A Hardware-Software Cooperative Interval-Replaying for FPGA-based Architecture Evaluation

Hongwei Cui[1], Shuhao Liang[1], Yujie Cui[1], Weiqi Zhang[1], Honglan Zhan[2],
Chun Yang[3], Xianhua Liu[3] and Xu Cheng[3]
School of Computer Science, Peking University, Beijing, China
[1]{cuihongwei, liangshuhao, YujieCui, zhangweiqi}@pku.edu.cn, [2]laan.z@stu.pku.edu.cn,
[3]{yangchun, liuxianhua, chengxu}@mprc.pku.edu.cn,

***Abstract*—Open-source processors and FPGA provide more real and accurate results of the new microarchitecture design, but the long execution time for running large benchmarks on FPGA boards still hinders researchers. This paper proposes a hardware-software cooperative interval-replaying. It uses simulators to create checkpoints for arbitrary program intervals and provides an extensible and portable checkpoint loader to re-execute selected intervals. In addition, this paper extends RISC-V ISA and proposes an event-based sampling design to find hot program intervals with more representative microarchitecture characteristics. By using checkpoints in hot regions, researchers can quickly verify the effectiveness of microarchitecture designs on FPGA and alleviate the speed bottleneck of FPGA. The correctness and effectiveness of the checkpoint scheme and the event-based sampling design are evaluated on FPGA. The experimental results show that the solution is effective.**

***Index Terms*—Open-source processors, FPGA, RISC-V, Checkpoint, Sampling**

I. Introduction

With the development of RISC-V projects, researchers can utilize the RISC-V ecosystems and FPGA boards to execute the complete benchmarks on a real-world operating system to evaluate the effectiveness of their wosks. Unfortunately, the resource limitation of FPGA boards still hinders researchers from utilizing open-source processors. For example, some SPEC CPU2006 benchmarks will take more than 30 hours on the SonicBOOM processor [1], which runs at 50 MHz on the Digilent Genesys 2 FPGA board.

Previous research works proposed the checkpointing technique for speeding up program simulation, such as ELFies [2] for X86 ISA and Intrinsically Checkpointed Assembly Code (ITCY) [3] for Alpha ISA. However, ELFies needs to rely on the PinPlay [4] framework, and ITCY needs to insert extra codes into the program binary.

This paper proposes a hardware-software cooperative interval-replaying solution. It first provides an event-based sampling design based on RISC-V ISA for choosing the desired program intervals for replaying. Second, it provides an extensible general checkpoint design to assist researchers in replaying the intervals of single-threaded programs on FPGA boards. We evaluate our solution with SPEC CPU2006 and SPEC CPU2017 on the SonicBOOM processor.

II. Hardware-Software Cooperative Interval-Replaying

This section first introduces the event-based hardware sampling design used to help researchers to choose the desired program interval. Next, it presents the main ideas and challenges of the general checkpoint design for interval replaying.

A. Choosing Hot Program Interval

Choosing the program interval with more obvious microarchitectural characteristics will help researchers to evaluate new designs efficiently and quickly. To this end, this work provides an event-based hardware sampling design to obtain the hardware events information of each program interval on open-source RISC-V processors.

This design utilizes the reserved RISC-V user-level control and status register (CSR) to configure and assist hardware sampling, including configuring the sampling period and the sampled hardware event. In order to capture the triggered sampling signal on software, this design sets the signal to a special syscall and uses the process trace (ptrace) to capture it, which avoids modifying the OS codes.

B. Replaying Program Intervals

This work presents a general checkpoint design for replaying the desired program intervals on FPGA boards. It first uses the gem5 simulator [5] to collect program runtime information and create checkpoint files. The information collected includes the logical registers' values and the first instruction address at the beginning of the interval, the executed syscalls' information, and the necessary memory states.

Second, it provides an extensible and general checkpoint loader for reading checkpoint files and re-executing program intervals on FPGA boards. This loader is independent of checkpoints and OS, making it easier to debug and extend new features. To increase the generality and extendibility of our design, we address the following challenges.

a) The system call handling challenge: Since the execution of syscalls relies on OS resources, to ensure its correctness and increase the generality of the design, we choose to reconstruct the architectural effects of syscalls. Therefore,

978-3-9819263-7-8/DATE23/© 2023 EDAA

we save all input and output parameters and memory updates associated with the syscalls executed in the interval. Before replaying the interval, these syscalls will be replaced with jump instructions that redirect the processor to a specific function that reconstructs their effects.

b) Interval replaying exit challenge: During the interval replaying, the replaying process needs to be actively terminated after the desired interval has been completely re-executed. Otherwise, it may be terminated abnormally due to accessing or executing uninitialized memory.

To this end, we will find a special interval exit instruction for triggering the replaying exit. An interval exit instruction needs to satisfy the following conditions.

- It is executed after the end of the specified interval.
- It is executed for the first time after the last syscall in the interval. If there is no syscall, this instruction is executed for the first time from the beginning of the interval.

As the interval exit instruction is only executed once after the last syscall, we will replace it with a jump for terminating the replaying process actively.

c) Extensible interval replaying design challenge: This work proposes an extensible and general checkpoint loader for replaying intervals. This loader is a self-contained program with complete text and data segments and can directly use all library functions. The main challenge is the memory layout collisions between the loader and the target program, including the stack collision and the text and data segments collision. To avoid these collisions, we use gem5 to initialize the target program's stack address and the target program's break pointer (brk) to other places to reserve enough memory space for the loader's stack space and text and data segments.

III. EVALUATION

A. Experimental Setup

This work performs evaluations with the SonicBOOM processor deployed on the Digilent Genesys 2 FPGA board. Table 1 lists the critical processor parameters. The SPEC CPU2006 and CPU2017 benchmarks with reference inputs are used.

TABLE I
THE PROCESSOR PARAMETERS

Component	Parameter
Core	50 MHz, 2-wide fetch, 4-wide issue, 16-entry load/store queue, 64-entry reorder buffer, TAGE branch predictor
L1-I Cache	16 KB, 64 B line, 4-way, 2 cycle access time
L1-D Cache	16 KB, 64 B line, 4-way, 2 cycle access time
L2 Cache	512 KB, 8-way, 40 cycle

For each benchmark, we sample every 200 million instructions and record the values of hardware counters. Figure 1 presents the sampling results for *523.xalancbmk_r*, including three hardware performance metrics. The results show that our design can correctly sample programs and obtain the microarchitecture behaviours of each program interval.

Furthermore, we generate the checkpoints for each benchmark for ten random intervals. Each interval contains 1 billion instructions for warming up and 200 million for evaluating the

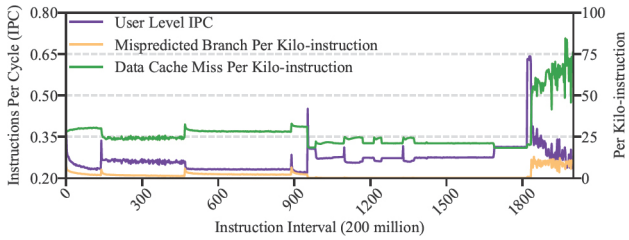

Fig. 1. The sampling results of *523.xalancbmk_r*.

replaying accuracy. Figure 2 presents the average relative percent error of the hardware performance metrics of the intervals replaying compared to the sampling results. The result shows that our design can accurately restore the microarchitecture behaviours of the specified program intervals on FPGA boards, especially for SPECInt benchmarks. At the same time, the low relative percent error also indicates that our solutions to the syscall handling and the memory layout collisions are effective.

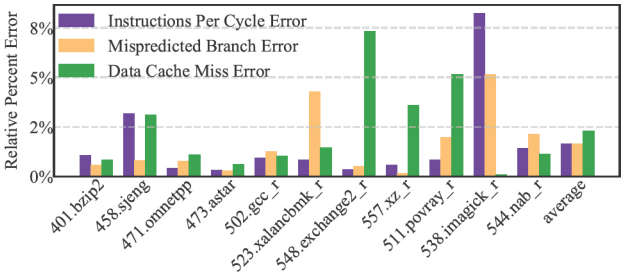

Fig. 2. The relative percent error of the hardware performance metrics of the benchmark intervals replaying compared to the sampling results.

IV. CONCLUSION

This work proposed a hardware-software cooperative interval-replaying solution for alleviating the problems caused by the resource limitation of FPGA boards. It supports architecture researchers in finding the desired program intervals and replaying them on FPGA boards. Experimental results show that our solution is effective.

V. ACKNOWLEDGEMENT

We greatly thank the anonymous reviewers for their insightful comments. This work was supported by the National Key R&D Program of China (Grant no. 2022YFB4500500). Chun Yang and Xu Cheng are the Corresponding authors of this paper.

REFERENCES

[1] J. Zhao *et al.*, "SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine," In *4th RISC-V Workshop*, 2020.
[2] H. Patil *et al.*, "ELFies: Executable region checkpoints for performance analysis and simulation," in *CGO*, 2021.
[3] J. Ringenberg and T. N. Mudge, "SuiteSpecks and SuiteSpots: A methodology for the automatic conversion of benchmarking programs into intrinsically checkpointed assembly code," in *ISPASS*, 2009.
[4] H. Patil *et al.*, "PinPlay: A framework for deterministic replay and reproducible analysis of parallel programs," in *CGO*, 2010.
[5] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, 2011.