RISC-V Opt-VP: An Application Analysis Platform Using Bounded Execution Trees

Jan Zielasko[1,2], Rune Krauss[1], Marcel Merten[1], Rolf Drechsler[1,2*]

[1]Institute of Computer Science, University of Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, Germany

Abstract

Tailoring hardware to applications significantly increases their performance, which is required to meet the rising demand for resource-limited devices in the area of embedded systems. While RISC-V facilitates application-specific solutions due to its extensibility, Virtual Prototypes (VPs) enable early software development before the actual hardware is built shortening the time-to-market. Although the RISC-V VP ecosystem already offers many useful tools to aid development there is still room for improvement, especially in analyzing applications for hardware optimization. To address the aforementioned issue and expand the mentioned ecosystem with a tool, this work presents RISC-V Opt-VP, which generates bounded execution trees in order to analyze applications. An embedded application analysis case study illustrates that promising instruction sequences are found which can also be merged to further improve their execution coverage, enabling efficient hardware designs.

Introduction

With the ever-increasing demand for high-performance applications in areas such as embedded systems, it is becoming increasingly important to optimize hardware in order to meet time-to-market constraints [1].

RISC-V is an instruction set architecture that is characterized by its modularity and extensibility [2]. It facilitates efficient and application-specific solutions, which is particularly ideal for resource-limited devices in areas such as embedded systems and IoT.

Virtual Prototypes (VPs) such as RISC-V VP[1] [3] enable early software development and testing by providing an executable hardware platform implemented using SystemC transaction-level modeling [4].

The RISC-V VP ecosystem offers some efficient tools to aid development. These include a co-simulator [5] and 3D visualizations of symbolic execution for debugging purposes [6]. However, there is still a need to analyze applications for hardware optimization.

To address this issue and expand tool diversity, in this extended abstract we propose *RISC-V Opt-VP*, which constructs bounded execution trees based on applications for tailoring hardware. Using an embedded application analysis case study, we illustrate that promising instruction sequences are found which can be merged to increase execution coverage, enabling efficient hardware designs. To stimulate further research, our tool is provided as open-source software[2] [7, 8].

*Corresponding author: `Jan.Zielasko@DFKI.de`. This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within projects Scale4Edge under grant no. 16ME0127, ECXL under grant no. 01IW22002 and VE-HEP under grant no. 16KIS1342.

[1] https://github.com/agra-uni-bremen/riscv-vp
[2] https://github.com/agra-uni-bremen/opt-vp

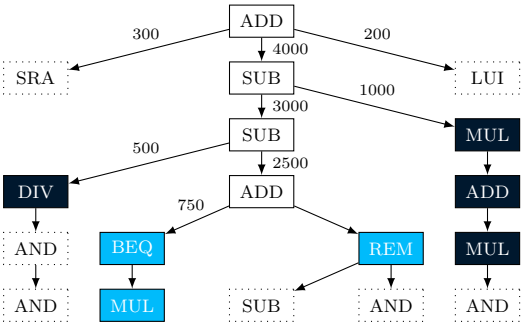

Figure 1: ☐ *Default*, ■ *Subsequence*, ■ *Variant*

RISC-V Opt-VP

RISC-V Opt-VP extends RISC-V VP [3] for tailoring hardware to application requirements. It traces the execution of applications by creating a k-tree with a predetermined maximum depth k for each encountered instruction: Figure 1 shows a simplified 6-tree for `ADD`. All trees are analyzed using a scoring function that evaluates a set of defined metrics such as affected instructions (length) and execution coverage so that the most promising instruction sequence with the highest score (here: `ADD`, `SUB`, `SUB`, `ADD`) can be identified. `SRA` and `LUI` (highlighted by dotted rectangles) are missing as the score for these instructions was not sufficient.

As applications can consist of instruction sequences that only cover a small fraction of the total execution, it is worthwhile to improve execution coverage through merging sequences. In order to offer more opportunities for merges, there is a ■ *Subsequence* and ■ *Variant* mode in addition to the ☐ *Default* optimization. While the subsequence mode extends promising instruction sequences past its end, the variant mode iterates over each branch to generate additional sequences.

Table 1: *Embedded application analysis case study*

	Best sequence		Merged sequence		
Application	Length	Score	Mode	Merges	Score
aha-mont64	11	1,786,752	Default	9	2,969,930
crc32	12	2,101,248	Default	6	4,027,420
edn	5	2,956,800	Subseq.	16	3,980,830
huffbench	1	661,439	Default	10	1,754,030
matmult-int	5	3,732,740	Subseq.	7	3,839,160
md5sum	24	958,464	Variant	12	4,473,090
minver	5	500,570	Default	6	954,229
nettle-aes	31	1,018,784	Full	11	8,016,290
nettle-sha	1	670,684	Subseq.	14	1,161,800
nsichneu	1	1,227,108	Variant	4	1,286,260
picojpeg	1	814,212	Full	14	2,494,210
primecount	2	1,399,464	Default	7	3,047,680
qrduino	1	619,974	Default	11	3,152,640
slre	7	749,049	Default	8	1,565,790
ud	2	298,888	Full	14	1,223,260
Average	7	1,299,745	Default	10	2,929,775

Case Study

In this section, we summarize an embedded application analysis case study created using RISC-V Opt-VP.

Embedded applications were taken from the de facto standard Embench™ suite[3] and compiled using the *-O3* level. Every application was traced to generate an execution k-tree for each encountered instruction, where $k = 50$ was determined experimentally. In the first experiment, our tool iterated over each tree and computed the most promising instruction sequence with the score described in the last section, subtracting a correction value to adjust its suitability for targeted hardware optimizations. The best sequence was then selected from all the trees. To compare this result, in the second experiment, we sorted all promising sequences w. r. t. the number of instruction mappings and merged them while respecting data dependencies. In addition to the used single modes, *Subseqence* and *Variant* were combined, which is called *Full* below.

The experimental results are shown in Table 1. Suitable sequences for hardware optimization could be identified for all applications and merging more than doubles their coverage with an average of 10 merges. For matmult-int, the best sequence with the highest score corresponds to an effective coverage of around 40 % resulting in merges hardly improving this score. However, there are many sequences of length 1 where merges are worthwhile: The coverage for nettle-sha can be almost doubled by using the subsequence mode. Modes like *Full* are also useful, e. g. for nettle-aes, where the score of the merged sequence is about 8 times higher compared to the best discovered sequence.[4]

[3] https://embench.org

[4] As sequences were sorted, using the full mode does not automatically lead to the highest score due to different mappings.

Discussion and Future Work

In this extended abstract we presented a VP-driven tool called *RISC-V Opt-VP* to analyze applications for hardware optimization. Using an embedded application analysis case study, we found that the generation of bounded execution trees can be used to identify promising instruction sequences that cover a large fraction of the total execution. Additionally, merging instructions more than doubles the achieved coverage, which together enables efficient hardware designs. Based on these high-level results, hardware can be tailored to applications by accelerating analyzed sequences that require a high execution time. By identifying similarities in different sequences and merging them, the design of a single hardware accelerator or new instruction that covers a large fraction of the total execution is possible, while the performance loss is negligible compared to building multiple accelerators, thus saving expensive hardware costs.

Future work will be directed towards further investigation of hardware design and construction of actual hardware. On the one hand, further metrics used by our scoring function, such as input and output of instruction sequences, are to be studied in order to fine-tune the identified sequences to a specific optimization. On the other hand, it is planned to design coarse-grained reconfigurable architectures to accurately measure possible hardware acceleration for the covered sequences.

References

[1] L. De Micco, F. Vargas, and P. Fierens. "A Literature Review on Embedded Systems". In: *Latin America Transactions* 18.2 (2020), pp. 188–205.

[2] A. Waterman et al. "The RISC-V Instruction Set". In: *IEEE Hot Chips 25 Symposium (HCS)*. IEEE, 2013.

[3] V. Herdt et al. "RISC-V based virtual prototype". In: *Journal of Systems Architecture* 109 (2020).

[4] F. Ghenassia. *Transaction-Level Modeling with SystemC*. New York: Springer, 2010.

[5] N. Bruns, V. Herdt, and R. Drechsler. "Processor Verification Using Symbolic Execution: A RISC-V Case-Study". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[6] J. Zielasko et al. "3D Visualization of Symbolic Execution Traces". In: *Forum on Specification & Design Languages (FDL)*. IEEE, 2022, pp. 1–8.

[7] J. Zielasko and R. Drechsler. "Virtual Prototype Driven Application Specific Hardware Optimization". In: *Forum on Specification & Design Languages (FDL)*. IEEE, 2023, pp. 1–8.

[8] J. Zielasko et al. "Improving Virtual Prototype Driven Hardware Optimization by Merging Instruction Sequences". In: *27th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 2024, pp. 73–78.