Flexible Timing Simulation of RISC-V Processors with Sniper

Neethu Bal Mallya
National University of Singapore
Singapore
neethu@comp.nus.edu.sg

Cecilia Gonzalez-Alvarez
Ghent University
Belgium
Cecilia.GonzalezAlvarez@ugent.be

Trevor E. Carlson
National University of Singapore
Singapore
tcarlson@comp.nus.edu.sg

ABSTRACT

RISC-V has become an important option for both academia and industry when considering new microprocessor designs. The open instruction set allows designs to be tailored for next-generation processor goals.

Sniper is a next-generation parallel multicore simulator, which allows trading-off simulation speed for accuracy with a range of simulation options. Combining flexibility with ease of use allows computer architects to use it for the evaluation of next-generation microarchitectural features.

This work presents an extended version of Sniper which enables support for instruction set architecture (ISA) flexibility and introduces support for RISC-V. In this work, we provide a detailed look at the updated Sniper infrastructure that can be used for exploring future architectural and microarchitectural directions on the RISC-V ISA.

CCS CONCEPTS

• **Computer systems organization** → **Reduced instruction set computing**; • **General and reference** → Performance;

KEYWORDS

RISC-V, Sniper, Simulation Infrastructure

ACM Reference Format:
Neethu Bal Mallya, Cecilia Gonzalez-Alvarez, and Trevor E. Carlson. 2018. Flexible Timing Simulation of RISC-V Processors with Sniper. In *Proceedings of Second Workshop on Computer Architecture Research with RISC-V (CARRV 2018)*, 4 pages.

1 INTRODUCTION

Simulation is essential for the design and evaluation of new computer architectures. The Instruction Set Architecture (ISA) is the interface between hardware and software and is a major portion of what makes up an architecture. Simulating the performance of the microarchitectural implementation of an ISA is crucial component for design space exploration of next-generation designs. Computer architects, therefore, rely on timing simulation to drive the design process to quantify potential performance improvements.

RISC-V, the latest RISC-based ISA from Berkeley [6], is distributed with an open license, a significant departure from mainstream processor ISAs today. With its straight-forward and modular ISA, RISC-V was engineered to suit a variety of processors, from extremely small implementations [2, 14] to large, high-performance implementations [9].

Building simulation infrastructure to evaluate high-performance processors demands parallelism and higher levels of abstraction to keep simulations within a typically limited time budget. Sniper [7] is a next-generation parallel multi-core open-source simulator based on interval simulation [13]. It is a fast and flexible simulator which allows trading off simulation speed for accuracy with a range of simulation options. Sniper first supported Intel's x86 ISA, and this work provides an overview of the proposed extension of the Sniper infrastructure to flexibly support additional architectures. In this work, we present an overview of Sniper internals and the updates needed to support new ISAs. Sniper comes with a number of analytical techniques to ease in the evaluation of processor designs; enabling new ISAs is one way to improve the flexibility of Sniper to new dimensions.

This paper is organized as follows. An overview of related work is presented in Section 2. Section 3 discusses the Sniper simulator itself, with Section 4 describing the details of our extensions to support RISC-V. Section 5 describes future directions, and finally we conclude in Section 6.

2 BACKGROUND

RISC-V is a general purpose architecture that serves as a basis for a variety of projects from industry and academia [1]. The open ISA and the collection of related software tools enable easy collaboration for users to optimize their design for specific functionality. The fixed user-level ISA also ensures software compatibility and longevity of the architecture [18].

The existing RISC-V architecture-level software implementations include ISA functional simulators [4], full system emulators [3] and timing simulators [15, 17]. The current work explores two simulators, namely Spike [4] and rv8 [10]. Spike is the reference RISC-V ISA functional simulator. It serves as a reference model and can be extended to add new instructions. The rv8 simulator is an additional RISC-V instruction set simulation suite comprised of a high performance x86-64 binary translator, a user mode simulator, a full system emulator, and an ELF binary analysis tool.

Sniper [7, 8] is a parallel multi-core simulator based on mechanistic core models. Sniper provides a range of flexible simulation options to explore a variety of different homogeneous and heterogeneous multicore architectures, as well as a Python-based runtime environment that allows for analysis and simulator control.

In the original execution-driven simulation mode, Sniper runs as a tool in Intel's Pin [16] dynamic instrumentation framework to produce timing results for the target microarchitecture. In the standalone replay mode, Sniper can be used for collecting and playing back instruction traces. The Sniper instruction trace format (SIFT) files are collected and stored on disk (in the case of single-threaded applications) or generated on the fly and used for bi-directional communication between the frontend and backend Sniper components. The details of the SIFT-based replay mode is discussed in Section 3. This alternative mode can be used to run multiple single-threaded and multi-threaded workloads on a single timing simulator instance. It is this SIFT-based connection that allows for the use of a variety of tools to act as the Sniper functional frontend, which is explored in this work to port RISC-V architecture to Sniper.

3 SNIPER OVERVIEW

3.1 Software Components

This section provides a high-level description and organization of Sniper internals. The main components of the Sniper simulator include the frontend, SIFT traces and backend. Figure 1 shows an overview of Sniper's internals.

Frontend. The functional frontend of Sniper supports different tools that can be re-modeled as bi-directional SIFT trace recorders. This component collects the application's dynamic instruction state that connects to a standalone Sniper timing instance. Typically, this is done with binary instrumentation tools such as Pin, but instruction set simulators allow for cross-ISA evaluation. Its main components are:

- Control: Module that opens and closes the communication files (SIFT) and changes the instrumentation mode between fast-forwarding, warmup and detailed simulation. For instance, if only a region of interest (ROI) in the application is to be simulated in detail, the code sections outside of the ROI could be simulated in functional cache warming mode (where the memory subsystem is warmed before ROI execution) or could be fast-forwarded without cache warming.
- Instruction instrumentation callbacks: Module that intercepts each executed instruction. It is responsible of counting instructions in fast-forward mode, sending instruction information required by the SIFT format in detailed instrumentation mode, and handling Magic Instructions (special instructions that do not modify the architectural state of the processor but are used by the application code to communicate with the Sniper backend. e.g. start and end ROI).
- System call instrumentation: Module for the interception of system calls to let Sniper backend simulate them.
- Thread instrumentation: Module that intercepts the creation, synchronization and termination of threads.

Sniper Instruction Trace File Format (SIFT). SIFT is a trace file format developed for use with Sniper which contains the dynamic instruction stream generated by the frontend. The format describes instruction execution order, but also contains additional dynamic information (like memory addresses) that are needed for detailed timing simulation.

Depending on Sniper's simulation mode, the traces can be stored for later use, or used immediately. In the execution-driven simulation mode, Sniper's backend consumes the traces generated by the frontend in real time, and therefore they are not stored. However, in the standalone replay mode, SIFT traces can be recorded and stored. These traces can be later executed with Sniper backend to obtain performance estimates.

Scheduler and Backend. This is the main component of the timing simulator. The configurable thread scheduler controls and maps program execution to the simulating cores. Each application thread in the original program will have a matching thread in the Sniper backend. In the case of multi-program workloads, each application has a dedicated thread in the backend. The core models estimate thread progress and the cache hierarchy and branch predictors are modeled by the execution-driven performance models.

3.2 Modeling Abstractions

Sniper uses a number of abstractions that provide more direct control, or simplify its implementation.

Core abstractions. Sniper currently provides two high-abstraction core models based on interval simulation [12, 13]. The simulator supports the original interval analysis-based version [7] as well as the instruction-window (IW) centric model [8] that allows for modeling additional restrictions (and provides higher accuracy) of industry processor designs. These mechanistic modeling techniques provide a more accurate representation of the core timing.

Perfect Memory Dependence Prediction. Perfect memory dependence prediction is something that is currently not possible to do on a single-pass, execution driven simulator. Currently, Sniper supports perfect memory dependence prediction, and does not limit forward progress because of imperfect prediction of memory dependencies. To support imperfect memory dependence prediction, we will require the implementation of the dependence prediction unit, and will need to adding false dependencies and additional execution delays caused by mispredictions of this prediction unit to simulate the slowdown present in the hardware.

Operating System Emulation. Operating System emulation allows for an abstract understanding of the application. A variety of OS wakeup scenarios are presented and this allows for abstract system modeling in Sniper.

4 IMPLEMENTATION

The improvements to Sniper and the corresponding tools were based on recent versions of Sniper, Spike and rv8[1].

Frontend. Two of the existing RISC-V simulators, namely rv8 and Spike, were updated to support SIFT generation. The Spike simulator and the user mode x86-64 binary translator (`rv-jit`) provided by rv8 simulation suite is explored in this work as the dynamic instrumentation frontend of Sniper.

The `Sift::Writer` class was used to generate the traces in these functional simulators. The dynamic information - memory addresses for loads and stores, branch directions (taken/not taken), and executed/masked information for predicated instructions are passed through the `Instruction()` function. Static instruction information is obtained through the `getCode` callback which needs to be provided in the `Sift::Writer` constructor. The `getCode` function copies the requested range of the instruction memory into `Sift::Writer`. This copy of instruction binary can be used at the

[1]The versions of the tools used are: Sniper(`v6.1`), Spike (`2dbcb01ca1c026b8`) and rv8 (`7fbe9fbac87365ac`)

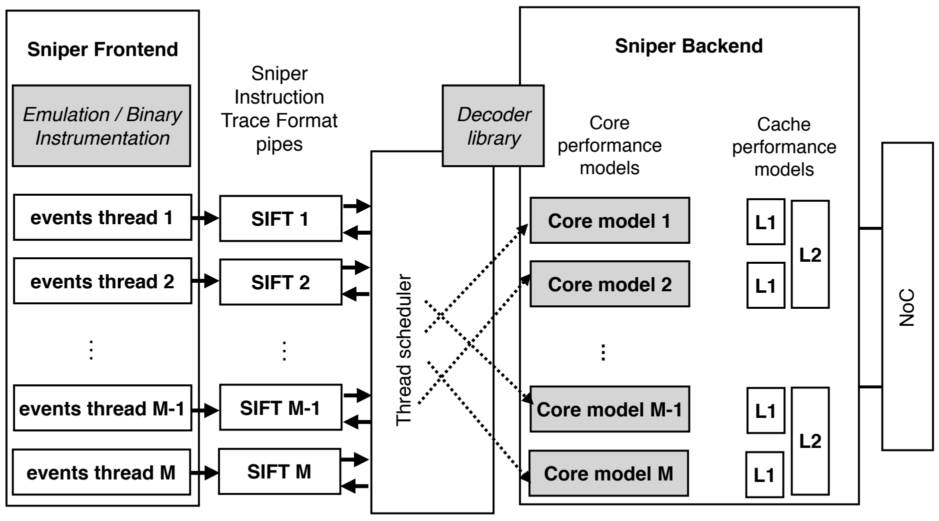

Figure 1: Sniper Internals. Shaded modules represent the main changes applied to port Sniper to a new architecture.

backend decoder to disassemble the instruction and derive static information.

SIFT. The SIFT format was designed at a high-level and no major modifications were needed to the data format. The SIFT traces from rv8/Spike were used to run Sniper in replay mode, but simultaneous generation and consumption of traces are also supported.

Backend. The Sniper backend is quite flexible with respect to instructions, dependencies and the representation of micro-operations. But there are a number of components that needed to be updated to support the alternative microarchitectures.

- *Decoder Library* Sniper uses a series of architectural agnostic methods to implement the decoding phase of the processor. A flexible decoder format was added in the backend to determine the static instruction information. To support the new microarchitecture, sub-classes for `Decoder` and `InstructionDecoded` were created and all the methods were implemented.
- *Core Model* The parameters like description of ports / functional units, latencies and branch prediction model were updated in the Sniper Core Model.
- *Configuration files* The architectural parameters like details of memory hierarchy are configurable in runtime. These parameters were adapted to the target architecture implementation. A new configuration file was also added to the stackable configuration options of Sniper to resemble a BOOM [9] processor[2].

5 CURRENT IMPLEMENTATION AND NEXT STEPS

Our current implementation allows for a flexible architecture base, with lower-level customization, similar to ASIM [11]. Nevertheless, our simulation platform occurs at a higher level than ASIM or gem5, to allow for more abstract micro-architecture implementations (perfect branch prediction and memory dependence prediction).

Our next steps include comparison with timing simulators to provide an understanding of which features need to be provided in order to maintain high accuracy when simulating processors such as the RocketChip [5] and BOOM [9] processor cores.

[2]We used the default BoomConfig from http://github.com/ccelio/boom-template (ad2412bd19c9dd7d)

6 CONCLUSION

This work presented an infrastructure extension of Sniper multi-core simulator to enable multi-architectural support. We present an overview of the updates needed to flexibly support new ISAs on Sniper, and the result is a working simulator that can be used for exploring future architectural developments on RISC-V. Our next steps are to improve the features of Sniper to allow for a detailed comparison with cycle-level processor implementations.

ACKNOWLEDGEMENTS

This research was supported by a grant from the Singapore Ministry of Education Academic Research Fund Tier 1. Additional support is provided by the European Research Council (ERC) Proof-of-Concept Grant No. 713632.

REFERENCES

[1] 2018. LowRISC Foundation Members Directory. Available at `https://riscv.org/membership`.
[2] 2018. LowRISC Homepage. Available at `http://www.lowrisc.org`.
[3] 2018. RISC-V QEMU. Available at `https://github.com/riscv/riscv-qemu`.
[4] 2018. RISC-V Spike. Available at `https://github.com/riscv/riscv-tools`.
[5] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[6] Krste Asanović and David A. Patterson. 2014. *Instruction Sets Should Be Free: The Case For RISC-V.* Technical Report UCB/EECS-2014-146. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html

[7] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC).* 52:1–52:12. http://dx.doi.org/10.1145/2063384.2063454

[8] Trevor E. Carlson, Wim Heirman, Stijn Eyerman, Ibrahim Hur, and Lieven Eeckhout. 2014. An Evaluation of High-Level Mechanistic Core Models. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 3, Article 5 (Aug. 2014), 25 pages. https://doi.org/10.1145/2629677

[9] Christopher Celio, David A. Patterson, and Krste Asanović. 2015. *The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor.* Technical Report UCB/EECS-2015-167. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html

[10] Michael D. Clark and Bruce Hoult. 2017. rv8: a high performance RISC-V to x86 binary translator. In *First Workshop on Computer Architecture Research with RISC-V (CARRV).* Boston, MA, USA. https://carrv.github.io/2017/papers/clark-rv8-carrv2017.pdf

[11] Joel Emer, Pritpal Ahuja, Eric Borch, Artur Klauser, Chi-Keung Luk, Srilatha Manne, Shubhendu S. Mukherjee, Harish Patil, Steven Wallace, Nathan Binkert, Roger Espasa, and Toni Juan. 2002. Asim: A Performance Model Framework. *Computer* 35, 2 (2002), 68–76. https://doi.org/10.1109/2.982918

[12] Stijn Eyerman, Lieven Eeckhout, and James E. Smith. 2007. Studying Compiler-Microarchitecture Interactions Through Interval Analysis. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT).* IEEE Computer Society, Washington, DC, USA, 406–422. https://doi.org/10.1109/PACT.2007.70

[13] Davy Genbrugge, Stijn Eyerman, and Lieven Eeckhout. 2010. Interval Simulation: Raising the Level of Abstraction in Architectural Simulation. In *Proceedings of the 16th IEEE International Symposium on High-Performance Computer Architecture (HPCA).* 307–318. https://doi.org/10.1109/HPCA.2010.5416636

[14] Jan Gray. 2016. GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator. In *Proceedings of the 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* 17–20. https://doi.org/10.1109/FCCM.2016.12

[15] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. 2017. FireSim: Cycle-Accurate Rack-Scale System Simulation using FPGAs in the Public Cloud. In *7th RISC-V Workshop.* Milpitas, CA, November.

[16] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).* 190–200. https://doi.org/10.1145/1065010.1065034

[17] 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).* Boston, MA, USA. https://carrv.github.io/2017/papers/roelke-risc5-carrv2017.pdf

[18] 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.* Technical Report UCB/EECS-2016-118. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html