Towards ML-based Performance Estimation of Embedded Software: A RISC-V Case Study *

Weiyan Zhang[1], Muhammad Hassan[1,2], Rolf Drechsler[1,2]
[1]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
[2]Institute of Computer Science, University of Bremen, Bremen, Germany
{weiyan.zhang, muhammad.hassan}@dfki.de, drechsler@uni-bremen.de

Abstract

Performance estimation of embedded software techniques mimic the behavior of real hardware, consistently navigating a balance between simulation accuracy and speed. Designers usually use real hardware, simulators, or static analyzers to obtain the performance. However, these methods suffer from serious drawbacks as real hardware is not available in the early stage of the design process, simulators either do not support any timing accuracy or require large execution time, and static analyzers need details of the hardware microarchitecture. Recently, *Machine Learning* (ML) has been successfully applied to estimate performance, in particular the clock cycles. It can significantly facilitate the exploration of a wide range of microarchitecture solutions at the early stage, applicable across various architectures. In this paper, we delve into the advancements of ML-based performance estimation on RISC-V processors, leveraging performance statistics obtained directly from a fast functional simulator using built-in counters. The proposed approach uses dynamic analysis for feature extraction and different ML algorithms including regression algorithms and *Neural Network* (NN) for the generation of *Predictive Models* (PMs). We present a comprehensive analysis of their effectiveness across diverse RISC-V implementations.

1 Introduction

Performance, measured in clock cycles, is a crucial design factor in system development. Predicting executed cycles aids in profiling embedded software, helping developers identify and address performance bottlenecks, such as optimizing resource utilization and enhancing overall system efficiency. Techniques for estimating performance in embedded software involve a trade-off between simulation accuracy and speed, categorized as simulation and analytic-based models.

Simulation based approaches model system architectures at different levels of abstraction. Functional simulator allows fast prototyping but lack precision. At the *Electronic System Level* (ESL) [1], SystemC-based *Virtual Prototype* (VP) is often used before the detailed hardware implementation is finalized. This abstraction gives some speedup over cycle-accurate modelling at a low abstraction level. *Register Transfer Level* (RTL) simulation offers a high degree of accuracy in verifying the functionality of digital circuit designs. It can detect errors at an early stage, leading to reduced cost in the design process. However, its simulation speed is comparatively slow. To effectively explore potential options for different choices of processor, performance estimation of embedded software at a higher level of abstraction is necessary.

Analytic-based models are another option that can be linear or nonlinear. The basic idea is to collect the reference executed cycles and performance-related parameters, such as dynamic instruction counts, during hardware execution, and apply *Machine Learning* (ML) algorithms to train the models. The performance-related parameters are selected in such a way that the approach has the lowest dependency on microarchitectural and software details. To predict the performance of new software, a fast functional simulator is used to quickly obtain their performance-related parameters, which are then applied to the trained PMs.

As a free and open-source *Instruction Set Architecture* (ISA), RISC-V demonstrates considerable potential, particularly for embedded systems in various utilization domains. The RISC-V ecosystem provides a variety of implementations at different abstraction levels, establishing it as a versatile choice for a wide range of applications. In this paper, **we propose an ML-based methodology to estimate the performance of embedded software across various microarchitectures**. Considering that the real physical hardware may not be available, we employ a combination of fast functional simulator, cycle-accurate RISC-V implementations, and ML techniques to generate PMs. Subsequently, the PMs are combined with a fast functional simulator to enable fast and accurate prediction of the performance of new embedded software. We demonstrate the generalizability of our approach across multiple microarchitectures for the RISC-V ISA in [2][3][4], applying it to a cycle-accurate simulator and four real-world, cycle-accurate RTL cores of the RISC-V ISA. Our evaluation focuses on the performance of our approach using standard benchmarks from TACLeBench [5].

2 Methodology

In this section, we present our proposed approach for performance estimation, which involves coupling a fast functional simulator with ML-based models. An overview of our approach is depicted in Fig. 1. It encompasses two crucial phases: the training phase and the prediction phase. In the subsequent subsections, we will delve into a detailed

*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.

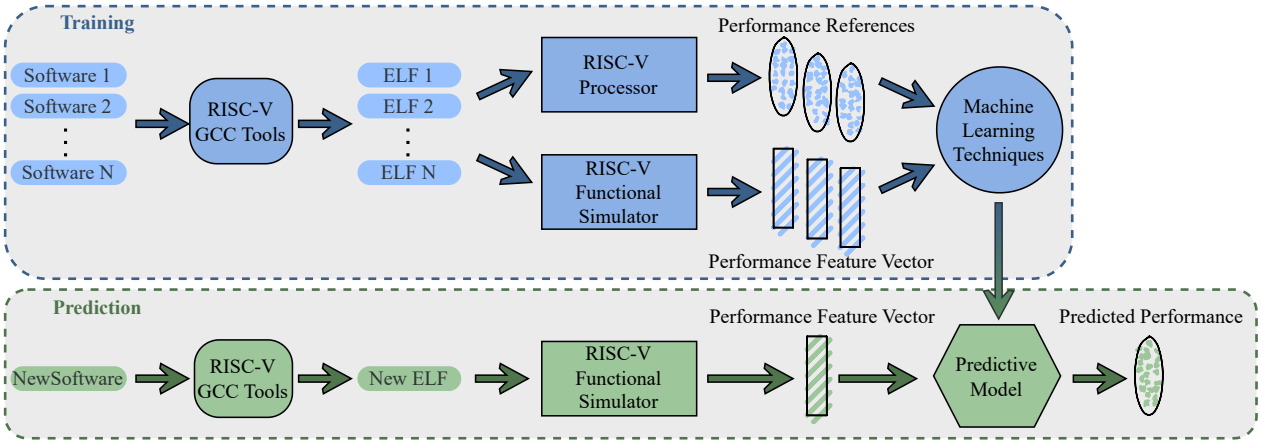

Figure 1 Performance estimation of embedded software workflow.

```
1 // Generate a specified number of SLTI instructions
2 // based on the value of num_slti
3 void slti_gen(int num_slti) {
4    int i = 0;
5    while (i < num_slti) {
6       bool slti = (i < 3);
7       i++;
8    }
9 }
```

Figure 2 Training software module generating the SLTI instruction.

explanation of each phase.

2.1 Predictive Model Training

During the training phase, a set of software is prepared. To generate the RISC-V binaries on our host computer, we take advantage of the cross-compilation. A set of training software is compiled with the RISC-V GNU Compiler Toolchain [6] to generate the *Executable and Linkable Formats* (ELFs). The ELFs of training software are then executed on a RISC-V processor to obtain the reference executed cycles from performance counters and on a fast functional simulator to obtain the dynamic information using built-in instruction counters. The RISC-V processor could be either a real physical hardware, such as a development board, a cycle-accurate VP, or a cycle-accurate implementation at RTL if physical hardware is not available. In this paper, we explore a cycle-accurate RISC-V VP [7] and four RISC-V implementations at the RTL (including Ibex [8], RSD [9], SweRV [10], and μRV32RTL [11]) to illustrate the generalizability of our approach across various RISC-V microarchitectures. The RTL cores were simulated using Verilator [12], an open-source Verilog/SystemVerilog simulator.

The foundation of our performance analysis is the extraction of dynamic instruction counts and reference executed cycles from runtime information. Dynamic instruction counts efficiently capture software execution dynamics and serve as crucial performance features. These features are then converted into vectors, forming the training dataset for model training. We employed supervised ML algorithms to predict the performance of new software. For RISC-V VP, we generated three models—PM1, PM2 and PM3—based on the different feature vectors. PM1 and PM2 [2] were trained using *Linear Regression* (LR) algorithm, incorporating the total instruction count and six RISC-V formats counts, respectively. In contrast, PM3 [3] employed the *Artificial Neural Network* (ANN) for training, utilizing individual instruction counts. For each RTL core, we implemented four classic ML algorithms: *Ordinary Least Squares* (OLS) regression, LR with *Mini Batch Gradient Descent* (MGD), ridge regression and ANN. OLS regression and LR with MGD are LR techniques that optimize the model's parameters differently. Ridge regression is a regularized form of LR that prevents overfitting. ANN is a more complex model capable of capturing nonlinear relationships among variables.

2.2 Performance Prediction

In this phase, the PM is tested by a set of new software selected from standard benchmarks. The benchmarks are compiled to generate the ELF files. Subsequently, each ELF file is executed on a functional simulator to obtain dynamic instruction counts (i.e. performance features). The performance feature vector is utilized as inputs to the PM to estimate the clock cycles.

To evaluate the performance of our PM, we calculated the *Absolute Percentage Error* (APE) metric for each performance counter of every benchmark, defined as

$$APE = \left| \frac{y - \hat{y}}{y} \right| * 100\%, \quad (1)$$

where y and $\hat{y}$ represent the actual and estimated clock cycle for each benchmark. The APE quantifies the extent of deviation between the real and estimated clock cycle.

To provide a more comprehensive evaluation, the *Mean Absolute Percentage Error* (MAPE) is computed. This involves summing up the APE values and dividing the sum by the total number of benchmarks.

3 Experimental Evaluation

3.1 Experimental Setup

In the training phase, for PM1, PM2, and PM3, we utilized 125 programs as the training software set. To generate the PM for each RTL core, we employed about 700 programs as the training software set. These programs were generated by providing various inputs to self-written sample programs and several standard benchmarks from

Table 1 Experimental Results of all Benchmarks used for Validation of PM1, PM2, and PM3.

Benchmark	#instr-exec.	VP		Ours		PM1		PM2		PM3	
		#Cycle	Time(ms)	Time(ms)	Speedup	#Cycle	APE	#Cycle	APE	#Cycle	APE
adpcm_dec	2 880 767	3 606 788	1 767	241	7.332	3 921 879	8.736%	3 836 719	6.375%	3 629 782	0.638%
adpcm_enc	2 898 910	3 636 185	1 751	244	7.176	3 946 579	8.536%	3 857 304	6.081%	3 653 801	0.484%
cubic	28 338 774	37 235 221	17 129	2 243	7.637	38580428	3.613%	38 487 287	3.363%	37 325 040	0.241%
deg2rad	510 732	659 172	332	56	5.929	695 312	5.483%	684 465	3.837%	658 399	0.117%
fft	3 678 523	5 020 639	2 573	319	8.066	5 007 945	0.253%	4 881 001	2.781%	5 207 390	3.720%
gsm_dec	9 168 157	12 320 332	5 942	779	7.628	12 481 536	1.308%	12 549 342	1.859%	11 838 296	3.913%
isqrt	1 002 079	1 838 966	844	123	6.862	1 364 232	25.82%	1 786 794	2.837%	1 747 940	4.950%
lms	5 814 944	7 430 919	3 472	487	7.129	7 916 470	6.534%	7 867 054	5.869%	7 620 148	2.547%
rad2deg	420 104	571 931	312	56	3.501	571 931	5.025%	558 272	2.517%	548 256	0.677%
st	3 684 067	4 842 675	2 319	313	4.083	5 015 492	3.569%	4 889 868	0.974%	4 926 141	1.724%
MAPE							6.887%		3.649%		1.900%

Time is reported with unit millisecond (ms).

TACLeBench [5]. Fig. 2 illustrates a module within the self-written sample program. This particular module is responsible for generating SLTI instruction. It allows for precise control over the quantity of SLTI instructions by specifying a desired value. During compilation, the RISC-V compiler was configured as RV32I. We utilized Whisper [13] as the functional simulator to execute the ELF files. Verilator 4.028 was employed to obtain cycle-accurate RTL simulation results for the four cores. In the prediction phase, we selected 10 benchmarks from TACLeBench that differed entirely from the training software set. These benchmarks span diverse domains, including signal processing and mathematical problem-solving, and are freely available, specifically designed for embedded systems. After compilation, we again utilized Whisper as the functional simulator to execute the ELF files.

The application of the ML algorithms and PMs was programmed using Python 3.8. The algorithms were implemented with publicly available libraries, where the OLS regression and ridge regression were implemented using Scikit-learn 1.0 [14] and TensorFlow 2.6.0 [15] was used to implement LR, LR with MGD and ANN.

3.2 Performance and Simulation Time Analysis

Table 1 presents the experimental results of applying various standard benchmarks to our proposed performance estimation approach for cycle-accurate VP. The first column enumerates the names of the benchmarks, while the subsequent column displays the total instruction counts for each benchmark. Column *VP* includes the cycle count (*#Cycle*) and simulation time (*Time*) reported from VP. Column *Ours* displays the execution time of our proposed approach and the achieved speedup relative to VP. Considering that each PM requires approximately 0.06 ms to estimate all 10 benchmarks, which is negligible in comparison to the Whisper simulation time, the overall simulation time on Whisper can be interpreted as the time dedicated to estimating the number of cycles for the new software. Colums *PM1*, *PM2*, and *PM3* represent PMs based on the total instruction count, instruction format counts, and individual instruction counts, respectively. The subcolumns *#Cycle* and *APE* denote the number of cycles estimated by PM and the APE of PM compared to VP for each benchmark, respectively. The experimental results show that our approach achieves a speedup up to more than 8× in comparison to the cycle-accurate VP. Additionally, we assessed the quality of each generated PM using the MAPE metric. For the overall benchmarks, PM3 exhibits the best accuracy in performance estimation, with a MAPE of 1.900%. This superior accuracy can be attributed to leveraging more detailed runtime information (i.e., individual instruction counts) and utilizing ANN, which provide a more effective estimation model for capturing non-linear behavior.

Fig. 3 presents the results for PMs of RTL cores, depicting the APE and the corresponding speedup in comparison between the PMs and RTL cores. The x-axis represents 10 benchmarks, while the left y-axis uses a linear scale to show the results for APE in the form of bars. On the other hand, the right y-axis is logarithmically scaled to represent the results of speedup, with marked vertical lines. The logarithmic scale is chosen for the right y-axis due to its wide dynamic range and ability to visualize exponential relationships. For each core, we chose the best PM by finding the corresponding ML algorithm with the smallest MAPE. For Ibex, the PM based on the ANN is selected and the APEs are less than 3.5%, which means that this PM is capable of modeling Ibex core. Our approach achieves up to 41.4× faster simulation speed than RTL simulation using Ibex. PM for RSD is based on the ANN and the corresponding APE ranged from 7.5% to 42.0%. The algorithm's sensitivity stems from several factors. Notably, the reliance on instruction counts proves inadequate in capturing the comprehensive features influencing RSD's cycle count. Additionally, disparities in the distribution of training and testing datasets contribute to the algorithm's sensitivity. For simulation speed, our approach is 208.4× to 531.2× faster than RSD. For SweRV, the best PM is based on the LR with MGD. The APE ranges from 0.04% to 6.8%, which means that PM can roughly mimic the execution cycle behavior of the SweRV core. Our approach can simulate 80.4× to 200.6× faster than SweRV. The best PM for μRV32RTL is based on LR with MGD with a maximum APE of 1.1%, which means this PM can accurately model the μRV32RTL core in a linear fashion. Compared to μRV32RTL, our approach achieves a speedup of 563.3× to 2261.4× in simulation speed. μRV32RTL simulation is very slow compared to other cores. Therefore, for each benchmark, the PM for μRV32RTL always achieves the maximum speedup. It is meaningful to use our approach to estimate the number of cycles on μRV32RTL.

4 Conclusion and Future Work

In this paper, we present a novel ML-based approach for performance estimation of embedded software across various processors, using RISC-V processors as a case study. Our method utilizes the runtime trace of the software via a fast functional simulator to achieve accurate

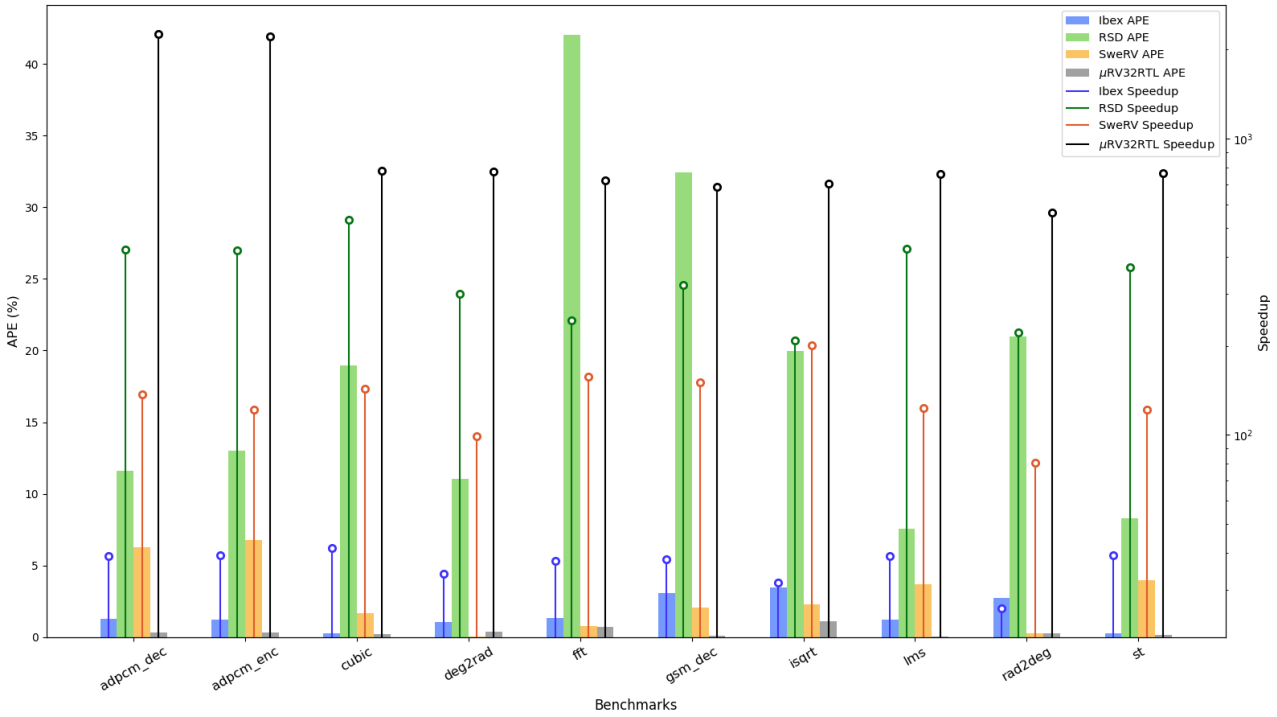

Figure 3 For the PMs of the RTL core, the APE of each benchmark, and the corresponding speedup of our approach.

performance estimation. We demonstrate its applicability through validation across a cycle-accurate simulator and four real-world, cycle-accurate RTL cores of the RISC-V ISA." The quality of our proposed approach was validated in terms of simulation speed and the accuracy of cycle count predictions, using a set of standard benchmarks.

In the future, we plan to extend this approach to other computer architectures. Furthermore, we intend to explore the use of *Large Language Models* (LLM) and fine-tune them to adapt our approach for estimating the performance of embedded software, exploring the potential efficiency gains through transfer learning.

5 Literature

[1] M. Goli and R. Drechsler, "Automated design understanding of systemc-based virtual prototypes: Data extraction, analysis and visualization," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 188–193.

[2] W. Zhang, M. Goli, and R. Drechsler, "Early performance estimation of embedded software on risc-v processor using linear regression," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 2022, pp. 20–25.

[3] W. Zhang, M. Goli, A. Mahzoon, and R. Drechsler, "ANN-based performance estimation of embedded software for risc-v processors," in *33rd International Workshop on Rapid System Prototyping (RSP)*, 2022.

[4] W. Zhang, M. Goli, M. Hassan, and R. Drechsler, "Efficient ml-based performance estimation approach across different microarchitectures for risc-v processors," in *Euromicro Conference Series on Digital System Design (DSD)*. o.A., 2023.

[5] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *International Workshop on Worst-Case Execution Time Analysis (WCET)*, ser. OpenAccess Series in Informatics (OASIcs), M. Schoeberl, Ed., vol. 55. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:10.

[6] "RISC-V GNU Compiler Toolchain," https://github.com/riscv-collab/riscv-gnu-toolchain.

[7] V. Herdt, D. Große, and R. Drechsler, "Fast and accurate performance evaluation for risc-v using virtual prototypes," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 618–621.

[8] "Ibex RISC-V Core," https://github.com/lowRISC/ibex.

[9] "RSD RISC-V Core," https://github.com/rsd-devel/rsd.

[10] "SweRV RISC-V Core," https://github.com/chipsalliance/Cores-VeeR-EH1/tree/1.0.

[11] S. Ahmadi-Pour, V. Herdt, and R. Drechsler, "The microrv32 framework: An accessible and configurable open source risc-v cross-level platform for education and research," *Journal of Systems Architecture*, vol. 133, p. 102757, 2022.

[12] W. Snyder, "Verilator," https://www.veripool.org/wiki/verilator.

[13] "Whisper ," https://github.com/chipsalliance/VeeR-ISS.

[14] "Scikit-learn," https://scikit-learn.org/stable/.

[15] "TensorFlow," https://www.tensorflow.org.