Supporting RISC-V Full System Simulation in gem5

Peter Yuen Ho Hin, Xiongfei Liao, Jin Cui, Andrea Mondelli, Thannirmalai Muthukaruppan Somu, Naxin Zhang
{petery.hin,liao.xiongfei,cuijin7,andrea.mondelli,thannirmalai.muthukaruppan.somu,Naxin.Zhang}@huawei.com
Huawei Research Centre

ABSTRACT

The RISC-V ISA and ecosystem have been becoming an increasingly popular in both industry and academia. gem5 is a widely used powerful simulation platform for computer architecture research. Previous works have added single-core and multi-core RISC-V support to gem5 but only for system call emulation. The full-system simulation of gem5, on the other hand, provides accurate analysis of systems as an actual system software is loaded and run on the hardware platform modelled in gem5. However, full-system simulation support in gem5 for RISC-V ISA is currently not available.

This paper presents our recent work on supporting RISC-V full-system simulation in gem5. After describing the implementation details of supporting extensible target system and debugging methodology for overcoming major challenges, we share our experiments of full-system simulations.

ACM Reference Format:
Peter Yuen Ho Hin, Xiongfei Liao, Jin Cui, Andrea Mondelli, Thannirmalai Muthukaruppan Somu, Naxin Zhang. 2021. Supporting RISC-V Full System Simulation in gem5. In *Proceedings of Computer Architecture Research with RISC-V (CARRV '21)*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

RISC-V is a free, open-source ISA [6][5] which has recently gained popularity from both the academia and the industry. RISC-V is designed to be simple, efficient yet future-proof by avoiding the pitfalls of existing ISAs and allowing extension of the instruction set. The RISC-V ISA also adopts a modular approach where vendors can implement any chosen set of the RISC-V ISA extensions. As such, this ISA is friendly to academic research and low volume applications, but powerful enough to be extended to warehouse-scale applications.

gem5 is a powerful open-source simulator [1] [2] widely used in computer architecture research. It strives to achieve a balance between speed, accuracy and development time. Its users can choose between different CPU models, system modes and memory systems to achieve system configurations with desired level of trade-offs. Such configurations can be easily setup up through gem5's Python interface, while performance-critical simulation logic is implemented in C++.

For each ISA, gem5 offers two modes of simulation: syscall emulation (SE) and full system (FS) simulation. With previous work [3, 4], gem5 is able to support most RISC-V instructions and system calls in SE mode where system calls are emulated. This mode provides a simplified method to run and analyze user-space workloads. FS simulation is needed for accurate analysis of system components and devices as an actual system software (often Linux kernel) is loaded by gem5.

Research areas which are made possible by FS simulation include virtual memory, virtualization, distributed systems , storage stack performances and network-related studies. However, the use of RISC-V for those areas was limited due to the lack of FS simulation support. This paper addresses this gap.

In this paper, we present our work of adding support for RISC-V full-system simulation, which has been included with a GNU/Linux Busybox distribution with kernel version 5.10 in the official gem5-21.0 release. We describe the target system setup and major challenges in Section 2, followed by the implementation details of new device models and platform in Section 3. Section 4 explains our debugging methodology. Finally, Section 5 presents validation and testing results using the full system setup.

2 DEVELOPMENT TARGET AND CHALLENGES

In this section, we first show the target system setup and then discuss the challenges to overcome for successful gem5 RISC-V full-system support.

2.1 Target System

The goal is to build a baseline RISC-V system which can be easily extended based on user needs. This target system has a core set of hardware sources including a minimum set of peripherals. It is able to run the system software, for example, one with bootloader and Linux kernel.

The hardware configuration of target system is shown in Figure 1, where only modules of interests are presented. Besides bus subsystem, there are two major sub-systems: CPU and HiFive Platform. The blocks represented by orange boxes are the devices newly added in our work for successful booting of system software.

In CPU sub-system, an extra MMU component, PMA checker, is added. The HiFive platform is based on SiFive's HiFive series of board and contains the minimal set of critical peripherals. The Core Local Interrupt Controller (CLINT) handles software and timer

Peter was Year 3 student with Nanyang Technological University, Singapore and participated this work during his internship with Huawei Singapore Research Centre. Xiongfei is the corresponding author of this paper. Andrea is with Huawei UK Research Centre; others are with Huawei Singapore Research Centre.

CARRV '21, June 14–19, 2021, Co-located with ISCA 2021

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

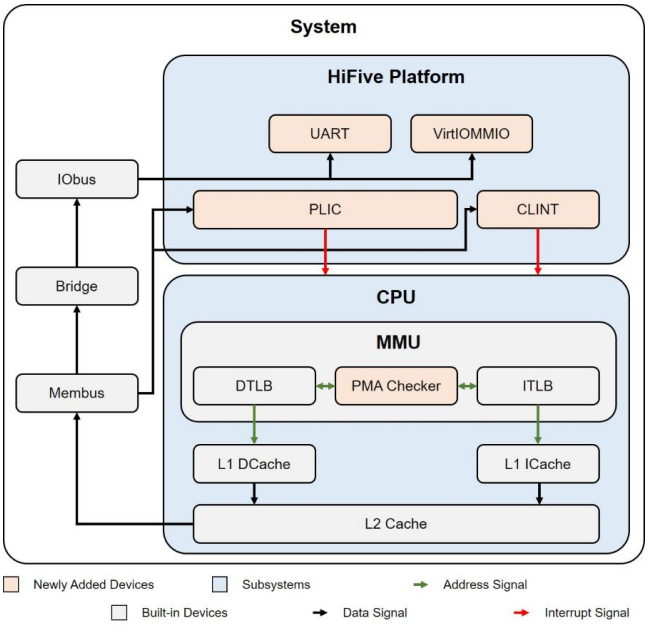

Figure 1: Hardware Configuration of Target System

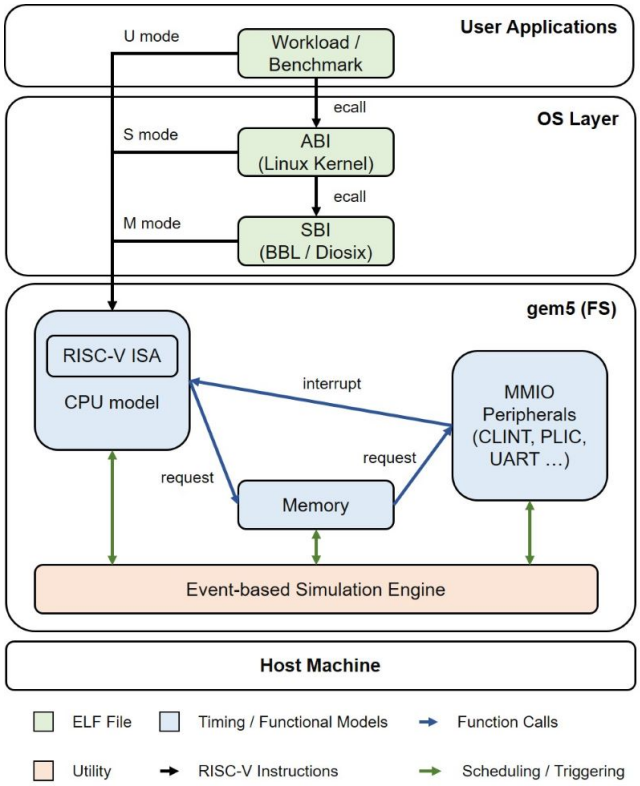

Figure 2: Software Layers of gem5 FS simulation

interrupts via a MMIO interface. The Platform Level Interrupt Controller (PLIC) is responsible for routing interrupts from external sources and peripheral devices to the hardware threads based on a priority scheme. The UART and VirtIOMMIO are not necessary for kernel boot-up but are essential for a usable operating system. UART provides an interactive command line terminal while the VirtIOMMIO provides a copy-on-write root filesystem which contains the workload scripts and operating system binaries.

Figure 2 shows the stack consisting of several software layers in gem5 full-system simulation. The gem5 (FS) block in the figure contains the hardware modules of system with desired configuration. It also models the interactions between hardware modules. CPU model with RISC-V ISA decoder handles the instructions from OS layer or user applications, which could be at different privileged modes. A gem5 FS simulation starts with parsing Python configuration script and building simulator executable based on configurations. Then, the simulator loads bootloader and Linux kernel to boot up the system. When kernel is up, user applications can be executed in background or via terminal.

Our gem5 RISC-V FS simulation also supports a hypervisor which doesn't need hardware-assistant virtualization, i.e., RISC-V H extension. Diosix is such a hypervisor.

2.2 Challenges

Compared to the SE mode, the FS simulation has a more complex configuration consisting of aforementioned newly added hardware components and software components such as the kernel and bootloader payloads. During a boot process, a fault can occur in any of the above components, even in interactions with the existing CPU models and memory models due to wrongly implemented privileged instructions and newly added device models.

There could be multiple possible reasons for a failure during a FS simulation as below. The DTB configuration errors can cause bootloader run into an erroneous status after parsing device-tree. In FS simulation, interrupt mechanism plays a critical role to keep the system running. The wrong privileged ISA implementation, interrupt triggering mechanism, or the interrupt handling logic within CPU and interrupt controller can lead to faults in simulation. As we found during debugging, in the complex scenarios, there were even errors due to CPU pipeline squashing and memory access of peripheral devices.

Finding out the root cause of a fault in above mentioned gem5 FS simulation is challenging due to the various reasons and complex scenarios. Identifying the code or instructions within bootloader or kernel that trigger the fault is more challenging because of the code size of them and the entrance into an infinite loop of process where the bootloader or the kernel enters a panic function and ends up in an idle loop when an error occurs. When that happens, millions of assembly instructions could have executed in simulation and it is impossible to trace the root cause manually. Therefore, a method of effective backtrace through millions of assembly instructions to locate the origin of an error is needed.

Additionally, gem5 is an event-driven simulator which simulates the desired tick-by-tick behaviour of the hardware. However, this introduces complexities to the debugging as call-stack information is limited to calls within the same simulation tick. Events such as memory read request and response would not be visible under

the same call stack. Hence, we would need a method of analyzing beyond the scope of current tick.

To smooth our work of supporting RISC-V FS simulation in gem5, in addition to enhancing traditional remote GDB debugging, we developed a sophisticated tool kit (a set of Python scripts) to analyze the instruction execution trace of a gem5 simulation together the comparison of the corresponding execution trace in Qemu. In Section 4, we will elaborate on our debugging methodology and how to apply it to overcome the above challenges.

3 IMPLEMENTATION DETAILS

In this section, we present the implementation details of the newly added devices or hardware modules. The UART module is built-in in gem5 while the VirtIOMMIO model is ported over from the ARM setup. The focus is put on the platform and other devices like CLINT, PLIC and PMAChecker.

We also talk about the other fixes on privileged instructions and CPU models. This section is closed by supports for checkpointing and device tree.

3.1 HiFive Platform

In gem5, system configurations are organized into container classes called platforms. A Platform class is a parent class with a standardized set of peripherals and utility functions that can be extended in a hierarchical manner to customize the setup to a specific board / system. In ARM, the common Platform class is **RealView** while in X86, the common Platform class is PC. In RISC-V, we name the platform HiFive, which corresponds to SiFive's HiFive series of board. The memory map conventions and peripheral addresses are chosen based on the SiFive U54MC SoC datasheet. The HiFive platform contains the minimal set of peripherals upon which other non-critical peripherals can be added to. Such a base configuration is used not only on HiFive boards but also on other SoCs such as the Kendryte K210.

The HiFive platform is designed to be easily extendable, with minimal changes needed to be made to port over devices from other ISAs. A PlicIntDevice class is provided to allow easy connection of a peripheral to the PLIC interrupt controller. A set of utility functions within the HiFive platform class also allows users to add new devices to a list and have the necessary connections made automatically.

3.2 CLINT

CLINT, as introduced earlier, handles software and timer interrupts. RISC-V hardware threads can set timer interrupts or send inter-processor interrupts to other threads by writing to memory-mapped registers of CLINT in machine mode.

In SiFive's setup, CLINT is often connected to an external RTC clock signal that increments the MTIME register, which then triggers interrupts based on the MTIMECMP register values for each hardware thread. In the current setup, a dummy RTC model is constructed to allows for a configurable clock frequency but is not yet implemented as a MMIO device. Aside from timer interrupts, the MTIME register also supplies the value for the RDTIME instruction.

Software interrupts in CLINT are posted and cleared using an MMIO interface. In machine modes, inter-processor interrupts (IPI) are made possible by allowing access to the MSIP register address of other cores.

3.3 PLIC

PLIC is responsible for routing of external interrupts to different contexts, each of which corresponds to an interrupt pending bit in a certain privilege mode. PLIC can route interrupts from up to 1023 external sources (1 to 1023) to up to 15872 contexts. A context in PLIC can corresponding to a hardware thread, or a (hardware thread, privilege level) tuple depending on the hardware implementation. In the current gem5 implementation, each hardware thread corresponds to two contexts, one for M mode and one for S mode.

Each source will be assigned a 32-bit priority and each context can enable or disable interrupts from any sources. At the same time, each context can also set a threshold on the minimum interrupt priority to trigger an external interrupt. A simplified working example of PLIC is illustrated in Figure 3. In accordance with the specifications, a 3-cycle delay is simulated between the interrupt sources and the external interrupt signals.

Claiming and completion of external interrupts are also implemented via MMIO. The gem5 implementation ensures that any external interrupt can only be claimed by one context and that a context cannot claim multiple interrupts before completing the last claimed interrupt.

3.4 PMAChecker

Aside from the MMIO devices, the RISC-V ISA also requires the implementation of a Physical Memory Attribute (PMA) checking mechanism for checking attributes of the memory address such as atomicity, memory-ordering, coherence, cacheability and idempotency. The RISC-V specification suggests possible memory attributes to check for but does not specify any standards on the implementation on this hardware component.

As such, the gem5 implementation of the PMA checker is an abstract components which adds certain flags and attributes to the memory request after address translation. Currently, the PMA checker only checks for uncacheability but further checks can be implemented easily when the need arises. This unit is necessary for the proper functioning of all MMIO peripherals as it ensures that memory requests to these devices will not be cached.

3.5 Fixes on Privileged Instructions and CPU Models

Prior to this paper's work, the RISC-V ISA is already mostly supported in SE mode. However, a few fixes were made by us on the CSR instructions and interrupt handling logic. The current ISA supports still has some minor discrepancy with the RISC-V ISA specifications but these discrepancies should not affect the functionality of the system.

In order to support full system booting on MinorCPU and DerivO3CPU, the following changes were made in our work. Firstly, PLIC avoids posting interrupt to hardware threads which are already handling an external interrupt. This should not be necessary in real hardware but was needed for MinorCPU support. MinorCPU lacks the internal logic to prevent receiving interrupt signals while it is inside an interrupt handler. Secondly, a pipeline squash was

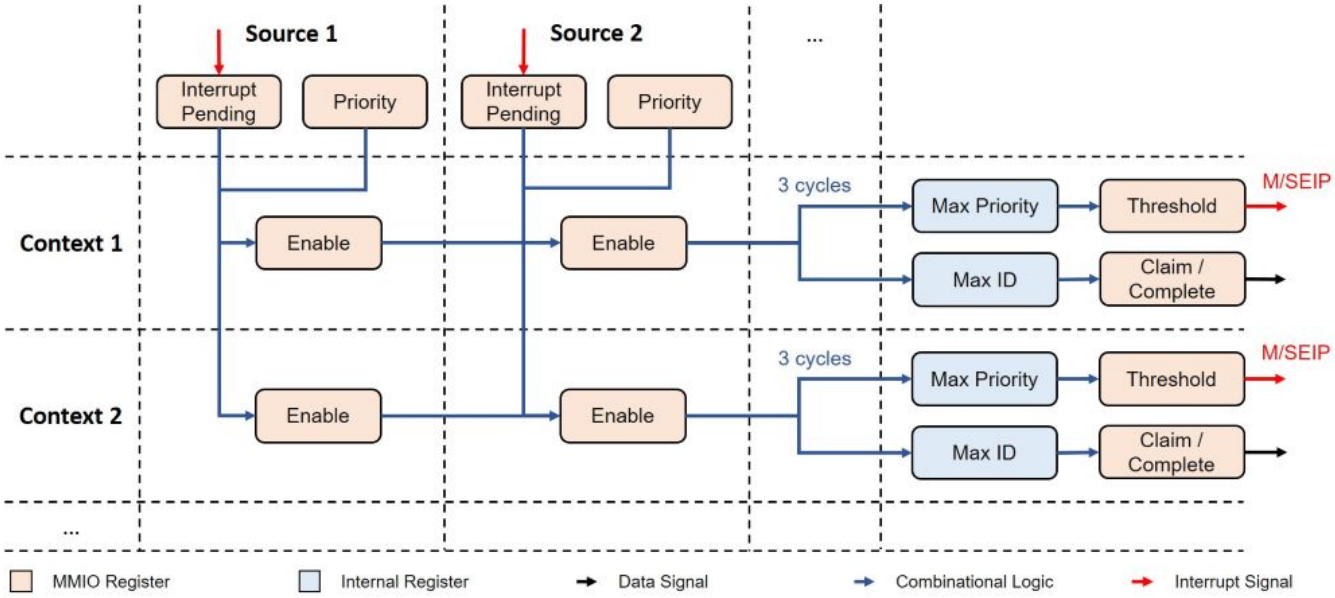

Figure 3: PLIC Illustration

forced after CSR writes to the SATP register. This is necessary such that AUIPC instructions in the pipeline do not use a wrong PC value when the MMU is activated.

3.6 Checkpointing and Device Tree

Besides a command line interface, another essential feature of a full system we added in gem5 is the ability to store and restore checkpoints. Booting a full system simulation can be very time consuming, especially when multiple peripheral devices are connected to the system. As such, researchers often use a fast CPU model such as AtomicSimpleCPU for the boot process and checkpoint the system after boot-up. Subsequently, researchers can run benchmark using more accurate CPU models by restoring from the checkpoint without the need to boot up the system. In our gem5 RISC-V full system support, we have tested and verified the checkpointing functionality on all CPU models.

Our full system support also comes with devicetree generation functionality. In Linux systems, the device and peripheral setup is made known to the kernel using a devicetree. Since gem5 systems are configured using a Python interface, it is necessary to modify the devicetree binary passed to the kernel everytime the system configuration is modified. In our full system support, we have added devicetree generation feature to each peripheral device such that users can avoid the trouble of having to manually match the devicetree binary with the Python configuration.

4 DEBUGGING METHODOLOGY

To overcome the challenges mentioned in Section 2, we enhance remote GDB support and introduce the methodology of trace analysis. In this section, we present details of them and share how they were applied in debugging FS simulation.

4.1 Remote GDB and Trace Analysis

A remote GDB stub is commonly attached to the workload running on the simulator. Using breakpoints and stack information, the origin of an error can be quickly traced. Prior to our work, RISC-V remote GDB in gem5 only supports printing values of integer registers. Support for printing values of float-point and CSR registers is added to allow for more efficient checking of privileged instruction implementation and errors.

Whenever a boot attempt of FS simulation fails, the method of remote GDB cannot be directly applied because it is difficult to identify the instruction causing trouble. Different from the common user workloads, the kernel and bootloader do not exit on an error like a typical workload. Instead, a panic function is called, which eventually goes into an idle loop. Given the size and complexity of the kernel and bootloader, together with the large number of simulation instructions (could be millions), it is not feasible to iteratively backtrack till the origin of the error using remote GDB in a forward-straight way.

We introduce the method of trace analysis and develop a toolkit which consists of several Python scripts for trace analysis. In order to avoid manual inspection of millions of lines of the output traces, we used QEMU as a reference for the exactly same execution path. Using the same system setup described in a same DTS file, we boot QEMU emulator and gem5 full-system simulator side-by-side and collect both execution traces which are in different formats. We then use the toolkit to parse both traces and perform comparisons on the execution path to find the location of the translation block where the two traces diverges. Subsequently we use aforementioned enhanced remote GDB to automatically insert breakpoints into the blocks to identify where errors come from.

Table 1: Simulated Run-time (in milliseconds) of Running Multhreading Benchmarks Using Four O3 Cores

Number of threads	PARSEC Benchmark				
	blackscholes	canneal	freqmine	streamcluster	swaptions
1	39.0	754.0	1363	160	431
2	20.0	392.0	777	-	218
4	10.1	257.9	-	-	-

4.2 Experience of Debugging FS Simulation

Using the above debugging methods together with gem5 build-in debugging log, we were able to effectively locate and fix any errors that occur during gem5 FS simulations.

An example of such errors is an incorrect return value from a load instruction, which was caused by the incorrect implementation of a peripheral device or the accidentally caching of the MMIO address range. By trace comparison with QEMU, we quickly identified the instruction that caused the panic condition. Looking into the instruction, we figured out the reason for the fault.

In some cases, trace analysis helps identify the location but cannot reveal the root cause. We can further leverage the remote GDB functionality to check or write to the interrupt pending and enable bits. Take the debugging of interrupt trigger mechanism for instance. The remote GDB functionality allowed us to efficiently verify the trigger conditions and timing behaviour of interrupt devices such as PLIC and CLINT. It also helped in debugging the interrupt handling logic in gem5 RISC-V CPUs by checking through the register state changes in desired clock cycles.

Aside from the above tools trace analysis and remote GDB, gem5's built-in debug logs were extensively used as well. They are helpful for printing out the internal states of targeted gem5 devices. For example, by enabling the debug logs of DerivO3CPU, we were able to inspect the behaviour of the pipeline stages in each clock cycle and thus identify issues such as accidentally triggered squashes. After fixing CLINT and SATP write side effects, DerivO3CPU is supported in RISC-V FS simulation.

5 EXPERIMENTS OF FS SIMULATION

5.1 Full-System Linux Boot-up

To verify our implementation, we booted up the target system in Section 2 using the Berkeley bootloader together with the Linux kernel v5.10. For simplicity, the system consists of four CPU cores, each with one hardware thread. The filesystem used is a port of the BusyBox disk image.

The Linux system has been successfully booted under all widely used four CPU models offered by gem5: Atomic Simple, Timing Simple, Minor and DerivO3. We further logged in the system and executed commands using terminal. The commands within BusyBox can be run without errors. Checkpoint and restore functionalities have also been tested with switching CPU models.

This correct functionality of commands shows that the kernel's process management and scheduler is working. Since the scheduler relies on CLINT's timer interrupts, we are sure that CLINT's timing functionality is implemented correctly. The ability to read, write and move files also demonstrates the correct functionality of the file system, which is controller by the VirtIOMMIO device. Furthermore, the proper functioning of the interactive terminal shows that the UART and PLIC models are correctly configured.

Table 2: Benchmark Blackscholes Simulated Run-time (in milliseconds) Using 1 Core

CPU	Number of Threads	Linux	Linux on top of Diosix
O3	8	38.948	94.468
	4	38.053	92.420
	2	36.959	91.781
	1	36.236	91.219

The most important components of Linux OS, including process management, memory management, device drivers and inter-process communication, have been checked to work correctly. We are confident that the target system and FS simulation support have been correctly implemented.

5.2 Running Multi-threaded Workloads

We created an instance of target system with Symmetric Multi-Processing (SMP) configuration. This configuration has four O3 CPU cores (each with one hardware thread) and 1024M DRAM. We further created a port of the PARSEC benchmark and selected five benchmarks. During experiments, we run multiple multi-threaded workloads on above Linux platform using different numbers of threads to see how this SMP configuration works.

Table 1 shows the simulated run-time taken to complete each execution of benchmarks under different thread counts. As some simulations are still running when this paper is submitted, a few cells are filled with "-" to indicate that the related data could NOT be included in camera ready paper.

As shown in the table, when a workload with a same data set to be handled by different numbers of threads, the execution with 2X threads could reduce the simulated run-time, i.e., the execution time of workload in simulation, roughly by half, compared to the one with 1X thread(s). For example, benchmark Blackscholes takes 39ms to finish the execution using 1 thread. After using 4 threads, it only needs 10.1ms to complete. The speed-up is close to 4. For other benchmarks, there are speed-up more or less. Hence, we can conclude that our full-system setup has successful support for running multi-threaded workloads on SMP configuration with multiple hardware threads.

5.3 Running Workloads on Linux as Guest OS of Diosix Hypervisor

Our full-system setup also allows for analysis of more complicated system setups. We created an instance of target system with the hardware configuration of 1 CPU core, which has one hardware

thread. This hardware configuration boots up two software configurations: 1) Linux FS; and 2) Diosix (a M mode hypervisor) with Linux as one of its Guest OS. Table 2 shows comparison of the simulated run-time for the blackscholes benchmark under different simulation runs using different numbers of software threads.

From the third column, we can see that the simulated run-time is increased following the increase of number of threads used to run the benchmark. This shows the thread context switch overhead on a CPU core with single hardware thread.

By comparing the numbers on a same row, we can see that the number on the third column is much less than the one on the fourth column. The extra delay is caused by Diosix for 1) intercepting and running system calls from benchmark; and 2) scheduling other guest OSes running on top of Diosix.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented our work on adding support for gem5 RISC-V FS simulation. We have implemented a core extensible target system for gem5 FS simulation by adding new devices and fixing errors in privileged instructions and CPU models. We further elaborate on our debugging methodology which is beneficial to other developers as well. At last, we share our experiments of FS simulations in 1) booting up Linux system, 2) running workloads on top of the system and 3) booting up Diosix hypervisor with two guest OSes and running workloads on one of the guest Linux OS.

Our gem5 FS simulation support provides the essential infrastructure to a multitude of future research topics on the RISC-V ISA. We give several examples as below.

gem5 FS simulation is crucial in the analysis of the virtual memory system architecture. The FS simulation allows the modelling of the system with virtual memory that integrates proposals such as page based attributes mechanism and new address translation modes for RISC-V in the OS kernel.

Security research involves multiple components of the system: the cores, the bus, the secure enclave, the external devices, the OS and the firmware. FS simulation can be used to evaluate proposals implemented in gem5 at system level.

The hardware-assistant virtualization technique is critical to high-performance processors in HPC and cloud domains. The RISC-V H extension and KVM support can be added on top of the gem5 FS simulation so as to enhance the capability of system-level performance evaluation.

REFERENCES

[1] Nathan et. al. Binkert. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.
[2] Jason Lowe-Power et. al. 2020. The gem5 Simulator: Version 20.0+. arXiv:2007.03152 [cs.AR]
[3] Alec Roelke and Mircea R Stan. 2017. Risc5: Implementing the RISC-V ISA in gem5. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*.
[4] Ta Tuan, Cheng Lin, and Batten Christopher. 2018. Simulating Multi-Core RISC-V Systems in gem5. In *Second Workshop on Computer Architecture Research with RISC-V (CARRV)*.
[5] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David A Patterson, and Krste Asanovic. 2016. The RISC-V instruction set manual volume II: Privileged architecture version 1.9. (2016).
[6] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanović. 2016. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1. (2016).