FastISS RISC-V VP++: A Simulation Performance Evaluation of RVV Workloads

Manfred Schlägl and Daniel Große

Institute for Complex Systems, Johannes Kepler University Linz, Austria
manfred.schlaegl@jku.at, daniel.grosse@jku.at

Abstract

In this paper, we consider the SystemC-based open-source RISC-V VP++ *with support for the* RISC-V "V" Vector Extension *(RVV), whose interpreter-based* Instruction Set Simulator *(ISS) has recently been significantly optimized, as presented in [1]. While the original paper examined simulation performance gains using classical, non-vectorized workloads, this paper focuses on the gains on a workload vectorized using RVV.*

Introduction

The open and royalty-free *Instruction Set Architecture* (ISA) RISC-V [2] has gained significant traction in recent years. Particularly noteworthy is the high degree of modularity of RISC-V, with a large number of standardized extensions that can be integrated as a tailor-made system to perfectly meet application-specific requirements. One outstanding extension is the *RISC-V "V" Vector Extension* (RVV), which adds 600+ new instructions and 32 vector registers, and with this brings extensive *Single Instruction, Multiple Data* (SIMD) capabilities to the RISC-V architecture. Unlike *Single Instruction, Single Data* (SISD), SIMD allows operations to be performed not only on individual data elements, but on whole *vectors* of elements simultaneously. With this, SIMD exploits the *Data-Level Parallelism* (DLP) often found in algorithms for modern multimedia and machine learning applications, significantly improving their data throughput and overall performance [3].

The *RISC-V VP++* considered in this paper is an open source, SystemC TLM (IEEE 1666, [4]) based *Virtual Prototype* (VP) with support for RVV [5, 6]. VPs are high-level, executable models of the entire *Hardware* (HW) platforms which can run unmodified production *Software* (SW) [7] and therefore allow early design space exploration, parallelization of HW and SW development, and system evaluation and validation [8]. Central elements of HW platforms are processors, which are modeled as interpreter-based RISC-V RV32 and RV64 *Instruction Set Simulator*s (ISSs) in *RISC-V VP++*. Recently, extensive optimizations were made to these ISSs to significantly increase simulation performance [1]. In the remainder of this paper, the optimized ISS will be referred to as *FastISS*. The core optimizations of *FastISS* are the addition of (i) the *Dynamic Basic Block Cache* (DBBCache), which generates an alternative representation of the executed code, the *Dynamic Basic Block Graph* (DBBG), to efficiently cache data needed for instruction processing (instruction fetch, decode and dispatch), and (ii) the *Load/Store Cache* (LSCache), which enables direct translation of in-simulation virtual addresses to host system memory addresses, thereby almost completely eliminating calls to the memory interface. Based on the DBBCache, other optimization techniques have also been applied, such as *computed goto*, *threaded code*, *fast and slow paths*, ... The authors of [1] evaluated their *FastISS RISC-V VP++* based on a set of classic, non-vectorized benchmark workloads, and achieved a simulation performance of up to 406.97 *Million Instructions per Second* (MIPS) and average speedup factors of 8.98 over the original unoptimized *RISC-V VP++* and 1.65 over the *Spike* simulator.

In this paper, we focus on the performance gains that can be achieved by *FastISS RISC-V VP++* for workloads vectorized with RVV. We compare the simulation performance based on achieved *Frames Per Second* (FPS) by a classic game in a non-vectorized and vectorized variant on the original *RISC-V VP++* and its new *FastISS* version.

Performance Evaluation

For our performance evaluation we consider fbDOOM-RISCV provided by Semidynamics, which is based on a Linux port of a classic game from the 1990s [9]. Semidynamics optimized fbDOOM-RISCV to speed up execution using RVV and adapted the build system to produce an executable without the optimizations (*non-vectorized*) and an executable with the optimizations (*vectorized*). For our experiments, we further adapt fbDOOM-RISCV: (i) We disable the limitation to 35 FPS, and (ii) we add instrumentation to measure the average FPS and *Million Executed Instructions Per Frame* (MIPF). Note that the FPS are calculated relative to the real wall clock time of the simulation host and not the simulation time, i.e. the measurements refer to the simulation performance and not the performance

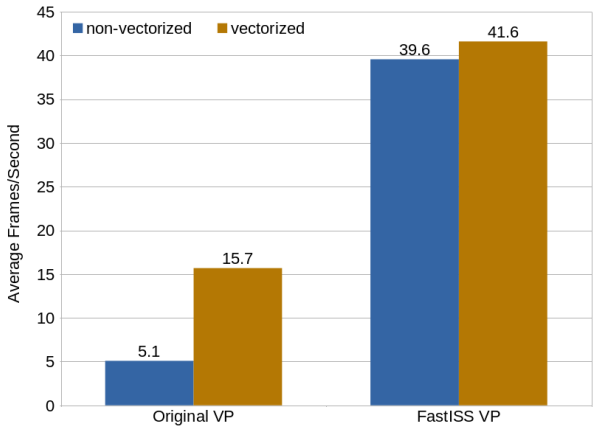

Figure 1: *Average FPS on achieved on* RISC-V VP++

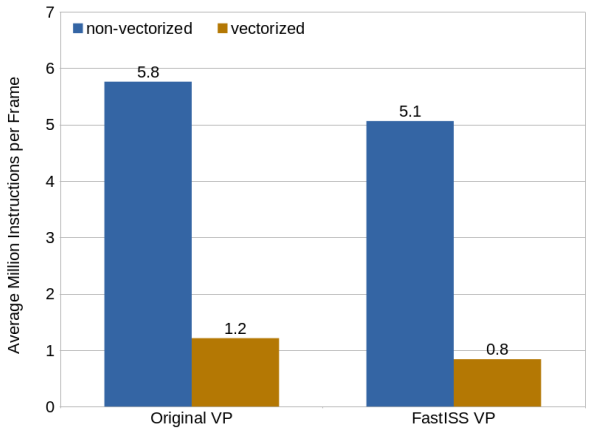

Figure 2: *Average MIPF executed by* RISC-V VP++

of RVV itself. As build and execution environment we chose a recent version of *GUI-VP Kit* which was initially introduced in [10]. *GUI-VP Kit* comes with support for RVV, and provides a full RISC-V development and simulation environment for interactive graphical Linux applications on *RISC-V VP++*. Its recent version uses *gcc-14.1* and *linux-6.10.4*. All measurements are performed on a host system with an AMD® Ryzen™ 7 PRO 6850U 8-core processor running at 2.7 GHz, with 32 GiB RAM.

Figure 1 and Figure 2 present the results of the fbDOOM-RISCV FPS and MIPF measurements, respectively. The *non-vectorized* version is shown in blue, the *vectorized* in brown. The left-hand side shows the results for the original *RISC-V VP++* with its unoptimized RV64 ISS (*Original VP*), the right-hand side shows the results for *RISC-V VP++* with the optimized RV64 *FastISS* (*FastISS VP*).

As can be seen in Figure 1 there is a significant increase in FPS and therefore simulation performance from the *Original VP* to the *FastISS VP*. For *vectorized*, which is the focus of this paper, we observe an improvement by a factor of 2.65. However for *non-vectorized* the improvement is significantly higher with 7.78. Another observation is that there is a significant difference in performance increase for *non-vectorized* and *vectorized*. For the *FastISS VP* we see an increase of 1.05, while for the *Original VP* the improvement is 3.09. Both observations can be explained as follows. In *vectorized* implementations, the work is shifted from many but simple instructions (SISD) to fewer but more complex instructions (SIMD). This can also be clearly seen in Figure 2: Instead of 5.8 and 5.1 MIPF for *non-vectorized*, we see only 1.2 and 0.8 MIPF for *vectorized*. In terms of an ISS, this means that there is a shift in the overall computational cost from instruction processing (instruction fetch, decode and dispatch) to instruction execution. Since the optimizations in the *FastISS VP* focus almost exclusively on reducing the overhead of ISS instruction processing, *non-vectorized* implementations can generally benefit more from the optimizations than *vectorized* implementations.

This concludes our simulation performance evaluation of RVV workloads on the *FastISS RISC-V VP++* presented in [1]. The presented experiments show a significant improvement in simulation performance by a factor of **2.65** for a vectorized workload. However, it is also shown that non-vectorized workloads can generally benefit more from the *FastISS* optimizations than vectorized workloads.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] Manfred Schlägl and Daniel Große. "Fast Interpreter-Based Instruction Set Simulation for Virtual Prototypes". In: *DATE*. 2025. URL: https://ics.jku.at/files/2025DATE_Fast_Interpreter-based_ISS.pdf.

[2] Andrew Waterman and Krste Asanović. *The RISC-V Instruction Set Manual; Volume I and II*. SiFive Inc. and UC Berkeley. 2019.

[3] Michael J. Flynn. "Very high-speed computing systems". In: *IEEE* 54.12 (1966), pp. 1901–1909.

[4] *IEEE Standard for Standard SystemC Language Reference Manual*. DOI: 10.1109/IEEESTD.2023.10246125. URL: https://doi.org/10.1109/IEEESTD.2023.10246125.

[5] Manfred Schlägl, Moritz Stockinger, and Daniel Große. "A RISC-V "V" VP: Unlocking Vector Processing for Evaluation at the System Level". In: *DATE*. 2024, pp. 1–6. DOI: 10.23919/DATE58400.2024.10546838. URL: https://ics.jku.at/files/2024DATE_RISCV-VP-plusplus_RVV.pdf.

[6] Manfred Schlägl, Christoph Hazott, and Daniel Große. "RISC-V VP++: Next Generation Open-Source Virtual Prototype". In: *Workshop on Open-Source Design Automation*. 2024. URL: https://ics.jku.at/files/2024OSDA_RISCV-VP-plusplus.pdf.

[7] Vladimir Herdt, Daniel Große, and Rolf Drechsler. *Enhanced Virtual Prototyping: Featuring RISC-V Case Studies*. Springer, 2020. DOI: 10.1007/978-3-030-54828-5.

[8] Tom De Schutter. *Better Software. Faster!: Best Practices in Virtual Prototyping*. Synopsys Press, Mar. 2014. ISBN: 978-1617300134.

[9] *fbDOOM-RISCV*. https://github.com/semidynamics/fbDOOM-RISCV.

[10] Manfred Schlägl and Daniel Große. "GUI-VP Kit: A RISC-V VP Meets Linux Graphics - Enabling Interactive Graphical Application Development". In: *GLSVLSI*. 2023, pp. 599–605. DOI: 10.1145/3583781.3590253. URL: https://ics.jku.at/files/2023GLSVLSI_GUI-VP_Kit.pdf.