Early Performance Estimation of Embedded Software on RISC-V Processor using Linear Regression

Weiyan Zhang[1] Mehran Goli[1,2] Rolf Drechsler[1,2]
[1]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[2]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
weiyan.zhang@dfki.de {mehran, drechsler}@uni-bremen.de

***Abstract*—RISC-V-based embedded systems are becoming more and more popular in recent years. Performance estimation of embedded software at an early stage of the design process plays an important role in efficient design space exploration and reducing time-to-market constraints. Although several cycle-accurate RISC-V simulators at different levels of abstraction have been proposed, they have an inherently high cost, both for the development of the simulation setting and for obtaining the software performance in terms of the number of cycles through simulation. This results in a significant burden on designers to perform design space exploration.**

In this paper, we present a novel ML-based approach, enabling designers to fast and accurately estimate the performance of a given embedded software implemented on the RISC-V processor at the early stage of the design process. The proposed approach is evaluated against a real-world cycle-accurate RISC-V *Virtual Prototype* (VP) using a set of standard benchmarks. Our experiments demonstrate that our approach allows obtaining highly-accurate performance estimation results in a short execution time. In comparison to the cycle-accurate RISC-V VP model, the proposed approach achieves up to more than $5\times$ faster simulation speed and less than 2.5% prediction error on average.

I. Introduction

RISC-V [1] is a free and open *Instruction Set Architecture* (ISA), providing designers with a variety of advantages including openness, modularity, simplicity, and extensibility. Due to these benefits both academia and industry have shown an increasing interest in using RISC-V for their applications. The advantages of RISC-V also make it ideal for a wide variety of modern embedded systems in several application areas such as IoT and Edge devices.

In modern embedded system design, the importance of software and its impact on the overall system performance is continually increasing. Such embedded systems are designed to handle specific tasks and usually are connected to a set of tight constraints such as performance, time-to-market, and overall cost of the product [2]. Therefore, software performance analysis for fast design space exploration is a crucial issue. In order to cut down the cost of the final product and meet the design constraints, a reliable performance estimation of the software at the early stage of the design process is required. This can help make decisions of hardware/software partitioning and provides designers with information for the design space exploration of software design to achieve a high overall performance of the entire system.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project VerSys under contract no. 01IW19001, and by the University of Bremen's graduate school System Design (SyDe).

978-1-6654-9431-1/22/$31.00 ©2022 IEEE

There are various implementations of the RISC-V processor at different levels of abstraction that differ in simulation speed, level of accuracy, and implementation details [3]. They range from slow RTL implementation [4] with precise cycle count to very fast functional simulators [5] without cycle count or timing information. At the algorithmic level, functional simulators are used to validate the functionality of a given embedded software with respect to the target ISA, achieving the same functions as the modeled hardware. As they do not consider any cycle count, they have the fastest simulation speed over all types of simulators. At the *Electronic System Level* (ESL) [6], [7], *Virtual Prototype* (VP) is considered as the first implementation of the real hardware that allows cycle-accurate simulation [8], [9]. Essentially, a VP is an executable software model of the entire hardware platform that is commonly implemented in SystemC (C++-based library) [10], [11] and is widely used for hardware/software co-design [12]. While the exact number of cycles required by an embedded software can be obtained by cycle-accurate simulation on the VP, it has an inherently high cost, both for the development of the simulation setting and for obtaining the number of cycles through simulation, even when simulation parallelization is used. Therefore, the VP simulation speed is considerably slower than functional simulators.

We are motivated by the large differences in simulation speed between functional simulators and cycle-accurate simulators (even at the ESL using VP) in *Application-Specific Instruction Set Processor* (ASIP) designs that put a significant burden on design space exploration of embedded software. Hence, we aim at filling this research gap and particularly focusing on providing a reliable and fast analytical performance estimation approach targeting embedded software on RISC-V processors. In this paper, we propose a novel analytical approach by taking advantage of a hybrid technique where dynamic analysis (through a fast functional simulator and a RISC-V VP) is used for features extraction and linear regression (a fast ML-based technique) is utilized for the performance estimation model generation. The proposed approach allows designers to perform a fast and accurate performance estimation of a given embedded software on RISC-V processors at the early stage of the design process.

We evaluate the accuracy and performance of the proposed approach for two ISAs (RV32I and RV32IM) against the real-world RISC-V VP [13], [14] using different standard embedded software benchmarks from three benchmark suites: Embench [15], TACLeBench [16] and RV8-bench [17]. Experimental results demonstrate that, on average, the simulation

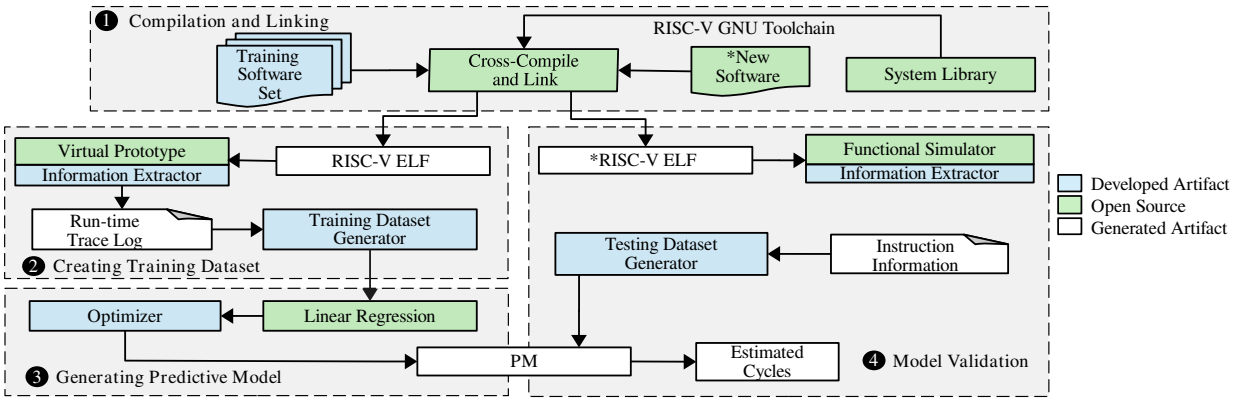

Fig. 1: An overview of the proposed performance estimation methodology workflow.

speed is $4\times$ to $5\times$ faster than VP and the obtained prediction error is less than 2.5% for the best predictive model (RV32IM).

The rest of paper is organized as follows: Section II describes the related work, Section III presents the experimental procedure, Section IV presents and analyzes the results and Section V concludes this paper and introduces future work.

II. Related Work

Early performance estimation is widely used in different applications such as parallel applications [2] and streaming applications [18]. It can be divided into two broad categories: simulation-based [13], [19] and analytical models [20], [21]. According to the different degrees of accuracy, the simulation-based approach can be classified into *Instruction Set Simulator* (ISS) and *Source-Level Timing Simulation* (SLTS). There exist a number of RISC-V ISS at different levels of abstraction ranging from slow *Register Transfer Level* (RTL) simulation with cycle accuracy such as BRISC-V [22], cycle-accurate VP models at the ESL [13], [14] to very fast functional simulation at the algorithmic level with no cycle and timing information such as RV8 [23]. VPs can provide designers with total abstract models of hardware platforms and verify software in terms of functionality and performance without any actual hardware. However, the simulation speed is considerably slower than functional simulators and analytical models. Among the aforementioned ISS techniques, the trade-off between accuracy and simulation speed is very challenging. SLTS is based on a mapping between source code and binary code and can enable high-speed performance simulation. However, it is very difficult to obtain an accurate mapping between source code and binary code, resulting in inaccurate simulation results. The method in [24] presents a transaction accurate simulation methodology by utilizing GNU gcov to profile execution statistics of given C code during the native simulation and using annotation based on the collected information.

Analytical models are powerful techniques for fast design space exploration. They can provide linear and non-linear solutions for complex systems and can be used in different architectures. In [25], performance models are built for the ARM926EJ-S and the LEON3 processor. Linear regression is performed on three different numerical features of embedded software. In [21], linear regression is used to predict the software performance and evaluate the method against ARM v5 implementation. The method in [26] derives the performance regression models and validates the models against the POWER4 RTL model and hardware implementation. The method in [27] takes advantage of neural networks to estimate the software performance and compared the results with PowerPC 750 cycle-accurate model.

To the best of our knowledge, there exists no analytical approach for performance estimation of embedded software on RISC-V processors. We aim at filling this research gap by proposing an analytical performance estimation model to obtain an approximation of the exact number of cycles of a given embedded software on RISC-V processors. The results of our approach can significantly help designers in performing a fast high-level design space exploration.

III. Performance Estimation Methodology

The proposed performance estimation framework and its overall workflow are illustrated in Fig. 1. The framework is divided into four phases which are:

1) generating a set of training embedded software and compiling them through the RISC_V GNU Toolchain,
2) creating a set of training data using the extracted data from VP as the input of the learning phase,
3) generating performance *Predictive Model* (PM)
4) validating the generated PM using new embedded software.

In the following, we explain each phase of the proposed framework in detail.

A. Source Codes Compilation and Linking

In order to obtain a robust ML-based performance estimation model that ensures the quality of results for a given new embedded software, a comprehensive set of training data is necessarily required. As shown in the first phase of Fig. 1, a comprehensive set of training software is provided ranging from simple to complex programs with different input variables that can cover all ISA instructions. In this paper, two RISC-V ISAs are considered which are the base integer instruction set (RV32I) and its M extension (RV32IM). For RV32IM, in addition to using the same method to generate training programs, part of the programs selected from standard benchmarks. Based on the binary encoding of an operation,

```
1 void ISS::performance_and_sync_update(Opcode::Mapping
      executed_op){
2  if (executed_op == Opcode::ADDI){
3   I_type += 1; // ADDI belongs to I-type format
4  } else if (executed_op == Opcode::SUB){
5   R_type += 1; // SUB belongs to R-type format
6  }
7  // catagorize other executed instructions
8 }
```

Fig. 2: Part of the *Information Extractor* module integrated into the RISC-V VP.

all instructions can be categorized into six formats; R-type, I-type, S-type, B-type, U-type, and J-type. Since each training software needs to be run on the RISC-V processor for run-time information extraction (i.e., the accurate cycle counts and trace log of instructions) in the next phase, its RISC-V compatible executable model must be generated.

To do this, we take advantage of the cross-compilation from our host computer to RISC-V processor where RISC-V GNU compiler Toolchain [28] is used to produce the RISC-V executable ELF files of each training embedded software. We also perform the same process for new software programs which are used in the performance predictive model validation phase. This is done by cross-compiling the source codes and optionally linking them with the system library. For each ISA, the RISC-V GNU compiler Toolchain is configured separately.

B. Training Dataset Creation

The performance (accurate number of cycles) of a given embedded software running on a processor is very related to the number and type of instructions. In order to create the training dataset, this information needs to be extracted. Hence, the executable file of each embedded training software generated in the previous phase is run on the RISC-V processor to obtain two goals which are extracting 1) the accurate number of cycles and 2) a detailed run-time trace of executed instructions. To do this, we take advantage of a *Virtual Prototype* (VP) implementation of the RISC-V processor. The RISC-V VP mimics the behavior of real hardware and is cycle-accurate. By running each embedded software on the VP, the accurate number of cycles is obtained. However, the VP itself does not provide designers with a detailed count of each instruction format. To retrieve this information, we implement a run-time information extractor module and integrate it into the VP. Fig. 2 shows a part of the *information extractor* module, counting the number of instructions for each RISC-V instruction format. Therefore, by executing the ELF file of each training software on the RISC-V VP, the number of instructions for each format, the total instructions count, and the total number of cycles are extracted. As illustrated in Fig. 1 – Phase 2, this information is stored in the *Run-time Trace Log* file. For each training software, the detailed parameters stored in this file are shown in Table I.

In the next step, we take advantage of the extracted information in the *Run-time Trace Log* file to build the training dataset D based on the following definition.

$$D = \{d_j | d_j = \{y_j, (x_{j,1}, x_{j,2}, ...)\}; 1 \leq j \leq N\} \quad (1)$$

TABLE I: Overview of parameters

Parameters	Description
x_1	Total instruction count
x_2	R-type: instructions using 3 register operands
x_3	I-type: instructions with immediate, loads
x_4	S-type: store instructions
x_5	B-type: branch instructions
x_6	U-type: instructions with upper immediate
x_7	J-type: jump instructions
y	Total cycles

Where D is a set of observations consists of a pair of inputs (predictors) and an output. For jth observation, the inputs are the parameters $x_{j,1}$, $x_{j,2}$ and the output is y_j.

In order to find the best performance PM, for each ISA, two different training datasets are created which are based on the total instruction count D_{total} and the number of each instruction format D_{format}. The training datasets of these two models are represented as follows:

$$D_{total} = \{d_j | d_j = \{y_j, x_{j,1}\}; 1 \leq j \leq N\} \quad (2)$$

$$D_{format} = \{d_j | d_j = \{y_j, (x_{j,2}, ..., x_{j,7})\}; 1 \leq j \leq N\} \quad (3)$$

C. Performance Predictive Model Generation

After generating the training dataset, we take advantage of machine learning techniques to find the relationship among extracted features and then use them to make predictions. The linear regression algorithm is one of the most well-known and best-understood machine learning algorithms that is widely used in different application domains for predictive models generation [29], [30]. Hence, we consider a linear regression model to estimate the executed cycles count of a given embedded software running on the RISC-V processor.

In general, linear regression [31] is a linear approach to model the relationship between independent variables (input parameters) and dependent variable (output) in a simple way. A simple linear regression model has the form as follows:

$$y = \beta_0 + \sum_{i=1}^{m} \beta_i x_i + \varepsilon \quad (4)$$

where y is the output variable and x_i are input parameters with $1 \leq i \leq m$. The parameters β_0 and β_i are called regression coefficients. The parameter ε indicates the error of model due to the lack of fit. In order to take advantage of the computation power of the computer and speed up the learning phase, we vectorize the linear model as $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ with $\mathbf{y} = \begin{bmatrix} y_1, & ... & y_j, & ... & y_N \end{bmatrix}^{\mathrm{T}}$. The parameter y_j is the output of jth embedded software. The training dataset $\mathbf{X}$ has the dimension of $N \times (m+1)$ with $\mathbf{X} = \begin{bmatrix} \underline{\mathbf{x}}_1^{\mathrm{T}}, & ... & \underline{\mathbf{x}}_j^{\mathrm{T}}, & ... & \underline{\mathbf{x}}_N^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$. For jth embedded software, the $m+1$ input parameters are formed as $\underline{\mathbf{x}}_j = \begin{bmatrix} 1, & x_{j,1}, & ... & x_{j,m} \end{bmatrix}$. The regression coefficients and errors are represented as $\boldsymbol{\beta} = \begin{bmatrix} \beta_0, & ... & \beta_m \end{bmatrix}^{\mathrm{T}}$ and $\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_0 & ... & \varepsilon_N \end{bmatrix}^{\mathrm{T}}$, respectively. The parameter N indicates the number of embedded software compiled and executed on RISC-V VP to generate the training dataset.

In *Optimizer* module (Fig. 1 – Phase 3), the suitable regression coefficients are calculated by minimizing the *Mean Squared Error* (MSE) illustrated in (5) interactively until it reaches convergence.

Algorithm 1 Generating Predictive Model

Data: training dataset $D = \{\mathbf{y}, \mathbf{X}\}$, learning rate η, maximum number of iterations K, cost function J
Output: PM

1: initialize regression coefficients of β with uniform distribution from range $\left(-\sqrt{\frac{6}{m+1}}, +\sqrt{\frac{6}{m+1}}\right)$
2: **for** iteration in $1...K$ **do**
3: $\quad\hat{\mathbf{y}} \leftarrow \mathbf{X}\boldsymbol{\beta}$
4: $\quad J \leftarrow f_1\left(\mathbf{y}, \hat{\mathbf{y}}\right)$
5: $\quad\beta_0 \leftarrow \beta_0 - \eta\frac{\partial J}{\partial \beta_0}$
6: $\quad$**for** i in $1...m$ **do**
7: $\quad\quad\beta_i \leftarrow \beta_i - \eta\frac{\partial J}{\partial \beta_i}$
8: $\quad$**end for**
9: $\quad$convergence test: calculate J
10: **end for**
11: **return** $\beta_0, ..., \beta_m$
12: $R^2 \leftarrow f_2\left(\mathbf{y}, \hat{\mathbf{y}}\right)$
13: PM $\leftarrow \beta_0, ..., \beta_m$

$$MSE = \frac{1}{N}\left\|\boldsymbol{\varepsilon}\right\|_2^2 \tag{5}$$

In order to assess the strength of the linear relationship between inputs and the corresponding output and indicate how good the performance of the model is, the coefficient of determination R^2 is considered. The metric R^2 is represented as a value between zero and one, where the value of one indicates a perfect fit and can be used for a highly accurate model for future estimation, while the value of zero indicates that the model fails to accurately model the training dataset at all. The metric R^2 is defined based on the following equation:

$$R^2 = 1 - \frac{\left\|\boldsymbol{\varepsilon}\right\|_2^2}{\left\|\mathbf{y} - \bar{\mathbf{y}}\right\|_2^2} \tag{6}$$

where $\bar{\mathbf{y}}$ is the mean of output vector $\mathbf{y}$.

Algorithm 1 shows the method used in the *Optimizer* module to generate the PM using the training dataset D, learning rate η, cost function J and the maximum number of iterations K. The parameter $\hat{\mathbf{y}}$ is the vector of estimated outputs. The algorithm consists of three main steps in each iteration. First, the estimated output vector and cost function based on current regression coefficients are calculated (Line 3 to 5) The cost function is defined as MSE. In the next step, each regression coefficient is updated based on the learning rate and cost function (Line 5 to 8). The learning rate is in the range between zero and one. It determines how quickly the cost function can achieve convergence. Finally, when the cost function converges, the regression coefficients are obtained (Line 9). The coefficient of determination can be calculated and PM is expressed in mathematical form as follows:

$$\hat{y} = \beta_0 + \sum_{i=1}^{m}\beta_i x_i \tag{7}$$

D. Performance Predictive Model Validation

In this phase, the PM is tested by new embedded software. The new software is compiled and the corresponding RISC-V ELF file is generated as described in Section III-A. Since the generated ELF file is compact and is not a human-readable file, we take advantage of RV8 [23] – an open-source RISC-V functional simulator – and integrated it into our framework to provide an expanded and unrolled assembly version of the ELF file. In general, analyzing the result of disassembler (a compact assembly code) is very difficult as it does not contain information such as the length of the loop and jump instructions based on the comparison. On the other hand, the trace result of the RV8 is a flattened and unrolled version of the assembly code where a detailed trace of the instructions is available. By this, it is possible to analyze the generated file and extract the required information to be passed as the input of PM for the performance prediction of the given software.

While RV8 allows fast execution of the ELF file and supports most extensions and the ability to trace the software execution and register values, it does not provide any information about the numbers of each instruction format. Hence, as illustrated in Fig. 1 – Phase 4, the *Information extractor* module is integrated into RV8 in a similar way as shown in Fig. 2. After executing the ELF file on RV8, the total instruction count and the number of instructions for each format are extracted and stored in the *Instruction Information* file. Then, the testing dataset D_{test} is generated in a similar way as the training dataset, except that the cycle count is not included. It is represented as the following equation.

$$D_{test} = \{x_1, ..., x_m\} \text{ with } m = \begin{cases} 1; \text{ total instruction count} \\ 6; \text{ different format counts} \end{cases} \tag{8}$$

where x_1 to x_m are input variables and parameter m indicates the number of input variables. According to the PM generated in third phase and the testing dataset, the total cycle count of new software $\hat{y}$ is estimated based on (7).

IV. Experimental Evaluation

This section presents the experimental evaluation of our proposed performance estimation approach. The benchmarks are provided by Embench [15], TACLeBench [16] and RV8-bench [17], covering different domains, such as filters, matrix manipulations, and sorting algorithms. They are freely available and specifically target embedded systems. We evaluated the obtained results of the generated PMs by comparing them against the real-world RISC-V VP model [13], [14] in terms of simulation speed and number of cycles accuracy. Two ISAs are considered by configuring RISC-V GNU Toolchain with different parameters. For each ISA, two PMs were generated based on the total number of instructions and the number of instruction formats. Hence, we divide the experimental evaluation into two parts. First, characteristics of PMs for RV32I ISA are presented. Second, we explain the features of the generated PMs for RV32IM ISA. The underlying linear regression for each PM was implemented using TensorFlow [32]. All experiments were carried out on a Linux system with 38 GB RAM and an AMD Ryzen 7 PRO 4750U processor running at 1.4 GHz.

Table II and Table III illustrate the experimental results of applying different standard benchmarks to our proposed performance estimation approach. The first column lists the names of benchmarks. The following two columns shows the total instruction counts and *Lines of Code* (LoC) of each benchmark. Column *VP* includes the cycle count (*# Cycle*) and simulation time (*Time*) reported from VP. Column *Ours* shows the execution time of the proposed approach and the obtained speed-up compared to VP. The reported execution

TABLE II: Experimental Results of all Benchmarks used for Validation of PMs on RV32I

Benchmark	#instr-exec.	LoC	VP		Ours		PM1		PM2	
			#Cycle	Time (ms)	Time (ms)	Speed-up	#Cycle	Error	#Cycle	Error
adpcm_enc[2]	2 898 910	413	3 636 185	1 877	561	3.346	3 946 579	+8.54%	3 857 302	+6.08%
ammunition[2]	724 331 550	2 449	1 046 797 153	511 228	119 414	4.281	986 105 480	-5.80%	1 010 774 689	-3.44%
cubic[1]	32 156 510	125	39 541 102	19 168	5 268	3.645	43777896	+10.71%	42 008 676	+6.24%
dhrystone[3]	7 160 022 427	241	11 380 028 888	5 508 120	1 199 379	4.592	9 747 659 677	-14.34%	11 137 231 472	-2.13%
gsm_enc[2]	26 855 420	1 793	34 399 663	17 019	4 299	3.959	36 560 989	+6.28%	36 729 549	+6.77%
miniz[3]	5 155 692 007	6 558	8 392 723 486	3 847 009	930 693	4.133	7 018 964 499	-16.37%	7 600 800 104	-9.44%
norx[3]	14 809 875	368	21 410 709	13 293	2 624	5.066	20 162 145	-5.83%	20 626 833	-3.66%
picojpeg[1]	13 751 505	263	22 157 267	10 300	2 156	4.777	18 721 310	-15.51%	22 923 623	+3.46%
st[1]	18 327 848	117	23 469 543	10 807	3 083	3.501	24 951 546	+6.31%	23 874 972	+1.73%
ud[1]	14 192 285	95	20 543 388	9 672	2 369	4.083	19 321 388	-5.95%	19 516 397	-4.50%
MAE								9.56%		4.50%
R^2							0.950		0.998	

The Benchmarks are provided by [1]Embench [15], [2]TACLeBench [16] and [3]RV8-bench [17]. **Time** is reported with unit millisecond (ms).

TABLE III: Experimental Results of all Benchmarks used for Validation of PMs on RV32IM

Benchmark	#instr-exec.	LoC	VP		Ours		PM1		PM2	
			#Cycle	Time (ms)	Time (ms)	Speed-up	#Cycle	Error	#Cycle	Error
ammunition[2]	329 312 597	2 449	535 361 750	232 868	55 143	4.223	492 679 360	-7.97%	533 658 750	+0.32%
audiobeam[2]	3 815 806	6 649	3 815 806	2 582	671	3.848	5 708 771	-4.71%	5 477 313	-0.47%
cosf[2]	279 995	631	385 620	210	60	3.500	418 900	+8.63%	370 838	-3.83%
cubic[3]	8 369 615	125	11 824 285	5 563	1 434	3.880	12 521 652	+5.90%	12 113 882	+2.45%
epic[2]	34 890 806	982	47 062 629	21 516	5 502	3.906	52 199 584	+10.91%	45 778 636	-2.73%
fft[2]	3 056 010	492	4 414 382	2 017	511	4.123	4 572 053	+3.57%	4 303 558	-2.51%
fmref[2]	6 402 343	684	8 845 407	4 031	1 054	3.834	9 578 448	+8.29%	8 618 082	-2.57%
g723_enc[2]	859 693	263	1 447 336	685	170	4.029	1 286 178	-11.13%	1 529 837	+5.70%
lms[2]	2 251 965	114	3 043 479	1 468	386	3.803	3 369 134	+10.70%	3 079 731	+1.19%
st[2]	1 937 169	127	2 664 519	1 263	323	3.827	2 898 173	+8.77%	2 585 080	-2.98%
MAE								8.06%		2.48%
R^2							0.955		0.996	

The Benchmarks are provided by [1]Embench [15], [2]TACLeBench [16] and [3]RV8-bench [17]. **Time** is reported with unit millisecond (ms).

time consists of two parts that are 1) the required time for RV8 to generate the run-time trace of a given software, and 2) the time used by PMs to estimate the cycle count of the new software. Since the time used by PMs is negligible in comparison to RV8 simulation time, the simulation time on RV8 can be seen as the time used to estimate the number of cycles of the new software. Columns *PM1* and *PM2* represent the predictive models based on the total number of instructions and the number of instruction formats, respectively. The following sub columns *Cycle* and *Error* indicates the number of estimated cycles and the error percentage of each software in comparison to the VP, respectively. In addition, we evaluated the quality of each generated PM using two metrics *MAE* and R^2 introduced in (5) and (6), respectively. The average values for the above metrics are reported in Table II and Table III.

A. Predictive Models for RV32I ISA

In this experiment, we took advantage of 125 programs as training software to create a training dataset. The total instruction counts of the training software set range from 1.5×10^6 to 3.7×10^7. For the validation phase, 10 benchmarks were used for each PM. Please note that the software used in the validation phase was completely different from the software used in the training phase to create the training dataset. The generated PM based on the total number of instructions is presented as the following equation.

$$\hat{y} = 2.36 + 1.36 * x_1 \tag{9}$$

The generated PM based on the number of instruction formats is shown in (10). The absolute values of the regression coefficient are used to rank the inputs. Due to its larger coefficient, the fifth input variable (x_6), i.e. U-type count, appears to be more important for determining the number of cycles.

$$\hat{y} = 2.46 - 0.82 * x_2 + 1.94 * x_3 + 2.11 * x_4 + 1.20 * x_5 + 2.76 * x_6 - 2.6 * x_6 \tag{10}$$

The experimental results in Table II show that our approach achieves a speed-up up to more than $5\times$ in comparison to the cycle-accurate VP. While the quality of both PMs based on the average *MAE* and R^2 metrics lays in acceptable boundaries, the linear relationship of PM2 is stronger than PM1 due to its higher R^2 (0.998) and lower average *MAE* (4.50%). This can also be seen in the Table where for each benchmark, PM2 provides better estimation results than PM1 and has a smaller error rate. The main reason for the high prediction accuracy is that PM2 takes advantage of the instruction formats classification in the training phase where more detailed information for the underlying regression algorithm is provided.

B. Predictive Models for RV32IM ISA

In the second experiment, 80 programs were used as training software to create the training dataset. The total instruction counts of the training software set range from 2.0×10^5 to 1.1×10^8. The PMs are validated using 10 benchmarks. The software programs used for validation were completely different from the software used to generate the training dataset. The following equation can be used to calculate the PM based on the total number of instructions.

$$\hat{y} = 6.29 + 1.50 * x_1 \tag{11}$$

The generated PM based on the number of instruction formats is expressed as (12). The third input variable (x_4), S-type count, has a greater impact on determining the number of cycles due to its greater coefficient.

$$\hat{y} = 17.32 + 1.16 * x_2 + 1.70 * x_3 + 4.29 * x_4 - 0.43 * x_5 - 0.52 * x_6 + 1.06 * x_6 \tag{12}$$

The experimental results in Table III demonstrate that our proposed approach can estimate the number of cycles up to $4.2\times$ faster than VP. Similar to the first experiment, for each benchmark, PM2 has higher accuracy than PM1 thanks to the instruction formats classification in its training phase where more detailed information for the underlying regression algorithm is provided. While the quality of both PMs based on the average *MAE* and R^2 metrics lays in acceptable boundaries, the linear relationship of PM2 is stronger than PM1 due to its higher R^2 (0.996) and lower average *MAE* (2.48%).

The results of both experiments demonstrate that the generated PMs (especially PM2 generated based on the instruction formats classification) can accurately predict the embedded software performance. The remaining accuracy gap is mostly due to the following reasons: 1) the similarity between the training software set and the testing benchmarks can be large for some cases and 2) more input variables are needed. For both cases, adding more training software from various domains can overcome these issues and improve the quality of the estimation results.

Overall, the proposed approach allows designers to analyze the performance of a given embedded software on RISC-V processors up to more than $5\times$ faster than even cycle-accurate VP at the ESL (which have orders of magnitude faster simulation speed than RTL simulators). The gain of performance using our proposed performance estimation approach is paid with a slight loss of accuracy, but the estimates are accurate enough to perform design space exploration of embedded software and overall help designers to make high-level decisions of system designs.

V. Conclusion and Future Work

In this paper, we proposed a novel analytical approach to estimate the performance of a given embedded software on RISC-V processors. The proposed approach is based on applying linear regression along with a functional simulator to obtain the number of cycles. The quality of the proposed approach for two ISAs was validated against a real-world cycle-accurate RISC-V VP model in terms of simulation speed and number of cycles accuracy using several standard benchmarks. We demonstrated the applicability and effectiveness of linear regression in the reliable estimation of embedded software performance. Moreover, we have shown applying machine learning in performance estimation can balance the simulation cost and prediction accuracy at the early stage. High-level performance estimation allows designers to perform design space exploration, where fast evaluations of the software performance are required, in order to decide on the adequacy of a given processor or of a given partitioning of tasks among processors, w.r.t system performance requirements. As future work, we plan to extend our approach to more RISC-V extensions, such as F and D extensions.

References

[1] "RISC-V ISA Specification," https://riscv.org/technical/specifications/, 2019.

[2] J. Flores-Contreras, H. A. Duran-Limon, A. Chavoya, and S. H. Almanza-Ruiz, "Performance prediction of parallel applications: a systematic literature review," *Journal of Supercomputing*, vol. 77, no. 4, pp. 4014–4055, 2021.

[3] "RISC-V Software Ecosystem Overview," https://github.com/riscvarchive/riscv-software-list.

[4] Y. Chi, Y.-k. Choi, J. Cong, and J. Wang, "Rapid cycle-accurate simulator for high-level synthesis," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019, pp. 178–183.

[5] J. T. Taylor and W. T. Taylor, "Functional simulator," in *Patterns in the Machine*. Springer, 2021, pp. 97–106.

[6] G. Martin, B. Bailey, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[7] 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)*, 2020, pp. 188–193.

[8] ——, *Automated Analysis of Virtual Prototypes at the Electronic System Level: Design Understanding and Applications*. Springer Nature, 2020.

[9] M. Goli, J. Stoppe, and R. Drechsler, "Automated nonintrusive analysis of electronic system level designs," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 39, no. 2, pp. 492–505, 2020.

[10] "IEEE standard for standard SystemC language reference manual," *IEEE Computer Society*, 2012.

[11] M. Goli, J. Stoppe, and R. Drechsler, "AIBA: an Automated Intra-cycle Behavioral Analysis for SystemC-based design exploration," in *International Conference on Computer Design (ICCD)*, 2016, pp. 360–363.

[12] T. De Schutter, *Better Software. Faster!: Best Practices in Virtual Prototyping*. Happy About, 2014.

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

[14] "RISC-V VP," https://github.com/agra-uni-bremen/riscv-vp.

[15] "Embench," https://github.com/embench/embench-iot.

[16] "TACLeBench," https://github.com/tacle/tacle-bench.

[17] "RV8-bench," https://github.com/michaeljclark/rv8-bench.

[18] M. Flasskamp, G. Sievers, J. Ax, C. Klarhorst, T. Jungeblut, W. Kelly, M. Thies, and M. Porrmann, "Performance estimation of streaming applications for hierarchical MPSoCs," in *Rapid Simulation and Performance Evaluation: Methods and Tools*, 2016, pp. 1–6.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[20] M. Lattuada and F. Ferrandi, "Performance estimation of embedded software with confidence levels," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012, pp. 573–578.

[21] B. Franke, "Fast cycle-approximate instruction set simulation," in *Software & Compilers for Embedded Systems*, 2008, pp. 69–78.

[22] S. Bandara, A. Ehret, D. Kava, and M. A. Kinsy, "BRISC-V: an open-source architecture design space exploration toolbox," *CoRR*, vol. abs/1908.09992, 2019. [Online]. Available: https://ascslab.org/research/briscv/index.html

[23] R. Simulator, https://github.com/michaeljclark/rv8.

[24] K. Huang, X. Zhang, S. Xiu, D. Zheng, M. Yu, D. Ma, K. Huang, G. Chen, and X. Yan, "Profiling and annotation combined method for multimedia application specific mpsoc performance estimation," *Frontiers Inf. Technol. Electron. Eng.*, vol. 16, no. 2, pp. 135–151, 2015.

[25] M. Lattuada and F. Ferrandi, "Performance modeling of embedded applications with zero architectural knowledge," in *International Conference on Hardware/Software Codesign and System Synthesis*, 2010, pp. 277–286.

[26] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," *ACM SIGOPS operating systems review*, vol. 40, no. 5, pp. 185–194, 2006.

[27] M. S. Oyamada, F. Zschornack, and F. R. Wagner, "Applying neural networks to performance estimation of embedded software," *Journal of Systems Architecture*, vol. 54, no. 1-2, pp. 224–240, 2008.

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

[29] M. Goli and R. Drechsler, "PREASC: Automatic portion resilience evaluation for approximating SystemC-based designs using regression analysis techniques," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 5, 2020.

[30] M. Goli, J. Stoppe, and R. Drechsler, "Resilience evaluation for approximating SystemC designs using machine learning techniques," in *Symposium on Rapid System Prototyping (RSP)*, 2018, pp. 97–103.

[31] A. C. Rencher and G. B. Schaalje, *Linear models in statistics*. John Wiley & Sons, 2008.

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