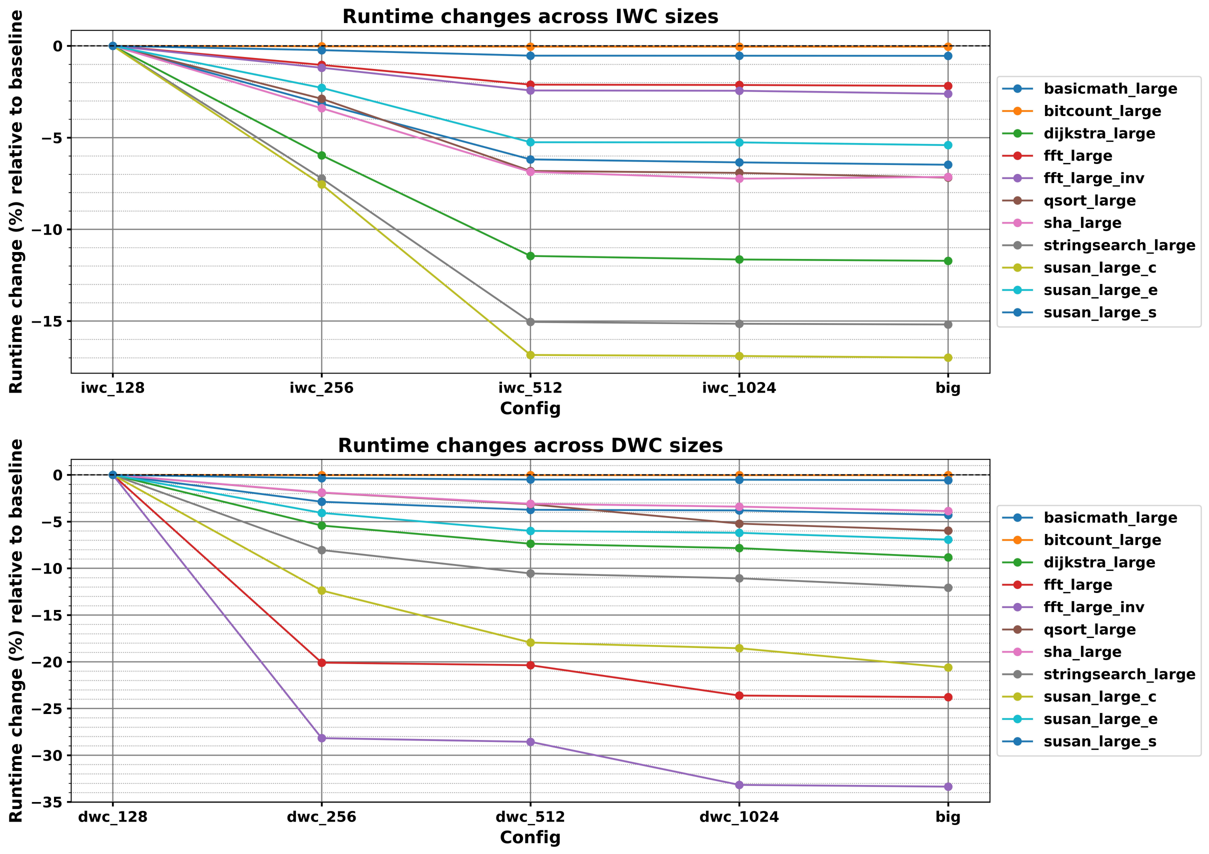

Figure 3. IWC (top) & DWC (bottom) performance comparison.

Figures 2 and 3 also show that beyond a certain size threshold, performance gains taper off, as indicated by the nearly flat (parallel to the x-axis) lines across most benchmarks. Specifically, minimal performance improvement is observed beyond the points at which the ITLB has 32 entries, the DTLB has 64 entries, the IWC is 512B, and the DWC is 1kB in the corresponding graphs. Aiming towards cost-efficiency of the used hardware, we construct the `optimal` configuration with these exact sizes. Figure 4 shows that our cost-efficient configuration only pessimizes performance $\approx 1.11\%$ on average, with a maximal loss of $\approx 3.5\%$ for a single outlier benchmark. This is an attractive tradeoff; we have reduced the ITLB by **4x**, the DTLB by **2x**, the IWC by **4x**, and the DWC by **2x**—significantly affecting chip area and design complexity—while only missing out on slightly more than **1%** of performance on average.

5 Related Work

The RISC-V H extension is a relatively recent addition to the privileged architecture of the RISC-V ISA, officially ratified in December 2021 [4]. This extension introduces significant opportunities for both academia and industry to explore advanced virtualization features within the RISC-V ecosystem. Several projects have been initiated to integrate the hypervisor extension into different hardware models [10]. From a hardware standpoint, functional support for the H extension has been implemented in gem5 [11], while hardware implementations based on in-order CPUs in the Rocket core [16], the CVA6 core [17], and the Lagarto I core [12]. Beyond hardware, software tools and simulators have also adopted the hypervisor extension. For example, QEMU [8], a widely used open-source emulator, and Spike [5], the official RISC-V ISA simulator, both offer support for the H extension.

6 Conclusion & Future Work

In this paper, we presented a cycle-accurate timing model of two-stage address translation for RISC-V integrated into gem5's out-of-order CPU. This model enables, for the first time, a full cloud stack evaluation of Linux guest VMs under the Xvisor type-1 hypervisor, capturing the performance impact of two-stage page walks with instruction and data page-walk caches (PWCs), L1 ITLB/DTLBs, and page-table walkers (PTWs). Through systematic sensitivity studies using representative MiBench workloads, we quantified the trade-offs between TLB and PWC sizes, identifying a cost-efficient, near-optimal MMU configuration. Beyond these

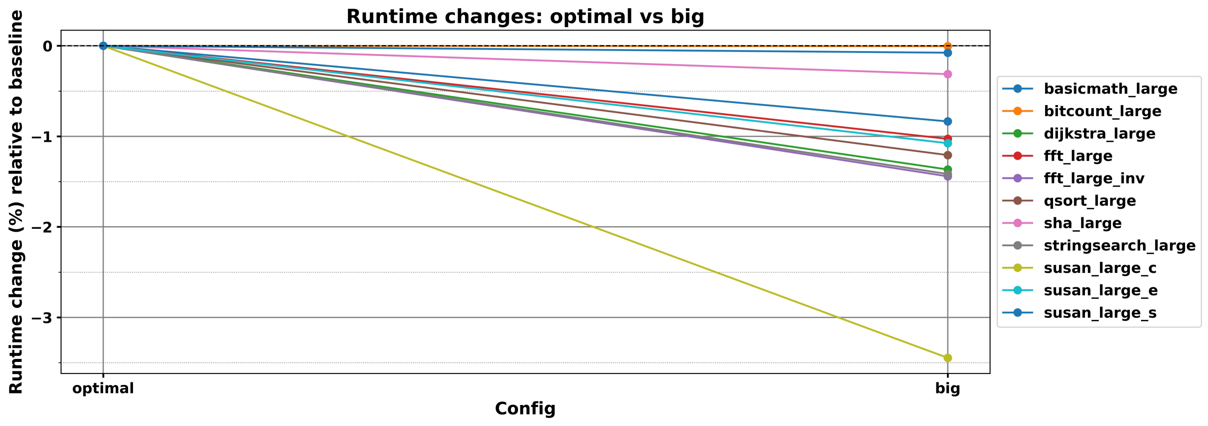

Figure 4. optimal (cost-efficient config) vs big (largest config) performance comparison.

specific results, our infrastructure demonstrates the ability to extract detailed performance metrics and actionable insights across a wide range of MMU configurations, providing researchers and architects with a powerful tool to evaluate, optimize, and guide the design of RISC-V cloud systems.

Looking forward, future work can explore adding models for additional microarchitectural enhancements to accelerate MMU functionality, such as more optimized page-walk pipelines, prefetching strategies, various caching schemes, and integration with speculative execution mechanisms. Extending the model to capture these features will provide deeper insights into performance-critical MMU design choices and further enable realistic evaluations of RISC-V cloud infrastructures.

References

[1] 2003. gem5 GitHub Repository. https://github.com/gem5/gem5. Accessed: 2024-07-30.

[2] 2011. Xvisor: an open-source bare-metal monolithic hypervisor. https://xhypervisor.org. Accessed: 2024-07-30.

[3] 2020. Unit tests for RISC-V Hypervisor extension. https://github.com/josecm/riscv-hyp-tests. Accessed: 2024-07-30.

[4] 2024. The RISC-V Instruction Set Manual Volume II: Privileged Architecture. https://drive.google.com/file/d/17GeetSnT5wW3xNuAHI95-SI1gPGd5sJ_/view?usp=drive_link Document Version 20240411, Accessed: 2024-07-30.

[5] 2024. Spike RISC-V ISA Simulator. https://github.com/riscv-software-src/riscv-isa-sim. Accessed: 2024-07-30.

[6] 2024. The Linux Kernel Archives. https://www.kernel.org. Accessed: 2024-07-30.

[7] 2025. Implementation of the RISC-V H-extension in gem5. https://github.com/gem5/gem5/pull/1387. Pull request #1387, accessed: 2025-08-28.

[8] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX annual technical conference, FREENIX Track*, Vol. 41. California, USA, 10–5555.

[9] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (aug 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[10] Ramon Canal, Cristiano Chenet, Angelos Arelakis, José-Maria Arnau, Josep Ll. Berral, Aaron Call, Stefano Di Carlo, Juan José Costa, Dimitris Gizopoulos, Vasileios Karakostas, Francesco Lubrano, Konstantinos Nikas, Yiannis Nikolakopoulos, Beatriz Otero, George Papadimitriou, Ioannis Papaefstathiou, Dionisios Pnevmatikatos, Daniel Raho, Alvise Rigo, Eva Rodríguez, Alessandro Savino, Alberto Scionti, Nikolaos Tampouratzis, and Alex Torregrosa. 2023. VITAMIN-V: Virtual Environment and Tool-Boxing for Trustworthy Development of RISC-V Based Cloud Services. In *2023 26th Euromicro Conference on Digital System Design (DSD)*. 302–308. https://doi.org/10.1109/DSD60849.2023.00050

[11] George-Marios Fragkoulis, Nikos Karystinos, George Papadimitriou, and Dimitris Gizopoulos. 2024. Advancing cloud computing capabilities on gem5 by implementing the risc-v hypervisor extension. *arXiv preprint arXiv:2411.12444* (2024).

[12] Jaume Gauchola, JuanJosé Costa, Enric Morancho, Ramon Canal, Xavier Carril, Max Doblas, Beatriz Otero, Alex Pajuelo, Eva Rodríguez, Javier Salamero, and Javier Verdú. 2024. Hypervisor Extension for a RISC-V Processor. arXiv:2406.17796 [cs.AR] https://arxiv.org/abs/2406.17796

[13] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*. 3–14. https://doi.org/10.1109/WWC.2001.990739

[14] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A.

Wood, Hongil Yoon, and Éder F. Zulian. 2020. The gem5 Simulator: Version 20.0+. arXiv:2007.03152 [cs.AR] https://arxiv.org/abs/2007.03152

[15] Anup Patel, Mai Daftedar, Mohamed Shalan, and M. Watheq El-Kharashi. 2015. Embedded Hypervisor Xvisor: A Comparative Analysis. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 682–691. https://doi.org/10.1109/PDP.2015.108

[16] Bruno Sá, José Martins, and Sandro Pinto. 2022. A First Look at RISC-V Virtualization From an Embedded Systems Perspective. *IEEE Trans. Comput.* 71, 9 (2022), 2177–2190. https://doi.org/10.1109/TC.2021.3124320

[17] Bruno Sá, Luca Valente, José Martins, Davide Rossi, Luca Benini, and Sandro Pinto. 2023. CVA6 RISC-V Virtualization: Architecture, Microarchitecture, and Design Space Exploration. arXiv:2302.02969 [cs.AR] https://arxiv.org/abs/2302.02969